ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення автоматизованих систем

Пояснювальна записка

до кваліфікаційної роботи на
ступінь вищої освіти магістр

на тему: «**Розробка інформаційної системи оптимізації маршрутів на основі об'єктно-орієнтованого програмування**»

Виконав: студент 6 курсу, групи ІСДМ–61
спеціальності
126 Інформаційні системи та технології
(шифр і назва спеціальності)

ШітівМ.М
(прізвище та ініціали)

Керівник Сеньков О.В
(прізвище та ініціали)

Рецензент_____
(прізвище та ініціали)

Київ – 2023

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра_Інформаційних систем та технологій_____
Ступінь вищої освіти – «Магістр»_____
Спеціальність підготовки _126 Інформаційні системита технології_____

ЗАТВЕРДЖУЮ
Завідувач кафедри ІСТ
_____ К.П.Сторчак
"____" _____2022 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

_____Шітів Михайло Миколайович_____
(прізвище, ім'я, по-батькові)

1. Тема роботи: «Розробка інформаційної системи оптимізації маршрутів на основі об'єктно-орієнтованого програмування.»
Керівник роботи: Сеньков Олег Вікторович, доцент кафедри, _____
(прізвище, ім'я, по-батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від _____року №
2. Строк подання студентом роботи _____
3. Вхідні дані до роботи:
1. Visual Studio 2019_____
2. СУБД MS Access_____
3. Науково-технічна література з питань, пов'язаних з темою роботи._____

4. Зміст розрахунково пояснювальної записки (перелік питань, які потрібно розробити):
1. Аналіз алгоритмів пошуку оптимального шляху_____
2. Дослідити технології зручну для використання_____
3. Розробка та тестування інформаційної системи_____

5. Перелік графічного матріалу
1. Передумови вибору теми_____
2. Результати аналізу та тестування алгоритмів_____
3. Презентація фунціоналу інформаційної системи_____

6. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір ноуково-технічної літератури	02.11.2021 - 05.11.2021	
2	Вивчення матеріалів для подальшої взаємодії з ними	05.11.2021 - 09.11.2021	
3	Аналіз алгоритмів пошуку оптимального шляху	09.11.2021 - 13.11.2021	
4	Вибір технологій та середовища проектування	13.11.2021 - 14.11.2021	
5	Розробка інформаційної системи	14.11.2021 - 21.12.2021	
6	Тестування інформаційної системи	21.12.2021 - 23.12.2021	
7	Вступ, висновки, реферат	23.12.2021 - 24.12.2021	
8	Розробка демонстраційних матеріалів	24.12.2021 - 25.12.2021	
9	Попередній захист роботи	27.12.2021	

Студент _____ Шітів М.М.
(Підпис) (прізвище та ініціали)

Керівник роботи _____ Сеньков О.В.
(Підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина магістерської роботи : 140 с., 8 табл., 30 рис., 1 дод., 37 джерел.

БАЗА ДАНИХ, МАРШРУТИ, МІСТА, ІНФОРМАЦІЙНА СИСТЕМА, ПЕРЕВЕЗЕННЯ, ОПТИМАЛЬНИЙ ШЛЯХ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ.

Об`єкт дослідження – оптимізація маршрутів на основі об'єктно-орієнтованого програмування.

Предмет дослідження – методи та засоби оптимізації маршрутів.

Мета роботи – є розробка програмного забезпечення оптимізації маршрутів на основі досліджених алгоритмів та аналогічних систем.

Методи дослідження —системний аналіз, оптимізація, методики розробки інтелектуальних інформаційних систем.

Проведено дослідження перспективних алгоритмів для визначення оптимальних маршрутів, показало що алгоритм Дейкстри показує високу ефективність у задачах прокладання маршрутів з великою кількістю міст. Головна його перевага - можливість використання для вирішення складних неформалізованих проблем для яких не існує приватних методів вирішення, що дозволяє ефективно вирішувати нестандартні завдання. Отже, для розробки інформаційної системи оптимізації маршрутів було взято за основу реалізацію алгоритму Дейкстри.

В результаті проведеної роботи вирішено актуальне технічне завдання для оптимізації маршрутів пасажирських та транспортних перевезень. У процесі вирішення завдання розроблено інженерну методику автоматизованої процедури проведення симуляції маршрутів та розрахунку найкоротших маршрутів між містами та населеними пунктами.

Зміст

ВСТУП

Маршрутизація – це процес визначення основі даних з таблиці маршрутизації оптимального шляху від вузла-джерела до вузлу-одержувачу за умов надлишкових зв'язків [1].

У процесі побудови маршрутизації виділяють дві частини: визначення подальшого шляху пакета та безпосередньо його пересилання цим шляхом.

Відповідно до цих частин, процес маршрутизації можна розділити на два ієрархічно пов'язані рівні:

– рівень маршрутизації. На цьому рівні відбувається робота з таблицею маршрутизації. Таблиця маршрутизації служить визначення мережного адреси наступного маршрутизатора, який буде переданий пакет, чи безпосередньо одержувача пакета. Після визначення адреси передачі вибирається певний вихідний фізичний порт маршрутизатора передачі мережного пакета. Цей процес називається визначенням маршруту переміщення пакета. Настроювання таблиці маршрутизації ведеться протоколами маршрутизації. На цьому ж рівні визначається перелік необхідних сервісів, що надаються;

– рівень передачі пакетів. Перед тим, як передати пакет, необхідно перевірити контрольну суму заголовка пакета, визначити адресу (канального рівня) одержувача пакета і здійснити безпосередньо відправлення пакета з урахуванням черговості, фрагментації, фільтрації та інших дій. Ці дії виконуються на підставі команд, які надходять із рівня маршрутизації.

Маршрутизація є однією з найважливіших процедур передачі даних. Процедура маршрутизації гарантує, що дані переміщуються з однієї мережі до іншої з оптимальною швидкістю та мінімальною затримкою. Цілісність даних у процесі побудови маршрутизації гарантована.

Алгоритм маршрутизації – частина програмного забезпечення мережного рівня, і відповідає за визначення за якою лінією відправляти пакет

далі [2]. Незалежно від того чи вибирається маршрут для сесії або для кожного пакета алгоритм маршрутизації повинен мати ряд властивостей: коректність, простота, стійкість, стабільність, справедливість і оптимальність.

Не адаптивні алгоритми не враховують поточне завантаження мережі та стан топології. Усі можливі маршрути обчислюються заздалегідь і завантажуються маршрутизатори під час завантаження мережі. Така маршрутизація називається статичною маршрутизацією.

Адаптивні алгоритми маршрутизації, навпаки, визначають маршрут виходячи з поточного завантаження мережі та топології. Адаптивні алгоритми відрізняються тим, де і як вони отримують інформацію (локально від сусідніх маршрутизаторів або глобально від усіх), коли вони змінюють маршрут (кожні ΔT секунд, коли змінюється навантаження, коли змінюється топологія), яка метрика використовується при оптимізації (відстань, кількість стрибків) , очікуваний час передачі).

Маршрутизація по вектору відстані - це алгоритм маршрутизації, ідея якого в тому, що кожен маршрутизатор в підмережі має таблицю відстаней до кожного маршрутизатора в підмережі. Періодично маршрутизатор обмінюється інформацією зі своїми сусідами та оновлює інформацію у таблиці. Кожен елемент таблиці складається з двох полів: перше – номер лінії, по якій треба відправляти пакети, щоб досягти потрібного місця, друге – величина затримки до місця призначення. Ця величина затримки може бути виміряна у різних одиницях: стрибках, мілісекундах, довжині черги на лінії тощо.

1 ЗАДАЧА МАРШРУТИЗАЦІЇ ТА АЛГОРИТМИ ЇЇ РОЗВ'ЯЗАННЯ

1.1 Транспортні перевезення та оптимізація маршрутної мережі

Транспорт - це галузь матеріального виробництва, що здійснює перевезення людей та вантажів [3].

У структурі суспільного виробництва транспорт відноситься до сфери виробництва матеріальних послуг.

Транспортна логістика є різновидом прикладної логістики, причому її матеріальний аспект проявляється у вигляді надання матеріальних і транспортних послуг [4]. До матеріальних послуг відносяться види діяльності, при яких не відбувається перетворення форм матерії та продуктом яких є особлива споживча вартість, що виражається у суспільній корисності самої праці. До матеріальних послуг відносяться види діяльності, що тягнуть за собою збільшення вартості раніше створених благ, наприклад, зберігання, транспортування, упаковка та ін. У загальному випадку послуга визначається як діяльність, пов'язана з обміном цін і спрямована на задоволення попиту споживачів, при якій не відбувається передача права власності на конкретний матеріал продукт. Транспортні послуги є особливим видом діяльності транспорту, що потребує відповідного технологічного, фінансового, інформаційного, правового та ресурсного забезпечення. У поняття послуг транспорту також входять супутні операції, пов'язані з підготовкою та здійсненням перевізного процесу, вантажно-розвантажувальні операції, упаковка вантажів, інформаційне забезпечення (моніторинг послуги) та ін. Транспортні послуги мають низку особливостей, які необхідно враховувати при застосуванні логістичних принципів у процесі управління підприємством транспортного комплексу, а саме [4]:

– послуга існує лише у процесі її виробництва, а отже, не може накопичуватися;

– якість послуги визначається якістю процесу надання послуги, оскільки продаж послуги є продаж самого процесу її надання;

– послуга має споживчу вартість у певний час на певному напрямку, що обмежує можливість її заміни;

– існує нерівномірність попиту послуги, як тимчасова, і просторова;

– пропозиція, як правило, не володіє достатньою гнучкістю в пристосуванні до попиту, що швидко змінюється;

– можливості транспорту для згладжування коливань попиту обмежені.

Якість транспортних послуг окреслюється відповідність рівня споживчих властивостей послуги вимогам ринку. Якість послуг оцінюється за різницею двох умовних величин – очікування споживача та фактичних параметрів. Ця різниця називається розбіжністю, вона оцінює ступінь задоволеності покупця якістю послуги.

Серед найважливіших параметрів якості послуг виділяють такі [5]:

– відчутність, тобто. середовище послуги (обладнання, зовнішній вигляд персоналу, інтер'єр офісу, забезпечення моніторингу послуги та ін.);

– надійність – послідовність виконання «точно вчасно»;

– відповідальність – гарантії виконання послуг;

– доступність - простота процедури встановлення контактів з підприємством, надання клієнту вибору зручного часу надання послуг;

– безпека – відсутність ризику та недовіри з боку клієнта;

– ввічливість та комунікабельність персоналу, взаєморозуміння з покупцем.

До завдань функціонування транспортного підприємства належать раціональна організація перевізного процесу, спільне планування транспортних, виробничих та інших процесів. Таким чином, при моделюванні процесів у транспортних логістичних системах використовуються як методи розв'язання задач функціонування, так і методи оптимізації транспортного підприємства.

При моделюванні стану та функціонування транспортних логістичних систем застосовується два підходи: детерміністсько-оптимальний та імовірнісно-адаптивний.

Використання детерміністсько-оптимального підходу при оптимальному плануванні дозволяє отримати найкращі варіанти планів, а застосування методів економіко-математичного моделювання – вибирати показники плану, що варіюються, за умовами екстремуму вжитої міри його ефективності [6]. Моделювання на основі детерміністсько-оптимального підходу дозволяє отримати варіанти розвитку транспортного підприємства з урахуванням зміни як стану системи, так і її функціонування.

Головною умовою досягнення високої ефективності управління є взаємозалежна оптимізація функціонування та стану системи. Однак у рамках однієї моделі вирішити це завдання неможливо. Тому необхідно розбиття спільної задачі на кілька локальних, що входять до загальної системи завдань транспортної логістики [7]. Недоліками детерміністсько-оптимального підходу є неможливість вирішення тих проблем прийняття рішень, які в даний час не можуть бути математично формалізовані, а також відмова від аналізу та вдосконалення організаційних структур.

Імовірнісно-адаптивний підхід до моделювання завдань підприємства, крім володіння всіма перевагами детерміністсько-оптимального підходу, характеризується такими особливостями [8]:

– дозволяє створювати людино-машинні системи планування з метою більш повного та ефективного використання у процесі планування досвіду фахівців;

– забезпечує персоніфікацію плану як системи взаємопов'язаних рішень;

– дозволяє розглядати організаційні проблеми;

– допускає облік випадкових факторів при виборі найбільш адаптивних варіантів планів.

Функціонування транспорту носить переважно адаптивний характер. Однак процеси, що включають елементи невизначеності, на транспорті не є суто випадковими процесами, і роль організаційної складової в них є надзвичайно високою. При виборі цієї моделі, розроблені тільки на основі імовірнісного або детермінованого підходу, часто не відповідають транспортній системі.

Тому моделювання розвитку підприємств транспорту має здійснюватися з використанням інтегрованого підходу, шляхом поєднання кількох моделей, які спроможні виробляти рішення щодо ефективного розвитку, так і описувати процеси його адаптації до умов зовнішнього середовища, що змінюються, в умовах невизначеності та нестачі інформації.

Значна частина логістичних операцій на шляху руху матеріального потоку від первинного джерела сировини до кінцевого споживання здійснюється із застосуванням різних транспортних засобів. Витрати виконання цих операцій становлять до 50% від суми загальних витрат за логістику [9].

Транспорт органічно вписується у виробничі та торговельні процеси. Тому транспортна складова бере участь у багатьох задачах логістики [10]. Разом з тим існує досить самостійна транспортна галузь логістики, в якій багатоаспектна узгодженість між учасниками транспортного процесу може розглядатися поза прямим зв'язком з пов'язаними виробничо-складськими ділянками руху матеріального потоку.

До завдань транспортної логістики насамперед відносять завдання, вирішення яких посилює узгодженість дій безпосередніх учасників транспортного процесу [11].

Застосування логістики в транспорті, як і у виробництві чи торгівлі, перетворює контрагентів і конкуруючих сторін на партнерів, взаємодоповнюють одне одного у транспортному процесі.

Логістика, як зазначалося, це єдина техніка, технологія, економіка та планування. Відповідно, до завдань транспортної логістики слід віднести забезпечення технічного та технологічного поєднання учасників транспортного процесу, узгодження їх економічних інтересів, а також використання єдиних систем планування [12]. Коротко охарактеризуємо кожне із цих завдань.

Технічна сполученість у транспортному комплексі означає узгодженість параметрів транспортних засобів як усередині окремих видів, і у міжвидовому розрізі. Ця узгодженість дозволяє застосовувати модальні перевезення, працювати з контейнерами та вантажними пакетами. Технологічна сполученість має на увазі застосування єдиної технології транспортування, прямі навантаження, без перевантаження повідомлення.

Економічна сполученість - це загальна методологія дослідження кон'юнктури ринку та побудови тарифної системи.

Спільне планування означає розробку та застосування єдиних планів графіків.

До завдань транспортної логістики відносять також [13]:

– створення транспортних систем, у тому числі створення транспортних коридорів та транспортних ланцюгів;

– забезпечення технологічної єдності транспортно-складського процесу;

– спільне планування транспортного процесу зі складським та виробничим;

– вибір виду транспортного засобу;

– вибір типу транспортного засобу;

– визначення раціональних маршрутів доставки та ін.

Специфіка логістичного підходу у питаннях організації та управління громадським пасажирським транспортом.

Стосовно пасажирського транспорту логістика є сукупністю проектних рішень, технічних засобів і методів організації та управління, які забезпечують заданий рівень обслуговування пасажирів, їх безпечну та безперервну доставку «від дверей до дверей» у певний час за мінімальних витрат [14]. Застосування логістики на пасажирському транспорті дозволяє оптимізувати перевізний процес, що розглядається як логістична система операторів та об'єктів інфраструктури, через логічні зв'язки, що беруть участь у процесі надання транспортних послуг.

Створення раціональної транспортної системи як регіону, і міста передбачає використання логістичного підходу ще етапі проектування і проведення містобудівних робіт. Це дозволяє суттєво скоротити потребу населення перевезень шляхом наближення місця проживання до місць праці, проведення дозвілля тощо, і навпаки. Структура проектованої пасажирської транспортної мережі має будуватися за принципом скорочення повних витрат часу пасажира, включаючи час підходу до пункту зупинки, час очікування транспортного засобу, час поїздки, час пересадки тощо.

Логістичний підхід до створення технічної інфраструктури пасажирського транспорту полягає у забезпеченні найкоротших зв'язків між основними пасажирськими пунктами, в обладнанні цих пунктів необхідними спорудами, обліку обсягів пасажиропотоків та вимог комфортного проїзду при розрахунку та виборі оптимального рухомого складу та типів транспортних засобів.

Вирішення транспортних проблем регіонів і великих міст з інфраструктурою, що історично склалася, істотно ускладняється необхідністю забудови житлових масивів на значній відстані від місць концентрації виробництва. У зв'язку з цим збільшується потреба у перевезеннях, підвищується транспортна втома пасажирів.

Нині спрощена класифікація кореспонденції пасажирів передбачає такі переміщення залежно від мети поїздки: трудові (зокрема і навчання), ділові

(службові поїздки протягом робочого дня) і соціальні чи культурно- побутові. Глибшою є просторово-часова класифікація поїздок. Класифікація за часом передбачає використання двох характеристик поїздки – періодичність поїздки та фіксованість поїздки за часом доби. За ознакою періодичності розрізняють поїздки постійні, що характеризуються сезонною нерівномірністю, періодичні та разові. Фіксованість за часом передбачає розмежування кореспонденції на вільні та фіксовані. Вільні міграції характеризуються свободою вибору пасажиром часу поїздки. При цьому пасажири змушені підлаштовуватися під розклад руху. У цьому випадку формування та розподіл пасажиропотоку за часом доби залежить від розкладу руху транспорту. Таким чином, за умови дотримання транспортом графіком руху реалізується потокоформуюча функція розкладу. Фіксовані пересування характеризуються необхідністю транспорту підлаштовуватись під певний час або інтервал руху, що передбачає концентрацію рухомого складу за часом доби.

Для повнішого опису можливих транспортних ситуацій необхідно враховувати також і просторову характеристику пересування. Класифікація за напрямами передбачає виділення двох груп поїздок – концентровані за напрямами та рівномірно розподілені на території міста. Обслуговування концентрованих пасажиропотоків передбачає наявність у маршрутній мережі міста спеціальних маршрутів з мінімальним числом проміжних зупинок (напів-експресного та експресного сполучення). Однак у рамках традиційного підходу до організації транспортного обслуговування існує тенденція до більш рівномірного розподілу елементів транспортної мережі територією міста, що пов'язано зі збільшенням щільності маршрутної мережі, мінімізацією інтервалу руху та усередненням довжини перегонів між зупинками.

Подібний підхід до організації перевезень можна визнати цілком виправданим у разі, коли необхідно забезпечити задоволення потреб населення у перевезеннях за принципом «з будь-якої зони міста до будь-якої зони протягом доби», тобто. під час обслуговування рівномірно розподілених

поїздок. Такі кореспонденції характерні для культурно-побутових чи соціальних переміщень. Логістична схема показана рис. 1.1.

Рисунок 1.1 – Схема логістичних систем пасажирського транспорту

Однак значну частку у загальному обсязі перевезень займають трудові переміщення, фіксовані за часом і концентровані у просторі, що мають ознаки стійких технологічних відносин. Обслуговування такого роду поїздок необхідно вибудовувати за принципом «Між певними зонами міста у певний момент часу або з певним інтервалом руху».

Відповідно до цього принципу, при організації обслуговування таких кореспонденцій виправдано використання логістичних технологій перевезень, оскільки є ключові ознаки можливості їх застосування – визначеність пунктів відправлення та призначення, а також переважне значення фактора часу. До цієї групи поїздок як умовно-фіксованих входять і деякі вільні переміщення,

наприклад, на масові видовищні заходи, у місця заміського відпочинку. Ефективність цих видів перевезень може бути істотно підвищена шляхом впровадження та використання принципів логістичного керування пасажирським транспортом.

Проектування та створення систем пасажирських перевезень має відповідати наведеним класифікаційним ознакам. Це необхідно враховувати під час розподілу маршрутів вулично-дорожньою мережею міста, визначенням потрібної кількості рухомого складу та типів транспортних засобів для обслуговування намічених маршрутів, вибору режиму руху тощо.

Специфіка громадського транспорту у тому, що у ньому відбивається взаємозв'язок потоків матеріальних і людських ресурсів. Транспортна логістика передбачає можливість надання логістичного обслуговування (послуг) споживачеві матеріального потоку. Логістика громадського транспорту нерозривно пов'язана з процесом відтворення і є комплексом транспортних послуг, що надаються пасажирам.

Такі послуги називатимемо пасажирськими. Відсутність логістичного підходу до управління громадським транспортом створює проблеми його ефективного використання, зокрема [15]:

– планування перевезень пасажирів насамперед ґрунтується на звітних даних без належного економічного обґрунтування;

– залишаються маловивченими фактори, що визначають обсяг та структуру пасажирських перевезень;

– значні недогляди допускаються при плануванні роботи рухомого складу та обслуговуючого персоналу, зайнятого пасажирськими перевезеннями, експлуатаційних витрат та собівартості перевезень;

– тарифна система громадського транспорту у підвищенні експлуатаційної швидкості, зростанні продуктивності праці, зниження собівартості, підвищенні рентабельності пасажирських перевезень та культури обслуговування пасажирів.

Логістичний підхід до управління пасажирськими потоками вимагає об'єднання окремих ділянок перевізного процесу в єдину систему, здатну забезпечити якісні транспортні послуги населенню за мінімальних витрат. Логістична система – це складне організаційно-економічне ціле, що виконує функції управління матеріальними сервісними та супутніми їм інформаційними та фінансовими потоками. Вона складається з кількох підсистем - ланок і має розвинені зв'язки із зовнішнім середовищем.

Особливостями реальних ланок логістичної системи громадського транспорту є [16]:

– економічний суверенітет;
– відмінності в цілях та характері функціонування;
– різноманітність форм власності транспортних підприємств;
– відмінності у потужності, ступені концентрації та споживанні ресурсів;
– різна залежність результатів діяльності від зовнішніх факторів та суміжних ланок логістичної системи;
– відмінності у мобільності логістичної взаємодії.

Майбутнє пасажирських перевезень залежить від пріоритетного вирішення наступних трьох основних завдань [17]:

– забезпечення гарантованого транспортного обслуговування соціально незахищених верств населення, що не мають індивідуальних транспортних засобів;
– забезпечення економічної стабільності в регіонах;
– забезпечення мінімізації екологічних збитків.

Узагальнено структуру логістичної системи пасажирських перевезень можна подати у вигляді поєднання трьох складових, що відповідають рівням транспортного обслуговування. Цими складовими є транспортне, транспортне та після транспортне обслуговування. До транспортного обслуговування

включає планування поїздки, забезпечення зручності підходу пасажирів до зупинкових пунктів громадського транспорту. Транспортне обслуговування реалізується безпосередньо через доставку пасажирів із використанням спеціального рухомого складу з пункту відправлення до пункту призначення з необхідним рівнем комфорту. Після транспортне обслуговування полягає у забезпеченні зручності підходу пасажирів до пунктів призначення чи пересадки в інший вид транспорту.

Функціональним призначенням логістичних систем керування пасажирськими перевезеннями є забезпечення вирішення наступних груп завдань [18]:

– диспозиційних - аналіз, прогнозування, прийняття рішень, планування, оперативне управління, контроль;

– транспортних – здійснення міських, приміських, міжміських міжнародних перевезень;

– станційних – організація продажу квитків, культурно-побутового обслуговування тощо;

– інформаційних – керування пасажиропотоками, контроль перевезень, довідкове забезпечення;

– інших спеціальних – надання супутніх транспортних послуг, страхування, кредитування, фінанси тощо.

Функціональним призначенням логістичних систем керування пасажирськими перевезеннями є забезпечення вирішення наступних груп завдань:

– диспозиційних - аналіз, прогнозування, прийняття рішень, планування, оперативне управління, контроль;

– транспортних – здійснення міських, приміських, міжміських міжнародних перевезень;

– станційних – організація продажу квитків, культурно-побутового обслуговування тощо;

– інформаційних – керування пасажиропотоками, контроль перевезень, довідкове забезпечення;

– інших спеціальних – надання супутніх транспортних послуг, страхування, кредитування, фінанси тощо.

При проектуванні та створенні логістичних систем пасажирських перевезень необхідно враховувати такі основні принципи системності - комплексний розгляд елементів логістичної системи, починаючи від етапу формування попиту на перевезення та закінчуючи його задоволенням.

Вочевидь, що використання логістичних підходів з організацією роботи пасажирського транспорту забезпечує оптимальні з погляду витрат варіанти задоволення транспортних потреб населення.

1.2 Аналіз алгоритмів пошуку оптимального шляху

Завданням всіх алгоритмів маршрутизації є мінімізація вартості шляху до адресата за будь-якими критеріями. Алгоритми, що визначають маршрут на підставі найменшої кількості проміжних вузлів, належать до найпростіших. Більш складні алгоритми можуть проводити мінімізацію вартості шляху за різними критеріями, наприклад, за затримкою при передачі пакетів, пропускною спроможністю каналів зв'язку або грошової вартості передачі по даній лінії зв'язку.

Алгоритми для вирішення задачі маршрутизації можна розділити на точні (exact algorithm) та неточні (non-exact algorithm) [19]. Точні алгоритми включають перебір всіх можливих варіантів, в окремих випадках рішення можуть бути швидко знайдені, але в цілому здійснюється перебір $n!$ циклів. Другі в загальних випадках застосовуються для завдань, які неможливо вирішити точно (обчислення певних інтегралів, розв'язання нелінійних рівнянь, вилучення квадратного кореня…), якщо існуючі точні рішення вимагають значних і невиправданих часових витрат при високій складності

задачі, і як частина складнішого алгоритму допомогою якого завдання вирішується точно.

1.2.1 Точні алгоритми

У свою чергу, існує дві групи точних алгоритмів - одна з них використовує методи релаксації лінійного програмування TSP: алгоритм Гоморі, метод внутрішньої точки, метод гілок і кордонів; друга, менша група, використовує методи динамічного програмування. Характерною особливістю методів обох груп є гарантія знаходження оптимальних рішень при загальній трудомісткості процесу.

Повний перебір (Brute Force)

Один із найбільш очевидних методів вирішення завдання комівояжера – метод повного перебору чи грубої сили [20]. Його суть полягає у переборі всіх можливих варіантів шляхів, алгоритм вирішення можна записати як:

– визначити загальну кількість можливих гамільтонових контурів;

– визначити вагу кожного гамільтонова контуру, склавши вагу всіх його ребер;

– вибрати гамільтон контур з мінімальною вагою, який і буде оптимальним.

Метод повного перебору має низку переваг - він гарантує знаходження рішення задачі TSP, при цьому він прямолінійний і простий у виконанні. У той же час алгоритм вважається неефективним при роботі з великим обсягом даних, тому що для знаходження оптимального маршруту вимагає знайти вагу $(n-1)!$ гамільтонових контурів.

Табл. 1.1 демонструє кількість часу, необхідне вирішення завдання комівояжера методом повного перебору при комп'ютерної потужності, що дозволяє вважати 1 мільйон гамільтонових контурів за секунду.

Таблиця 1.1 – Розрахунковий час рішення TSP шляхом повного перебору.

Розрахунковий час вирішення TSP методом повного перебору	
Кількість міст	Розрахунковий час
10	1/3 секунди
13	8 хвилин
15	1 рік
20	193 року

Метод гілок та кордонів (Branch and Bound)

Метод гілок та кордонів часто використовується для знаходження оптимального вирішення задач комбінаторної оптимізації [21]. Його суть полягає в розбитті множини на завдання та виключення свідомо неоптимальних рішень.

Нехай граф V містить усі міста, Π - безліч всіх перестановок міст, що покриває всі можливі рішення. Розглянемо перестановку $\pi \in \Pi$, в якій кожному місту призначається наступник - i для πi міста. Таким чином, тур можна записати як $(1, \pi(1), \pi(\pi(1)), \ldots, 1)$. Якщо кількість міст у турі дорівнює n, тоді перестановку називають циклічною. Завдання про призначення ставить собі за мету знайти циклічні перестановки, а завдання комівояжера переслідує ту ж мету, але з обмеженням, що у цих перестановок має бути мінімальна вартість. Метод гілок і кордонів в першу чергу знаходить рішення задачі про призначення, вартість якої для n міст досить велика і асимптотично дорівнює $O(n^3)$.

Якщо було знайдено повний тур, то отримане значення є рішенням завдання комівояжера. В іншому випадку проблема поділяється на кілька областей, кожна з яких виключає деякі дуги туру, таким чином виключаючи самий тур. Метод, з допомогою якого обчислюється, яку дугу слід видалити, називають правилом розгалуження. Важливе зауваження – не повинно існувати дубльованих завдань, їхня загальна кількість має бути мінімізовано.

Будемо використовувати критерій, що гарантує незалежність задач – розглядається включений набір дуги та вибирається мінімальна кількість дуг, які не належать до набору. Позначимо *E* як безліч виключених дуг і *I* як безліч включених. Розкладемо *I*. Виберемо *t* дуги областей $x_1 x_2 \dots x_n$ які не належать *I*. Завдання поділено на *t* нащадків так, щоб у j_{th}; нащадка були $E_j <$ виключених дуг. Напишемо у вигляді формули:

$$\left.\begin{matrix} E_j = E \cup \{x_j\} \\ I_j - I\{x_1, x_2, \dots x_{j-1}\} \end{matrix}\right\} k = 1,2, \dots j \qquad (1.1)$$

Та x_j - виключена дуга j_{th} завдання та включена дуга в $(j + 1)_{st}$ області. Це означає тур, отриманий рішенням $(j + 1)_{st}$ завдання може мати x_j дугу, але тур, отриманий рішенням $(j + 1)_{st}$, не містить цієї дуги. Це гарантує відсутність маршрутів, що дублюються.

Кількість можливих рішень дорівнює $(n - 1)!/2$, для $n=50$ це приблизно 3×1062. Цей метод найчастіше використовується при кількості вузлів від 40 до 60.

Необхідність цілком вирішувати завдання лінійного програмування у всій області допустимих рішень вважатимуться головним недоліком вищеописаного методу. Для завдань з великим обсягом даних метод гілок і кордонів є невиправдано трудомістким, в той же час алгоритм є надійним методом вирішення цілих завдань.

Алгоритм Гоморі (The Cutting Plane)

У 1954 році була представлена робота Данцига, Фалкерсона і Джонсон, що описує новий метод вирішення завдання комівояжера, який також може бути використаний для вирішення будь-якої проблеми [22]:

$$minimize\ c^T x\ subject\ x\ to\ \in S \qquad (1.2)$$

де $c \neq 0$, S – кінцеве підмножина деякого R^m, і таким чином ми зможемо знайти точки S. Це ітераційний алгоритм – кожне повторення починається з лінійної програмної релаксації. Запишемо у вигляді формули:

$$minimize\ c^T x\ subject\ x\ to\ Ax \leq b \qquad (1.3)$$

де багатогранник P, визначений як $\{x : Ax \leq b\}$, містить S і обмежений. Оскільки P обмежений, ми можемо знайти оптимальне рішення x^* як екстремальну точку P. Якщо x^* належить S, то оптимальне рішення знайдено (1.2); інакше деяка лінійна нерівність задовольняє всі точки S і порушує x^*. Таку нерівність називають алгоритмом Гоморі, який докладно описав його 1958 року, методом відсікаючих площин або просто відсікань.

Даний метод використовується для побудови точних або наближених завдань, особливо часто зустрічається у поєднанні з методом гілок та кордонів і тоді називається методом гілок та відсікань. Обидва методи засновані на вирішенні послідовності релаксованих задач лінійного програмування. В алгоритмі Гоморі релаксовані завдання поступово покращують апроксимацію цілої задачі, зменшуючи околицю оптимального рішення. Якщо оптимальність не вдалося отримати, тоді необхідно шукати наближене рішення з похибкою.

У методу відсікань є перевага над методом гілок і кордонів - перші зручніші для апаратного обчислення, так як для їх вирішення не потрібно великий обсяг оперативної пам'яті для зберігання дерева рішень.

1.2.2 Неточні алгоритми

Алгоритм Дейкстри

В основу алгоритму, запропонованого Дейкстри, покладено метод, при якому вершинам приписуються тимчасові позначки, причому позначка вершини дає верхню межу шляху до цієї вершини [23]. Побудова найкоротших шляхів здійснюється поетапно, починаючи з деякого вузла до решти вузлів.

Величини позначок поступово зменшуються за допомогою деякої ітераційної процедури, і на кожному кроці ітерації точно одна з тимчасових позначок стає постійною. Останнє вказує на те, що позначка вже не є верхнім кордоном, а дає точну довжину найкоротшого шляху. Робота алгоритму закінчується, коли всі вершини мають постійні позначки.

Алгоритм Дейкстри застосовується тільки для графів з позитивними вагами дуг. Загалом принцип роботи алгоритму Дейкстри виглядає так: нехай дано граф $G(X, L)$, де $\{x_1, \dots x_n\} = X$ – безліч вершин графа.

Відповідність L показує взаємозв'язки вершин. Дугам графа приписані ваги $c(x_i, x_y) \geq 0$, що задаються матрицею C розмірністю $n \cdot n$.

Через $x_s \in X$позначено початкову вершину, щодо якої шукаються всі найкоротші шляхи.

Нехай $L(x_i)$ - позначка вершини x_i, тобто довжина шляху від початкової вершини x_s до вершини x_i.

Присвоєння початкових значень.

Крок 1. Покласти $L(x_s) = 0$ і вважати цю позначку постійною. $L(x_i) = \infty$ для всіх $x_i \neq x_s$ і вважати ці позначки тимчасовими. Тоді покласти $p = s$.

Оновлення позначок.

Крок 2. Для всіх $x_i \in L(x_p)$, позначки яких тимчасові, змінити позначки відповідно до наступного виразу:

$$L(x_i) \leftarrow min\left[L(x_i), L\left(x_p\right) + c(x_p, x_i)\right] \qquad (1.4)$$

Перетворення позначки на постійну.

Крок 3. Серед усіх вершин із тимчасовими познаками знайти таку, для якої $L(x_i) = \min(L(x_i))$.

Крок 4. Вважати позначку вершини x_i^* іх постійною та покласти $x_p = x_i^*$.

Крок 5. Якщо всі вершини позначені як постійні, то позначки дають довжину найкоротших шляхів. Зупин. Якщо деякі позначки є тимчасовими, перейдіть на крок 2.

Якщо знайдено всі довжини найкоротших шляхів, самі шляхи можна отримати, використовуючи метод за допомогою наступної формули:

$$L(x_i^*) + c(x_i^*, x_i) = L(x_i), \qquad (1.5)$$

де x_i^* - вершина, що передує x_i в найкоротшому шляху від x_s до x_i .

Алгоритм Дейкстри має квадратичну обчислювальну складність.

Алгоритм найближчого сусіда (Nearest Neighbour)

Один із найпростіших евристичних методів вирішення TSP. Головне правило алгоритму – завжди вибирати сусіднє місто (сусіда). Розв'язання задачі складається з наступних кроків [24]:

1. вибрати будь-яке місто;
2. знайти найближче місто, не включений у маршрут, і перейти до нього;
3. перевірити чи залишилися міста, не включені до маршруту, якщо відповідь позитивна – повторити другий крок;
4. щоб завершити тур додати ребро між останнім вибраним містом та першим.

У випадку трудомісткість вирішення завдання дорівнює $O(n^2)$. Нижня межа вартості оптимального маршруту на 10% вище за нижню межу Хелд-Карп.

Генетичні алгоритми

Генетичні алгоритми належать до методів оптимізації, основою яких лягли біологічні процеси, які у природі. Чарльз Дарвін у своїй еволюційній теорії ввів визначення природного відбору, згідно з якою особи, більш присолені до умов навколишнього середовища, мають більше шансів на виживання та продовження роду, і навпаки – непристосовані осипи зазнають виборчого знищення. Основою відбору є мутації генів та їх комбінації, що формуються при розмножені та передаються потомству. У результаті природного відбору виживають екземпляри з найбільшою функцією пристосованості. Це чисельна характеристика, яка може змінюватись в залежності від умов конкретного завдання. Пристосовані особини схрещуються (кросовер) та виробляють потомство. Випадкові мутації також можуть впливати в розвитку популяції.

Ми можемо позначити задачу оптимізації як задачу знаходження функції $f(x_5, x_6, \ldots, x_n)$, що носить назву функція пристосованості, яка дозволяє виділити найбільш пристосованих особин популяції для розмноження і найменше пристосування. Потрібно, щоб на області визначення виконувалася нерівність $f(x\_1, x\_2, \ldots, x\_n) \geq 0$, область є обмеженою. Параметри функції записуються як рядки, що складаються з бітів. Рядок, отриманий конкатенацією рядків, називається особиною:

$$1010\ 10110\ 101 \ldots 10101$$

$$|x1|x2|x3|\ldots|xn| \qquad (1.6)$$

Генетичний алгоритм універсальний, оскільки лише функція пристосованості та кодування рішень залежить від умов поставленого завдання. За виконання алгоритму враховуються такі правила: початкова популяція вибирається випадково, кількість її особин залишається незмінним, кожна їх записується як рядок з певної довжиною кодування.

Кожен крок алгоритму можна розбити на три етапи:

1) пропорційний залежно від пристосованості відбір особин поточного покоління, мають право виробляти потомство, і формування їх проміжної популяції;

2) проміжну популяцію ділять на пару і з певною ймовірністю схрещують, у результаті нове покоління потрапляє сама пара чи її нащадки за її присутності. Нащадки формуються з відсічених частин батьківських рядків, розділеною певною точкою.

3) відбувається мутація отриманого покоління, яка допускає передчасної збіжності. Кожен біт особини з ймовірністю трохи більше 1% записується як протилежний вихідному. Як заздалегідь заданих критеріїв отримання оптимального рішення можуть бути певна кількість зміни поколінь або сходження популяції - умова виконується, якщо всі рядки є майже ідентичними і знаходяться в області екстремуму. Рішенням алгоритму буде особина з максимальним значенням функції пристосованості.

1.2.3 Порівняння найбільш перспективних алгоритмів

Для вибору найкращого алгоритму було проведено їх порівняння на завданнях великої розмірності міст (вершин). Результати дослідження, проведені на 2,2 GHz 16GB of 1600MHz DDR3L SDRAM Intel Core i7 MacBook і включають порівняння таких параметрів як довжина оптимально шляху в кілометрах, витрачений час в секундах і кількість ітерацій, представлені на табл. 1.2.

Таблиця 1.2 – Порівняння алгоритму найближчого сусіда, генетичного та жадібних алгоритмів при вирішенні завдань зі 100 та 1000 містами

Порівняння алгоритмів для вирішення TSP при 100 містах			
Вибраний алгоритм	Довжина оптимального маршруту (км)	Витрачений час (сек)	Кількість ітерацій
Алгоритм найближчого сусіда	26654	2.5	100
Генетичний	225379	45	10000
Алгоритм Дейкстри	23211	0.07	18
Порівняння алгоритмів для вирішення TSP за 1000 міст			
Вибраний алгоритм	Довжина оптимального маршруту (км)	Витрачений час (сек)	Кількість ітерацій
Алгоритм найближчого сусіда	83838	98.5	1000
Генетичний	282766	468	10000
Алгоритм Дейкстри	72701	117	151

Підсумовуючи сказане вище, можна сказати, що головним недоліком генетичного алгоритму є відсутність гарантії, що отримане рішення є оптимальним, як і саме його перебування за прийнятний оптимально час.

Тоді як, алгоритм Дейкстри показує високу ефективність у задачах з великою кількістю міст. У ситуаціях, коли завдання великої розмірності відсутня впорядкованість вступних даних, алгоритм Дейкстри є єдиною альтернативою алгоритму повного перебору. Головна його перевага - його можна використовувати для вирішення складних неформалізованих проблем для яких не існує приватних методів вирішення, що дозволяє ефективно вирішувати нестандартні завдання.

1.3 Аналіз існуючих аналогів

З ростом попиту на навігаційні системи та планувальників маршрутів, збільшилась і кількість досліджень у цій галузі. Насамперед з’явилось чимало систем, що будують найкоротший шлях між двома точками. Більшість з них можна поділити на два основних типи: вбудовані та онлайн-планувальники. З-

поміж чисельних програм обох типів нижче наведено найпопулярніші рішення географічної маршрутизації.

1.3.1 TomTom

TomTom - велика міжнародна компанія, яка пропонує автономні навігаційні пристрої [25]. Їх пристрої є одними з найпопулярніших, головним чином завдяки інтуїтивно зрозумілому інтерфейсу, швидкості та точності обчислення маршрутів. Пристрої можуть розраховувати маршрути для подорожі на машині, велосипеді або пішки. На жаль, їх алгоритм маршрутизації та джерела даних не є відкритими для розробників. Нещодавно TomTom також випустила онлайнову версію свого планувальника маршрутів, проте ця версія не має багатьох функцій, які пропонує вбудована версія. Серед іншого, їй бракує можливості планувати маршрути на велосипеді або пішки. Також ані онлайн-версія, ані вбудована не мають української локалізації.

1.3.2 Via Michelin

Via Michelin - це служба онлайн-маршрутизації, заснована на картах відомого видавця дорожніх карт Michelin [26]. Служба є безкоштовною і швидко працює, але знову ж таки, інформація про алгоритми є конфіденційною. Via Michelin не доступна українською мовою, будує лише найкоротші маршрути і також не завжди точно працює для веломаршрутів.

1.3.3 GoogleMaps

Google запустив свою власну службу маршрутизації, що називається Google Maps [27]. Це дуже швидкий, безкоштовний сервіс. Очевидно, що Google виконує певну попередню обробку або кешування, щоб зробити їхню службу маршрутизації такою швидкою, проте деталі та алгоритм маршрутизації залишаються в секреті. Карти Google доступні українською

мовою, але Google Maps API не дозволяє модифікувати алгоритм або враховувати пріоритети об’єктів при побудові маршруту.

1.3.4 YourNavigation.org

YourNavigation.org - це демонстраційний веб-сайт для проекту YOURS [28]. Метою проекту є створення веб-сайту маршрутизації на основі даних OpenStreetMap (OSM) з використанням інших відкритих програм. Він використовує механізм маршрутизації з відкритим кодом, який називається Gosmore. Сервіс не дуже швидко працює, і на їхньому веб-сайті згадується, що Gosmore не призначений для генерації маршрутів довжиною більше 200 км. Планувальних надає можливість обрати один з двох типів маршруту: найкоротший або найшвидший, а також один з дев’яти видів транспорту, проте доступний лише англійською мовою.

1.3.5 OpenRouteService.org

OpenRouteService.org - це інший онлайн-сервіс, який використовує дані OSM [29]. Як і YourNavigation.org, це некомерційний сервіс. Служба використовує алгоритм A*, і повільніша на довгих маршрутах за інші аналоги. OpenRouteService.org доступна українською мовою, має можливість налаштування типів доріг, типу маршруту (найшвидший/найкоротший), типу транспортного засобу та його виду палива. Проте так само, як і інші аналогічні сервіси, не враховує типи проміжних пунктів маршруту та не дозволяє конфігурувати їх пріоритетність при побудові маршруту відносно вподобань користувача.

1.3.6 Порівняння сервісів

Результати порівняння п’яти сервісів маршрутизації наведено у табл. 1.3. Для додатків, що працюють найшвидше, алгоритми маршрутизації

виявилися конференційними, інші два додатки з відкритими алгоритмом використовують A*. Згідно результатів видно, що з п’яти досліджених сервісів лише два мають україномовну версію та не підтримують пріоритетів об’єктів на карті.

Таблиця 1.3 – Порівняння існуючих сервісів

Назва	Алгоритм	Джерело даних	Метод вибору точок та напряму	Підтримка української мови	Підтримка пріоритетів об’єктів мапи
TomTom	*конфіденційний*	Власні дані	Адреса	Немає	Немає
Google Maps	*конфіденційний*	Власні дані	Адреса або точка на карті	Є	Немає
Via Michelin	*Конфіденційний*	Michelin maps	Адреса або точка на карті	Немає	Немає
YourNavigation.org	A*	OSM	Адреса або точка на карті	Немає	Немає
OpenRoute Service.org	A*	OSM	Адреса або точка на карті	Є	Немає

За результатами порівняння виявлено, що жоден з додатків не підтримує побудову туристичних маршрутів. Таким чином, актуальною є розробка системи, яка б шукала шлях між точками, враховуючи області видимості проміжних пунктів. Цей підхід дозволить на основі алгоритмів побудови найкоротших маршрутів, виконати класифікацію об’єктів на карті та побудувати маршрут в автоматичному режимі.

1.4 Постановка завдання

Огляд сучасних підходів та засобів до проектування та розроблення програмного забезпечення дозволив обрати для створення власної системи ефективні технології та інструментальні засоби: IDE – Visual Studio 2019; Мова програмування – C#.

На основі виконаного аналізу предметної області можна сформулювати постановку задачі.

Розробити систему, основне призначення якої оптимізації маршрутів за допомогою жадного алгоритму Дейкстри.

Розробити зручний графічний інтерфейс для роботи з програмою. Основні дії та взаємодія між користувачем та системою повинні супроводжуватися відповідними повідомленнями для користувача.

Створити навігаційне меню для можливості швидкого та зручного отримання доступу до потрібної функції в системі.

Розробити супровідну документацію до створеної системи.

Дипломна робота припускає розробку додатка засобами об’єктно-

Результат роботи:

– Побудова найкоротших маршрутів між містами для транспортних та пасажирських перевезень.

Серед усіх функцій, які виконує система можна виділити загальні операції:

– реакція програми на вибір меню користувача;
– реакція програми на натискання кнопок в програмі;
– реакція програми на введення неправильних даних.

Функціональні вимоги:

– можливість додавати, редагувати та видаляти дані про міста;
– проведення симуляції розташування міст та надання відстані між ними;
– можливість зберігання та завантаження даних симуляції;
– можливість побудови найкоротших маршрутів.

Нефункціональні вимоги:

– для роботи програми на комп’ютері повинна бути встановлена бібліотека класів .NET Framework 4.7;

– для забезпечення роботи програми потрібно мати лише клавіатуру та мишку.

1. 5 Висновок

У даному розділі було проведено аналіз методів рішення TSP, представлено короткий опис алгоритмів, перераховані їх переваги та недоліки. Більш детально було описано генетичний алгоритм та алгоритм Дейкстри, також підбито підсумки порівняльного дослідження алгоритмів. За результатами даного порівняння було виявлено, що для завдань з великою кількістю міст, одним із найефективніших є алгоритм Дейкстри, який був обраний для реалізації програмного забезпечення.

Також було проаналізовано існуючі рішення в області маршрутизації. На основі отриманих даних було проведено постановку задачі на розробку інформаційної системи оптимізації маршрутів на основі об'єктно-орієнтованого програмування.

2 ВИБІР ТЕХНОЛОГІЙ ПРОЕКТУВАННЯ ТА ОСОБЛИВОСТІ АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Вибір технологій

Для побудови користувацького інтерфейсу будемо використовувати середовище розробки Visual Studio 2019 та мову програмування C#.

C# широко використовується професіоналами для розробки великих програмних продуктів завдяки наступним аспектам мови [30]:

– Сучасна мова програмування загального призначення.
– Підтримка об’єктно-орієнтованої парадигми.
– Підтримка компонентно-орієнтованої парадигми.
– Мова легка для вивчення навчитися.
– Добре структурована.
– Дозволяє розробляти ефективні програми.
– Мова має підтримку різних комп'ютерних платформах.
– Це частина .Net Framework.

C# спроектовано таким чином, що мова відповідає традиційним мовам високого рівня, C та C++ і є об'єктно-орієнтованою мовою програмування. Мова дуже схожа на Java, має численні сильні функції програмування, які роблять його привабливим для багатьох програмістів у всьому світі.

.NET Framework - це платформа для розробників із відкритим кодом, яку можна використовувати для створення широкого кола програм. Цей безкоштовний крос-платформний фреймворк підтримує декілька мов і має великі бібліотеки коду, які спрощують створення додатків для мобільних пристроїв, робочих столів та Інтернету.

Платформа .NET була розроблена для досягнення наступних цілей [31]:

– Сумісність;
– Підтримка різних платформам;
– Мовна незалежність;

– Бібліотека базових класів;
– Легка розробка;
– Безпека.

Для розробки користувацького інтерфейсу платформа .Net має декілька технологій, одна з яких – WinForms. Не дивлячись на те, що ця технологія досить не нова, її важко назвати застарілою. [32] Вона надає широкий спектр різних інструментів для побудови зручного та сучасного інтерфейсу. Крім того, IDE, які підтримують C# та .Net, надають зручний інтерфейс для графічної побудови користувацького застосунку, який розробляється.

Отже, C# та платформа .Net має низку характеристик, які задовольняють вимоги щодо розробки клієнтської частини системи складського обліку. Набір готових класів у стандартній бібліотеці, лаконічний зрозумілий синтаксис мови та зручний конструктор користувацького інтерфейсу зробить розробку зручною та достатньо швидкою. Об’єктно-орієнтована парадигма дозволить спроектувати систему таким чином, що розширення функціоналу буде без накладних розходів ресурсів розробки. Платформа .Net забезпечить безпеку, ефективність програмного забезпечення, а також підтримку декількох платформ.

Для розробки інформаційної бази використовувався Microsoft Access. MS Access – це СУБД, що входить до складу пакету офісних програм Microsoft Office [33]. Дана система управління базами даних має широкий спектр функцій (зв'язні запити, сортування, зв'язки із зовнішніми таблицями та базами даних).

Переваги використання:

– простий інтерфейс користувача, що дозволяє розробляти додатки, використовуючи вбудовані бібліотеки;
– зберігає всі дані у одному файлі;

– пропонує велику кількість «Майстрів», які допомагають уникнути рутинних дій і полегшують роботу досвідченому в програмуванні користувачеві;

– поширеність, яка зумовлена тим, що Access є продуктом компанії Microsoft;

– постійно оновлюється виробником, підтримує безліч мов;

– орієнтованість на користувача з різною професійною підготовкою,;

– широкі можливості імпорту/експорту даних у різні формати;

– Наявність розвинених інтегрованих засобів розробки додатків. Більшість програм, що розповсюджуються серед користувачів, містить той чи інший обсяг коду VBA (Visual Basic for Applications);

– Наявність вбудованої мови макрокоманд.

Недоліки:

– обмежені можливості щодо забезпечення розрахованого на багато користувачів середовища;

– має нескладні способи захисту з використанням пароля БД (можливе застосування додаткових заходів щодо захисту від несанкціонованого доступу з використанням процедур VBA);

– в питаннях підтримки цілісності даних відповідає лише моделям БД невеликої та середньої складності;

– Не розповсюджується безкоштовно.

Отже СУБД Access було мною вибрано туму, що розроблена програма може легко переноситись на будь-які системи сімейства Windows і не потребує додаткового встановлення сервера бази даних. Реляційність бази даних дає можливість ефективно використовувати пам’ять та уникнути дублювання інформації. Завдяки відкритому коду не потрібно витрачати зайві ресурси на різні ліцензії, тобто розробка та використання системи на базі СУБД Access значно дешевша. Підтримка проекту відбувається вже багато років

розробниками з усього світу, через що надійність та безпека продукту на високому рівні. Таким чином, СУБД Access повністю покриває вимоги щодо розробленого проекту.

2.2 Середовище розробки Visual Studio

Microsoft Visual Studio - повнофункціональне інтегроване середовище розробки (IDE) за допомогою популярних мов програмування, серед яких C, C++, VB.NET, C#, F#, JavaScript, Python (рис. 2.1).

Функціональність Visual Studio охоплює всі етапи розробки програмного забезпечення, надаючи сучасні інструменти для написання коду, проектування графічних інтерфейсів, складання, налагодження та тестування програм. Можливості Visual Studio можуть бути доповнені шляхом підключення потрібних розширень [34].

Редактор коду Visual Studio підтримує підсвічування синтаксису, вставку фрагментів коду, відображення структури та пов'язаних функцій. Істотно прискорити роботу допомагає технологія IntelliSense – автозавершення коду у міру введення з клавыатури символів.

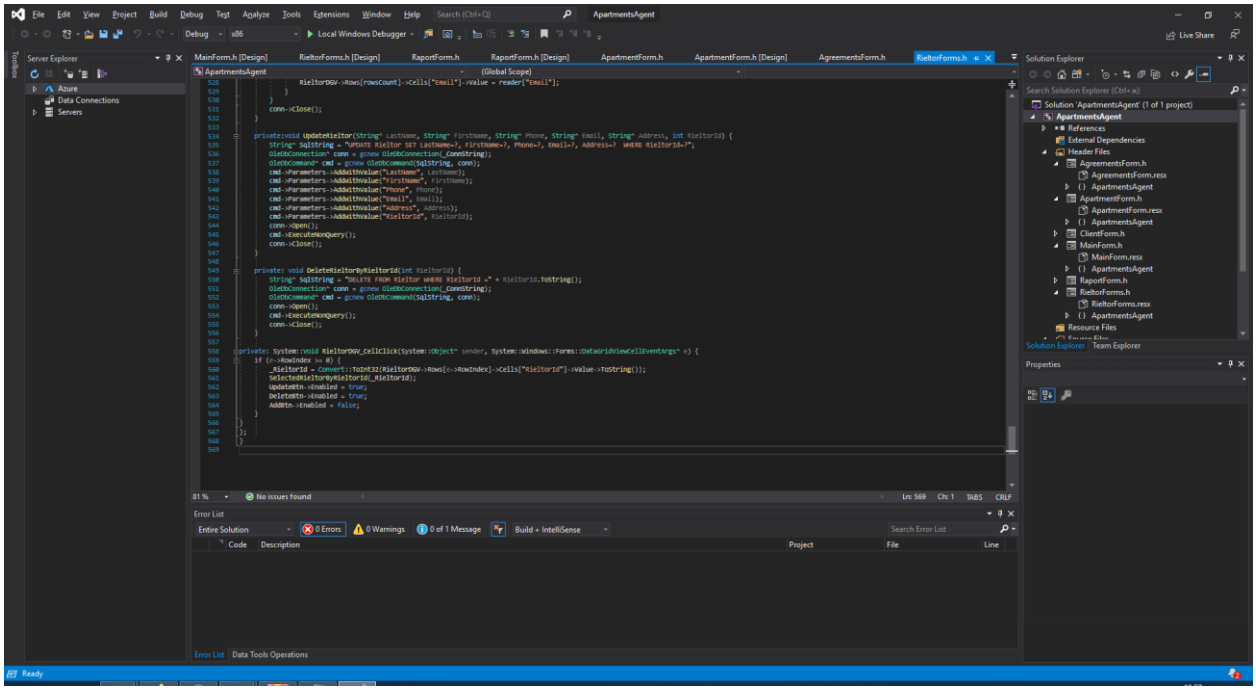


Рисунок 2.1 – Інтерфейс середовища розробки Visual Studio

Вбудований налагоджувач Visual Studio використовується для пошуку та виправлення помилок у вихідному коді, у тому числі на низькому апаратному рівні. Інструменти діагностики дозволяють оцінити якість коду з погляду продуктивності та використання пам'яті.

Дизайнер форм Visual Studio незамінний при розробці програм з графічним інтерфейсом, допомагає спроектувати зовнішній вигляд майбутньої програми та роботу кожного елемента інтерфейсу.

Для командних проектів Visual Studio пропонує підтримку групової роботи, дозволяючи виконувати спільне редагування та налагодження будь-якої частини коду в реальному часі, а як систему управління версіями використовувати Team Foundation або Git.

Основним розширенням файлу, асоційованим з Microsoft Visual Studio, є SLN - Visual Studio Solution File (Файл рішення Visual Studio), при відкритті якого в програму завантажуються всі дані та проекти, пов'язані з програмним рішенням, що розробляється.

Функціональна структура середовища включає [35]:

– редактор вихідного коду, який включає безліч додаткових функцій, як автодоповнення IntelliSense, рефракторинг коду тощо;

– налагоджувач коду;

– редактор форм, призначений для спрощеного конструювання графічних інтерфейсів;

– веб-редактор;

– дизайнер класів;

– дизайнерсхем баз даних.

Visual Studio також дозволяє створювати та підключати сторонні доповнення (плагіни) для розширення функціональності.

3.3 Переваги та недоліки середовища розробки

Інтегроване середовище розробки (IDE) Visual Studio пропонує високорівневі функціональні можливості, що виходять за рамки базового керування кодом.

Нижче наведено основні переваги IDE-середовища Visual Studio:

– Підтримка багатьох мов під час розробки. Visual Studio дозволяє писати код своєю мовою або будь-якими іншими мовами.

– Менше за код для написання. Для створення більшості програм потрібна пристойна кількість стандартного стереотипного коду та Web-сторінки ASP. NET тому не виняток. Наприклад, додавання Web-елемента керування, приєднання обробників подій та коригування форматування потребує встановлення в розмітці сторінки ряду деталей. У Visual Studio такі деталі встановлюються автоматично.

– Інтуїтивний стиль кодування. За промовчанням Visual Studio форматує код у міру його введення, автоматично вставляючи необхідні відступи та застосовуючи колірне кодування для виділення елементів типу коментарів.

– Висока швидкість розробки. Зручні функції, на зразок функції IntelliSense, функції пошуку та заміни та функції автоматичного додавання та видалення коментарів, дозволяють розробнику працювати швидко та ефективно.

– Можливості налагодження. Пропоновані Visual Studio інструменти налагодження є найкращим засобом для відстеження загадкових помилок і діагностування дивної поведінки.

– Висока швидкість розробки програм для Microsoft Windows.

– Низький поріг входження завдяки простому синтаксису мови.

– Можливість компіляції як у машинний код, так і в P-код (на вибір програміста).

– Безпека типів забезпечує захист від помилок, пов'язаних із застосуванням покажчиків та доступом до пам'яті.

– Можливість використання більшості функцій WinAPI для розширення функціональних можливостей програми. Це питання найбільш повно досліджено Деном Епплманом, який написав книгу "Visual Basic Programmer's Guide to the Win32 API".

Недоліки:

– Підтримка операційних систем лише сімейства Windows та Mac OS X (Виняток - VB1 for DOS).

– Відсутність повноцінного механізму наслідування реалізації об’єктів. Існуюче у мові успадкування дозволяє успадковувати лише інтерфейси, але з їх реалізацію.

– Практично всі вбудовані функції мови реалізовані через бібліотеку часу виконання, що в свою чергу сильно уповільнює швидкість роботи програм.

2.4 Аналіз вимог. Use-case діаграми. Основні прецеденти

Згідно з поставленою задачею можна висунути такий список вимог до проекту:

Таблиця 2.1 – Функціональні вимоги до додатку «Оптимізація маршрутів»

Вимоги	Опис
REQ-1	Система повинна дозволяти здійснювати реєстрацію користувача в програмі
REQ-2	Система повинна вести журнал подій, що відбулися в програмі
REQ-3	Система повинна дозволяти користувачеві додати та редагувати інформацію про облікові записи користувачів

REQ-4	Система повинна дозволяти користувачеві додати та редагувати інформацію про міста та населені пункти
REQ-5	Система повинна дозволяти користувачеві проводити симуляції маршрутів

Таблиця 2.2 – Нефункціональні вимоги до додатку «Оптимізація маршрутів»

Вимоги	**Опис**
REQ-6	Додаток повинен мати простий дизайн та зручну навігації
REQ-7	Поля повинні бути не порожніми, унікальними відносно вже існуючих записів

Таблиця 2.3 – Актори та цілі додатку «Оптимізація маршрутів»

Актори	**Цілі**
Користувач	Мета користувача полягає в роботі з системою
База даних	Мета бази даних полягає у зберігання інформації

Таблиця 2.4 – Опис варіантів використання додатка «Оптимізація маршрутів»

Варіант використання	**Ім’я**	**Опис**
UC1	Ідентифікація в системі	Дозволяє користувачу пройти ідентифікацію в системі
UC2	Вивід каталогу користувачів	Дозволяє користувачеві з правами системного адміністратора вивести каталог всіх користувачів системи

UC3	Додати користувача	Дозволяє користувачеві додати нового користувача системи
UC4	Редагувати користувача	Дозволяє користувачеві редагувати інформацію вибраного із списку користувача
UC5	Видалити користувача	Дозволяє користувачеві видалити інформацію вибраного із списку користувача
UC6	Вивід каталогу міст	Дозволяє користувачеві вивести каталог всіх міст
UC7	Додати місто	Дозволяє користувачеві додати нове місто
UC8	Редагувати місто	Дозволяє користувачеві редагувати інформацію вибране із списку місто
UC9	Видалити місто	Дозволяє користувачеві видалити інформацію вибраного із списку міста
UC10	Проведення симуляції маршрутів	Дозволяє користувачеві провести симуляцію маршрутів із вибраною кількістю міст
UC11	Збереження результатів симуляції	Дозволяє користувачеві зберегти результати проведеної симуляції
UC12	Завантаження симуляції	Дозволяє користувачеві здійснити завантаження збережених даних симуляції
UC13	Вивід системних подій	Дозволяє вивести всі події, які відбулися в системі

З поставлених вимог (див. розділ 1) тепер можна виставити повний опис вимог із сценаріями, що будуть вхідними даними для візуального моделювання мовою UML.

UC1 Ідентифікація в системі

Актор: користувач.

Ціль актора: пройти ідентифікацію в системі.

Задіяний актор: база даних.

Передумова: користувач ввівши нікнейм на пароль натискає кнопку «Підтвердити».

Післяумова: система відображає головне вікно програми з меню.

UC2 Вивід каталогу користувачів

Актор: користувач.

Ціль актора: вивести інформацію про всіх користувачів системи.

Задіяний актор: база даних.

Передумова: користувач натиснув на пункт меню «Користувачі».

Післяумова: система відображає екран для виведення списку всіх зареєстрованих користувачів системи.

UC3 Додати користувача

Актор: користувач.

Ціль актора: додати нового клієнта.

Задіяний актор: база даних.

Передумова: користувач ввівши всю необхідну інформацію про користувача натискає на кнопку «Додати».

Післяумова: система додає нового користувача та відображає екран із списком всіх користувачів.

UC4 Редагувати користувача

Актор: користувач.

Ціль актора: редагувати інформацію про вибраного користувача.

Задіяний актор: база даних.

Передумова: користувач вибирає необхідного із списку користувача.

Післяумова: система відображає екран для редагування інформації про вибраного користувача.

UC5 Видалити користувача

Актор: користувач.

Ціль актора: видалити інформацію про вибраного користувача.

Задіяний актор: база даних.

Передумова: користувач вибирає необхідного із списку користувача.

Післяумова: система відображає екран для видалення інформації вибраного користувача.

UC6 Вивід каталогу міст

Актор: користувач.

Ціль актора: вивести інформацію про всі міста та населені пункти, що додані у програмі.

Задіяний актор: база даних.

Передумова: користувач натиснув на пункт меню «Міста та населені пункти».

Післяумова: система відображає екран для виведення списку всіх міст та населених пунктів.

UC7 Додати місто

Актор: користувач.

Ціль актора: додати нове міст або населений пункт.

Задіяний актор: база даних.

Передумова: користувач ввівши всю необхідну інформацію про місто та натискає на кнопку «Додати».

Післяумова: система додає нове міст та відображає екран із списком всіх міст.

UC8 Редагувати місто

Актор: користувач.

Ціль актора: редагувати інформацію вибраного міста або населеного пункту.

Задіяний актор: база даних.

Передумова: користувач вибирає необхідне місто або населений пункт із списку.

Післяумова: система відображає екран для редагування інформації вибраного міста або населеного пункту.

UC9 Видалити місто

Актор: користувач.

Ціль актора: видалити інформацію вибраного міста або населеного пункту.

Задіяний актор: база даних.

Передумова: користувач вибирає необхідне місто або населений пункт із списку.

Післяумова: система відображає екран для видалення інформації вибраного міста або населеного пункту.

UC10 Проведення симуляції маршрутів

Актор: користувач.

Ціль актора: провести симуляцію маршрутів із вказаної кількості міст.

Задіяний актор: база даних.

Передумова: користувач переходить по меню програми «Управління» –> «Симулятор маршрутів».

Післяумова: система відображає екран з можливістю симуляції маршрутів.

UC11 Збереження результатів симуляції

Актор: користувач.

Ціль актора: зберегти результати проведеної симуляції.

Задіяний актор: база даних.

Передумова: користувач ввівши додаткову інформацію про симуляцію натискає кнопку «Зберегти».

Післяумова: система зберігає всю інформацію та сповіщає користувача системи.

UC12 Завантаження симуляції

Актор: користувач.

Ціль актора: завантажити збережену інформацію про проведені симуляції маршрутів.

Задіяний актор: база даних.

Передумова: користувач переходить по меню програми «Управління» –> «Завантаження маршрутів».

Післяумова: система відображає екран з можливістю завантаження даних проведених симуляцій.

UC13 Вивід системних подій

Актор: користувач.

Ціль актора: вивести список всіх подій в системі.

Задіяний актор: база даних.

Передумова: користувач вибирає пункт меню «Системний журнал».

Післяумова: система відображає екран із списком всіх системних подій.

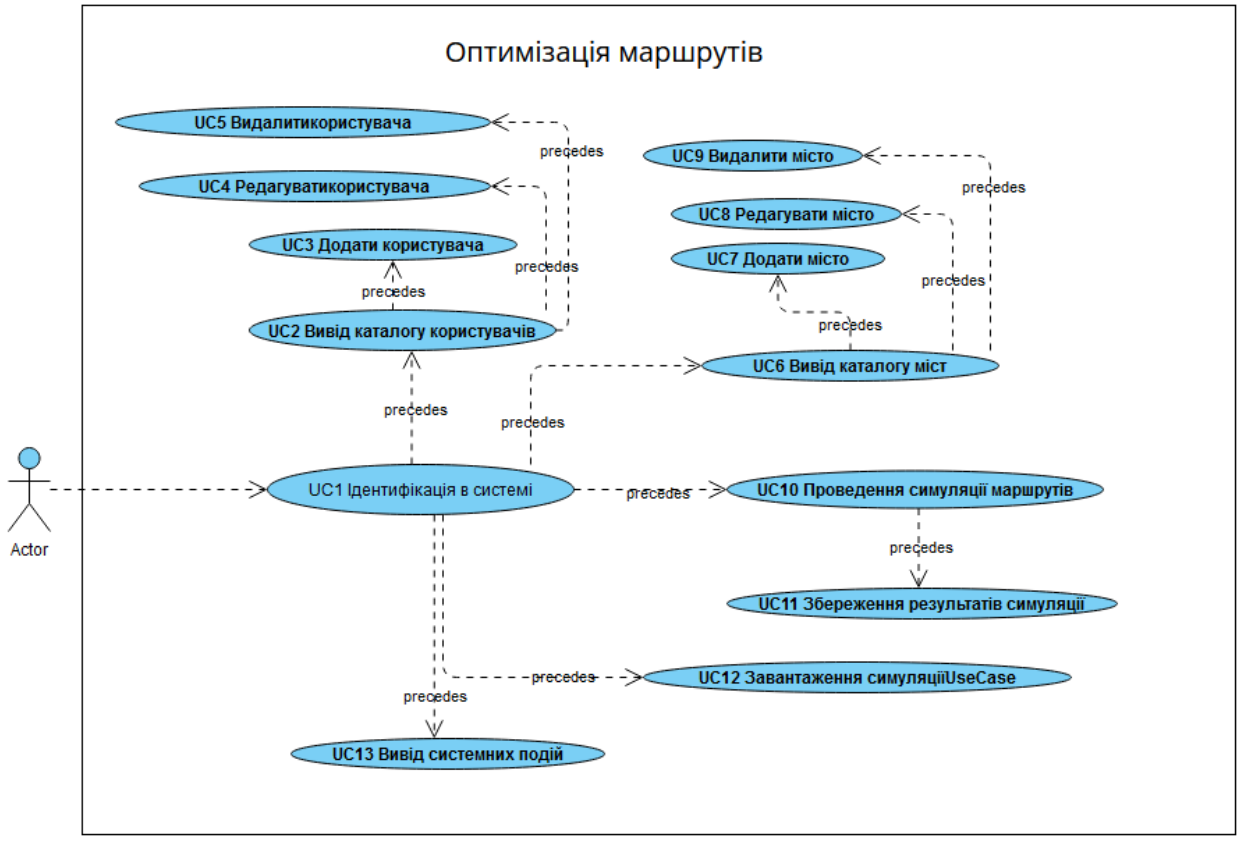


Рисунок 2.2 – Діаграма use-case

2.5 Архітектура проекту

Розроблений продукт повинен відповідати характеристикам якості, таким як: стійкість, корисність, доступність, масштабованість, відкритість, гнучкість, можливість тестування. Це вимагає від процесу розробки додаткові обмеження/правила, а саме:

– дотримання шаблонів і стилів;

– документування розробки на різних рівнях;
– тестування компонентів, окремих модулів, підсистем;
– управління проектами, процесами.

З урахуванням вимог до забезпечення стійкості та гнучкості системи при її розробці було обрано шаблон Layers, який розбиває систему на дві частини: клієнт та сервер.

Проектування системи буде покладатись на предметну область (DDD підхід) та принципи SOLID [27].

При розробці клієнта був обраний користувацький інтерфейс Windows Forms.

Сервер, в свою чергу, буде складатись з таких модулів:

1) BLL (англ. Business Logic Layer) – логіка та всі необхідні обчислення додатку на мові бізнесу;
2) DAL (англ. Data Acces Layer) – рівень доступу до даних.
3) DB (англ. Data Base) – база даних для зберігання даних.

З урахування всіх вище перерахованих шаблонів структура проекту буде виглядати наступним чином:

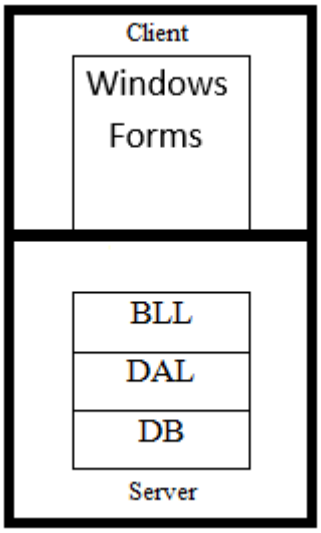


Рисунок 2.3 – Структура проекту

Кожен компонент буде реалізувати контракт (інтерфейс), який надає на гнучкість компоненту.

Оскільки бізнес постійно і відносно швидко змінюється, то кожен із компонентів повинен швидко адаптуватись під ці зміни. Для цього використаємо шаблон Dependency Injection (DI).

2.6 Особливості розробки бази даних. ERD діаграма з описанням сутностей

Схема "сутність-зв'язок" (також ERD або ER-діаграма) - це різновид блок-схеми, де показано, як різні "сутності" (люди, об'єкти, концепції і так далі) пов'язані між собою всередині системи. ER-діаграми найчастіше застосовуються для проектування та налагодження реляційних баз даних у сфері освіти, дослідження та розробки програмного забезпечення та інформаційних систем для бізнесу.

ER-діаграми (або ER-моделі) покладаються на стандартний набір символів, включаючи прямокутники, ромби, овали та сполучні лінії для відображення сутностей, їх атрибутів та зв'язків. Ці діаграми влаштовані за тим самим принципом, як і граматичні структури: сутності виконують роль іменників, а зв'язку — дієслів.

ER-діаграми - "родичі" схем структури даних (DSD), де замість зв’язків між самими сутностями відображається відношення між елементами всередині них. ER-діаграми часто використовуються у поєднанні з діаграмами DFD, що схематично показують рух потоків інформації в рамках процесу або системи.

У ER-моделях та моделях даних зазвичай виділяють до трьох рівнів деталізації:

Концептуальна модель даних – схема найвищого рівня з мінімальною кількістю подробиць. Перевага цього підходу полягає у можливості відобразити загальну структуру моделі та всю архітектуру системи. Менш

масштабні системи можуть обійтися без цієї моделі. І тут можна відразу переходити до логічної моделі [13].

Логічна модель даних: містить більш детальну інформацію, ніж концептуальна модель. На цьому рівні визначаються докладніші операційні та транзакційні сутності. Логічна модель залежить від технології, у якій застосовуватиметься.

Фізична модель даних: на основі кожної логічної моделі даних можна становити одну або дві фізичні моделі. В останніх має бути достатньо технічних подробиць для складання та впровадження самої бази даних.

Можна звернути увагу на той факт, що схожі рівні масштабу та деталізації зустрічаються і в інших видах схем (наприклад, у діаграмах DFD), проте дана класифікація відрізняється від трьохсхемного підходу у розробці ПЗ, де розподіл інформації здійснюється за дещо іншим принципом. Щоправда, іноді розробники застосовують ER-діаграми з додатковими ієрархіями, якщо дизайн бази даних потребує більше інформаційних рівнів. Наприклад, розробник може додати нові групи за принципом розширення вгору (суперкласи) та вниз (підкласи). А саме:

– *Лише реляційні дані*. Слід чітко розуміти, що мета ER-діаграм – показати зв'язки та відносини між елементами, тому вони відображають лише реляційну структуру.

– *Лише для структурованих даних*. Дані мають бути чітко розбиті на поля, стовпці та рядки, інакше користі від ER-діаграми буде мало. Це стосується і частково структурованих даних, оскільки лише деякі з них будуть придатними для роботи.

– *Складність інтеграції з базою даних*. Застосування ER-моделей для інтеграції з існуючою базою даних — непросте завдання через різницю в архітектурі.

Згідно завдання будуємо ERD діаграму.

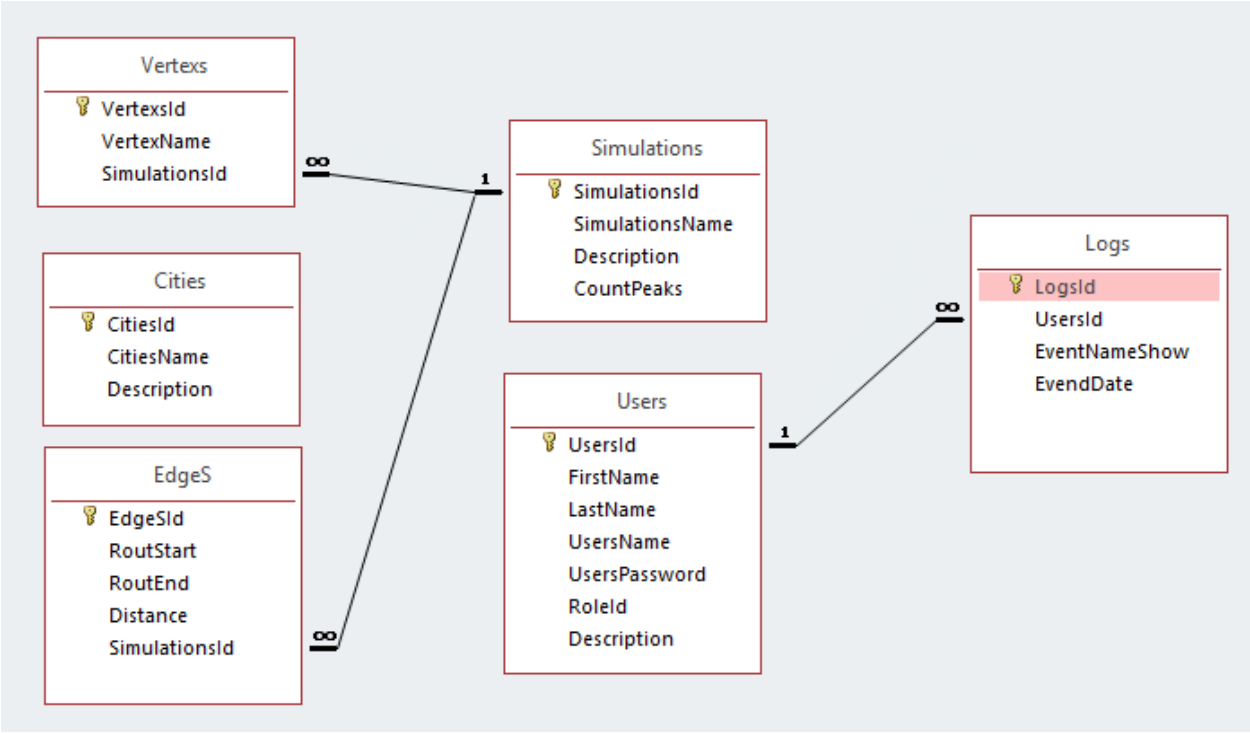


Рисунок 2.4 – ERD діаграма

Як ми бачимо на рис. 2.3 наша база даних складається із 6 сутностей. Кожна сутність має свою таблицю, а саме:

1. Таблиця «Cities» - зберігає інформацію про міста.
2. Таблиця «Logs» - зберігає інформацію про активність користувачів системи і їхні дії.
3. Таблиця «EdgeS» - зберігає інформацію про маршрути проведеної симуляції.
4. Таблиця «Simulations» - зберігає інформацію про проведену симуляцію.
5. Таблиця «Users» - містить інформацію про всіх користувачів системи та їхні облікові дані.
6. Таблиця «Vertexs» - зберігає інформацію про міста симуляції.

2.7 Особливість реалізації бізнес логіки – діаграма домена

Будуємо діаграму класів домена. Кожен клас описує конкретну таблицю бази даних для зручного опрацювання даних (рис. 2.5).

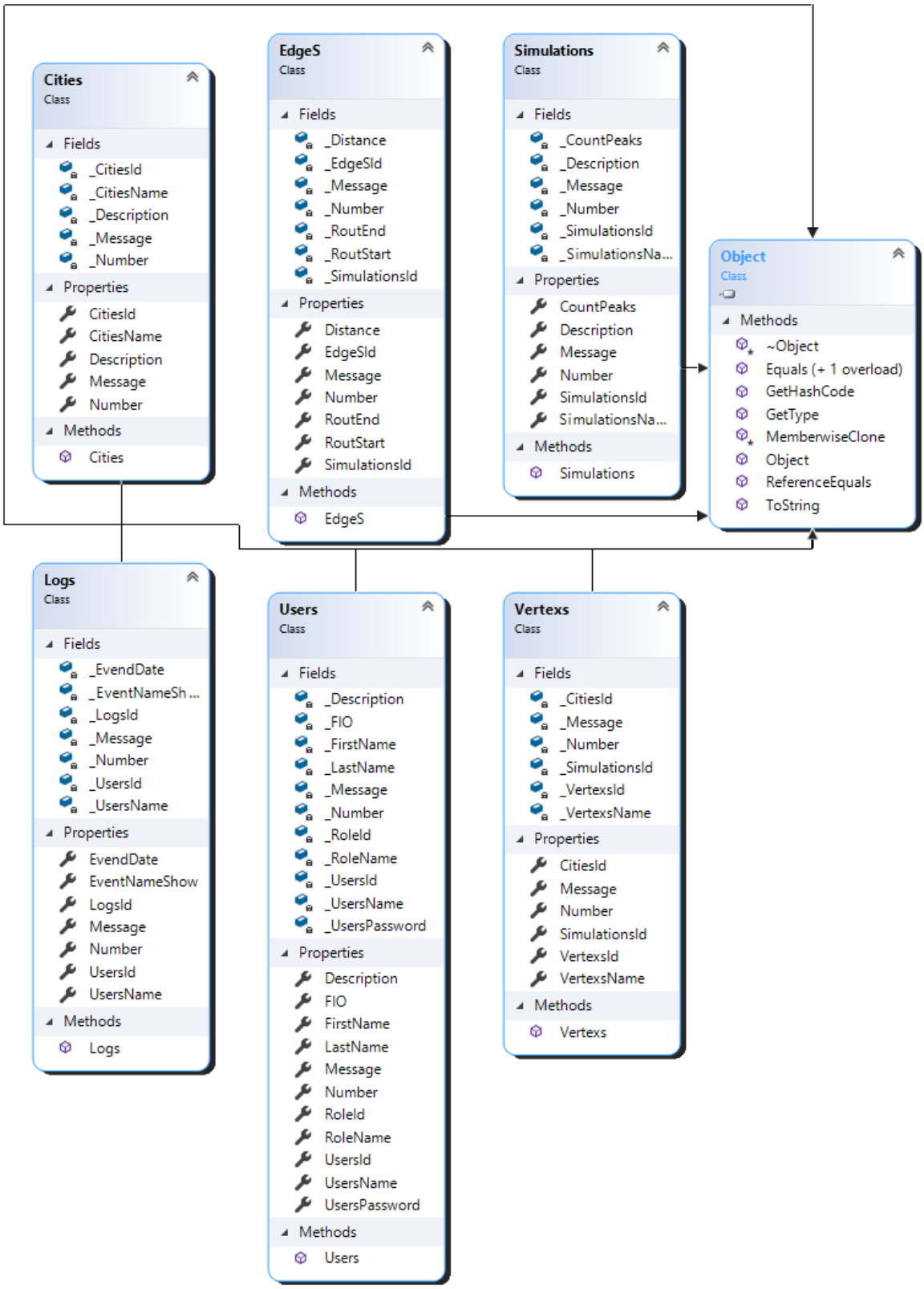


Рисунок 2.5 – Діаграма класів домена

Як можна побачити із рис. 2.5, у проекті реалізовано рівно стільки класів, скільки і сутностей у базі даних.

2.8 Особливості розробки рівня UI

User interface (UI) елементи – це частини, які дизайнери використовують для створення програм або веб-сайтів. Вони додають інтерактивність в інтерфейс користувача, надаючи користувачеві точки зіткнення при навігації по них. Кнопки, смуги прокручування, пункти меню та чекбокси.

Інтерфейсу користувача (UI) використовує UI елементи для створення візуальної мови і забезпечення узгодженості продукту, що робить його зручним для користувача і простим у навігації без особливих зусиль з боку користувача.

UI елементи зазвичай поділяються на одну з наступних чотирьох груп:

– елементи керування введенням (Input Controls) – дозволяють користувачам вводити інформацію до системи.

– компоненти навігації (Navigation Components) – допомагають користувачам переміщатися продуктом або веб-сайтом. Загальні навігаційні компоненти включають панелі вкладок та головне меню програми.

– інформаційні компоненти (Informational Components) – діляться інформацією з користувачем.

– контейнери (Containers) – містять зв’язаний контент разом.

В даному проекті для розробки користувацького інтерфейсу я використав Windows Forms.

Windows Forms - це платформа користувача інтерфейсу для створення класичних додатків Windows. Вона забезпечує один з найефективніших способів створення класичних додатків за допомогою візуального конструктора в Visual Studio.

На рис. 2.6 показана діаграма програми «Оптимізація маршрутів» з правами адміністратора системи.

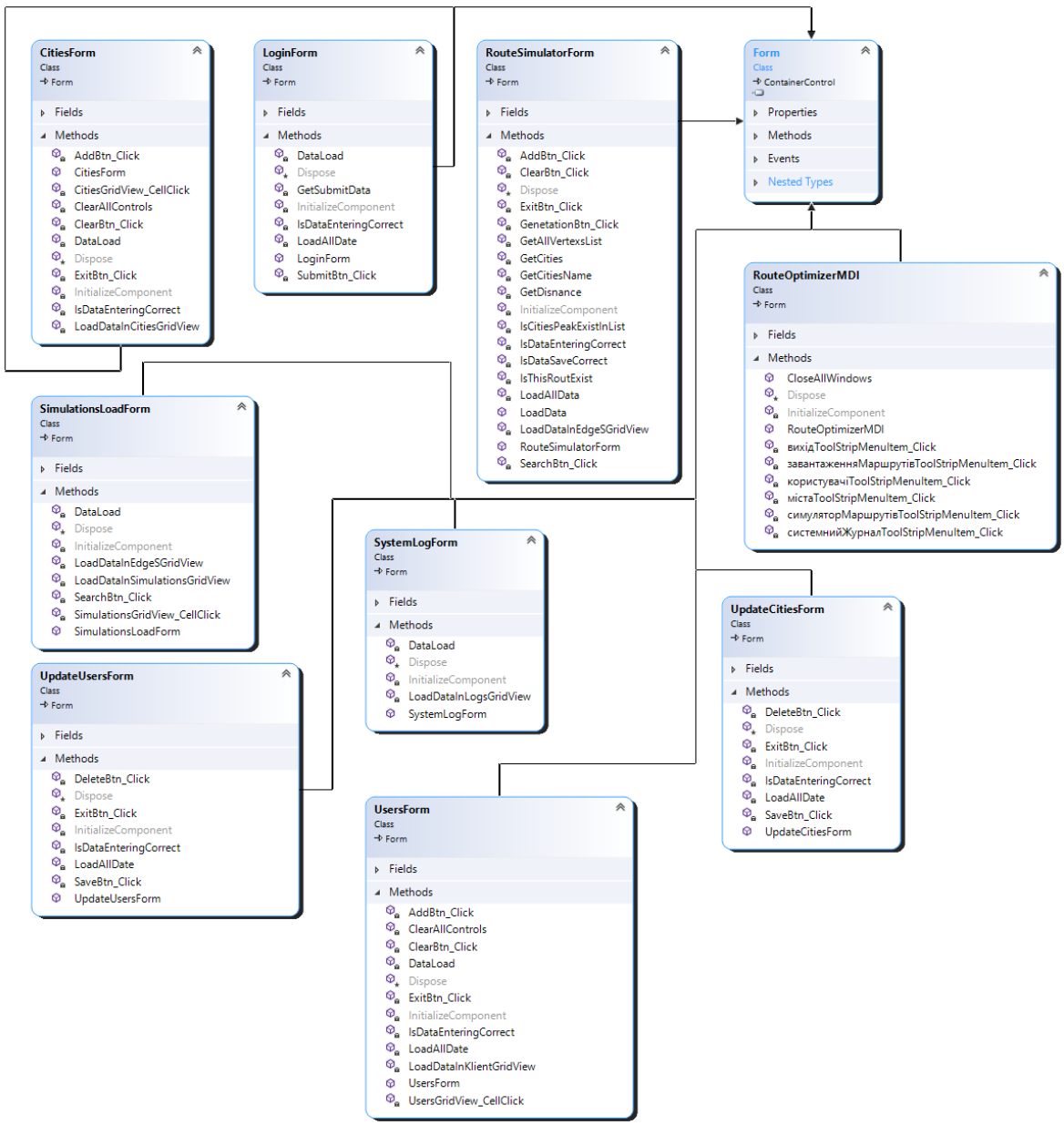


Рисунок 2.6 – Діаграма інтерфейсу програми «Оптимізація маршрутів»

Програма складається з дев’яти класів рівня UI та є похідними від класу Form, тобто мають графічний інтерфейс.

Клас «RouteOptimizerMDI» становить головне вікно програми. Найважливішим елементом управління в головному вікні є розташування головного меню, за допомогою якого можна відкривати інші форми та вийти з програми.

Клас «CitiesForm» призначений для опрацювання даних про міста та населені пункти.

Клас «UpdateCitiesForm» призначений для редагування інформації про міста та населені пункти.

Клас «LoginForm» призначений для авторизації користувачів системи.

Клас «SystemLogForm» призначений для перегляду подій системи.

Клас «UsersForm» призначений для опрацювання даних про користувачів системи.

Клас «UpdateUsersForm» призначений для редагування інформації вибраного із списку користувача.

Клас «RouteSimulatorForm» призначений для проведення симуляції маршрутів.

Клас «SimulationsLoadForm» призначений для завантаження даних проведеної симуляції.

Створення об'єктів інших класів та виклик їх методів відбувається в результаті взаємодії користувача з елементами графічного інтерфейсу програми.

2.9 Особливості розробки DAL

Шар доступу до даних (Data Access Layer – DAL) у програмному забезпеченні – це шар комп'ютерної програми, який надає спрощений доступ до даних, що зберігаються у постійному сховищі якогось типу, такому як реляційна база даних. Цей акронім в основному використовується в оточенні Microsoft.NET.

DAL може повертати посилання на об'єкт (у термінах об'єктно-орієнтованого програмування) з його атрибутами замість рядків полів із таблиці бази даних. Це дозволяє створювати клієнтські (або модулі користувача) модулі з більш високим рівнем абстракції. Така модель може бути реалізована шляхом створення класу з методами доступу до даних, які

безпосередньо посилаються на відповідний набір процедур бази даних. Інша реалізація може потенційно отримувати або записувати записи або з файлової системи. DAL приховує складність сховища даних, що лежить в основі даних.

Замість використання таких команд як «створити», «видалити» або «оновити» в певній таблиці в базі, клас і кілька процедур, що зберігаються, можуть бути створені в базі. Ці процедури можуть викликатися методом усередині класу, який поверне об'єкт, що містить запитані значення. Або команди створення, видалення та оновлення можуть бути виконані всередині простих функцій, що зберігаються у шарі доступу до даних.

Також методи бізнес-логіки із програми можуть бути співвіднесені до шару доступу до даних.

Для роботи з базою даних було реалізовано 7 класів. Назва кожного класу починається із відповідній їй назві таблиці в базі даних та закінчується приставкою «Provider».

На рис. 2.7 приведено діаграму класів з методами для роботи з базою даних.

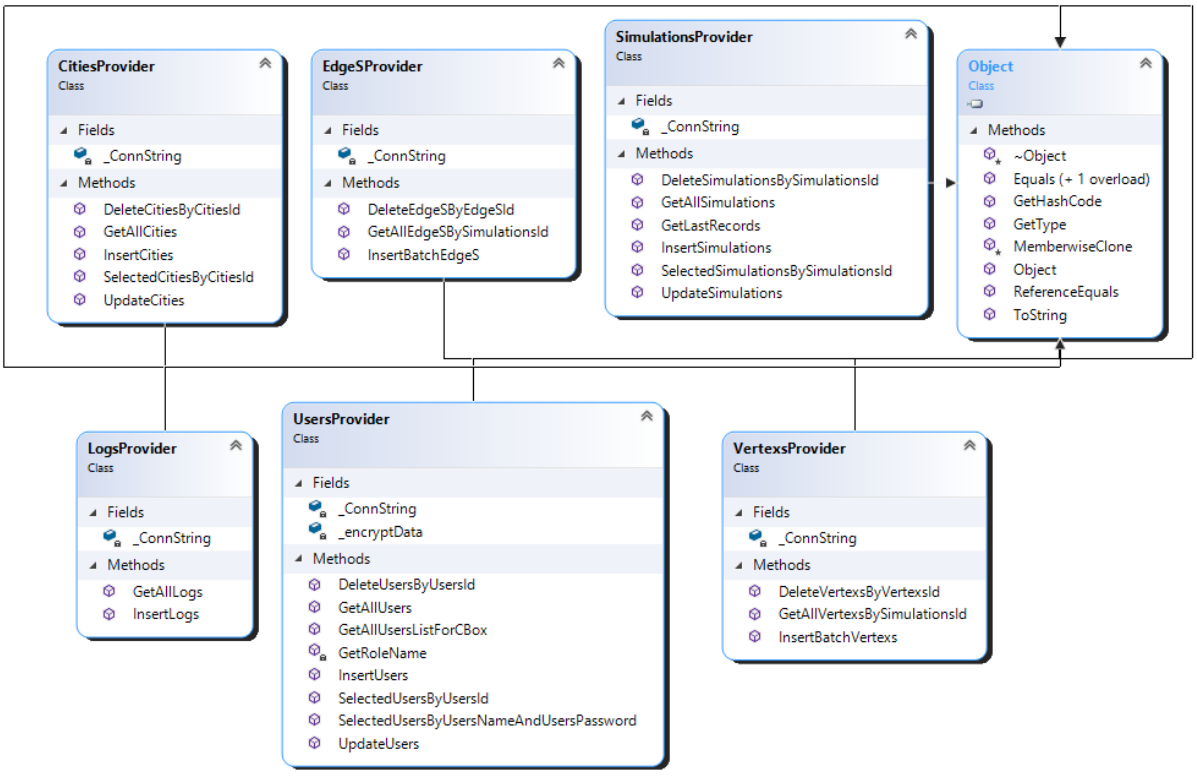


Рисунок 2.7 – Діаграма класів з методами для роботи з базою даних

2.10 Висновок

Побудова сучасних інформаційних система займає дуже багато часу. Вона починається з аналізу предметної області, детального планування системи, описання вимог, моделювання її поведінки за допомогою uml діаграм. Визначається шаблони, які будуть використовуватись при розробці.

Після планування починається стадія розробки. Розробка починається зі створення користувацького інферфейсу і закінчується базою даних.

При розробці слід враховувати, що вимоги змінюються швидко і потрібно будувати систему так, щоб вона була гнучкою.

Розробка системи виконується по окремим компонентам. Кожний створений компонент потрібно детально тестувати, щоб мінімізувати помилки на наступних етапах розробки.

Якщо дотримуватися всіх вищеперерахованих вимог, то можна побудувати гнучку і стійку інформаційну систему.

3 РОЗРОБКА, ВИКОРИСТАННЯ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Розробка програмних модулів системи

В даному підрозділі наведений код роботи програми, що був використаний для реалізації проекту. Для зменшення кількості коментарів у програмі всі назви класів, об’єктів у програмі мають зрозумілі назви.

Після постановки задачі, маючи всі необхідні дані, проводимо детальний аналіз алгоритму Дейкстри для подальшого його реалізації у вигляді класу і з назвою «Dijkstra».

Алгоритм працює покроково - на кожному кроці він «відвідує» одну вершину і намагається зменшувати мітки.

Робота алгоритму завершується, коли всі вершини відвідані.

Мітка самої вершини a покладається рівною 0, мітки інших вершин - нескінченності. Це відображає те, що відстані від a до інших вершин поки невідомі. Всі вершини графа позначаються як невідвіданих.

Якщо все вершини відвідані, алгоритм завершується. В іншому випадку, з ще не відвіданих вершин вибирається вершина u, що має мінімальну позначку.

Ми розглядаємо різні маршрути, в яких u є передостаннім пунктом. Вершини, в які ведуть ребра з u, назвемо сусідами цієї вершини. Для кожного сусіда вершини u, крім позначених відвідані, розглянемо нову довжину шляху, що дорівнює сумі значень поточної мітки u і довжини ребра, що з'єднує u з цим сусідом.

Якщо отримане значення довжини менше значення мітки сусіда, замінимо значення мітки отриманим значенням довжини. Розглянувши всіх сусідів, позначимо вершину u як відвіданих і повторимо крок алгоритму.

Алгоритм закінчує роботу, коли всі вершини відвідані.

Блок–схема алгоритму Дейкстри зображена на рис. 3.1.

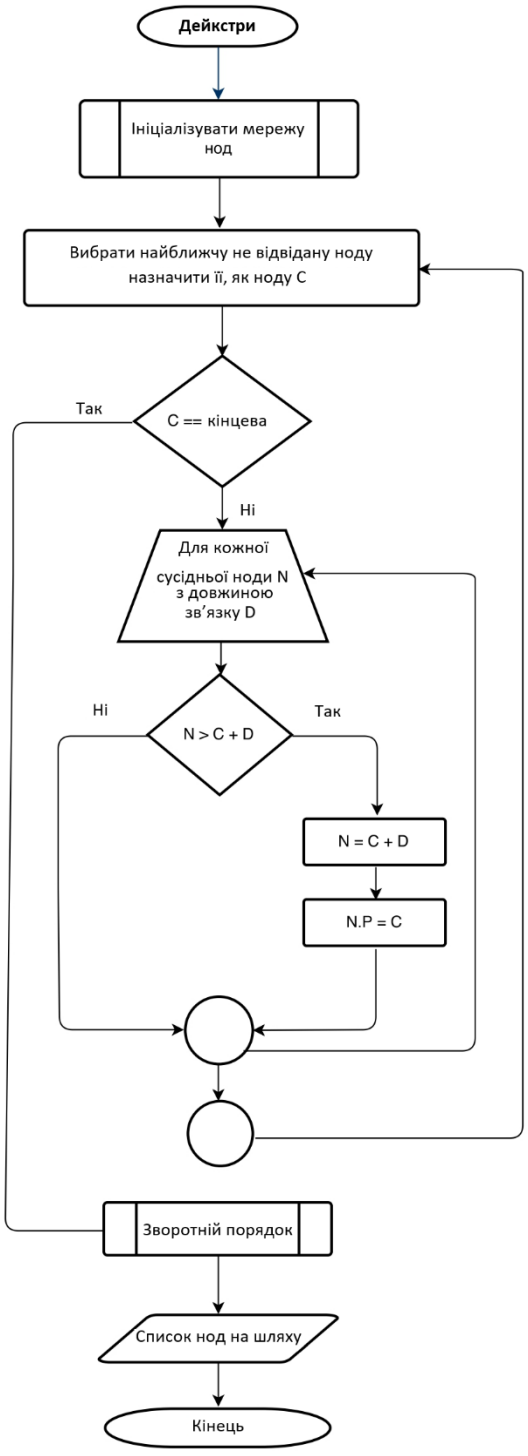


Рисунок 3.1 – Алгоритм Дейкстри

Перелік класів, які були розроблені для реалізації алгоритму Дейкстри:

1. Клас «GraphVertexInfo» - містить інформацію про вершину.

2. Клас «Graph» - містить повну інформацію про граф. Список всіх вершин графа знаходиться у публічній властивості даного класу «List<GraphVertex>». Також даний клас має 4 методи:

2.1. Метод «Graph» – конструктор класу.

2.2. Метод «AddVertex» – призначений для додавання вершин класу.

2.3. Метод «FindVertex» – призначений для пошуку вершин даного графа і містить вхідний параметр, в який необхідно передати назву вершини, для того, щоб знайти її.

2.4. Метод «AddEdge» – призначений для додавання ребра графа. Для того, щоб додати ребро графа необхідно передати такі параметри: назву першої вершини, назву другої вершини та її вагу.

3. Клас «GraphVertex» – описує саму вершину графа, містить дві властивості: назву вершини та список ребер з вагою. Містить наступні методи:

3.1. Метод «AddEdge» – призначений для додавання інформації про ребро графа. Вхідним параметром є клас «GraphEdge» (ребро графа).

3.2. Метод «AddEdge», який є подібним до попереднього, але

4. Клас «GraphEdge» – призначений для опису ребра графа і містить інформацію про зв’язані вершини графа та вагу ребра. Також містить метод конструктор для передачі параметрів для властивостей зв’язаної вершини та ваги графа.

5. Клас «Dijkstra» – реалізовує алгоритм Дейкстри. Містить властивість «infos», в якій зберігається список інформації всіх вершин графа. Також даний клас містить наступні методи:

5.1. Метод-конструктор «Dijkstra», для створення екземпляру цього класу необхідно передати параметр конструктору з типом вище описаного класу «Graph».

5.2. Метод «InitInfo» – ініціалізує інформацію про вершини графа.

5.3. Метод «GetVertexInfo» призначений для того, щоб получити інформацію про вершину графа.

5.4. Метод «FindUnvisitedVertexWithMinSum» – призначений для пошуку невідвіданої вершини з мінімальним значенням суми.

5.5. Метод «FindShortestPath» – пошук найкоротшого шляху по назвах вершин. Для його виклику необхідно передати 2 параметри, а саме: назву стартової та фінішної вершин.

5.6. Метод «FindShortestPath» – призначений для пошуку найкоротшого шляху між вершинами. Щоб його застосувати необхідно передати 2 параметри типу класа «GraphVertex». Даний метод повертає весь пройдений шлях типу string.

5.7. Метод «SetSumToNextVertex» обчислює суми ваг ребер для наступної вершини.

5.8. Метод «GetPath» призначений для формування найкоротшого шляху між вершинами. Для його виклику необхідно передати 2 параметри типу класа «GraphVertex».

Далі засобами СУБД було побудовано модель бази даних.

Сучасні бази даних зберігають великі об’єми інформації, тому обробляти їх вручну переглядаючи та редагуючи дані в таблицях, стає дуже важко важко, тому для підвищення ефективності користування базами даних застосовують запити.

Запит– це засіб отримання інформації з бази даних.

Для написання запитів було використано мову SQL (Structured Query Language).

В базі даних, було розроблено необхідні запити на додавання інформації до таблиць, редагування та видалення. Всі запити були реалізовані безпосередньо із розробленого додатку.

Після того, як базу даних було створено під’єднаємо її за допомогою засобів середовища Visual Studio 2019 для розробки подальшої системи. Усі таблиці пізніше будуть підключатись до форм, на яких можливе додавання, редагування та видалення даних.

Для під’єднання БД до середовища Microsoft Visual Studio 2019 у файлі проекту "App.config" створюємо змінну "CONNECT" та задаємо значення параметрів (лістинг 3.1).

Лістинг 3.1 Параметри змінної "CONNECT"

<add key="CONNECT" value="Provider=Microsoft.Jet.OLEDB.4.0;Data Source=|DataDirectory|\DataBase.mdb;"/>

Для роботи з створеною базою даних використовуємо простір імен System.Data. OleDb.

Для того, щоб створити меню, додаємо елемент menuStrip у головне вікно проекту (рис. 3.2).

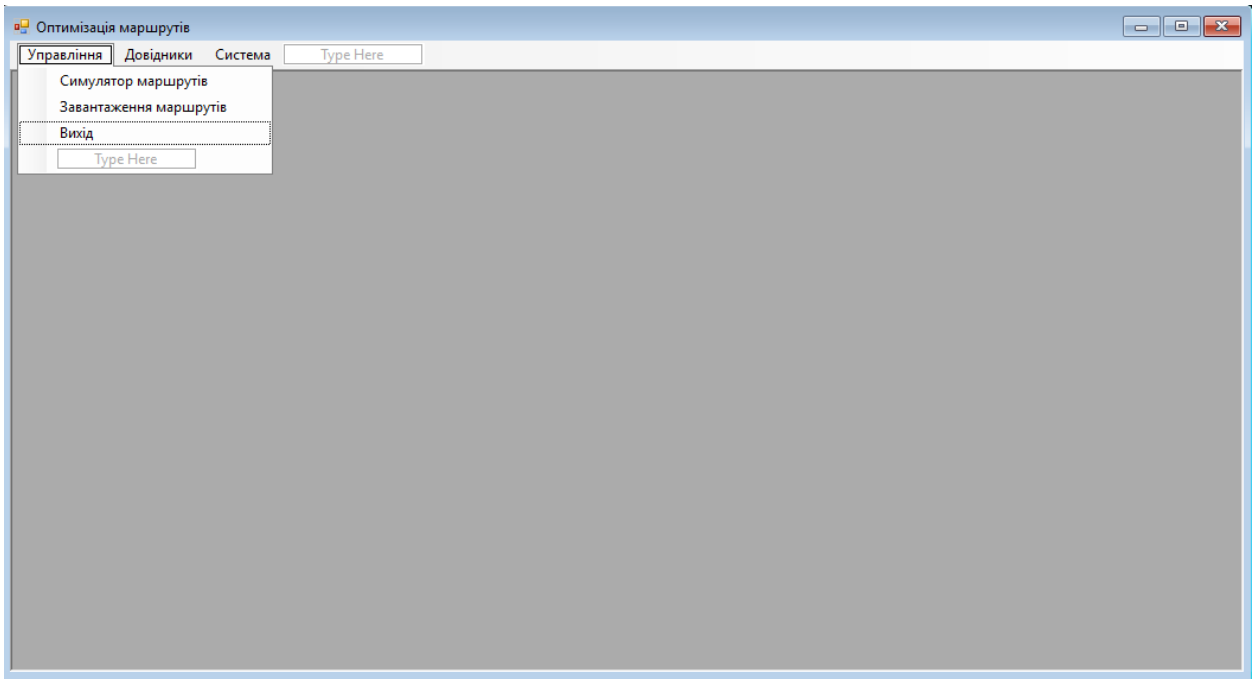


Рисунок 3.2 – Головне меню програми

На кожен із пунктів меню додано певний код, який відповідає за створення екземплярів форм та закриття попередньо відкритого вікна. Аналогічний код застосовуємо для всіх пунктів меню (рис. 3.3).

```
1 reference
private void симуляторМаршрутівToolStripMenuItem_Click(object sender, EventArgs e) {
    CloseAllWindows();
    RouteSimulatorForm routeSimulatorForm = new RouteSimulatorForm();
    routeSimulatorForm.MdiParent = this;
    routeSimulatorForm.WindowState = FormWindowState.Maximized;
    routeSimulatorForm.Show();
}
```

Рисунок 3.3 – Програмування пунктів меню

Для роботи з базою даних було створено відповідні класи, всі вони розміщені у папці із назвою «Providers» в проекті рішення.

Для з’єднання з базою даних використовується метод «Open» екземпляру класу «OleDbConnection». Для закриття з’єднання з базою даних використовується метод «Close» того ж екземпляру класу.

Для опрацювання даних про маршрути був створений клас «EdgeSProvider», який містить в собі 3 публічних методів для опрацювання інформації: додавання списку маршрутів симуляції, вибірка всіх маршрутів по ідентифікатору симуляції, та видалення інформації про всі маршрути по ідентифікатору симуляції.

Код методу для додавання списку маршрутів симуляції на рис. 3.4.

```
4 references
class EdgeSProvider {
    private string _ConnString = System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];
    1 reference
    public void InsertBatchEdgeS(List<EdgeS> EdgeS) {
        string SqlString = "INSERT INTO EdgeS (RoutStart, RoutEnd, Distance, SimulationsId) Values(?, ?, ?, ?)";
        using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
            using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
                cmd.CommandType = CommandType.Text;
                conn.Open();
                for (int i = 0; i < EdgeS.Count; i++) {
                    cmd.Parameters.AddWithValue("RoutStart", EdgeS[i].RoutStart);
                    cmd.Parameters.AddWithValue("RoutEnd", EdgeS[i].RoutEnd);
                    cmd.Parameters.AddWithValue("Distance", EdgeS[i].Distance);
                    cmd.Parameters.AddWithValue("SimulationsId", EdgeS[i].SimulationsId);
                    cmd.ExecuteNonQuery();
                    while (cmd.Parameters.Count > 0) {
                        cmd.Parameters.RemoveAt(0);
                    }
                }
                conn.Close();
            }
        }
    }
```

Рисунок 3.4 – Метод «InsertBatchEdgeS» для додавання списку маршрутів

Код методу для вибірки всіх маршрутів по ідентифікатору симуляції представлений на рис. 3.5.

```
1 reference
public List<EdgeS> GetAllEdgeSBySimulationsId(int SimulationsId) {
  int i = 0;
  string SqlString = "SELECT * FROM EdgeS WHERE SimulationsId=" + SimulationsId;

  List<EdgeS> EdgeS = new List<EdgeS>();
  using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
    using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      conn.Open();
      using (OleDbDataReader reader = cmd.ExecuteReader()) {
        while (reader.Read()) {
          EdgeS oneEdgeS = new EdgeS();
          oneEdgeS.Number = ++i;
          oneEdgeS.EdgeSId = Convert.ToInt32(reader["EdgeSId"]);
          oneEdgeS.RoutStart = reader["RoutStart"].ToString();
          oneEdgeS.RoutEnd = reader["RoutEnd"].ToString();
          oneEdgeS.Distance = Convert.ToInt32(reader["Distance"]);
          oneEdgeS.SimulationsId = Convert.ToInt32(reader["SimulationsId"]);
          EdgeS.Add(oneEdgeS);
        }
      }
      conn.Close();
    }
  }
  return EdgeS;
}
```

Рисунок 3.5 – Метод «GetAllEdgeSBySimulationsId» для вибірки маршрутів

Для видалення даних маршрутів симуляції був розроблений метод «DeleteEdgeSByEdgeSId», що показаний на рис. 3.6.

```
1 reference
public void DeleteClientByClientId(int ClientId) {
  string SqlString = "DELETE FROM Client WHERE ClientId=" + ClientId.ToString();
  using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
    using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      conn.Open();
      cmd.ExecuteNonQuery();
      conn.Close();
    }
  }
}
```

Рисунок 3:6 – Метод «DeleteEdgeSByEdgeSId» для видалення інформації вибраної із списку симуляції

Наступним кроком було створення форми для проведення симуляції (рис. 3.7).

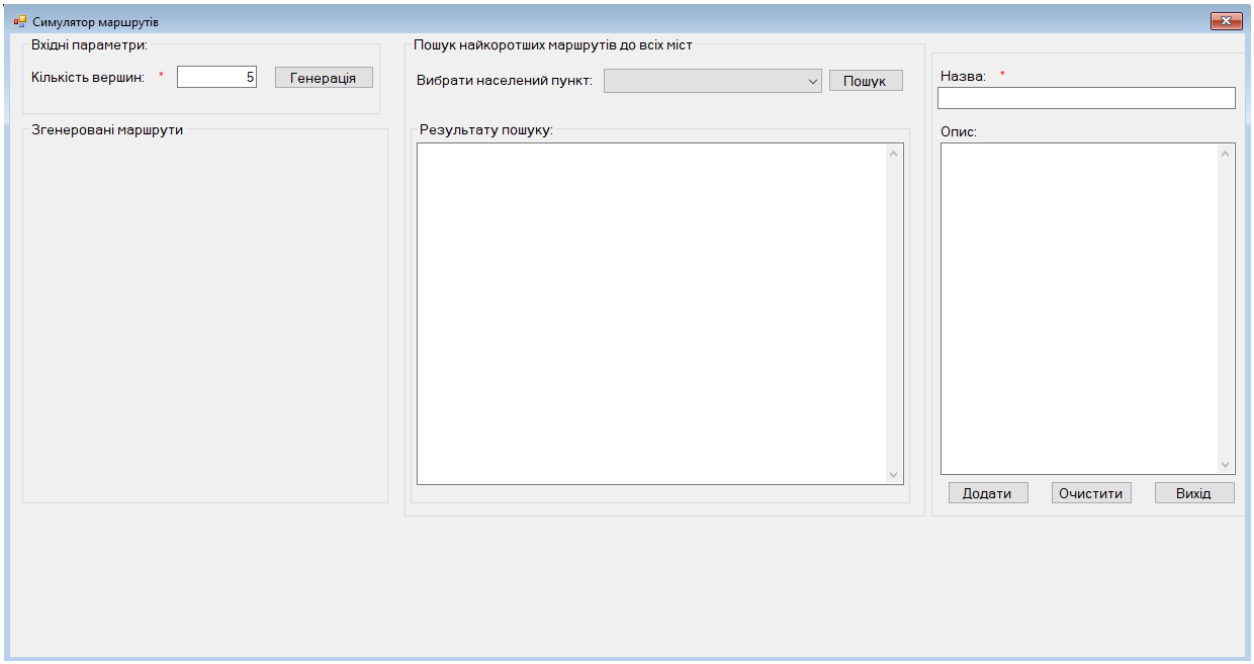


Рисунок 3.7 – Форма для симуляції маршрутів

При завантаженні форми про реєстрацію клієнтів у конструкторі форми був реалізований метод «LoadAllDate», який завантажує дані про міста та населені пункти (рис. 3.8).

```
1 reference
private void LoadAllData() {
    _CitiesList = _CitiesProvider.GetAllCities();
}
```

Рисунок 3.8 – Метод для вибірки інформації про міста та населені пункти

Для проведення симуляції реалізована подія «GenetationBtn_Click», код якої представлено на рис. 3.9. Код методу «LoadData», який викликається у події наведений у додатку.

```
1 reference
private void GenetationBtn_Click(object sender, EventArgs e) {
    if (IsDataEnteringCorrect()) {
        _Graph = new Graph();
        _EdgeSList.Clear();
        //Генеруємо міста
        _VertexsList = GetAllVertexsList(Convert.ToInt32(CountPeaksTBox.Text));
        LoadData();
    }
}
```

Рисунок 3.9 – Подія старту симуляції

Для збереження інформації про симуляцію у формі вікна реалізовано метод «AddBtn_Click», що спрацьовує на натискання кнопки «Додати» (рис. 3.10).

```
1 reference
private void AddBtn_Click(object sender, EventArgs e) {
    if (IsDataSaveCorrect()) {
        LogsProvider logsProvider = new LogsProvider();
        SimulationsProvider simulationsProvider = new SimulationsProvider();
        simulationsProvider.InsertSimulations(SimulationsNameTBox.Text, DescriptionTBox.Text,
          Convert.ToInt32(CountPeaksTBox.Text));
        int lastSimulationsId = simulationsProvider.GetLastRecords();
        for (int i = 0; i < _VertexsList.Count; i++) {
            _VertexsList[i].SimulationsId = lastSimulationsId;
        }
        for (int i = 0; i < _EdgeSList.Count; i++) {
            _EdgeSList[i].SimulationsId = lastSimulationsId;
        }
        _VertexsProvider.InsertBatchVertexs(_VertexsList);
        _EdgeSProvider.InsertBatchEdgeS(_EdgeSList);
        MessageBox.Show("Симуляцію успішно збережено!");
        logsProvider.InsertLogs(LoginForm.CurrentUser.UsersId, "Було збережено симуляцію під назвою: " +
         SimulationsNameTBox.Text, DateTime.Now);
        SimulationsNameTBox.Text = String.Empty;
        DescriptionTBox.Text = String.Empty;
    }
}
```

Рисунок 3.10 – Метод для збереження симуляції

Як можна побачити, в методі на рис. 3.10 викликається метод «IsDataEnteringCorrect», що призначений для перевірки даних на правильність введення, для подальшого їх зберігання в базі даних. Код методу «IsDataSaveCorrect» показаний на рис. 3.11.

```
1 reference
private bool IsDataSaveCorrect() {
    bool isCorrect = true;
    if (_validation.IsDataEntering(SimulationsNameTBox.Text)) {
        SimulationsNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
        SimulationsNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
    }
    if (_validation.IsDataInThisScope(3, 30, Convert.ToInt32(CountPeaksTBox.Text))) {
        CountPeaksValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
        CountPeaksValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
    }
    return isCorrect;
}
```

Рисунок 3.11 – Метод для перевірки введених даних

Отже, в даному розділі було проведено частковий опис основним моментів функціонування розробленого програмного забезпечення.

3.2 Результати функціонального тестування розробленого додатку

Тестування — це процес аналізу та дослідження, який надає змогу виявити інформацію про якість продукту відносно умов, в яких він буде застосовуватись. Методика тестування також включає в себе процес пошуку дефектів, помилок, несправностей. Також це є випробуванням програмних складових з метою оцінити готовність програмного продукту до використання. Результат тестування оцінюється за наступними критеріями:

— відповідність вимогам, які надавалися розробниками та проектувальниками;

— відповідність вихідних даних;

— прийнятний час виконання функцій;

— практичність;

— відповідність вимогам замовника.

Кількість тестів навіть для простих програмних компонентів може бути ледь не нескінченним, тому тактика тестування має полягати в тому, що будуть проведені тільки необхідні тести з урахуванням доступного часу та

ресурсів. Як результат, програмні засоби тестуються стандартним виконанням програми з метою виявлення багів, помилок або інших дефектів.

Існує багато видів тестування: одні зазвичай виконують самі розробники, а інші — спеціалізовані групи. В нашому ж випадку буде використовуватись тестування системи.

Тестування системи — це виконання програмного забезпечення в його остаточної конфігурації, інтегрованого з іншими програмними та апаратними системами.

Одним із способів вивчення поставленого питання є дослідження методики «чорної скриньки», Основна роль тестування методів «чорної скриньки» – це інтерфейс програмного забезпечення. Ці тести демонструють:

— як будуть виконуватись функції програми;

— як будуть прийматися вихідні дані;

— як будуть формуватись результати;

— як буде збережена цілісність зовнішньої інформації.

Тестування програмних засобів буде доречно проводити використовуючи методику «чорної скриньки».

Вона базується на використанні шаблонів тестування або ж тест-кейсів. Це означає, що буде створено декілька ситуацій у яких перевіряється працездатність додатку, коректності основних функцій.

Таблиця 3.1 – Тест-кейси

Код тест-кейса	Опис тест-кейса		
	Хід тестування		Очікуваний результат
	Дата тестування	Результат	Примітка
001	Перевірка реєстрації користувача програми		

	1. Запустити додаток; 2. Провести автентифікацію користувача із роллю «адміністратор».		Вивід відповідного вікна із функціоналом для роботи користувача
	9.12.2022	Пройдено	–
002	Додавання інформації про нове місто		
	1. Запустити додаток; 2. Провести автентифікацію; 3. Додати місто; 4. Редагувати дані міста; 5. Видалити місто.		Список міст відображається відповідно до введених та змінених користувачем даних
	9.12.2022	Пройдено	–
003	Проведення симуляції		
	1. Запустити додаток; 2. Провести автентифікацію; 3. Ввести інформацію про кількість міст; 4. Натиснути кнопку «Генерація».		Програма проводить симуляцію маршрутів відповідно до введених даних користувачем
	9.12.2022	Пройдено	–
004	Перевірка збережених даних симуляції		
	1. Запустити додаток; 2. Провести автентифікацію; 3. Вибрати відповідний пункт меню; 4. Вибрати відповідну симуляцію.		Успішне завантаження даних вибраної симуляції
	9.12.2022	Пройдено	–

3.3 Інструкція користувачеві програми

3.3.1 Мінімальні вимоги для запуску ПЗ

Важливим фактором, який необхідно врахувати при розробці програмного забезпечення є потреба в ресурсах.

Мінімальні вимоги до апаратних засобів для запуску та функціонування розробленого програмного забезпечення:

– Процесор Pentium IV/Xeon2.4Ггц.
– Оперативна пам'ять: 1024 Мб вище.
– Вільний дисковий простір менше 120Мб.
– Миша.
– Клавіатура.
– Монітор.

Вимоги до програмних засобів:

– Операційна система сімейства Windows: починаючи з Windows XP – до Windows 10.
– Наявність бібліотеки класів .NET framework 4.7 або вище.

3.3.2 Опис процедури розгортання програмного продукту, створеного на платформі .NET

Розгортання програмного продукту на комп'ютері користувача у вигляді автономного застосунку:

1. Створіть на цільовому диску каталог для застосунку, наприклад «Оптимізація маршрутів»;
2. Скопіюйте каталог «Debug» із проекту.
3. Для перевірки коректності запуску програми виконайте подвійне клацання лівою кнопкою миші на файлі «HotelApp.exe» (операційна система Windows).

3.3.2 Використання програмного продукту

Запуск програми

Запуск програми в операційній системі сімейства Windows здійснюється одним з стандартних способів:

а) подвійним клацанням лівою кнопкою миші на ярлику програми;

б) викликом контекстного меню з вибором його пункту "Відкрити";

в) натисканням кнопки "Пуск" панелі завдань із подальшим вибором пункту "Усі програми" та подвійним клацанням лівою кнопкою миші на ярлику програми.

Вхід користувача в систему

На початку необхідно запустити додаток «Оптимізація маршрутів». Після запуску програми буде виведено вікно, де користувачу буде запропоновано ввести ім’я та пароль.

Після запуску програми буде виведено вікно, де користувачу буде запропоновано ввести ім’я та пароль (рис.3.12).

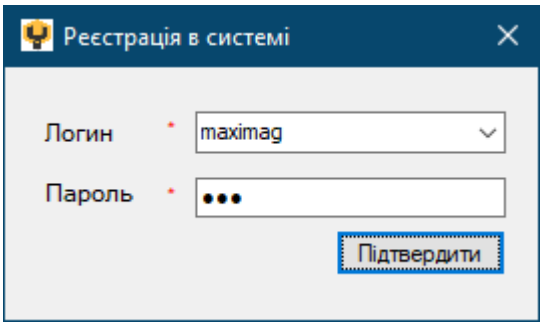


Рисунок 3.12 – Реєстрація користувача в системі

Якщо ж дані буде введена інформація буде не коректною, програма буде попереджувати про це користувача відповідним повідомленням (рис.3.13).

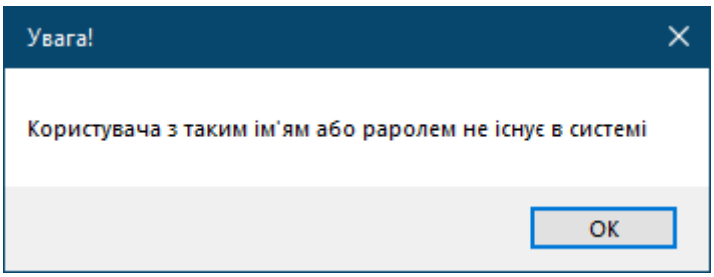


Рисунок 3.13 – Попередження про неправильне введення даних

Для швидшого пошуку можна вибрати ім’я із випадаючого списку або під час введення даних програма поставить у верх списку те ім’я, яке буде вводитись з клавіатури.

Після успішної ідентифікації користувача системи буде відкрите головне вікно програми з основним меню (рис.3.14).

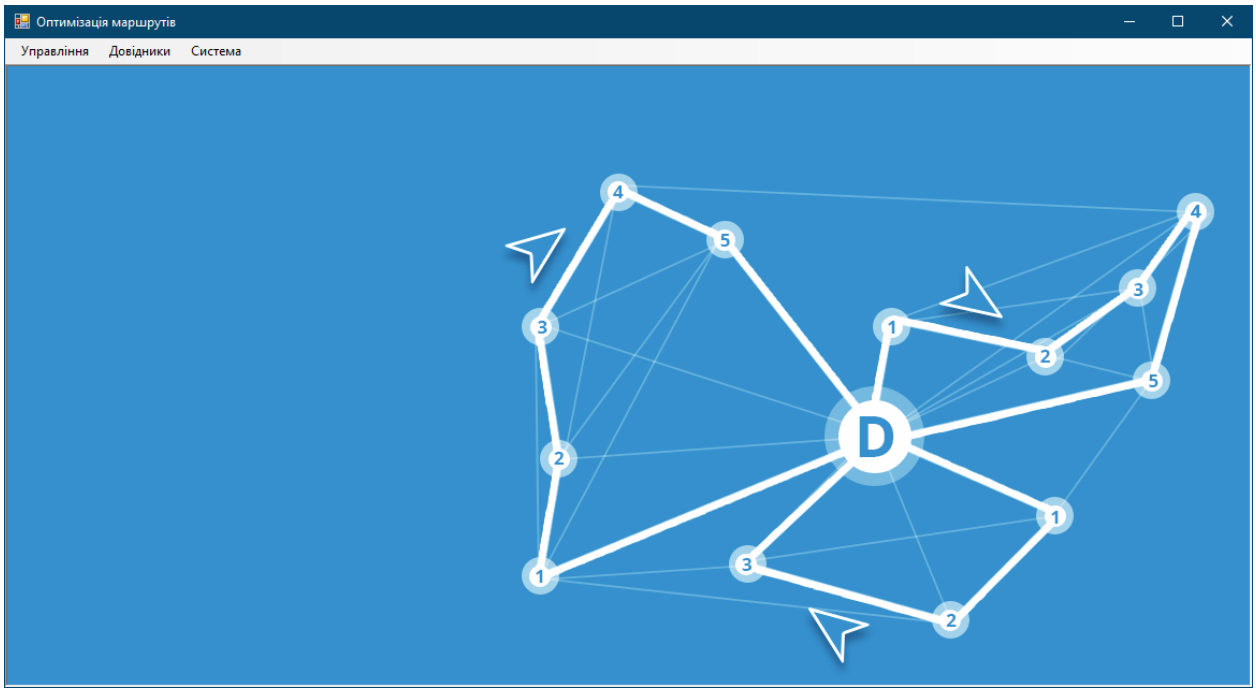


Рисунок 3.14 – Головне меню програми

Для проведення симуляції маршрутів необхідно наповнити інформаційну систему, а саме: інформацію про міста та населені пункти.

Програма дозволяє також редагувати та видаляти введені дані про про міста та населені пункти.

Отже, для початку потрібно додати інформацію про міста та населені пункти. Для цього необхідно перейти по меню програми «Довідники» –> «Міста та населені пункти», у відповідному вікні (рис. 3.15) ввести всі необхідні дані про місто або населений пункт та натиснути кнопку «Додати».

Інформація про нове сміство з’явиться у правій частині екрану у списку всіх міст та населених пунктів.

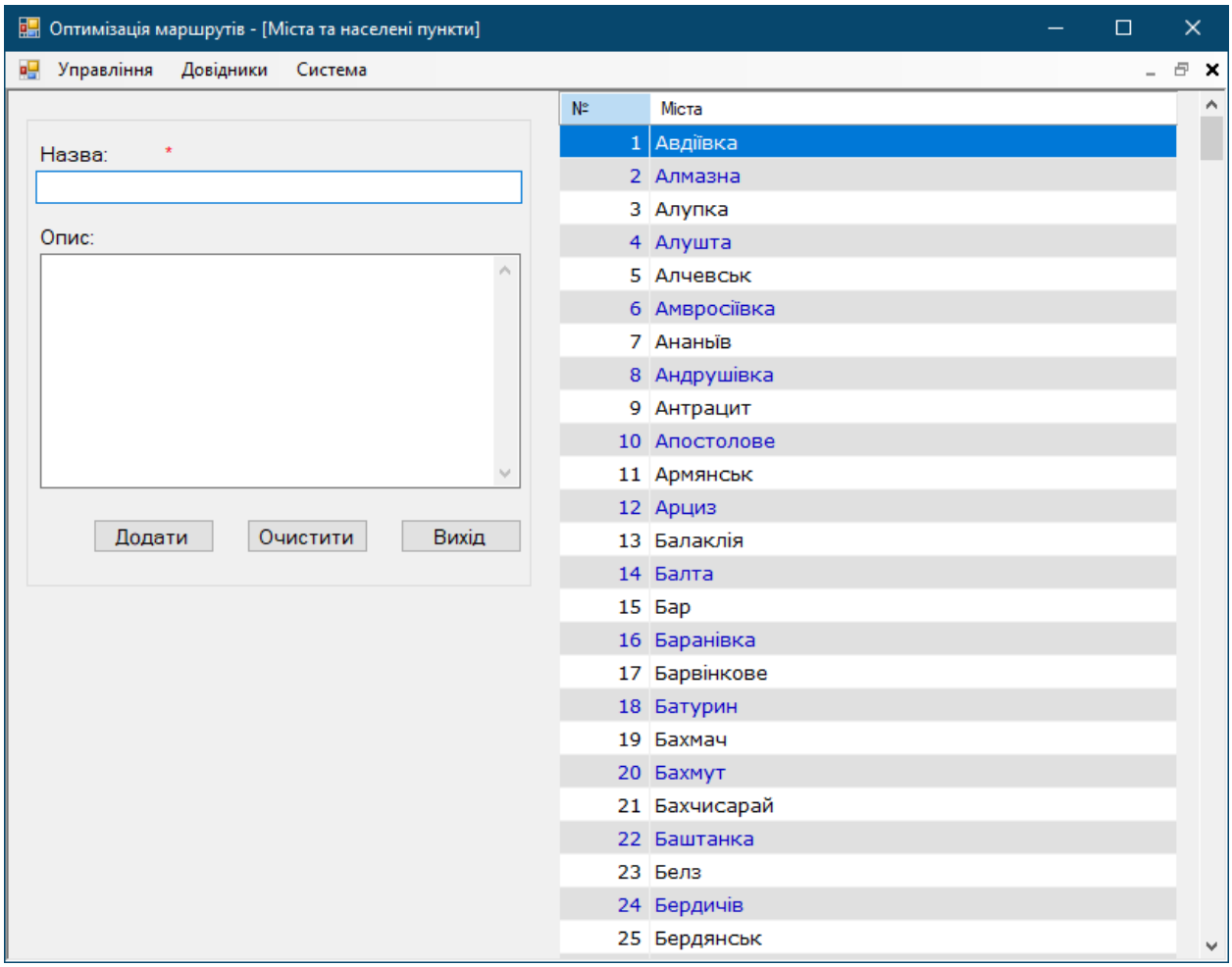


Рисунок 3.15 – Вікно для опрацювання даних про міста та населені пункти

Також реалізована можливість редагування та видалення даних у випадку якщо це є необхідним. На рис 3.16 зображено вікно для редагування даних про міста та населені пункти.

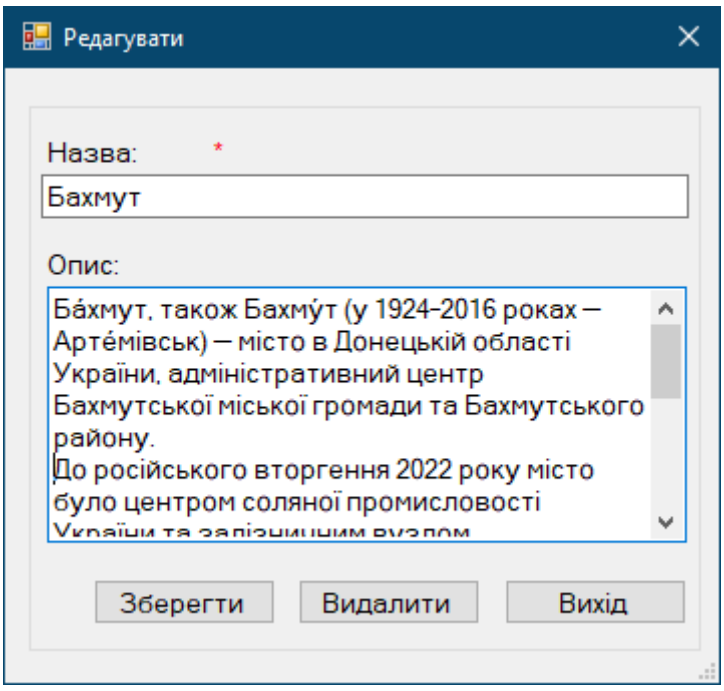


Рисунок 3.16 – Вікно для редагування даних

Для проведення симуляції маршрутів необхідно перейти по меню програми «Управління» –> «Симулятор маршрутів», після чого відкриється вінкно, що представлено на рис. 3.17. Обов’язковим параметром, який необхідно вказати у даному вікні є кількість міст. Вказавши цей параметр та натиснувши кнопку «Генерувати», програма випадковим чином вибере із бази даних міста та населені пунки, після чого згенерує маршрути між ними та виведе результати на екран.

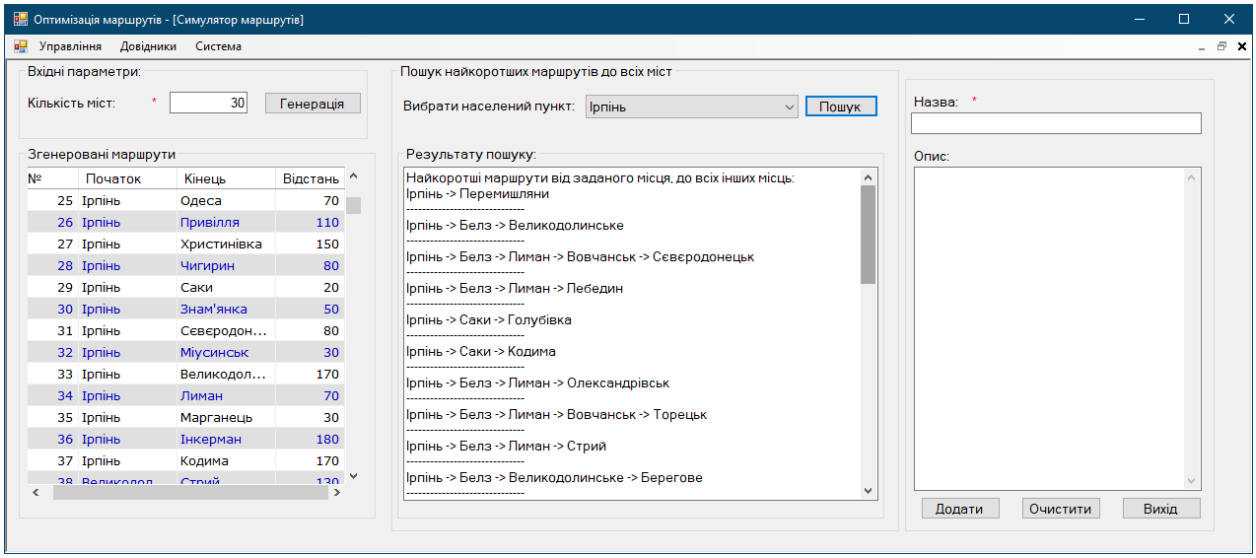


Рисунок 3.17 – Вікно для проведення симуляції маршрутів

Вибираючи населений пункт або місто у відповідному полі, можна прокласти маршрути із цього пункту до всіх інших пунктів.

Вказавши назву симуляції, зробивши її опис та натиснувши кнопку «Додати», дані можна зберегти у базі даних.

Для завантаження даних із бази даних необхідно перейти по меню програми «Управління» –> «Завантаження маршрутів» (рис. 3.18).

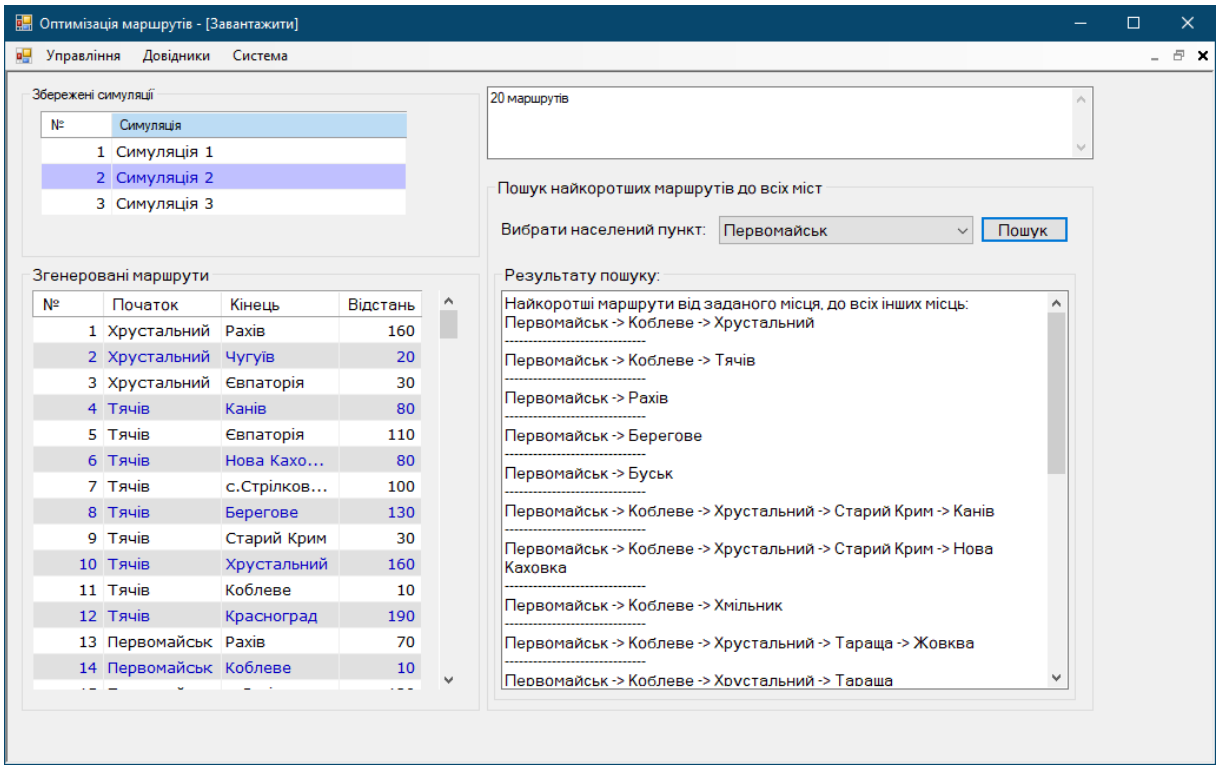


Рисунок 3.18 – Вікно для завантаження проведених симуляцій

Користувач із роллю «Системний адміністратор» має доступ до управління обліковими записами системи (рис. 3.19).

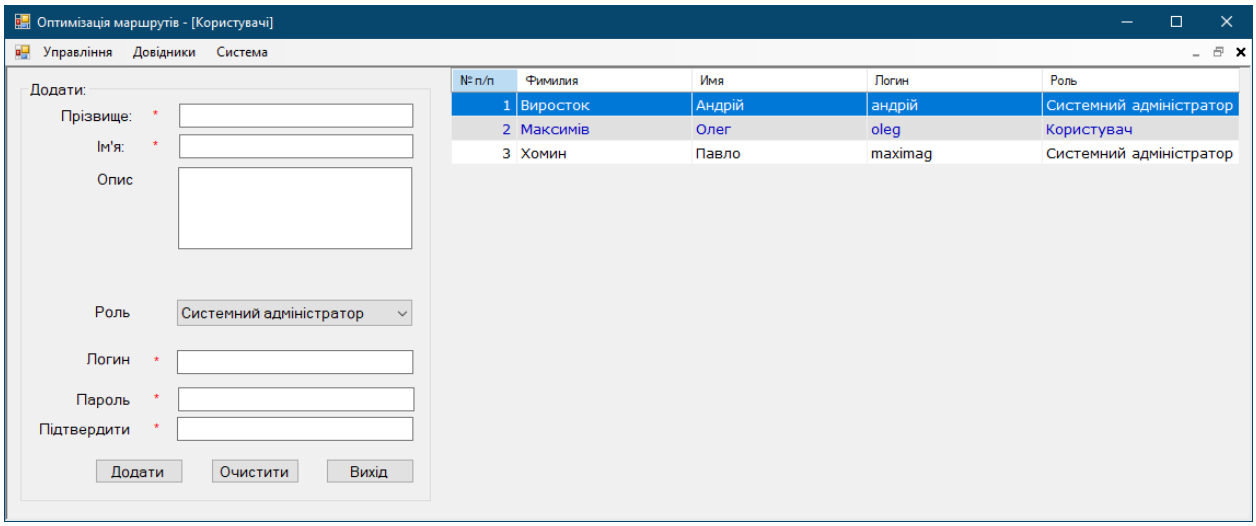


Рисунок 3.19 – Вікно для управління користувачами системи

При необхідності дані та пароль будь-якого користувача можна змінити (рис. 3.20).

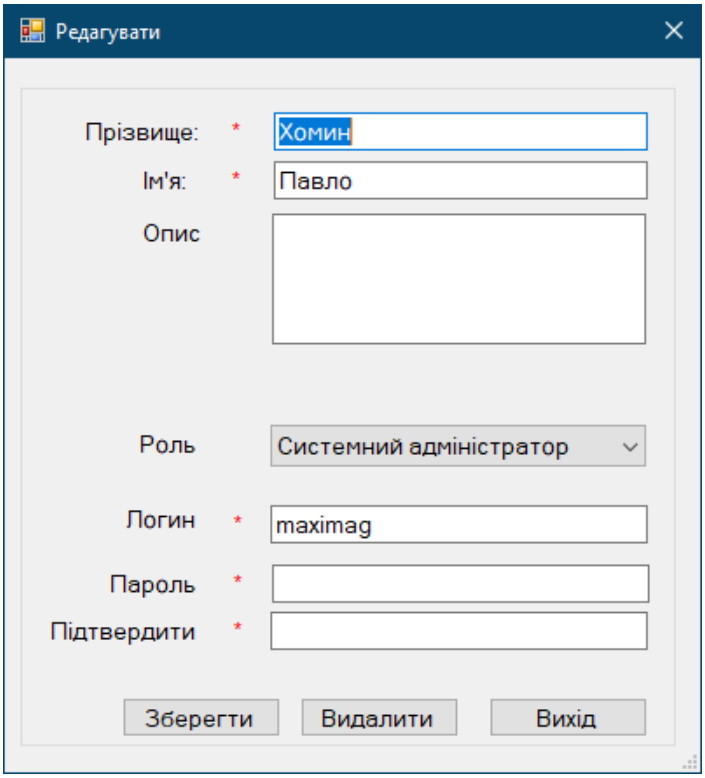


Рисунок 3.20 – Редагування даних користувача системи

Якщо ж поля «Пароль» та «Підтвердити» не співпадають, програма виведе повідомлення про це (рис. 3.21).

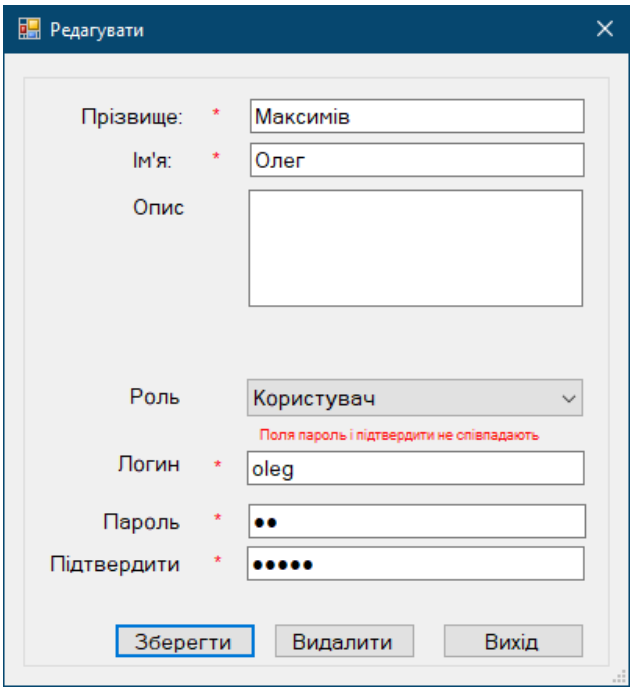
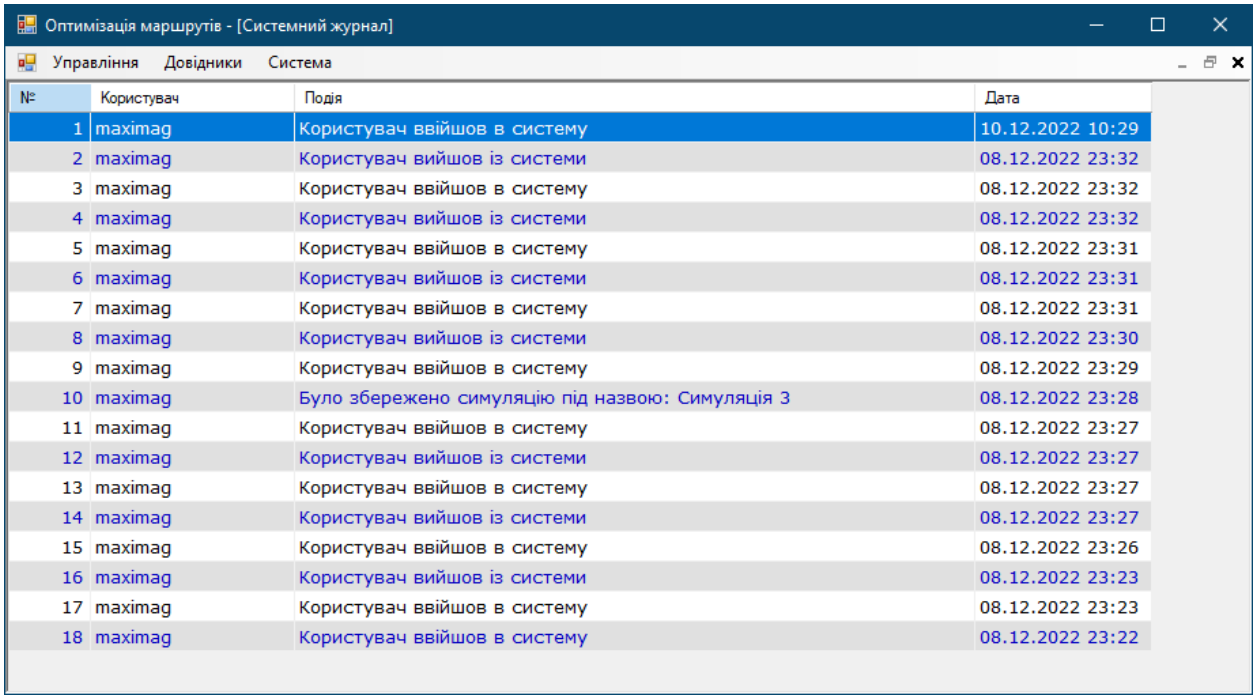


Рисунок 3.21 – Пароль» та «Підтвердити» не співпадають

Також в системі можна бачити активність користувачів, та те, що вони робили в системі. Це можливо зробити за допомогою облікового запису, який має права системного адміністратора. Для цього перейдемо по меню «Система» –> «Системний журнал». В цьому вікні виводяться всі події системи.



№	Користувач	Подія	Дата
1	maximag	Користувач ввійшов в систему	10.12.2022 10:29
2	maximag	Користувач вийшов із системи	08.12.2022 23:32
3	maximag	Користувач ввійшов в систему	08.12.2022 23:32
4	maximag	Користувач вийшов із системи	08.12.2022 23:32
5	maximag	Користувач ввійшов в систему	08.12.2022 23:31
6	maximag	Користувач вийшов із системи	08.12.2022 23:31
7	maximag	Користувач ввійшов в систему	08.12.2022 23:31
8	maximag	Користувач вийшов із системи	08.12.2022 23:30
9	maximag	Користувач ввійшов в систему	08.12.2022 23:29
10	maximag	Було збережено симуляцію під назвою: Симуляція 3	08.12.2022 23:28
11	maximag	Користувач ввійшов в систему	08.12.2022 23:27
12	maximag	Користувач вийшов із системи	08.12.2022 23:27
13	maximag	Користувач ввійшов в систему	08.12.2022 23:27
14	maximag	Користувач вийшов із системи	08.12.2022 23:27
15	maximag	Користувач ввійшов в систему	08.12.2022 23:26
16	maximag	Користувач вийшов із системи	08.12.2022 23:23
17	maximag	Користувач ввійшов в систему	08.12.2022 23:23
18	maximag	Користувач ввійшов в систему	08.12.2022 23:22

Рисунок 3.22 – Вікно «Системний журнал»

Для виходу із програми необхідно перейти по меню «Управління» –> «Вихід».

Треба сказати, що дана система є не сильно функціональною, але вона є досить простою в користуванні. ЇЇ інтерфейс є інтуїтивно зрозумілим для користувача.

Тестування програми успішно проведено та не було виявлено жодних помилок системи.

3.4 Висновок

В даному розділі було частково описано процес розробки основного класу "Dijkstra". Складено алгоритм його роботи та описані основні класи та методи, що призначені для оптимізації маршрутів.

В ході виконання роботи мовою C# в середовищі Visual Studio 2019 реалізовано додаток для оптимізації маршрутів пасажирських та транспортних перевезень.

Реалізовано такі функціональні вимоги:

– можливість додавати, редагувати та видаляти облікові записи користувачів;

– можливість додавати, редагувати та видаляти дані про населені пункти та міста;

– можливість проведення симуляції маршрутів із вибраної кількості міст;

– можливість зберігання та завантаження даних проведеної симуляції;

– фіксування активності користувачів системи. Ведення документування подій системи.

Також будо проведено функціональне тестування розробленої системи та розроблено інструкцію користувача програми.

ВИСНОВКИ

Даний проект розроблявся для здійснення оптимізації маршрутів на основі об'єктно-орієнтованого програмування. Розробку інформаційної системи було виконано у середовищі Microsoft Visual Studio 2019 при використанні мови програмування C # та СУБД MS Access. Дана інформаційна система повинна значно полегшити роботу менеджерів, що вирішують проблеми вантажного та пасажирського перевезення. Також, розроблене ПЗ має зручний зручний та інтуїтивно зрозумілий інтерфейс користувача

У розробленому продукту реалізовано такі функціональні можливості:

– можливість додавати, редагувати та видаляти облікові записи користувачів;

– можливість додавати, редагувати та видаляти дані про можливість додавати, редагувати та видаляти облікові записи користувачів;

– можливість додавати, редагувати та видаляти дані про населені пункти та міста;

– можливість проведення симуляції маршрутів із вибраної кількості міст;

– можливість зберігання та завантаження даних проведеної симуляції;

– фіксування активності користувачів системи. Ведення документування подій системи.

У першій частині роботи було розглянуте поняття транспортної логістики. Виділено основні аспекти, які необхідно враховувати при застосуванні логістичних принципів у процесі управління підприємством транспортного комплексу. Побудовано схему логістичних систем пасажирського транспорту. Після чого було проведено аналіз точних та неточних алгоритмів пошуку оптимального шляху та встановлено, що неточні алгоритми мають велику перевагу при розрахунках маршрутів для великої кількості міст. Проведене дослідження перспективних алгоритмів показало, що алгоритм Дейкстри показує високу ефективність у задачах з великою

кількістю міст. Головна його перевага - можливість використання для вирішення складних неформалізованих проблем для яких не існує приватних методів вирішення, що дозволяє ефективно вирішувати нестандартні завдання. Отже, для розробки інформаційної системи оптимізації маршрутів було взято за основу реалізацію алгоритму Дейкстри.

У другій частині роботи розглянуто дослідження системи та описано її архітектуру. Зроблено аналіз вимог до програмного забезпечення та побудовано use-case діаграми основних прецедентів.

В третій частині було здійснено розробку програмного забезпечення та проведено тестування. Також, було розроблено інструкцію користувача програми. Результати тестування були успішними, помилок не було виявлено.

В результаті проведеної роботи вирішено актуальне технічне завдання для оптимізації маршрутів пасажирських та транспортних перевезень. У процесі вирішення завдання розроблено інженерну методику автоматизованої процедури проведення симуляції маршрутів та розрахунку найкоротших маршрутів між містами та населеними пунктами, і таким чином поставленої мети досягнуто. У ході виконання роботи отримано такі основні наукові та практичні результати:

1. Розроблено гнучку систему, призначену для оптимізації маршрутів пасажирських та транспортних перевезень.

2. Розроблені методи в даній роботі можуть бути використані для широкого класу завдань, тому можливий подальший розвиток розробленого програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Маршрутизація : [Електронний ресурс]. – Режим доступу: https://www.jnsm.com.ua/cgi-bin/u/book/sis.pl?Qry=%EC%E0%F0%F8%F0%F3%F2%E8%E7%E0%F6%B3%FF .

2. Алгоритми маршрутизації : [Електронний ресурс]. – Режим доступу: http://ni.biz.ua/7/7_9/7_95359_algoritmi-marshrutizatsii-printsip-optimalnosti-marshruta-vibor-kratchayshego-puti-zalivka.html

3. Транспорт - як галузь матеріального виробництва : [Електронний ресурс]. – Режим доступу: http://um.co.ua/4/4-16/4-169419.html

4. Гуторов О. І. Логістика: навч. посібник / О. І. Гуторов, О. І. Лебединська, Н. В. Прозорова / Харк. нац. аграр. ун-т. – Харків: Міськдрук. 2011. – 322 с.

5. Горяїнов О.М. Теорія і практика дисципліни «Логістика» (для менеджерів): Навчальний посібник. – Харків: НТМТ, 2009. – 522 с. (Серія «Скарбничка знань фахівця з логістики»)

6. Крикавський Є. Логістичне управління: Підручник. – Львів: Видавництво Національного університету “Львівська політехніка”, 2005. – 684с.

7. лькема В.Г., Сумець О.М. Логістика. Теорія та практика. Навчальний посібник. – К.: «Видавничий дім «Професіонал», 2008. – 272с.

8. Окландер М.А. Логістика: Навчальний посібник. – К.:Зовнішня торгівля, 2005. – 234с.

9. Пономарьова Ю.В. Логістика: Навчальний посібник. Київ: Центр навчальної літератури, 2003. – 192с.

10. Таньков К.М., Тридід О.М., Колодизєва Т.О. Виробнича логістика:Навчальний посібник. – Х.: Видавничий Дім “ІНЖЕК”, 2004. – 352с

11. Тридід О.М., Таньков К.М., Леонова Ю.О. Логістика. Навчальний посібник. – К.: Видавничий дім «Професіонал», 2008. – 176с.

12. Чухрай Н., Патора Р. Інновації та логістика товарів: Монографія. – Львів: Видавництво Національного університету “Львівська політехніка”, 2001. – 264с.

13. Горяїнов О.М., Рославцев Д.М. Автотранспорт в логістичних системах і ланцюгах. Монографія. – Харків: НТМТ, 2009. – 344с.

14. Германчук А.М. Маркетингова логістика: сутність і значення: [Електронний ресурс]. – Режим доступу: http://www.rusnauka.com/7_NND_2009/Economics/42641.doc.htm

15. Пономарьова Ю. В. Логістика : навч. посіб. / Ю. В. Пономарьова. – К. : Центр навчальної літератури, 2005. – 328 с.

16. Lambert D. M. Fundamentals of logistics management / D. M. Lambert, J. R. Stock, L. M. Ellram. – New York : Mc Graw-Hill, 2008. – 611 p.

17. Марунич В.С., Шморгун Л.Г. Організація та управління пасажирськими перевезеннями: підручник/ за ред. доц. В.С. Марунич, проф. Л.Г. Шморгуна – К.: Міленіум, 2017. – 528 с.

18. Велісевич М.К., Ігнатенко О.С., Марунич В.С. Концептуальні основи координації автомобільного та залізничного транспорту. – К.: Автошляхових України. – 1995. – Вип. 4. – С. 7-9

19. Kona H., Burde A., Dr. Zanwar D. R. A Review of Traveling Salesman Problem with Time Window Constraint // IJIRST – International Journal for Innovative Research in Science & Technology, 2015. Vol. 2, Issue 1. P. 253– 254.

20. Tannenbaum P. Excursions in Mathematics. University of Kansas, 2011. P. 25.

21. Branch and Bound Algorithm : [Електронний ресурс]. – Режим доступу: https://www.geeksforgeeks.org/branch-and-bound-algorithm/

22. Cutting Plane : [Електронний ресурс]. – Режим доступу: https://www.sciencedirect.com/topics/engineering/cutting-plane

23. Алгоритм Дейкстри : [Електронний ресурс]. – Режим доступу: https://disted.edu.vn.ua/media/doc
/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC

%20%D0%94%D1%96%D0%B5%D0%BA%D1%81%D1%82%D1%80%D0%B8.pdf

24. Nearest neighbour algorithm : [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Nearest_neighbour_algorithm

25. Marcin Wojnarski «TomTom Traffic Prediction for Intelligent GPS Navigation» // M. Wojnarski, IEEE International Conference on Data Mining Workshops – 2010 – С.20-21.

26. ViaMichelin Navigation. User Manual [Електронний ресурс] // ViaMichelin, URL: http://enav.download.viamichelin.com/nav/tel/manuels/gbr/User_Manual_GBR_VMN_New_Edition_v7.pdf .

27. The Algorithms Behind The Working Of Google Maps : [Електронний ресурс]. – Режим доступу: https://blog.codechef.com/2021/08/30/the-algorithms-behind-the-working-of-google-maps-dijkstras-and-a-star-algorithm/

28. YourNavigation.org About [Електронний ресурс] // YOURS, URL: https://wiki.openstreetmap.org/wiki/YOURS/.

29. OpenRouteService : [Електронний ресурс]. – Режим доступу: https://ask.openrouteservice.org/tos.

30. Литвинов О.А., Карпенко Н.В. Тестування інформаційних систем: модульне, інтеграційне, системне [Текст] – Д.: Ліра, 2016. – 283 с.

31. Литвинов О.А., Герасимов В.В., Карпенко Н.В. Об’єктно-орієнтована розробка інформаційних систем [Текст] – Д.: Ліра, 2018. – 448 с.

32. .NET 4.7 [Електронний ресурс] Режим доступу: https://temofeev.ru/info/articles/predstavlyaem-net-5/

33. Програма для роботи з базами даних Microsoft Access: [Електронний ресурс] // Режим доступу: https://www.microsoft.com/uk-ua/microsoft-365/access.

34. Джон Скит, C# для професіоналів: тонкощі програмування, 608 ст. ISBN 978-5-8459-1909-0, 978-1-617-29134-0; 2014, Вільямс.

35. Мова програмування C# і платформа .NET Core [Електронний ресурс] Режим доступу: https://metanit.com/sharp/tutorial/1.1.php

36. What's new in .NET 4.7 [Електронний ресурс] Режим доступу: https://docs.microsoft.com/en-us/dotnet/core/dotnet-five.

37. Критерії хорошої архітектури. Ієрархія принципів проектування https://org2.knuba.edu.ua/pluginfile.php/56721/mod_resource/content/1/Lek_4_%D0%90%D0%9F%D0%9F%D0%97.pdf

ДОДАТОК А. Лістинги програм

Лістинг 1. Код класу «RouteSimulatorForm»

```
using RouteOptimizerApp.AppCode;
using RouteOptimizerApp.Forms.Systems;
using RouteOptimizerApp.Provider;
using RouteOptimizerApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace RouteOptimizerApp.Forms.Control {
  public partial class RouteSimulatorForm : Form {
    Random _GetRandom = new Random();

    private Graph _Graph;
    private Dijkstra _Dijkstra ;
    private CitiesProvider _CitiesProvider = new CitiesProvider();
    private List<Cities> _CitiesList = new List<Cities>();
    private VertexsProvider _VertexsProvider = new VertexsProvider();
    private List<Vertexs> _VertexsList = new List<Vertexs>();
    private EdgeSProvider _EdgeSProvider = new EdgeSProvider();
    private List<EdgeS> _EdgeSList = new List<EdgeS>();
    private ValidationMy _validation = new ValidationMy();

    public RouteSimulatorForm() {
      InitializeComponent();
      LoadAllData();
```

```
}

private void LoadAllData() {
  _CitiesList = _CitiesProvider.GetAllCities();
}

private void GenetationBtn_Click(object sender, EventArgs e) {
  if (IsDataEnteringCorrect()) {
    _Graph = new Graph();
    _EdgeSList.Clear();
    //Генеруємо міста
    _VertexsList = GetAllVertexsList(Convert.ToInt32(CountPeaksTBox.Text));
    LoadData();
  }
}

private bool IsDataEnteringCorrect() {
  bool isCorrect = true;
  if (_validation.IsDataInThisScope(3, 30, Convert.ToInt32(CountPeaksTBox.Text))) {
    CountPeaksValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    CountPeaksValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  return isCorrect;
}



public void LoadData() {
  List<EdgeS> edgeSList = new List<EdgeS>();

  //Додавання вершин
  for (int i = 0; i < _VertexsList.Count; i++) {
```

```
  _Graph.AddVertex(_VertexsList[i].VertexsName);
  VertexCBox.Items.Add(_VertexsList[i].VertexsName);
 }

VertexCBox.SelectedIndex = 0;

//Додавання ребер
for (int i = 0; i < _VertexsList.Count; i++) {
  int genRoutCount = _GetRandom.Next(0, _VertexsList.Count - 1) + 1;
  int count = 0;
  while (count < genRoutCount) {
    EdgeS oneEdgeS = new EdgeS();
    int genCitiesId = _GetRandom.Next(0, _VertexsList.Count);
    if (i != genCitiesId) {
      oneEdgeS.RoutStart = _VertexsList[i].VertexsName;
      oneEdgeS.RoutEnd = GetCitiesName(genCitiesId, _VertexsList);
      oneEdgeS.Distance = _GetRandom.Next(1, 20) * 10;

      if (!IsThisRoutExist(oneEdgeS, edgeSList)) {
        edgeSList.Add(oneEdgeS);
        count++;
      }
    }
  }
}

for (int i = 0; i < edgeSList.Count; i++) {
  EdgeS oneEdgeS = new EdgeS();
  oneEdgeS.Number = i + 1;
  oneEdgeS.RoutStart = edgeSList[i].RoutStart;
  oneEdgeS.RoutEnd = edgeSList[i].RoutEnd;
  int distanse = GetDisnance(oneEdgeS, _EdgeSList);
  if ( distanse > -1) {
```

```
            oneEdgeS.Distance = distanse;
          } else{
            oneEdgeS.Distance = edgeSList[i].Distance;
          }
          _EdgeSList.Add(oneEdgeS);
        }


        for (int i = 0; i < _EdgeSList.Count; i++) {
          //ResultTBox.Text += _EdgeSList[i].RoutStart + "-" + _EdgeSList[i].RoutEnd + "-" + _EdgeSList[i].Distance + "\r\n";
          _Graph.AddEdge(_EdgeSList[i].RoutStart, _EdgeSList[i].RoutEnd, _EdgeSList[i].Distance);
        }
        LoadDataInEdgeSGridView(_EdgeSList);
      }

      private int GetDisnance(EdgeS CurEdgeS, List<EdgeS> EdgeSList) {
        for (int i = 0; i < EdgeSList.Count; i++) {
          if (CurEdgeS.RoutStart == EdgeSList[i].RoutEnd && CurEdgeS.RoutEnd == EdgeSList[i].RoutStart) {
            return EdgeSList[i].Distance;
          }
        }
        return -1;
      }

      private string GetCitiesName(int CitiesId, List<Vertexs> Vertexs) {
        return Vertexs[CitiesId].VertexsName;
      }

      private bool IsThisRoutExist(EdgeS genEdgex, List<EdgeS> EdgeList) {
```

```
            for (int i = 0; i < EdgeList.Count; i++) {
              if (genEdgex.RoutStart == EdgeList[i].RoutStart && genEdgex.RoutEnd == EdgeList[i].RoutEnd) {
                return true;
              }
            }
            return false;
          }

          private List<Vertexs> GetAllVertexsList(int PeaksCount) {
            List<Vertexs> selVertexsList = new List<Vertexs>();
            int count = 0;

            while (count < PeaksCount) {
              int randCitiesId = _GetRandom.Next(0, _CitiesList.Count - 1);
              if (!IsCitiesPeakExistInList(randCitiesId, selVertexsList)) {
                Cities oneCities = new Cities();
                oneCities = GetCities(randCitiesId, _CitiesList);
                count++;
                Vertexs oneVertexs = new Vertexs();
                oneVertexs.CitiesId = oneCities.CitiesId;
                oneVertexs.VertexsName = oneCities.CitiesName;
                selVertexsList.Add(oneVertexs);
              }
            }
            return selVertexsList;
          }

          private bool IsCitiesPeakExistInList(int CitiesId, List<Vertexs> Vertexs) {
            for (int i = 0; i < Vertexs.Count; i++) {
              if (CitiesId == Vertexs[i].CitiesId) {
                return true;
              }
            }
```

```
            return false;
          }
          private Cities GetCities(int CitiesId, List<Cities> CitiesList) {
            for (int i = 0; i < CitiesList.Count; i++) {
              if (CitiesId == CitiesList[i].CitiesId) {
                return CitiesList[i];
              }
            }
            return null;
          }

          private void SearchBtn_Click(object sender, EventArgs e) {
            if (VertexCBox.Text != "") {
              _Dijkstra = new Dijkstra(_Graph);
              ResultTBox.Text = "Найкоротші маршрути від заданого місця, до всіх інших місць: \r\n";
              for (int i = 0; i < Convert.ToInt32(CountPeaksTBox.Text); i++) {
                if (VertexCBox.Text != i.ToString()) {
                  var path = _Dijkstra.FindShortestPath(VertexCBox.Text, _VertexsList[i].VertexsName);
                  if (path != VertexCBox.Text) {
                    ResultTBox.Text += path + "\r\n";
                    ResultTBox.Text += "------------------------------\r\n";
                  }
                }
              }
            }
          }

          private void LoadDataInEdgeSGridView(List<EdgeS> EdgeSList) {
            EdgeSGridView.DataSource = null;
            EdgeSGridView.Columns.Clear();
            EdgeSGridView.AutoGenerateColumns = false;
```

```
            EdgeSGridView.RowHeadersVisible = false;

            EdgeSGridView.DataSource = EdgeSList;

            if (EdgeSList.Count > 0) {
              if (EdgeSList[0].Message == NamesMy.NoDataNames.NoDataInEdgeS) {
                DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
                messageColumn.DataPropertyName = "Message";
                messageColumn.Width = EdgeSGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;
                EdgeSGridView.Columns.Add(messageColumn);
              } else {
                DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
                DetailIdColumn.DataPropertyName = "ClientId";
                EdgeSGridView.Columns.Add(DetailIdColumn);
                EdgeSGridView.Columns[0].Visible = false;

                DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
                numberColumn.HeaderText = "№ ";
                numberColumn.DataPropertyName = "Number";
                numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
                numberColumn.Width = NamesMy.SizeOptins.NumberSize;
                numberColumn.ReadOnly = true;
                EdgeSGridView.Columns.Add(numberColumn);

                DataGridViewColumn RoutStartColumn = new DataGridViewTextBoxColumn();
                RoutStartColumn.HeaderText = "Початок";
                RoutStartColumn.DataPropertyName = "RoutStart";
                RoutStartColumn.Name = "RoutStart";
                RoutStartColumn.Width = 100;
                RoutStartColumn.ReadOnly = true;
                EdgeSGridView.Columns.Add(RoutStartColumn);
```

```
                DataGridViewColumn RoutEndColumn = new DataGridViewTextBoxColumn();
                RoutEndColumn.HeaderText = "Кінець";
                RoutEndColumn.DataPropertyName = "RoutEnd";
                RoutEndColumn.Name = "RoutEnd";
                RoutEndColumn.Width = 100;
                RoutEndColumn.ReadOnly = true;
                EdgeSGridView.Columns.Add(RoutEndColumn);

                DataGridViewColumn DistanceColumn = new DataGridViewTextBoxColumn();
                DistanceColumn.HeaderText = "Відстань";
                DistanceColumn.DataPropertyName = "Distance";
                DistanceColumn.Name = "Distance";
                DistanceColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
                DistanceColumn.Width = 70;
                DistanceColumn.ReadOnly = false;
                EdgeSGridView.Columns.Add(DistanceColumn);

              }
              for (int i = 0; i < EdgeSGridView.Columns.Count; i++) {
                EdgeSGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
              }
            }
          }

          private void AddBtn_Click(object sender, EventArgs e) {
            if (IsDataSaveCorrect()) {
              LogsProvider logsProvider = new LogsProvider();
              SimulationsProvider simulationsProvider = new SimulationsProvider();
              simulationsProvider.InsertSimulations(SimulationsNameTBox.Text,
DescriptionTBox.Text,
                Convert.ToInt32(CountPeaksTBox.Text));
              int lastSimulationsId = simulationsProvider.GetLastRecords();
              for (int i = 0; i < _VertexsList.Count; i++) {
```

```
                _VertexsList[i].SimulationsId = lastSimulationsId;
              }
              for (int i = 0; i < _EdgeSList.Count; i++) {
                _EdgeSList[i].SimulationsId = lastSimulationsId;
              }
              _VertexsProvider.InsertBatchVertexs(_VertexsList);
              _EdgeSProvider.InsertBatchEdgeS(_EdgeSList);
              MessageBox.Show("Симуляцію успішно збережено!");
              logsProvider.InsertLogs(LoginForm.CurrentUser.UsersId, "Було збережено
симуляцію під назвою: " +
               SimulationsNameTBox.Text, DateTime.Now);
              SimulationsNameTBox.Text = String.Empty;
              DescriptionTBox.Text = String.Empty;
            }
          }

          private void ClearBtn_Click(object sender, EventArgs e) {
            SimulationsNameTBox.Text = String.Empty;
            DescriptionTBox.Text = String.Empty;
          }

          private void ExitBtn_Click(object sender, EventArgs e) {
            this.Close();
          }



          private bool IsDataSaveCorrect() {
            bool isCorrect = true;
            if (_validation.IsDataEntering(SimulationsNameTBox.Text)) {
              SimulationsNameValiadtionLbl.Text                                    =
NamesMy.ProgramButtons.RequiredValidation;
            } else {
              SimulationsNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
              isCorrect = false;
```

```
            }
            if (_validation.IsDataInThisScope(3, 30, Convert.ToInt32(CountPeaksTBox.Text))) {
              CountPeaksValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
            } else {
              CountPeaksValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
              isCorrect = false;
            }
            return isCorrect;
          }
        }
      }
```

Лістинг 2. Код класу «SimulationsLoadForm»

```
using RouteOptimizerApp.AppCode;
using RouteOptimizerApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace RouteOptimizerApp.Forms.Control {
  public partial class SimulationsLoadForm : Form {
    private int _selectedRowIndex = 0;
    private Graph _Graph;
    private Dijkstra _Dijkstra;

    private SimulationsProvider _SimulationsProvider = new SimulationsProvider();
    private List<Simulations> _SimulationsList = new List<Simulations>();
    private Simulations _SelectSimulations = new Simulations();
    private EdgeSProvider _EdgeSProvider = new EdgeSProvider();
    private List<EdgeS> _EdgeSList = new List<EdgeS>();
    private VertexsProvider _VertexsProvider = new VertexsProvider();
    private List<Vertexs> _VertexsList = new List<Vertexs>();


    public SimulationsLoadForm() {
      InitializeComponent();
      DataLoad();
    }
```

```
  private void DataLoad() {
    int firstRowIndex = 0;
    if (SimulationsGridView.FirstDisplayedScrollingRowIndex > 0) {
      firstRowIndex = SimulationsGridView.FirstDisplayedScrollingRowIndex;
    }
    try {
      _SimulationsList = _SimulationsProvider.GetAllSimulations();
      LoadDataInSimulationsGridView(_SimulationsList);
      if (_selectedRowIndex == SimulationsGridView.Rows.Count) {
        _selectedRowIndex = SimulationsGridView.Rows.Count - 1;
      }
      if (_selectedRowIndex >= 0) {
        SimulationsGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
        SimulationsGridView.Rows[_selectedRowIndex].Selected = true;
      }
    } catch { }
  }

  private void LoadDataInSimulationsGridView(List<Simulations> SimulationsList) {
    SimulationsGridView.DataSource = null;
    SimulationsGridView.Columns.Clear();
    SimulationsGridView.AutoGenerateColumns = false;
    SimulationsGridView.RowHeadersVisible = false;

    SimulationsGridView.DataSource = SimulationsList;

    if (SimulationsList.Count > 0) {
      if (SimulationsList[0].Message == NamesMy.NoDataNames.NoDataInSimulations) {
        DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
        messageColumn.DataPropertyName = "Message";
        messageColumn.Width = SimulationsGridView.Width -
NamesMy.SizeOptins.MinusSizePanel;
        SimulationsGridView.Columns.Add(messageColumn);
      } else {
        DataGridViewColumn SimulationsIdColumn = new DataGridViewTextBoxColumn();
        SimulationsIdColumn.DataPropertyName = "SimulationsId";
        SimulationsGridView.Columns.Add(SimulationsIdColumn);
        SimulationsGridView.Columns[0].Visible = false;

        DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
        numberColumn.HeaderText = "№ ";
        numberColumn.DataPropertyName = "Number";
        numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        numberColumn.Width = NamesMy.SizeOptins.NumberSize;
        numberColumn.ReadOnly = true;
        SimulationsGridView.Columns.Add(numberColumn);

        DataGridViewColumn RoutStartColumn = new DataGridViewTextBoxColumn();
        RoutStartColumn.HeaderText = "Симуляція";
        RoutStartColumn.DataPropertyName = "SimulationsName";
        RoutStartColumn.Width = 250;
```

```
        RoutStartColumn.ReadOnly = true;
        SimulationsGridView.Columns.Add(RoutStartColumn);
      }
      for (int i = 0; i < SimulationsGridView.Columns.Count; i++) {
        SimulationsGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
    }
  }

  private void SimulationsGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
    if (e.RowIndex >= 0 && SimulationsGridView[0, e.RowIndex].Value.ToString() !=
_SimulationsList[0].Message) {
      _Graph = new Graph();
      VertexCBox.Items.Clear();
      _selectedRowIndex = Convert.ToInt32(SimulationsGridView[0,
e.RowIndex].Value.ToString());
      _SelectSimulations =
_SimulationsProvider.SelectedSimulationsBySimulationsId(_selectedRowIndex);
      DescriptionTBox.Text = _SelectSimulations.Description;
      ResultTBox.Text = String.Empty;
      _EdgeSList = _EdgeSProvider.GetAllEdgeSBySimulationsId(_selectedRowIndex);
      LoadDataInEdgeSGridView(_EdgeSList);
      _VertexsList = _VertexsProvider.GetAllVertexsBySimulationsId(_selectedRowIndex);
      for (int i = 0; i < _VertexsList.Count; i++) {
        _Graph.AddVertex(_VertexsList[i].VertexsName);
        VertexCBox.Items.Add(_VertexsList[i].VertexsName);
      }
      VertexCBox.SelectedIndex = 0;
      for (int i = 0; i < _EdgeSList.Count; i++) {
        _Graph.AddEdge(_EdgeSList[i].RoutStart, _EdgeSList[i].RoutEnd,
_EdgeSList[i].Distance);
      }
    }
  }

  private void LoadDataInEdgeSGridView(List<EdgeS> EdgeSList) {
    EdgeSGridView.DataSource = null;
    EdgeSGridView.Columns.Clear();
    EdgeSGridView.AutoGenerateColumns = false;
    EdgeSGridView.RowHeadersVisible = false;

    EdgeSGridView.DataSource = EdgeSList;

    if (EdgeSList.Count > 0) {
      if (EdgeSList[0].Message == NamesMy.NoDataNames.NoDataInEdgeS) {
        DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
        messageColumn.DataPropertyName = "Message";
        messageColumn.Width = EdgeSGridView.Width - NamesMy.SizeOptins.MinusSizePanel;
        EdgeSGridView.Columns.Add(messageColumn);
      } else {
        DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
        DetailIdColumn.DataPropertyName = "ClientId";
```

```
        EdgeSGridView.Columns.Add(DetailIdColumn);
        EdgeSGridView.Columns[0].Visible = false;

        DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
        numberColumn.HeaderText = "№ ";
        numberColumn.DataPropertyName = "Number";
        numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        numberColumn.Width = NamesMy.SizeOptins.NumberSize;
        numberColumn.ReadOnly = true;
        EdgeSGridView.Columns.Add(numberColumn);

        DataGridViewColumn RoutStartColumn = new DataGridViewTextBoxColumn();
        RoutStartColumn.HeaderText = "Початок";
        RoutStartColumn.DataPropertyName = "RoutStart";
        RoutStartColumn.Width = 100;
        RoutStartColumn.ReadOnly = true;
        EdgeSGridView.Columns.Add(RoutStartColumn);

        DataGridViewColumn RoutEndColumn = new DataGridViewTextBoxColumn();
        RoutEndColumn.HeaderText = "Кінець";
        RoutEndColumn.DataPropertyName = "RoutEnd";
        RoutEndColumn.Width = 100;
        RoutEndColumn.ReadOnly = true;
        EdgeSGridView.Columns.Add(RoutEndColumn);

        DataGridViewColumn DistanceColumn = new DataGridViewTextBoxColumn();
        DistanceColumn.HeaderText = "Відстань";
        DistanceColumn.DataPropertyName = "Distance";
        DistanceColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        DistanceColumn.Width = 70;
        DistanceColumn.ReadOnly = false;
        EdgeSGridView.Columns.Add(DistanceColumn);

      }
      for (int i = 0; i < EdgeSGridView.Columns.Count; i++) {
        EdgeSGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
    }
  }

  private void SearchBtn_Click(object sender, EventArgs e) {
    _Dijkstra = new Dijkstra(_Graph);
    ResultTBox.Text = "Найкоротші маршрути від заданого місця, до всіх інших місць:
\r\n";
    for (int i = 0; i < Convert.ToInt32(_SelectSimulations.CountPeaks); i++) {
      if (VertexCBox.Text != i.ToString()) {
        var path = _Dijkstra.FindShortestPath(VertexCBox.Text, _VertexsList[i].VertexsName);
        if (path != VertexCBox.Text) {
          ResultTBox.Text += path + "\r\n";
          ResultTBox.Text += "------------------------------\r\n";
```

```
        }
       }
      }
     }

  }
}
```

Лістинг 3. Код класу «CitiesForm»

```
using RouteOptimizerApp.AppCode;
using RouteOptimizerApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace RouteOptimizerApp.Forms.Dictionary {
  public partial class CitiesForm : Form {
    private int _selectedRowIndex = 0;
    private ValidationMy _validation = new ValidationMy();
    private CitiesProvider _CitiesProvider = new CitiesProvider();
    private List<Cities> _CitiesList = new List<Cities>();

    public CitiesForm() {
      InitializeComponent();
      DataLoad();
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _CitiesProvider.InsertCities(CitiesNameTBox.Text, DescriptionTBox.Text);
        DataLoad();
        ClearAllControls();
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllControls();
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void DataLoad() {
      int firstRowIndex = 0;
```

```
      if (CitiesGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = CitiesGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _CitiesList = _CitiesProvider.GetAllCities();
        LoadDataInCitiesGridView(_CitiesList);
        if (_selectedRowIndex == CitiesGridView.Rows.Count) {
          _selectedRowIndex = CitiesGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          CitiesGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          CitiesGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch { }
    }

    private void LoadDataInCitiesGridView(List<Cities> CitiesList) {
      CitiesGridView.DataSource = null;
      CitiesGridView.Columns.Clear();
      CitiesGridView.AutoGenerateColumns = false;
      CitiesGridView.RowHeadersVisible = false;

      CitiesGridView.DataSource = CitiesList;

      if (CitiesList.Count > 0) {
        if (CitiesList[0].Message == NamesMy.NoDataNames.NoDataInCities) {
          DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
          messageColumn.DataPropertyName = "Message";
          messageColumn.Width = CitiesGridView.Width - NamesMy.SizeOptins.MinusSizePanel;
          CitiesGridView.Columns.Add(messageColumn);
        } else {
          DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
          DetailIdColumn.DataPropertyName = "CitiesId";
          CitiesGridView.Columns.Add(DetailIdColumn);
          CitiesGridView.Columns[0].Visible = false;

          DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
          numberColumn.HeaderText = "№ ";
          numberColumn.DataPropertyName = "Number";
          numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
          numberColumn.Width = NamesMy.SizeOptins.NumberSize;
          CitiesGridView.Columns.Add(numberColumn);

          DataGridViewColumn CitiesNameColumn = new DataGridViewTextBoxColumn();
          CitiesNameColumn.HeaderText = "Міста";
          CitiesNameColumn.DataPropertyName = "CitiesName";
          CitiesNameColumn.Width = NamesMy.SizeOptins.NameSize;
          CitiesGridView.Columns.Add(CitiesNameColumn);


        }
```

```
      for (int i = 0; i < CitiesGridView.Columns.Count; i++) {
        CitiesGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
    }
  }

  private void ClearAllControls() {
    CitiesNameTBox.Text = String.Empty;
    DescriptionTBox.Text = String.Empty;
  }

  private bool IsDataEnteringCorrect() {
    bool isCorrect = true;
    if (_validation.IsDataEntering(CitiesNameTBox.Text)) {
      CitiesNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      CitiesNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    return isCorrect;
  }

  private void CitiesGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
    if (e.RowIndex >= 0 && CitiesGridView[0, e.RowIndex].Value.ToString() !=
_CitiesList[0].Message) {
      _selectedRowIndex = e.RowIndex;
      UpdateCitiesForm updateCitiesForm = new
UpdateCitiesForm(Convert.ToInt32(CitiesGridView[0, e.RowIndex].Value.ToString()));
      updateCitiesForm.ShowDialog();
      DataLoad();
    }
  }
 }
}
```

Лістинг 4. Код класу «UpdateCitiesForm»

```
using RouteOptimizerApp.AppCode;
using RouteOptimizerApp.Providers;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace RouteOptimizerApp.Forms.Dictionary {
 public partial class UpdateCitiesForm : Form {
  private int _CitiesId = 0;
```

```
    private Cities _selectedCities = new Cities();
    private CitiesProvider _CitiesProvider = new CitiesProvider();
    private ValidationMy _validation = new ValidationMy();

    public UpdateCitiesForm(int CitiesId) {
      InitializeComponent();
      _CitiesId = CitiesId;
      LoadAllDate();
    }

    private void SaveBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _CitiesProvider.UpdateCities(CitiesNameTBox.Text, DescriptionTBox.Text, _CitiesId);
        this.Close();
      }
    }

    private void DeleteBtn_Click(object sender, EventArgs e) {
      if (MessageBox.Show("Ви дійсно хочете видалити цей елемент?", "Видалити",
MessageBoxButtons.YesNo) == DialogResult.Yes) {
        _CitiesProvider.DeleteCitiesByCitiesId(_CitiesId);
        this.Close();
      }
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void LoadAllDate() {
      _selectedCities = _CitiesProvider.SelectedCitiesByCitiesId(_CitiesId);
      CitiesNameTBox.Text = _selectedCities.CitiesName;
      DescriptionTBox.Text = _selectedCities.Description;
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(CitiesNameTBox.Text)) {
        CitiesNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        CitiesNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      return isCorrect;
    }

  }
}
```

Лістинг 5. Код класу «LoginForm»

```
using RouteOptimizerApp.AppCode;
```

```
using RouteOptimizerApp.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace RouteOptimizerApp.Forms.Systems {
  public partial class LoginForm : Form {
    public static Users CurrentUser = new Users();

    private UsersProvider _UserProvider = new UsersProvider();
    private ValidationMy _validation = new ValidationMy();
    private LogsProvider _LogsProvider = new LogsProvider();
    private List<Users> _UserList = new List<Users>();
    public LoginForm() {
      InitializeComponent();
      LoadAllDate();
    }

    private void SubmitBtn_Click(object sender, EventArgs e) {
      GetSubmitData();
    }

    private void DataLoad() {
      _LogsProvider.InsertLogs(CurrentUser.UsersId, "Користувач ввійшов в систему",
DateTime.Now);
      this.Visible = false;
      (new RouteOptimizerMDI()).ShowDialog();
      _LogsProvider.InsertLogs(CurrentUser.UsersId, "Користувач вийшов із системи",
DateTime.Now);
      this.Close();
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(UserNameCBox.Text)) {
        UserNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        UserNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(UserPasswordTbx.Text)) {
        UserPasswordValidation.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        UserPasswordValidation.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
```

```
      return isCorrect;
    }

    private void LoadAllDate() {
      _UserList = _UserProvider.GetAllUsersListForCBox();
      UserNameCBox.DataSource = _UserList;
      UserNameCBox.AutoCompleteMode = AutoCompleteMode.SuggestAppend;
      UserNameCBox.AutoCompleteSource = AutoCompleteSource.ListItems;
      UserNameCBox.ValueMember = "UsersId";
      UserNameCBox.DisplayMember = "UsersName";
    }

    private void GetSubmitData() {
      try {
        if (IsDataEnteringCorrect()) {
          List<Users> listUsers = new List<Users>();
          listUsers =
_UserProvider.SelectedUsersByUsersNameAndUsersPassword(UserNameCBox.Text,
UserPasswordTbx.Text);
          if (listUsers.Count > 0) {
            CurrentUser = listUsers[0];
            DataLoad();
          } else {

MessageBox.Show(NamesMy.MessageBoxExaption.ThisUserLoginAndUserPasswordNotExistI
nSystem, NamesMy.MessageBoxExaption.CaptionMessage);
          }
        }
      } catch {
        MessageBox.Show("Немає з'єднання!");
      }
    }
  }
}
```

Лістинг 6. Код класу «SystemLogForm»

```
using RouteOptimizerApp.AppCode;
using RouteOptimizerApp.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace RouteOptimizerApp.Forms.Systems {
  public partial class SystemLogForm : Form {
    private int _selectedRowIndex = 0;
```

```
  private LogsProvider _LogsProvider = new LogsProvider();
  private List<Logs> _LogsList = new List<Logs>();
  public SystemLogForm() {
    InitializeComponent();
    DataLoad();
  }

  private void DataLoad() {
    int firstRowIndex = 0;
    if (LogsGridView.FirstDisplayedScrollingRowIndex > 0) {
      firstRowIndex = LogsGridView.FirstDisplayedScrollingRowIndex;
    }
    try {
      _LogsList = _LogsProvider.GetAllLogs();
      LoadDataInLogsGridView(_LogsList);
      if (_selectedRowIndex == LogsGridView.Rows.Count) {
        _selectedRowIndex = LogsGridView.Rows.Count - 1;
      }
      if (_selectedRowIndex >= 0) {
        LogsGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
        LogsGridView.Rows[_selectedRowIndex].Selected = true;
      }
    } catch { }
  }

  private void LoadDataInLogsGridView(List<Logs> LogsList) {
    LogsGridView.DataSource = null;
    LogsGridView.Columns.Clear();
    LogsGridView.AutoGenerateColumns = false;
    LogsGridView.RowHeadersVisible = false;

    LogsGridView.DataSource = LogsList;

    if (LogsList.Count > 0) {
      if (LogsList[0].Message == NamesMy.NoDataNames.NoDataInLogs) {
        DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
        messageColumn.DataPropertyName = "Message";
        messageColumn.Width = LogsGridView.Width - NamesMy.SizeOptins.MinusSizePanel;
        LogsGridView.Columns.Add(messageColumn);
      } else {
        DataGridViewColumn DetailIdColumn = new DataGridViewTextBoxColumn();
        DetailIdColumn.DataPropertyName = "LogsId";
        LogsGridView.Columns.Add(DetailIdColumn);
        LogsGridView.Columns[0].Visible = false;

        DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
        numberColumn.HeaderText = "№ ";
        numberColumn.DataPropertyName = "Number";
        numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
        numberColumn.Width = NamesMy.SizeOptins.NumberSize;
        LogsGridView.Columns.Add(numberColumn);
```

```
        DataGridViewColumn UsersNameColumn = new DataGridViewTextBoxColumn();
        UsersNameColumn.HeaderText = "Користувач";
        UsersNameColumn.DataPropertyName = "UsersName";
        UsersNameColumn.Width = 150;
        LogsGridView.Columns.Add(UsersNameColumn);

        DataGridViewColumn EventNameShowColumn = new DataGridViewTextBoxColumn();
        EventNameShowColumn.HeaderText = "Подія";
        EventNameShowColumn.DataPropertyName = "EventNameShow";
        EventNameShowColumn.Width = 500;
        LogsGridView.Columns.Add(EventNameShowColumn);

        DataGridViewColumn EvendDateColumn = new DataGridViewTextBoxColumn();
        EvendDateColumn.HeaderText = "Дата";
        EvendDateColumn.DataPropertyName = "EvendDate";
        EvendDateColumn.Width = NamesMy.SizeOptins.Date;
        LogsGridView.Columns.Add(EvendDateColumn);

      }
      for (int i = 0; i < LogsGridView.Columns.Count; i++) {
        LogsGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
      }
    }
   }

  }
}
```

Лістинг 7. Код класу «UpdateUsersForm»

```
using RouteOptimizerApp.AppCode;
using RouteOptimizerApp.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;


namespace RouteOptimizerApp.Forms.Systems {
  public partial class UpdateUsersForm : Form {
    private int _UserId = 0;
    private Users _selectedUser = new Users();
    private UsersProvider _UserProvider = new UsersProvider();
    private ValidationMy _validation = new ValidationMy();
    private RoleApp _RoleApp = new RoleApp();
    private List<Role> _RoleList = new List<Role>();
```

```
    public UpdateUsersForm(int UserId) {
      InitializeComponent();
      _UserId = UserId;
      LoadAllDate();
    }

    private void SaveBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _UserProvider.UpdateUsers(FirstNameTBox.Text, LastNameTBox.Text,
UserLoginTbx.Text, PasswordTbx.Text,
          Convert.ToInt32(RolesCBox.SelectedValue), DescriptionTbx.Text, _UserId);
        this.Close();
      }
    }

    private void DeleteBtn_Click(object sender, EventArgs e) {
      if (MessageBox.Show("Ви дійсно хочете видалити цей елемент?", "Видалити",
MessageBoxButtons.YesNo) == DialogResult.Yes) {
        _UserProvider.DeleteUsersByUsersId(_UserId);
        this.Close();
      }
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void LoadAllDate() {
      _RoleList = _RoleApp.GetRoleList();
      RolesCBox.DataSource = _RoleList;
      RolesCBox.ValueMember = "RoleId";
      RolesCBox.DisplayMember = "RoleName";

      _selectedUser = _UserProvider.SelectedUsersByUsersId(_UserId);
      FirstNameTBox.Text = _selectedUser.FirstName;
      LastNameTBox.Text = _selectedUser.LastName;
      UserLoginTbx.Text = _selectedUser.UsersName;
      RolesCBox.SelectedValue = _selectedUser.RoleId;
    }

    private bool IsDataEnteringCorrect() {
      bool isCorrect = true;
      if (_validation.IsDataEntering(FirstNameTBox.Text)) {
        FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
        FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
        isCorrect = false;
      }
      if (_validation.IsDataEntering(LastNameTBox.Text)) {
        LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
      } else {
```

```
      LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     if (_validation.IsPasswordMatch(PasswordTbx.Text, RePasswordTbx.Text)) {
      PasswordAndRePasswordDontMatchLbl.Visible = false;
     } else {
      PasswordAndRePasswordDontMatchLbl.Visible = true;
      isCorrect = false;
     }
     if (_validation.IsDataEntering(UserLoginTbx.Text)) {
      UserLoginValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
     } else {
      UserLoginValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     if (_validation.IsDataEntering(PasswordTbx.Text)) {
      PasswordValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
     } else {
      PasswordValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     if (_validation.IsDataEntering(RePasswordTbx.Text)) {
      RePasswordValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
     } else {
      RePasswordValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
     }
     return isCorrect;
    }
   }
}
```

Лістинг 8. Код класу «UsersForm»

```
using RouteOptimizerApp.AppCode;
using RouteOptimizerApp.Forms.Systems;
using RouteOptimizerApp.Provider;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace RouteOptimizerApp.Forms.Systems {
  public partial class UsersForm : Form {
    private int _selectedRowIndex = 0;
    ValidationMy _validation = new ValidationMy();
    private UsersProvider _UserProvider = new UsersProvider();
```

```
    private List<Users> _UserList = new List<Users>();
    private RoleApp _RoleApp = new RoleApp();
    private List<Role> _RoleList = new List<Role>();
    public UsersForm() {
      InitializeComponent();
      LoadAllDate();
      DataLoad();
    }

    private void AddBtn_Click(object sender, EventArgs e) {
      if (IsDataEnteringCorrect()) {
        _UserProvider.InsertUsers(FirstNameTBox.Text, LastNameTBox.Text,
UserLoginTbx.Text, PasswordTbx.Text,
          Convert.ToInt32(RolesCBox.SelectedValue), DescriptionTbx.Text);
        DataLoad();
        ClearAllControls();
      }
    }

    private void ClearBtn_Click(object sender, EventArgs e) {
      ClearAllControls();
    }

    private void ExitBtn_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void DataLoad() {
      int firstRowIndex = 0;
      if (UsersGridView.FirstDisplayedScrollingRowIndex > 0) {
        firstRowIndex = UsersGridView.FirstDisplayedScrollingRowIndex;
      }
      try {
        _UserList = _UserProvider.GetAllUsers();
        LoadDataInKlientGridView(_UserList);
        if (_selectedRowIndex == UsersGridView.Rows.Count) {
          _selectedRowIndex = UsersGridView.Rows.Count - 1;
        }
        if (_selectedRowIndex >= 0) {
          UsersGridView.FirstDisplayedScrollingRowIndex = firstRowIndex;
          UsersGridView.Rows[_selectedRowIndex].Selected = true;
        }
      } catch { }
    }

    private void LoadDataInKlientGridView(List<Users> UserList) {
      UsersGridView.DataSource = null;
      UsersGridView.Columns.Clear();
      UsersGridView.AutoGenerateColumns = false;
      UsersGridView.RowHeadersVisible = false;

      UsersGridView.DataSource = UserList;
```

```
      if (UserList.Count > 0) {
        if (UserList[0].Message == NamesMy.NoDataNames.NoDataInUsers) {
          DataGridViewColumn messageColumn = new DataGridViewTextBoxColumn();
          messageColumn.DataPropertyName = "Message";
          messageColumn.Width = UsersGridView.Width - NamesMy.SizeOptins.MinusSizePanel;
          UsersGridView.Columns.Add(messageColumn);
        } else {
          DataGridViewColumn deviseIdColumn = new DataGridViewTextBoxColumn();
          deviseIdColumn.DataPropertyName = "UsersId";
          UsersGridView.Columns.Add(deviseIdColumn);
          UsersGridView.Columns[0].Visible = false;

          DataGridViewColumn numberColumn = new DataGridViewTextBoxColumn();
          numberColumn.HeaderText = "№ п/п";
          numberColumn.DataPropertyName = "Number";
          numberColumn.DefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleRight;
          numberColumn.Width = NamesMy.SizeOptins.NumberSize;
          UsersGridView.Columns.Add(numberColumn);

          DataGridViewColumn lastNameColumn = new DataGridViewTextBoxColumn();
          lastNameColumn.HeaderText = "Фимилия";
          lastNameColumn.DataPropertyName = "LastName";
          lastNameColumn.Width = NamesMy.SizeOptins.Name;
          UsersGridView.Columns.Add(lastNameColumn);

          DataGridViewColumn firstNameColumn = new DataGridViewTextBoxColumn();
          firstNameColumn.HeaderText = "Имя";
          firstNameColumn.DataPropertyName = "FirstName";
          firstNameColumn.Width = NamesMy.SizeOptins.Name;
          UsersGridView.Columns.Add(firstNameColumn);

          DataGridViewColumn UserNameColumn = new DataGridViewTextBoxColumn();
          UserNameColumn.HeaderText = "Логин";
          UserNameColumn.DataPropertyName = "UsersName";
          UserNameColumn.Width = NamesMy.SizeOptins.Name;
          UsersGridView.Columns.Add(UserNameColumn);

          DataGridViewColumn roleNameColumn = new DataGridViewTextBoxColumn();
          roleNameColumn.HeaderText = "Роль";
          roleNameColumn.DataPropertyName = "RoleName";
          roleNameColumn.Width = NamesMy.SizeOptins.Name;
          UsersGridView.Columns.Add(roleNameColumn);
        }
        for (int i = 0; i < UsersGridView.Columns.Count; i++) {
          UsersGridView.Columns[i].HeaderCell.Style.BackColor = Color.LightGray;
        }
      }
    }

    private void ClearAllControls() {
```

```
    FirstNameTBox.Text = String.Empty;
    LastNameTBox.Text = String.Empty;
    DescriptionTbx.Text = String.Empty;
    UserLoginTbx.Text = String.Empty;
    PasswordTbx.Text = String.Empty;
    RePasswordTbx.Text = String.Empty;
  }

  private bool IsDataEnteringCorrect() {
    bool isCorrect = true;
    if (_validation.IsDataEntering(FirstNameTBox.Text)) {
      FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      FirstNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_validation.IsDataEntering(LastNameTBox.Text)) {
      LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      LastNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_validation.IsPasswordMatch(PasswordTbx.Text, RePasswordTbx.Text)) {
      PasswordAndRePasswordDontMatchLbl.Visible = false;
      PasswordAndRePasswordDontMatchLbl.Text =
NamesMy.ProgramButtons.RequiredValidation;
    } else {
      PasswordAndRePasswordDontMatchLbl.Visible = true;
      PasswordAndRePasswordDontMatchLbl.Text =
NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_validation.IsDataEntering(UserLoginTbx.Text)) {
      UserLoginValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      UserLoginValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_validation.IsDataEntering(PasswordTbx.Text)) {
      PasswordValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      PasswordValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    if (_validation.IsDataEntering(RePasswordTbx.Text)) {
      RePasswordValidationLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
    } else {
      RePasswordValidationLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
      isCorrect = false;
    }
    return isCorrect;
  }
```

```
    private void LoadAllDate() {
      _RoleList = _RoleApp.GetRoleList();
      RolesCBox.DataSource = _RoleList;
      RolesCBox.ValueMember = "RoleId";
      RolesCBox.DisplayMember = "RoleName";
    }

    private void UsersGridView_CellClick(object sender, DataGridViewCellEventArgs e) {
      if (e.RowIndex >= 0 && UsersGridView[0, e.RowIndex].Value.ToString() !=
_UserList[0].Message) {
        _selectedRowIndex = e.RowIndex;
        UpdateUsersForm updateUsersForm = new
UpdateUsersForm(Convert.ToInt32(UsersGridView[0, e.RowIndex].Value.ToString()));
        updateUsersForm.ShowDialog();
        DataLoad();
      }
    }
  }
}
```

Лістинг 9. Код класу «CitiesProvider»

```
using RouteOptimizerApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RouteOptimizerApp.Providers {
  class CitiesProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertCities(string CitiesName, string Description) {
      string SqlString = "INSERT INTO Cities (CitiesName, Description" +
        ") Values(?, ?)";

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("CitiesName", CitiesName);
          cmd.Parameters.AddWithValue("Description", Description);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }
```

```
public List<Cities> GetAllCities() {
  int i = 0;
  string SqlString = "SELECT * FROM Cities";

  List<Cities> listCities = new List<Cities>();
  using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
    using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      conn.Open();
      using (OleDbDataReader reader = cmd.ExecuteReader()) {
        while (reader.Read()) {
          Cities oneCities = new Cities();
          oneCities.Number = ++i;
          oneCities.CitiesId = Convert.ToInt32(reader["CitiesId"]);
          oneCities.CitiesName = reader["CitiesName"].ToString();
          oneCities.Description = reader["Description"].ToString();
          listCities.Add(oneCities);
        }
      }
      conn.Close();
    }
  }

  if (listCities.Count == 0) {
    Cities noCities = new Cities();
    noCities.CitiesId = 0;
    noCities.Message = NamesMy.NoDataNames.NoDataInCities;
    listCities.Add(noCities);
  }
  return listCities;
}

public Cities SelectedCitiesByCitiesId(int CitiesId) {
  string SqlString = "SELECT * FROM Cities Where CitiesId=" + CitiesId.ToString();

  Cities oneCities = new Cities();
  using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
    using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
      conn.Open();
      using (OleDbDataReader reader = cmd.ExecuteReader()) {
        while (reader.Read()) {
          oneCities.CitiesId = Convert.ToInt32(reader["CitiesId"]);
          oneCities.CitiesName = reader["CitiesName"].ToString();
          oneCities.Description = reader["Description"].ToString();
        }
      }
    }
    conn.Close();
  }
  return oneCities;
}
```

```
    public void UpdateCities(string CitiesName, string Description, int CitiesId) {
      string SqlString = "UPDATE Cities SET CitiesName=?, Description=?  " +
  "WHERE CitiesId=?";

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("CitiesName", CitiesName);
          cmd.Parameters.AddWithValue("Description", Description);
          cmd.Parameters.AddWithValue("CitiesId", CitiesId);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public void DeleteCitiesByCitiesId(int CitiesId) {
      string SqlString = "DELETE FROM Cities WHERE CitiesId=" + CitiesId.ToString();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

  }
}


public class Cities {
  private int _Number;
  private int _CitiesId;
  private string _CitiesName;
  private string _Description;
  private string _Message;

  public Cities() {
    _Number = 0;
    _CitiesId = 0;
    _CitiesName = String.Empty;
    _Description = String.Empty;
    _Message = String.Empty;
  }

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int CitiesId {
```

```
    set { _CitiesId = value; }
    get { return _CitiesId; }
  }
  public string CitiesName {
    set { _CitiesName = value; }
    get { return _CitiesName; }
  }
  public string Description {
    set { _Description = value; }
    get { return _Description; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 10. Код класу «DijkstraS»

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RouteOptimizerApp.Providers {
  class DijkstraS {
  }
}

/// <summary>
/// ІЬІнформація про вершину
/// </summary>
public class GraphVertexInfo {
  /// <summary>
  /// Вершина
  /// </summary>
  public GraphVertex Vertex { get; set; }

  /// <summary>
  /// Не відвідана вершина
  /// </summary>
  public bool IsUnvisited { get; set; }

  /// <summary>
  /// Сума ваг ребер
  /// </summary>
  public int EdgesWeightSum { get; set; }

  /// <summary>
```

```
  /// Попередня вершина
  /// </summary>
  public GraphVertex PreviousVertex { get; set; }

  /// <summary>
  /// Конструктор
  /// </summary>
  /// <param name="vertex">Вершина</param>
  public GraphVertexInfo(GraphVertex vertex) {
    Vertex = vertex;
    IsUnvisited = true;
    EdgesWeightSum = int.MaxValue;
    PreviousVertex = null;
  }
}

/// <summary>
/// Граф
/// </summary>
public class Graph {
  /// <summary>
  /// Список вершин графа
  /// </summary>
  public List<GraphVertex> Vertices { get; set; }

  /// <summary>
  /// Конструктор
  /// </summary>
  public Graph() {
    Vertices = new List<GraphVertex>();
  }

  /// <summary>
  /// Додавання вершини
  /// </summary>
  /// <param name="vertexName">Імя вершини</param>
  public void AddVertex(string vertexName) {
    Vertices.Add(new GraphVertex(vertexName));
  }

  /// <summary>
  /// Поиск вершины
  /// </summary>
  /// <param name="vertexName">Назва вершини</param>
  /// <returns>Знайдена вершина</returns>
  public GraphVertex FindVertex(string vertexName) {
    foreach (var v in Vertices) {
      if (v.Name.Equals(vertexName)) {
        return v;
      }
    }
```

```
    return null;
  }

  /// <summary>
  /// Додавання ребра
  /// </summary>
  /// <param name="firstName">Імя першої вершини</param>
  /// <param name="secondName">Імя другої вершини</param>
  /// <param name="weight">Вага ребра зєдуючого вершини</param>
  public void AddEdge(string firstName, string secondName, int weight) {
    var v1 = FindVertex(firstName);
    var v2 = FindVertex(secondName);
    if (v2 != null && v1 != null) {
      v1.AddEdge(v2, weight);
      v2.AddEdge(v1, weight);
    }
  }
}

/// <summary>
/// Вершина графа
/// </summary>
public class GraphVertex {
  /// <summary>
  /// Назва вершини
  /// </summary>
  public string Name { get; set; }

  /// <summary>
  /// Список ребер
  /// </summary>
  public List<GraphEdge> Edges { get; set; }

  /// <summary>
  /// Конструктор
  /// </summary>
  /// <param name="vertexName">Назва вершини</param>
  public GraphVertex(string vertexName) {
    Name = vertexName;
    Edges = new List<GraphEdge>();
  }

  /// <summary>
  /// Додати ребро
  /// </summary>
  /// <param name="newEdge">Ребро</param>
  public void AddEdge(GraphEdge newEdge) {
    Edges.Add(newEdge);
  }

  /// <summary>
  /// Додати ребро
```

```
  /// </summary>
  /// <param name="vertex">Вершина</param>
  /// <param name="edgeWeight">Вага</param>
  public void AddEdge(GraphVertex vertex, int edgeWeight) {
    AddEdge(new GraphEdge(vertex, edgeWeight));
  }

  /// <summary>
  /// Преобразование в строку
  /// </summary>
  /// <returns>Имя вершины</returns>
  //public override string ToString() => Name;
}

/// <summary>
/// Ребро графа
/// </summary>
public class GraphEdge {
  /// <summary>
  /// Зв'язана вершина
  /// </summary>
  public GraphVertex ConnectedVertex { get; set; }

  /// <summary>
  /// Вага ребра
  /// </summary>
  public int EdgeWeight { get; set; }

  /// <summary>
  /// Конструктор
  /// </summary>
  /// <param name="connectedVertex">Зв'язана  вершина</param>
  /// <param name="weight">Вага ребра</param>
  public GraphEdge(GraphVertex connectedVertex, int weight) {
    ConnectedVertex = connectedVertex;
    EdgeWeight = weight;
  }
}

public class Dijkstra {
  Graph graph;

  List<GraphVertexInfo> infos;

  /// <summary>
  /// Конструктор
  /// </summary>
  /// <param name="graph">Граф</param>
  public Dijkstra(Graph graph) {
    this.graph = graph;
  }
```

```
/// <summary>
/// Ініціалізація інформації
/// </summary>
void InitInfo() {
  infos = new List<GraphVertexInfo>();
  foreach (var v in graph.Vertices) {
    infos.Add(new GraphVertexInfo(v));
  }
}

/// <summary>
/// Дістати інформацію про вершину графа графа
/// </summary>
/// <param name="v">Вершина</param>
/// <returns>Інформація про вершину</returns>
GraphVertexInfo GetVertexInfo(GraphVertex v) {
  foreach (var i in infos) {
    if (i.Vertex.Equals(v)) {
      return i;
    }
  }

  return null;
}

/// <summary>
/// Пошук невідваної вершини з мінімальним значенням суми
/// </summary>
/// <returns>Інформація про вершину</returns>
public GraphVertexInfo FindUnvisitedVertexWithMinSum() {
  var minValue = int.MaxValue;
  GraphVertexInfo minVertexInfo = null;
  foreach (var i in infos) {
    if (i.IsUnvisited && i.EdgesWeightSum < minValue) {
      minVertexInfo = i;
      minValue = i.EdgesWeightSum;
    }
  }

  return minVertexInfo;
}

/// <summary>
/// Пошук найкоротшого шляху за назвами вершин
/// </summary>
/// <param name="startName">Назва стартової вершини/param>
/// <param name="finishName">Назва фінішної вершини</param>
/// <returns>Кратчайший путь</returns>
public string FindShortestPath(string startName, string finishName) {
  return FindShortestPath(graph.FindVertex(startName), graph.FindVertex(finishName));
}
```

```
/// <summary>
/// Пошук найкоротшого шляху по вершинах
/// </summary>
/// <param name="startVertex">Стартова вершина</param>
/// <param name="finishVertex">Фінішна вершина</param>
/// <returns>Найкоротний шлях</returns>
public string FindShortestPath(GraphVertex startVertex, GraphVertex finishVertex) {
  InitInfo();
  var first = GetVertexInfo(startVertex);
  first.EdgesWeightSum = 0;
  while (true) {
    var current = FindUnvisitedVertexWithMinSum();
    if (current == null) {
      break;
    }

    SetSumToNextVertex(current);
  }

  return GetPath(startVertex, finishVertex);
}

/// <summary>
/// Обчислення суми ваг ребер для наступної вершини
/// </summary>
/// <param name="info">Інформація про поточну вершину</param>
void SetSumToNextVertex(GraphVertexInfo info) {
  info.IsUnvisited = false;
  foreach (var e in info.Vertex.Edges) {
    var nextInfo = GetVertexInfo(e.ConnectedVertex);
    var sum = info.EdgesWeightSum + e.EdgeWeight;
    if (sum < nextInfo.EdgesWeightSum) {
      nextInfo.EdgesWeightSum = sum;
      nextInfo.PreviousVertex = info.Vertex;
    }
  }
}

/// <summary>
/// Формирование пути
/// </summary>
/// <param name="startVertex">Початкова вершина</param>
/// <param name="endVertex">Кінцева вершина</param>
/// <returns>Шлях</returns>
string GetPath(GraphVertex startVertex, GraphVertex endVertex) {
  string path = "";
  List<string> vertex = new List<string>();

  vertex.Add(endVertex.Name);
  while (startVertex != endVertex) {
    endVertex = GetVertexInfo(endVertex).PreviousVertex;
    vertex.Add(endVertex.Name);
```

```
    }

    for (int i = vertex.Count() - 1; i >= 0; i--) {
      path += vertex[i];
      if (i > 0) {
        path += " -> ";
      }
    }
    return path;
  }
          }
```

Лістинг 10. Код класу «EdgeSProvider»

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RouteOptimizerApp.Providers {
  class EdgeSProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];
    public void InsertBatchEdgeS(List<EdgeS> EdgeS) {
      string SqlString = "INSERT INTO EdgeS (RoutStart, RoutEnd, Distance, SimulationsId)
Values(?, ?, ?, ?)";
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          conn.Open();
          for (int i = 0; i < EdgeS.Count; i++) {
            cmd.Parameters.AddWithValue("RoutStart", EdgeS[i].RoutStart);
            cmd.Parameters.AddWithValue("RoutEnd", EdgeS[i].RoutEnd);
            cmd.Parameters.AddWithValue("Distance", EdgeS[i].Distance);
            cmd.Parameters.AddWithValue("SimulationsId", EdgeS[i].SimulationsId);
            cmd.ExecuteNonQuery();
            while (cmd.Parameters.Count > 0) {
              cmd.Parameters.RemoveAt(0);
            }
          }
          conn.Close();
        }
      }
    }

    public List<EdgeS> GetAllEdgeSBySimulationsId(int SimulationsId) {
      int i = 0;
      string SqlString = "SELECT * FROM EdgeS WHERE SimulationsId=" + SimulationsId;

      List<EdgeS> EdgeS = new List<EdgeS>();
```

```
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          using (OleDbDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              EdgeS oneEdgeS = new EdgeS();
              oneEdgeS.Number = ++i;
              oneEdgeS.EdgeSId = Convert.ToInt32(reader["EdgeSId"]);
              oneEdgeS.RoutStart = reader["RoutStart"].ToString();
              oneEdgeS.RoutEnd = reader["RoutEnd"].ToString();
              oneEdgeS.Distance = Convert.ToInt32(reader["Distance"]);
              oneEdgeS.SimulationsId = Convert.ToInt32(reader["SimulationsId"]);
              EdgeS.Add(oneEdgeS);
            }
          }
          conn.Close();
        }
      }
      return EdgeS;
    }


    public void DeleteEdgeSByEdgeSId(int SimulationsId) {
      string SqlString = "DELETE FROM EdgeS WHERE SimulationsId=" +
SimulationsId.ToString();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

  }
}



public class EdgeS {
  private int _Number;
  private int _EdgeSId;
  private string _RoutStart;
  private string _RoutEnd;
  private int _Distance;
  private int _SimulationsId;
  private string _Message;

  public EdgeS() {
    _Number = 0;
    _EdgeSId = 0;
    _RoutStart = String.Empty;
```

```
    _RoutEnd = String.Empty;
    _Distance = 0;
    _SimulationsId = 0;
    _Message = String.Empty;
  }

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int EdgeSId {
    set { _EdgeSId = value; }
    get { return _EdgeSId; }
  }
  public string RoutStart {
    set { _RoutStart = value; }
    get { return _RoutStart; }
  }
  public string RoutEnd {
    set { _RoutEnd = value; }
    get { return _RoutEnd; }
  }
  public int Distance {
    set { _Distance = value; }
    get { return _Distance; }
  }
  public int SimulationsId {
    set { _SimulationsId = value; }
    get { return _SimulationsId; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
        }
```

Лістинг 11. Код класу «LogsProvider»

```
using RouteOptimizerApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RouteOptimizerApp.Provider {
  class LogsProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];
    public void InsertLogs(int UsersId, string EventNameShow, DateTime EvendDate) {
```

```
    string SqlString = "INSERT INTO Logs (UsersId, EventNameShow, EvendDate) Values(?,
?, ?)";
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("UsersId", UsersId);
        cmd.Parameters.AddWithValue("EventNameShow", EventNameShow);
        cmd.Parameters.AddWithValue("EvendDate", EvendDate.ToString());
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }

  public List<Logs> GetAllLogs() {
    int i = 0;
    string SqlString = "SELECT Logs.LogsId, Logs.UsersId, Logs.EventNameShow,
Logs.EvendDate, Users.UsersName " +
    "FROM Logs INNER JOIN Users ON Users.UsersId = Logs.UsersId  ORDER BY
Logs.EvendDate DESC";
    List<Logs> listAllLogs = new List<Logs>();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            Logs oneLogs = new Logs();
            oneLogs.Number = ++i;
            oneLogs.LogsId = Convert.ToInt32(reader["LogsId"]);
            oneLogs.UsersId = Convert.ToInt32(reader["UsersId"]);
            oneLogs.EventNameShow = reader["EventNameShow"].ToString();
            oneLogs.EvendDate = Convert.ToDateTime(reader["EvendDate"]);
            oneLogs.UsersName = reader["UsersName"].ToString();
            listAllLogs.Add(oneLogs);
          }
        }
        conn.Close();
      }
    }

    if (listAllLogs.Count == 0) {
      Logs noLogs = new Logs();
      noLogs.LogsId = 0;
      noLogs.Message = NamesMy.NoDataNames.NoDataInLogs;
      listAllLogs.Add(noLogs);
    }
    return listAllLogs;

  }
 }
}
```

```
public class Logs {
  private int _Number;
  private int _LogsId;
  private int _UsersId;
  private string _UsersName;
  private string _EventNameShow;
  private DateTime _EvendDate;
  private string _Message;

  public Logs() {
    _Number = 0;
    _LogsId = 0;
    _UsersId = 0;
    _UsersName = String.Empty;
    _EventNameShow = String.Empty;
    _EvendDate = new DateTime();
    _Message = String.Empty;
  }

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int LogsId {
    set { _LogsId = value; }
    get { return _LogsId; }
  }
  public int UsersId {
    set { _UsersId = value; }
    get { return _UsersId; }
  }
  public string UsersName {
    set { _UsersName = value; }
    get { return _UsersName; }
  }
  public string EventNameShow {
    set { _EventNameShow = value; }
    get { return _EventNameShow; }
  }
  public DateTime EvendDate {
    set { _EvendDate = value; }
    get { return _EvendDate; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 12. Код класу «SimulationsProvider»

```
using RouteOptimizerApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RouteOptimizerApp.Providers {
  class SimulationsProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertSimulations(string SimulationsName, string Description, int CountPeaks) {
      string SqlString = "INSERT INTO Simulations (SimulationsName, Description,
CountPeaks" +
        ") Values(?, ?, ?)";

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("SimulationsName", SimulationsName);
          cmd.Parameters.AddWithValue("Description", Description);
          cmd.Parameters.AddWithValue("CountPeaks", CountPeaks);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public List<Simulations> GetAllSimulations() {
      int i = 0;
      string SqlString = "SELECT * FROM Simulations";

      List<Simulations> listSimulations = new List<Simulations>();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          using (OleDbDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              Simulations oneSimulations = new Simulations();
              oneSimulations.Number = ++i;
              oneSimulations.SimulationsId = Convert.ToInt32(reader["SimulationsId"]);
              oneSimulations.SimulationsName = reader["SimulationsName"].ToString();
              oneSimulations.Description = reader["Description"].ToString();
              oneSimulations.CountPeaks = Convert.ToInt32(reader["CountPeaks"]);
              listSimulations.Add(oneSimulations);
            }
          }
```

```
        conn.Close();
       }
      }

      if (listSimulations.Count == 0) {
       Simulations noSimulations = new Simulations();
       noSimulations.SimulationsId = 0;
       noSimulations.Message = NamesMy.NoDataNames.NoDataInSimulations;
       listSimulations.Add(noSimulations);
      }
      return listSimulations;
    }

    public Simulations SelectedSimulationsBySimulationsId(int SimulationsId) {
      string SqlString = "SELECT * FROM Simulations Where SimulationsId=" +
SimulationsId.ToString();

      Simulations oneSimulations = new Simulations();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
       using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
         while (reader.Read()) {
          oneSimulations.SimulationsId = Convert.ToInt32(reader["SimulationsId"]);
          oneSimulations.SimulationsName = reader["SimulationsName"].ToString();
          oneSimulations.Description = reader["Description"].ToString();
          oneSimulations.CountPeaks = Convert.ToInt32(reader["CountPeaks"]);
         }
        }
       }
       conn.Close();
      }
      return oneSimulations;
    }

    public void UpdateSimulations(string SimulationsName, string Description, int CountPeaks,
int SimulationsId) {
      string SqlString = "UPDATE Simulations SET SimulationsName=?, Description=?,
CountPeaks=?  " +
  "WHERE SimulationsId=?";

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
       using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        cmd.CommandType = CommandType.Text;
        cmd.Parameters.AddWithValue("SimulationsName", SimulationsName);
        cmd.Parameters.AddWithValue("Description", Description);
        cmd.Parameters.AddWithValue("CountPeaks", CountPeaks);
        cmd.Parameters.AddWithValue("SimulationsId", SimulationsId);
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
       }
```

```
      }
    }

    public void DeleteSimulationsBySimulationsId(int SimulationsId) {
      string SqlString = "DELETE FROM Simulations WHERE SimulationsId=" +
SimulationsId.ToString();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public int GetLastRecords() {
      int lastRecordNumber = 0;
      string SqlString = "Select LAST (SimulationsId) From Simulations ";
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          using (OleDbDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              lastRecordNumber = Convert.ToInt32(reader.GetValue(0));
            }
          }
        }
        conn.Close();
      }
      return lastRecordNumber;
    }

  }
}


public class Simulations {
  private int _Number;
  private int _SimulationsId;
  private string _SimulationsName;
  private string _Description;
  private int _CountPeaks;
  private string _Message;

  public Simulations() {
    _Number = 0;
    _SimulationsId = 0;
    _SimulationsName = String.Empty;
    _Description = String.Empty;
    _CountPeaks = 0;
    _Message = String.Empty;
  }
```

```
  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int SimulationsId {
    set { _SimulationsId = value; }
    get { return _SimulationsId; }
  }
  public string SimulationsName {
    set { _SimulationsName = value; }
    get { return _SimulationsName; }
  }
  public string Description {
    set { _Description = value; }
    get { return _Description; }
  }
  public int CountPeaks {
    set { _CountPeaks = value; }
    get { return _CountPeaks; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 13. Код класу «UsersProvider»

```
using RouteOptimizerApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RouteOptimizerApp.Provider {
  class UsersProvider {
    private EncryptData _encryptData = new EncryptData();
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertUsers(string FirstName, string LastName, string UsersName, string
UsersPassword,
   int RoleId, string Description) {
      string SqlString = "INSERT INTO Users (FirstName, LastName, UsersName,
UsersPassword, " +
       "RoleId, Description) Values(?, ?, ?, ?, ?, ?)";
```

```
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("FirstName", FirstName);
          cmd.Parameters.AddWithValue("LastName", LastName);
          cmd.Parameters.AddWithValue("UsersName", UsersName);
          cmd.Parameters.AddWithValue("UsersPassword",
_encryptData.Encrypt(UsersPassword));
          cmd.Parameters.AddWithValue("RoleId", RoleId);
          cmd.Parameters.AddWithValue("Description", Description);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public List<Users> GetAllUsers() {
      int i = 0;
      string SqlString = "SELECT * FROM Users ORDER BY LastName ASC";
      List<Users> listAllUsers = new List<Users>();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          using (OleDbDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              Users oneUsers = new Users();
              oneUsers.Number = ++i;
              oneUsers.UsersId = Convert.ToInt32(reader["UsersId"]);
              oneUsers.FirstName = reader["FirstName"].ToString();
              oneUsers.LastName = reader["LastName"].ToString();
              oneUsers.FIO = oneUsers.LastName + " " + oneUsers.FirstName;
              oneUsers.UsersName = reader["UsersName"].ToString();
              oneUsers.UsersPassword =
_encryptData.Decrypt(reader["UsersPassword"].ToString());
              oneUsers.RoleId = Convert.ToInt32(reader["RoleId"]);
              oneUsers.RoleName = GetRoleName(oneUsers.RoleId);
              oneUsers.Description = reader["Description"].ToString();
              listAllUsers.Add(oneUsers);
            }
          }
          conn.Close();
        }
      }

      if (listAllUsers.Count == 0) {
        Users noUsers = new Users();
        noUsers.UsersId = 0;
        noUsers.Message = NamesMy.NoDataNames.NoDataInUsers;
        listAllUsers.Add(noUsers);
      }
      return listAllUsers;
```

```
    }

    private string GetRoleName(int RoleId) {
      RoleApp roleApp = new RoleApp();
      for (int i = 0; i < roleApp.GetRoleList().Count(); i++) {
        if (RoleId == roleApp.GetRoleList()[i].RoleId) {
          return roleApp.GetRoleList()[i].RoleName;
        }
      }
      return "";
    }

    public Users SelectedUsersByUsersId(int UsersId) {
      string SqlString = "SELECT * FROM Users WHERE UsersId=" + UsersId.ToString();

      Users oneUsers = new Users();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          using (OleDbDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              oneUsers.UsersId = Convert.ToInt32(reader["UsersId"]);
              oneUsers.FirstName = reader["FirstName"].ToString();
              oneUsers.LastName = reader["LastName"].ToString();
              oneUsers.UsersName = reader["UsersName"].ToString();
              oneUsers.FIO = oneUsers.LastName + " " + oneUsers.FirstName;
              oneUsers.UsersPassword =
_encryptData.Decrypt(reader["UsersPassword"].ToString());
              oneUsers.RoleId = Convert.ToInt32(reader["RoleId"]);
              oneUsers.Description = reader["Description"].ToString();
            }
          }
        }
        conn.Close();
      }
      return oneUsers;
    }

    public List<Users> GetAllUsersListForCBox() {
      string SqlString = "SELECT UsersId, UsersName, UsersPassword FROM Users ORDER BY
UsersName ASC";
      List<Users> listAllUsers = new List<Users>();

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          using (OleDbDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              Users oneUsers = new Users();
              oneUsers.UsersId = Convert.ToInt32(reader["UsersId"].ToString());
              oneUsers.UsersName = reader["UsersName"].ToString();
```

```
            oneUsers.UsersPassword =
_encryptData.Decrypt(reader["UsersPassword"].ToString());
            listAllUsers.Add(oneUsers);
          }
        }
        conn.Close();
      }
    }

    if (listAllUsers.Count == 0) {
      Users noUsers = new Users();
      noUsers.UsersId = 0;
      noUsers.Message = NamesMy.NoDataNames.NoDataInUsers;
      listAllUsers.Add(noUsers);
    }
    return listAllUsers;
  }

  public List<Users> SelectedUsersByUsersNameAndUsersPassword(string UsersName, string
UsersPassword) {
    string SqlString = "SELECT * FROM Users WHERE UsersName='" + UsersName + "' AND
UsersPassword='" + _encryptData.Encrypt(UsersPassword) + "'";
    List<Users> UsersList = new List<Users>();

    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        using (OleDbDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            Users oneUsers = new Users();
            oneUsers.UsersId = Convert.ToInt32(reader["UsersId"]);
            oneUsers.FirstName = reader["FirstName"].ToString();
            oneUsers.LastName = reader["LastName"].ToString();
            oneUsers.UsersName = reader["UsersName"].ToString();
            oneUsers.FIO = oneUsers.LastName + " " + oneUsers.FirstName;
            oneUsers.UsersPassword =
_encryptData.Decrypt(reader["UsersPassword"].ToString());
            oneUsers.RoleId = Convert.ToInt32(reader["RoleId"]);
            oneUsers.Description = reader["Description"].ToString();
            UsersList.Add(oneUsers);
          }
        }
      }
      conn.Close();
    }
    return UsersList;
  }

  public void UpdateUsers(string FirstName, string LastName, string UsersName, string
UsersPassword,
  int RoleId, string Description, int UsersId) {
    string SqlString = "UPDATE Users SET FirstName=?, LastName=?, " +
```

```
   "UsersName=?, UsersPassword=?, RoleId=?, Description=? WHERE UsersId=?";

      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          cmd.Parameters.AddWithValue("FirstName", FirstName);
          cmd.Parameters.AddWithValue("LastName", LastName);
          cmd.Parameters.AddWithValue("UsersName", UsersName);
          cmd.Parameters.AddWithValue("UsersPassword",
_encryptData.Encrypt(UsersPassword));
          cmd.Parameters.AddWithValue("RoleId", RoleId);
          cmd.Parameters.AddWithValue("Description", Description);
          cmd.Parameters.AddWithValue("UsersId", UsersId);
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

    public void DeleteUsersByUsersId(int UsersId) {
      string SqlString = "DELETE FROM Users WHERE UsersId=" + UsersId.ToString();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          cmd.ExecuteNonQuery();
          conn.Close();
        }
      }
    }

  }
}


public class Users {
  private int _Number;
  private int _UsersId;
  private string _FirstName;
  private string _LastName;
  private string _UsersName;
  private string _FIO;
  private string _UsersPassword;
  private int _RoleId;
  private string _RoleName;
  private string _Description;
  private string _Message;

  public Users() {
    _UsersId = 0;
    _FirstName = String.Empty;
    _LastName = String.Empty;
```

```
    _UsersName = String.Empty;
    _FIO = String.Empty;
    _UsersPassword = String.Empty;
    _RoleId = 0;
    _Description = String.Empty;
  }

  public int Number {
    set { _Number = value; }
    get { return _Number; }
  }
  public int UsersId {
    set { _UsersId = value; }
    get { return _UsersId; }
  }
  public string FirstName {
    set { _FirstName = value; }
    get { return _FirstName; }
  }
  public string LastName {
    set { _LastName = value; }
    get { return _LastName; }
  }
  public string FIO {
    set { _FIO = value; }
    get { return _FIO; }
  }
  public string UsersName {
    set { _UsersName = value; }
    get { return _UsersName; }
  }
  public string UsersPassword {
    set { _UsersPassword = value; }
    get { return _UsersPassword; }
  }
  public int RoleId {
    set { _RoleId = value; }
    get { return _RoleId; }
  }
  public string RoleName {
    set { _RoleName = value; }
    get { return _RoleName; }
  }
  public string Description {
    set { _Description = value; }
    get { return _Description; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

Лістинг 14. Код класу «VertexsProvider»

```
using RouteOptimizerApp.AppCode;
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.OleDb;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace RouteOptimizerApp.Providers {
  class VertexsProvider {
    private string _ConnString =
System.Configuration.ConfigurationSettings.AppSettings["CONNECT"];

    public void InsertBatchVertexs(List<Vertexs> Vertexs) {
      string SqlString = "INSERT INTO Vertexs (VertexName, SimulationsId) Values(?, ?)";
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          cmd.CommandType = CommandType.Text;
          conn.Open();
          for (int i = 0; i < Vertexs.Count; i++) {
            cmd.Parameters.AddWithValue("VertexName", Vertexs[i].VertexsName.ToString());
            cmd.Parameters.AddWithValue("SimulationsId", Vertexs[i].SimulationsId);
            cmd.ExecuteNonQuery();
            while (cmd.Parameters.Count > 0) {
              cmd.Parameters.RemoveAt(0);
            }
          }
          conn.Close();
        }
      }
    }

    public List<Vertexs> GetAllVertexsBySimulationsId(int SimulationsId) {
      int i = 0;
      string SqlString = "SELECT * FROM Vertexs WHERE SimulationsId=" + SimulationsId;

      List<Vertexs> Vertexs = new List<Vertexs>();
      using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
        using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
          conn.Open();
          using (OleDbDataReader reader = cmd.ExecuteReader()) {
            while (reader.Read()) {
              Vertexs oneVertexs = new Vertexs();
              oneVertexs.Number = ++i;
              oneVertexs.VertexsId = Convert.ToInt32(reader["VertexsId"]);
              oneVertexs.VertexsName = reader["VertexName"].ToString();
```

```
            oneVertexs.SimulationsId = Convert.ToInt32(reader["SimulationsId"]);
            Vertexs.Add(oneVertexs);
          }
        }
        conn.Close();
      }
    }
    return Vertexs;
  }


  public void DeleteVertexsByVertexsId(int SimulationsId) {
    string SqlString = "DELETE FROM Vertexs WHERE SimulationsId=" +
SimulationsId.ToString();
    using (OleDbConnection conn = new OleDbConnection(_ConnString)) {
      using (OleDbCommand cmd = new OleDbCommand(SqlString, conn)) {
        conn.Open();
        cmd.ExecuteNonQuery();
        conn.Close();
      }
    }
  }

 }
}


public class Vertexs {
 private int _Number;
 private int _VertexsId;
 private string _VertexsName;
 private int _SimulationsId;
 private int _CitiesId;
 private string _Message;

 public Vertexs() {
  _Number = 0;
  _VertexsId = 0;
  _VertexsName = String.Empty;
  _SimulationsId = 0;
  _CitiesId = 0;
  _Message = String.Empty;
 }

 public int Number {
  set { _Number = value; }
  get { return _Number; }
 }
 public int VertexsId {
  set { _VertexsId = value; }
  get { return _VertexsId; }
 }
```

```
  public string VertexsName {
    set { _VertexsName = value; }
    get { return _VertexsName; }
  }
  public int SimulationsId {
    set { _SimulationsId = value; }
    get { return _SimulationsId; }
  }
  public int CitiesId {
    set { _CitiesId = value; }
    get { return _CitiesId; }
  }
  public string Message {
    set { _Message = value; }
    get { return _Message; }
  }
}
```

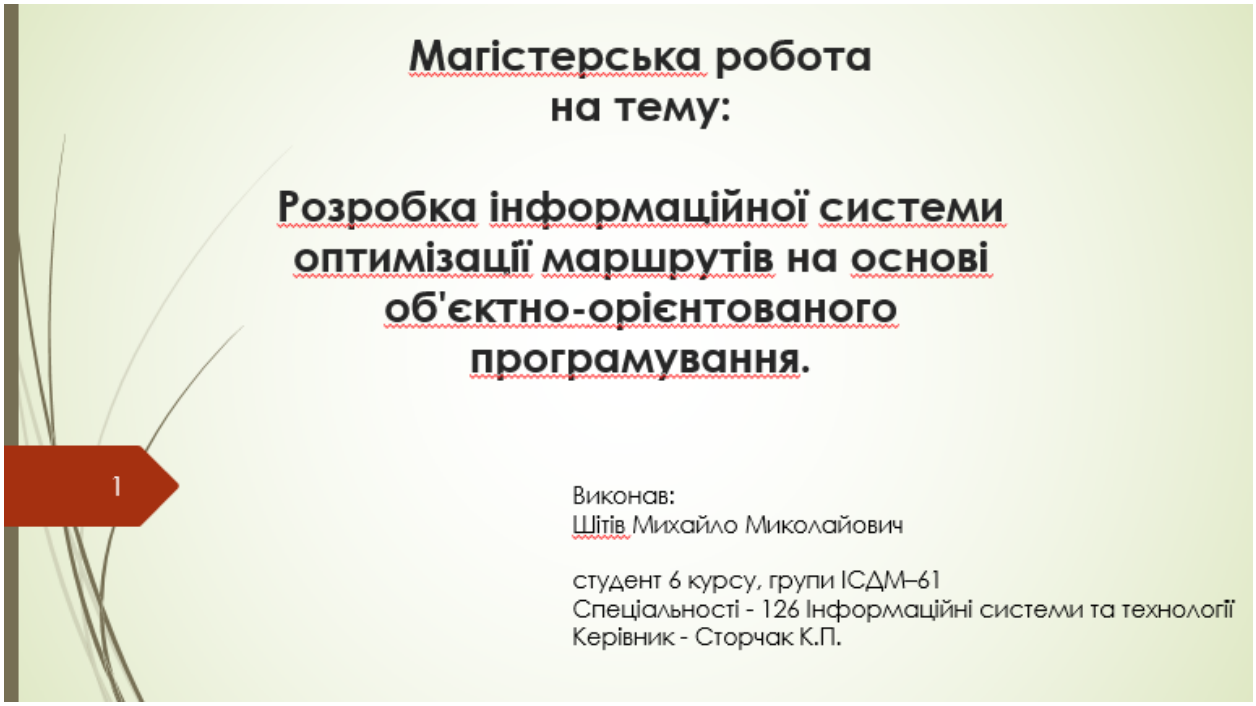
Лістинг 15. Код класу «RouteOptimizerMDI»

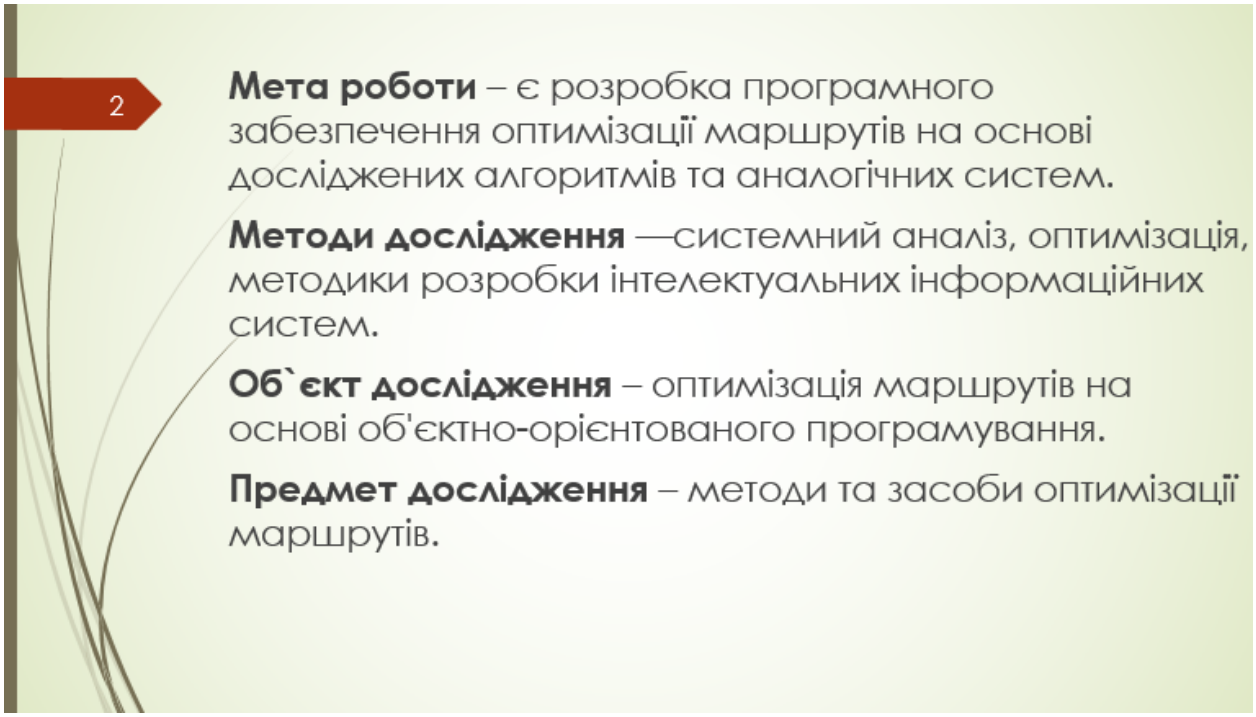
```
using RouteOptimizerApp.AppCode;
using RouteOptimizerApp.Forms.Control;
using RouteOptimizerApp.Forms.Dictionary;
using RouteOptimizerApp.Forms.Systems;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace RouteOptimizerApp {
  public partial class RouteOptimizerMDI : Form {

    public RouteOptimizerMDI() {
      InitializeComponent();
    }

    public void CloseAllWindows() {
      Form[] childArray = this.MdiChildren;
      foreach (Form childForm in childArray) {
        childForm.Close();
      }
    }

    private void користувачіToolStripMenuItem_Click(object sender, EventArgs e) {
      if (LoginForm.CurrentUser.RoleId == 1) {
        CloseAllWindows();
```

```
        UsersForm usersForm = new UsersForm();
        usersForm.MdiParent = this;
        usersForm.WindowState = FormWindowState.Maximized;
        usersForm.Show();
      } else {
        MessageBox.Show(NamesMy.MessageBoxExaption.YouDontHavePermission);
      }
    }

    private void системнийЖурналToolStripMenuItem_Click(object sender, EventArgs e) {
      if (LoginForm.CurrentUser.RoleId == 1) {
        CloseAllWindows();
        SystemLogForm systemLogForm = new SystemLogForm();
        systemLogForm.MdiParent = this;
        systemLogForm.WindowState = FormWindowState.Maximized;
        systemLogForm.Show();
      } else {
        MessageBox.Show(NamesMy.MessageBoxExaption.YouDontHavePermission);
      }
    }

    private void містаToolStripMenuItem_Click(object sender, EventArgs e) {
      CloseAllWindows();
      CitiesForm citiesForm = new CitiesForm();
      citiesForm.MdiParent = this;
      citiesForm.WindowState = FormWindowState.Maximized;
      citiesForm.Show();
    }

    private void симуляторМаршрутівToolStripMenuItem_Click(object sender, EventArgs e) {
      CloseAllWindows();
      RouteSimulatorForm routeSimulatorForm = new RouteSimulatorForm();
      routeSimulatorForm.MdiParent = this;
      routeSimulatorForm.WindowState = FormWindowState.Maximized;
      routeSimulatorForm.Show();
    }

    private void завантаженняМаршрутівToolStripMenuItem_Click(object sender, EventArgs e)
{
      CloseAllWindows();
      SimulationsLoadForm simulationsLoadForm = new SimulationsLoadForm();
      simulationsLoadForm.MdiParent = this;
      simulationsLoadForm.WindowState = FormWindowState.Maximized;
      simulationsLoadForm.Show();
    }

    private void вихідToolStripMenuItem_Click(object sender, EventArgs e) {
      this.Close();
    }

    private void RouteOptimizerMDI_Resize(object sender, EventArgs e) {
      this.BackgroundImage = Properties.Resources.back_img;
```

```
    }
  }
}
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ (Презентація)





3

Розрахунковий час рішення TSP шляхом повного перебору

Розрахунковий час вирішення TSP методом повного перебору	
Кількість міст	Розрахунковий час
10	1/3 секунди
13	8 хвилин
15	1 рік
20	193 року

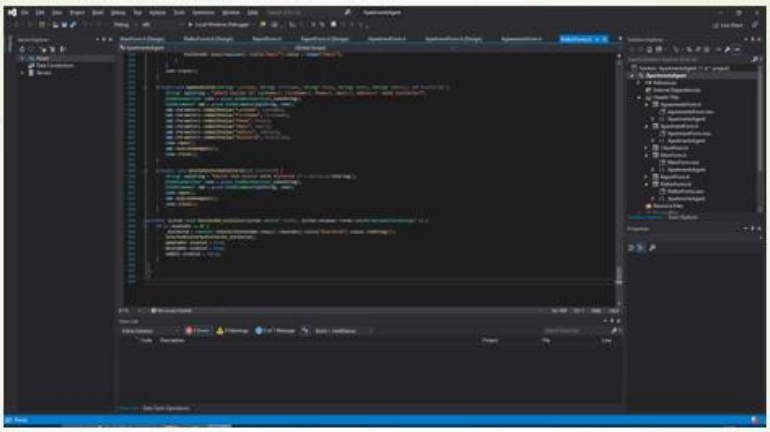
4

Порівняння алгоритму найближчого сусіда, генетичного та жадібних алгоритмів при вирішенні завдань зі 100 та 1000 містами

Порівняння алгоритмів для вирішення TSP при 100 містах			
Вибраний алгоритм	Довжина оптимального маршруту (км)	Витрачений час (сек)	Кількість ітерацій
Алгоритм найближчого сусіда	26654	2.5	100
Генетичний	225379	45	10000
Алгоритм Дейкстри	23211	0.07	18
Порівняння алгоритмів для вирішення TSP за 1000 міст			
Вибраний алгоритм	Довжина оптимального маршруту (км)	Витрачений час (сек)	Кількість ітерацій
Алгоритм найближчого сусіда	83838	98.5	1000
Генетичний	282766	468	10000
Алгоритм Дейкстри	72701	117	151

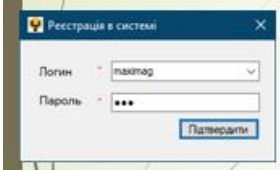
5

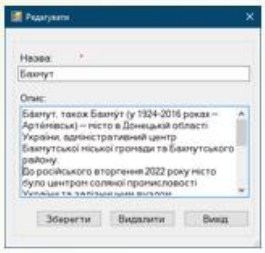
Інтерфейс середовища розробки Visual Studio

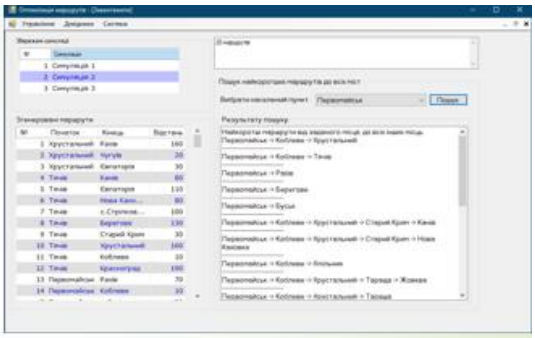


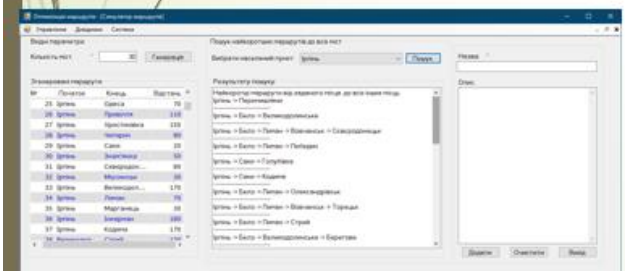
Розробку інформаційної системи виконано у середовищі Microsoft Visual Studio при використанні мови програмування C# та СУБД MS Access.

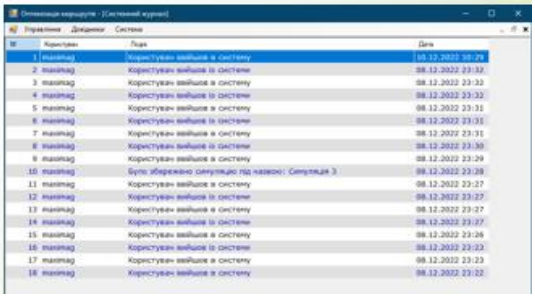
6











Висновки

У першій частині роботи було розглянуте поняття транспортної логістики. Виділено основні аспекти.

У другій частині роботи розглянуто дослідження системи та описано її архітектуру. Зроблено аналіз вимог до ПЗ.

В третій частині було здійснено розробку програмного забезпечення та проведено тестування.