Міністерство освіти і науки України
Кам’янець-Подільський національний університет імені Івана Огієнка
Фізико-математичний факультет
Кафедра комп’ютерних наук

**Кваліфікаційна робота магістра**

з теми: «**Система оптимізації та управління дорожніми маршрутами з голосовим інтерфейсом на платформі .NET MAUI**»

Виконав: здобувач вищої освіти групи KN1-M24,
спеціальності 122 комп'ютерні науки
Гадзира Тарас Ігорович
(прізвище та ім’я і по батькові здобувача вищої освіти)

Керівник: Федорчук В. А., д-р техн. наук, проф.
(прізвище та ініціали, науковий ступінь, учене звання)

Рецензент:
(прізвище та ініціали, науковий ступінь, учене звання)

м. Кам’янець-Подільський – 2025 р.

# ЗМІСТ

# ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface).

BLOB – Binary Large Object.

BCrypt – алгоритм хешування паролів, криптостійкий метод для безпечного зберігання облікових даних.

EF Core – Entity Framework Core.

HTTP – HyperText Transfer Protocol.

HTTPS – безпечна версія HTTP з шифруванням через TLS/SSL.

ID – ідентифікатор (Identifier).

JSON – формат обміну даними (JavaScript Object Notation).

LINQ – мова інтегрованих запитів (Language Integrated Query).

MAUI – .NET Multi-platform App UI.

NAudio – бібліотека для роботи з аудіо у .NET, підтримує запис звуку.

ORM – мапування об’єктно-реляційних даних (Object-Relational Mapping), технологія для зв’язку об’єктів коду з таблицями БД.

REST – архітектурний стиль веб-сервісів (Representational State Transfer).

SecureStorage – безпечне сховище у .NET MAUI для зберігання чутливих даних (наприклад, токенів, паролів).

Speech-to-Text – розпізнавання мовлення в текст.

SQL – Structured Query Language.

TSP – задача комівояжера (Travelling Salesman Problem), класична оптимізаційна задача пошуку найкоротшого циклічного маршруту.

UI – користувацький інтерфейс (User Interface).

URL – універсальний локатор ресурсу (Uniform Resource Locator), адреса веб-ресурсу в інтернеті.

БД – база даних.

ПК – персональний комп’ютер.

ШІ – Artificial Intelligence.

# АНОТАЦІЯ

Гадзира Т. І. Система оптимізації та управління дорожніми маршрутами з голосовим інтерфейсом на платформі .NET MAUI. – Кваліфікаційна робота за спеціальністю 122 «Комп'ютерні науки». – Кам'янець-Подільський національний університет імені Івана Огієнка, Кам'янець-Подільський, 2025.

У магістерській роботі розроблено кросплатформний додаток на основі .NET MAUI з інтеграцією голосового вводу через Google Cloud Speech-to-Text та оптимізації маршрутів за допомогою Google Distance Matrix API та алгоритму Branch & Bound. Система включає автономну роботу на різних ОС, голосовий ввід міст українською мовою, формування матриці відстаней, розв'язання задачі комівояжера у трьох постановках, керування маршрутами з категоріями, фільтрацією, сортуванням, збереженням виїздів та корпоративний обмін повідомленнями. Реалізовано локальну базу даних SQLite з EF Core, безпечне сховище SecureStorage та прототип гібридної архітектури для масштабування.

**Ключові слова:** .NET MAUI, задача комівояжера, Branch & Bound, Google Cloud Speech-to-Text, Google Distance Matrix API, голосовий ввід, кросплатформність, корпоративні повідомлення, оптимізація маршрутів, SQLite, EF Core.

Кваліфікаційна робота містить результати власних досліджень. Використання чужих ідей, результатів і текстів супроводжується відповідними посиланнями на джерела.

# ABSTRACT

Hadzyra T. I. System for optimizing and managing road routes with a voice interface on the .NET MAUI platform. – Qualification work in specialty 122 “Computer Science”. – Kamianets-Podilskyi Ivan Ogienko National University, Kamianets-Podilskyi, 2025.

The master's thesis developed a cross-platform application based on .NET MAUI with the integration of voice input via Google Cloud Speech-to-Text and route optimization using the Google Distance Matrix API and the Branch & Bound algorithm. The system includes autonomous operation on different OSes, voice input of cities in Ukrainian, formation of a distance matrix, solving the traveling salesman problem in three settings, route management with categories, filtering, sorting, saving trips and corporate messaging. Implemented a local SQLite database with EF Core, SecureStorage secure storage, and a prototype of a hybrid architecture for scaling.

**Keywords:** .NET MAUI, traveling salesman task, Branch & Bound, Google Cloud Speech-to-Text, Google Distance Matrix API, voice input, cross-platform, corporate messaging, route optimization, SQLite, EF Core.

The qualification work contains the results of your own research. The use of other people's ideas, results, and texts is accompanied by appropriate references to the sources.

# ВСТУП

**Актуальність теми** зумовлена стрімким розвитком цифрових технологій, мобільності та логістики у сучасному світі. Щоденно мільйони людей – водії, кур'єри, туристи, менеджери – стикаються з необхідністю швидкого та ефективного планування маршрутів між кількома пунктами. Зростання обсягів e-commerce, доставки та туризму вимагає інструментів, які мінімізують час, витрати та зусилля на організацію переміщень.

Голосовий ввід набуває особливого значення в умовах мобільності, коли ручне введення даних є незручним або небезпечним. Технології розпізнавання мовлення дозволяють значно прискорити взаємодію з додатком, роблячи його доступним навіть під час руху.

Задача комівояжера (TSP) залишається однією з фундаментальних в оптимізації маршрутів. Її розв'язання має практичне застосування в логістиці, транспорті, туризмі та плануванні. Розробка ефективних алгоритмів для мобільних пристроїв є важливим кроком до автоматизації цих процесів.

Кросплатформовість відповідає сучасним тенденціям розробки, коли один код працює на різних операційних системах. Це забезпечує широке охоплення аудиторії та знижує витрати на підтримку.

Корпоративна взаємодія через обмін повідомленнями та спільне використання маршрутів відкриває нові можливості для командної роботи в логістичних і транспортних компаніях.

Таким чином, розробка кросплатформового мобільного додатку з голосовим вводом та оптимізацією маршрутів є актуальною з наукової, практичної та соціальної точок зору, відповідаючи потребам цифрового суспільства, підвищуючи безпеку та ефективність повсякденних і професійних переміщень.

**Об'єктом дослідження** є процес планування та оптимізації маршрутів переміщення між кількома географічними пунктами з використанням мобільних цифрових технологій.

**Предметом дослідження** є кросплатформний програмний комплекс на базі .NET MAUI, що інтегрує голосовий ввід, обчислення відстаней через хмарні API та розв'язання задачі комівояжера для автоматизації логістичного планування.

**Метою дослідження** є розробка методів і засобів підвищення ефективності планування маршрутів шляхом створення спеціалізованого мобільного додатку, який забезпечує швидке введення даних голосом, точне обчислення відстаней, глобальну оптимізацію шляху та підтримку корпоративної взаємодії.

Для досягнення поставленої мети необхідно виконати такі **завдання**:

1. Проаналізувати сучасні підходи до розв'язання задачі комівояжера (TSP) у відкритих, замкнутих та вільних постановках.
2. Дослідити технології голосового розпізнавання мовлення та інтеграції з Google Cloud Speech-to-Text і Distance Matrix API для автоматизації введення та обробки геоданих.
3. Розробити архітектуру кросплатформового додатку на .NET MAUI з підтримкою, включаючи локальну БД (SQLite) та асинхронну взаємодію з UI.
4. Реалізувати модуль оптимізації маршрутів у трьох режимах: фіксований старт/кінець, замкнутий цикл та вільний вибір найкращого шляху.
5. Розробити механізми корпоративного обміну повідомленнями як прототип архітектури для майбутньої міграції на клієнт-серверну модель.
6. Провести тестування всіх модулів (голосовий ввід, матриця відстаней, Branch & Bound, БД), оцінити продуктивність, точність та обмеження.

**Методи дослідження.** Методологія дослідження зосереджена на комплексному аналізі процесів планування маршрутів у реальних логістичних сценаріях, включаючи поведінку користувачів під час введення геоданих, обробки матриць відстаней та прийняття рішень щодо оптимізації шляху. Використано емпіричні методи (тестування на реальних пристроях), математичне моделювання (задача комівояжера), порівняльний аналіз (Branch & Bound vs евристики), експериментальні випробування (час виконання,

точність API) та системний підхід до архітектури ПЗ. Дослідження проводилося з урахуванням обмежень мобільних платформ, мережевих затримок та вимог до офлайн-доступу.

**Практичне значення одержаних результатів.** Розроблений кросплатформовий мобільний додаток на базі .NET MAUI має високу практичну цінність для індивідуальних і корпоративних користувачів у сфері логістики, туризму, доставки та повсякденного планування. Реалізовані механізми голосового вводу, автоматичного формування матриці відстаней та оптимізації за трьома постановками TSP дозволяють скоротити час планування, зменшити витрати пального та підвищити безпеку за рахунок виключення ручного введення під час руху. Прототип корпоративного обміну повідомленнями відкриває шлях до інтеграції в системи управління флотом. Результати можуть бути застосовані при створенні комерційних логістичних додатків, вбудованих модулів для ERP-систем та освітніх тренажерів з геооптимізації.

**Наукова новизна** полягає у розробці інтегрованого підходу до розв’язання задачі комівояжера на мобільних пристроях з урахуванням голосового вводу, хмарних API та локального зберігання. Вперше реалізовано універсальний модуль оптимізації, що підтримує три постановки TSP (відкрита, замкнута, вільний старт/кінець) в рамках єдиної архітектури Branch & Bound з обрізкою гілок за поточним рекордом. Запропоновано гібридну модель даних (локальна SQLite + перспектива міграції на ASP.NET Core), що забезпечує офлайн-доступ для індивідуальних користувачів і масштабованість для корпоративного сегмента. Внесок у теорію – формалізація переходу між режимами TSP без зміни алгоритмічного ядра, що підвищує універсальність і повторне використання коду.

**Структура роботи.** Робота складається зі вступу, 5 розділів, висновків та списку використаних джерел, що налічує 27 джерел.

# РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

## 1.1 Задача пошуку оптимального маршруту між декількома пунктами та її вирішення

Задача пошуку найоптимальнішого шляху між декількома точками є однією з ключових у галузі оптимізації та теорії графів. Ці задачі часто виникають у логістиці, плануванні маршрутів, транспортних системах, комунікаційних мережах та навіть у біоінформатиці. Загальна мета полягає в мінімізації витрат, які можуть включати відстань, час, витрати пального або інші ресурси.

Однією з найбільш відомих задач у цій сфері є задача комівояжера (Travelling Salesman Problem, TSP). Її суть полягає в тому, щоб знайти найкоротший шлях, який проходить через набір заданих точок (міст, пунктів) рівно один раз і повертається в початкову точку. Це класична NP-складна задача, що означає, що для великих наборів точок оптимальне рішення не може бути знайдене за поліноміальний час, і доводиться використовувати наближені або евристичні методи [1].

Для вирішення задач такого типу застосовуються різні підходи. Ці методи дозволяють знайти оптимальне рішення, але вони ефективні лише для невеликої кількості точок. Один із них – це метод повного перебору, де розглядаються всі можливі маршрути й вибирається найкоротший. Однак кількість можливих маршрутів зростає експоненційно зі збільшенням кількості точок. Іншим підходом є метод гілок та меж (Branch and Bound), який будує дерево можливих рішень, відсікаючи гілки, що не можуть привести до оптимального рішення. Також використовують методи динамічного програмування, такі як алгоритм Беллмана-Хелда-Карпа, який забезпечує ефективніше розв’язання, але все ще з експоненційною складністю [2]. Для великих задач, де екзактні методи стають непрактичними, використовуються

евристики, які забезпечують "достатньо хороші" рішення за прийнятний час. Серед них:

1. Алгоритм найближчого сусіда (Nearest Neighbor): починається з однієї точки, кожного разу вибирається найближча ще не відвідана точка.

2. Жадібні алгоритми (Greedy): побудова маршруту на основі локальних оптимальних рішень.

3. Метод вставок: послідовне додавання нових точок у вже знайдений маршрут з мінімальним збільшенням загальної довжини.

Ці методи імітують природні або фізичні процеси, щоб знайти рішення. Вони часто використовуються для вирішення складних варіантів задач пошуку оптимального шляху.

1. Генетичні алгоритми (GA): базуються на принципах природного відбору.

2. Метод імітації відпалу (Simulated Annealing): моделює процес охолодження металів, поступово зменшуючи "температуру" пошуку, щоб уникати локальних мінімумів і знайти глобальне [3].

3. Мурашиний алгоритм (Ant Colony Optimization): імітує поведінку колоній мурах, які шукають найкоротший шлях між колонією та джерелом їжі, залишаючи "феромонні" сліди, що впливають на вибір інших мурах.

Таким чином, задача пошуку оптимального маршруту є багатогранною і складною, але розвиток алгоритмів і обчислювальних потужностей дозволяє успішно вирішувати навіть великі задачі з численними обмеженнями [4].

## 1.2 Технології розпізнавання мовлення та кросплатформної розробки

Розвиток мобільних технологій та штучного інтелекту відкрив нові можливості для взаємодії користувача з програмним забезпеченням. Одним із ключових напрямів є розпізнавання мовлення [4, 5], що дозволяє перетворювати усну мову на текст у реальному часі. Ця технологія [6, 7] значно підвищує зручність використання додатків, особливо під час руху,

роботи в польових умовах або за наявності обмежень у введенні даних вручну. У контексті планування маршрутів голосовий ввід міст усуває необхідність ручного набору назв, зменшує час взаємодії та знижує ймовірність помилок, що особливо важливо для водіїв, логістів і туристів.

Серед сучасних рішень лідирують Google Cloud Speech-to-Text [8], Microsoft Azure Speech Service [9], Amazon Transcribe [10] та Apple Speech Framework [11]. Google Cloud пропонує високу точність розпізнавання для понад 120 мов, включаючи українську, підтримку різних акцентів і шумозаглушення. Система працює за моделлю «хмара», що забезпечує найкращу якість, але вимагає стабільного інтернет-з'єднання. Локальні альтернативи, такі як Vosk API або Whisper (OpenAI), дозволяють працювати офлайн, але поступаються в точності, особливо для української мови. У розробленому додатку обрано Google Cloud Speech-to-Text через її надійність, підтримку української мови та простоту інтеграції через REST API. Для забезпечення автономності передбачено резервний режим із кешуванням часто використовуваних назв міст [12].

Паралельно з голосовим вводом важливим є вибір платформи для кросплатформної розробки. Сучасні фреймворки дозволяють створювати один кодовий базис для Android, iOS, Windows і macOS, що значно скорочує час і витрати на розробку. Серед лідерів – Flutter (Google), React Native (Meta), Xamarin і .NET MAUI (Microsoft). Flutter використовує мову Dart і компілюється в нативний код, забезпечуючи високу продуктивність, але має обмежену інтеграцію з .NET-екосистемою. React Native базується на JavaScript, що зручно для веб-розробників, однак страждає від проблем із продуктивністю та доступом до нативних API. Xamarin, як попередник MAUI, вже застаріває, хоча й підтримує C# [13].

Таким чином, поєднання Google Cloud Speech-to-Text і .NET MAUI є оптимальним рішенням: перше забезпечує високу точність розпізнавання, друге – єдину кодову базу, нативну продуктивність і повний контроль над логікою.

## 1.3 Аналіз аналогів – сучасних систем планування та оптимізації маршрутів

На ринку існує значна кількість рішень для планування маршрутів, що охоплюють різні сегменти – від індивідуального використання до корпоративної логістики. Проведений аналіз аналогів дозволяє оцінити їх функціональні можливості, технологічні обмеження та позиціонування, а також визначити нішу для розробленого додатку. Основна увага приділяється наявності підтримки задачі комівояжера, голосового вводу, кросплатформності, офлайн-режиму та вартості. Розглянуто п’ять ключових систем: Google Maps [16], Waze [17], Route4Me [18], OptimoRoute [19].

Google Maps є найпоширенішим навігаційним сервісом, що пропонує маршрутизацію в реальному часі, альтернативні шляхи та інтеграцію з Google Distance Matrix API. Система підтримує частковий голосовий ввід через Google Assistant, дозволяє додавати проміжні точки, але не виконує автоматичну оптимізацію порядку їх відвідування. Відсутність розв’язання задачі комівояжера та неможливість збереження персональних маршрутів із категоріями обмежують її використання в логістичних задачах. Перевагою є безкоштовність і кросплатформність, однак відсутній офлайн-режим для складних обчислень.

Waze вирізняється соціальною складовою та оновленням трафіку в реальному часі завдяки внескам користувачів. Додаток підтримує голосові команди, має простий інтерфейс і працює офлайн з обмеженим функціоналом. Проте Waze не пропонує API, не вирішує задачу комівояжера і не дозволяє створювати багатоточкові маршрути з оптимізацією. Його основне призначення – навігація в межах одного маршруту, а не планування складних поїздок.

Route4Me і OptimoRoute – це професійні хмарні платформи для оптимізації логістики. Вони розв’язують задачу комівояжера для тисяч точок, враховують обмеження за часом, вантажем та робочим графіком. Системи надають веб-інтерфейс, мобільні додатки для водіїв, звіти, аналітику та

інтеграцію з CRM. Route4Me підтримує імпорт даних із Excel, GPS-трекінг і динамічне перепланування. OptimoRoute додає прогнозування навантаження та балансування робочого дня. Порівняльний аналіз наведено в таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз сучасних систем планування та оптимізації маршрутів

| Критерій | Google Maps | Waze | Route4Me | OptimoRoute |
|---|---|---|---|---|
| Розв’язання TSP | – | – | + | + |
| Голосовий ввід міст | + | + | – | – |
| Кросплатформність | + | – | + | + |
| Офлайн-режим (оптимізація) | – | – | – | – |
| Збереження маршрутів | + | – | + | + |
| Корпоративні функції | – | – | + | + |
| Вартість | Безкоштовно | Безкоштовно | $99+/міс | $35+/корист. |

Аналіз показує, що жодна з розглянутих систем не поєднує в собі голосовий ввід, розв’язання задачі комівояжера, офлайн-доступність, кросплатформність і безкоштовність. Професійні рішення надто дорогі та залежні від мережі, а безкоштовні – обмежені в функціоналі. Розроблений додаток на основі .NET MAUI заповнює цю прогалину, пропонуючи автономне обчислення оптимального маршруту, зручний голосовий інтерфейс, систему категорій і корпоративних повідомлень – і все це в єдиній кросплатформній кодовій базі без абонентської плати.

### 1.4 Постановка задачі

На основі проведеного аналізу предметної області, сучасних систем планування маршрутів та технологій розпізнавання мовлення й кросплатформної розробки сформульовано задачу дипломної роботи.

Необхідно розробити кросплатформний мобільний додаток для оптимізації логістичних маршрутів з голосовим вводом пунктів призначення на основі .NET MAUI.

Додаток має забезпечувати:

1. Реєстрацію та автентифікацію користувачів з можливістю редагування профілю та безпечного зберігання паролів (BCrypt);
2. Голосовий ввід назв міст українською мовою через Google Cloud Speech-to-Text з локальним записом звуку та підтримкою офлайн-кешування;
3. Обчислення матриці відстаней між пунктами через Google Matrix API;
4. Розв’язання задачі комівояжера за допомогою алгоритму Branch & Bound з двома режимами: фіксований старт і кінець; вільний старт і кінець;
5. Збереження маршрутів у локальній базі даних (SQLite + EF Core) з можливістю категоризації, фільтрації за кількістю пунктів і категорією, сортування за відстанню, кількістю виїздів та назвою;
6. Корпоративні функції: обмін повідомленнями між користувачами однієї компанії, перегляд вхідних і надісланих повідомлень;
7. Кросплатформність, нативний UI та автономну роботу в офлайн-режимі (крім голосового розпізнавання та API відстаней).

Система повинна бути безкоштовною, інтуїтивно зрозумілою, продуктивною та надійною (обробка помилок мережі, мікрофона, API).

**Висновки до розділу 1**

У першому розділі проведено аналіз предметної області, математичної постановки задачі комівояжера, сучасних технологій розпізнавання мовлення та кросплатформної розробки, а також основні аналоги. Встановлено, що ринок потребує доступного, автономного та інтелектуального рішення, яке поєднує голосовий ввід, точну оптимізацію маршрутів і корпоративні функції. На основі цього сформульовано чітку задачу розробки кросплатформного мобільного додатку на основі .NET MAUI, що стане основою для подальшого проєктування та реалізації.

# РОЗДІЛ 2 ВИБІР ТА ОБҐРУНТУВАННЯ ТЕХНОЛОГІЧНОГО СТЕКУ

## 2.1 Порівняння кросплатформних фреймворків (MAUI, Flutter, React Native)

Вибір кросплатформного фреймворку є ключовим для забезпечення єдиної кодової бази, продуктивності та підтримки різних операційних систем. На ринку домінують три основні рішення: .NET MAUI (Microsoft) [20], Flutter (Google) [21] та React Native (Meta) [22]. Кожен із них має власну архітектуру, мову програмування та екосистему, що впливає на швидкість розробки, продуктивність, доступ до нативних API та довгострокову підтримку. У контексті розробки мобільного додатку для оптимізації маршрутів з голосовим вводом, інтеграцією Google API та локальною базою даних проведено порівняльний аналіз цих технологій за критеріями: мова, продуктивність, кросплатформність, інтеграція з .NET, доступ до мікрофона та камери, підтримка офлайн-режиму, спільнота та інструменти розробки.

.NET MAUI є частиною .NET 6+ і еволюцією Xamarin.Forms. Фреймворк використовує C# – мову з потужною типізацією, асинхронним програмуванням та інтеграцією з Entity Framework Core, що ідеально підходить для роботи з SQLite, BCrypt та обчислювальними алгоритмами, такими як Branch & Bound. MAUI забезпечує нативний рендеринг UI через платформні контролери, підтримує Android, iOS, Windows і macOS з одного проєкту. Доступ до мікрофона здійснюється через NAudio (Windows/Android) або Plugin.AudioRecorder, а також через вбудовані Permissions API. Продуктивність близька до нативної, особливо при обчисленнях. Інструменти – Visual Studio з гарячим перезавантаженням і XAML Hot Reload – значно прискорюють розробку.

Flutter базується на мові Dart і компілюється в нативний ARM-код через Skia. Це забезпечує високу продуктивність і однаковий UI на всіх платформах завдяки власному рендерингу. Flutter має багату бібліотеку віджетів, швидке оновлення через Hot Reload і підтримку Android, iOS, Windows, macOS, Linux

та веб. Проте інтеграція з .NET-екосистемою відсутня, що ускладнює використання EF Core, BCrypt чи Google Cloud SDK без додаткових обгорток. Доступ до мікрофона реалізується через плагіни (наприклад, flutter_sound), але потребує ручного налаштування для кожної платформи. Офлайн-режим підтримується добре, але відсутність нативної інтеграції з .NET робить Flutter менш придатним для проєктів, орієнтованих на C# [23].

Основні порівняльні характеристики кросплатформних фреймворків наведені в таблиці 2.1.

Таблиця 2.1

Порівняльна характеристика кросплатформних фреймворків

| Критерій | .NET MAUI | Flutter | React Native |
|---|---|---|---|
| Мова | C# | Dart | JavaScript/TS |
| Продуктивність (обчислення) | Висока | Висока | Середня |
| Кросплатформність | Android, iOS, Windows, macOS | Усі + веб | Android, iOS, Windows, веб |
| Інтеграція з .NET | Нативна | — | Через API |
| Доступ до мікрофона | NAudio, Plugin | flutter_sound | react-native-audio |
| Офлайн-режим | Повний | Повний | Повний |
| Інструменти | Visual Studio | VS Code, Android Studio | VS Code |
| Спільнота та підтримка | Microsoft | Google | Meta |

React Native використовує JavaScript/TypeScript і рендерить нативні компоненти через міст (bridge). Це дозволяє швидко створювати додатки з веб-розробниками, але міст створює затримки при інтенсивних обчисленнях, таких як TSP. Підтримка платформ включає Android, iOS, Windows (через

React Native for Windows) та веб. Доступ до мікрофона – через бібліотеки на кшталт react-native-audio-recorder-player, але стабільність нижча, ніж у MAUI. Інтеграція з .NET можлива лише через REST API або WebView, що ускладнює локальну логіку. Продуктивність гірша при CPU-інтенсивних задачах, а спільнота, хоча й велика, фрагментована через велику кількість сторонніх пакетів.

### 2.2 Обґрунтування вибору .NET MAUI

Вибір .NET MAUI як основного фреймворку для розробки кросплатформного мобільного додатку є стратегічним рішенням, що ґрунтується на технічних, архітектурних і економічних перевагах цієї платформи. На відміну від попередніх рішень Microsoft (Xamarin.Forms), MAUI є частиною єдиної екосистеми .NET 6+, що забезпечує уніфіковану кодову базу для мобільних, десктопних і хмарних застосунків. Це дозволяє використовувати одну мову програмування – C# – для всієї логіки, включаючи роботу з базами даних, криптографією, мережевими запитами та складними обчислювальними алгоритмами, такими як Branch & Bound для задачі комівояжера.

Однією з ключових переваг .NET MAUI є нативна продуктивність. Фреймворк не використовує WebView або інтерпретатор, а рендерить інтерфейс через платформні контролери: UIKit на iOS, WinUI на Windows, Android Views на Android. Це забезпечує швидкість відгуку, близьку до нативних додатків, що критично важливо під час запису звуку, обробки голосу та виконання обчислень. Реалізація голосового вводу через NAudio (Windows) і MediaRecorder (Android) відбувається безпосередньо, без проміжних шарів, що гарантує низьку латентність і точний контроль над аудіопотоком у форматі WAV (16 кГц, моно), необхідному для Google Cloud Speech-to-Text.

Інтеграція з Entity Framework Core – ще одна вагома перевага. MAUI дозволяє використовувати повноцінний ORM для роботи з локальною базою даних SQLite, що забезпечує типобезпечний доступ до моделей

(OptimizedRoute, RouteCategory, AppUser, Message), автоматичне відстеження змін, міграції та асинхронні запити. Це значно спрощує реалізацію функцій збереження маршрутів, фільтрації, сортування та корпоративних повідомлень. Використання BCrypt.Net для хешування паролів і SecureStorage для зберігання облікових даних відбувається природно в межах .NET-екосистеми, без необхідності підключення сторонніх бібліотек.

Важливим є також доступ до платформних API. MAUI надає єдиний інтерфейс для роботи з мікрофоном, мережею, файловою системою та геолокацією. Наприклад, запит дозволу на запис звуку реалізується через Permissions.RequestAsync<Permissions.Microphone>(), що працює однаково на всіх платформах. Інтеграція з Google Cloud SDK (Speech-to-Text, Distance Matrix) здійснюється через REST API з використанням HttpClient і Google.Apis.Auth, що підтримується нативно в .NET. Файл автентифікації копіюється в збірку через .csproj і встановлюється як змінна середовища – стандартна практика в .NET.

З точки зору інструментів розробки, Visual Studio з MAUI Workload забезпечує гаряче перезавантаження (Hot Reload), інтеграцію з Git, профайлери продуктивності та емулятори. Це прискорює ітерації розробки, особливо при налаштуванні UI через XAML і тестуванні алгоритмів.

### 2.3 Інтеграція Google Cloud Speech-to-Text та Distance Matrix API

Інтеграція хмарних сервісів Google Cloud є критично важливою для реалізації ключових функцій додатку – голосового вводу міст і обчислення реальних відстаней між пунктами. Використано два API: Google Cloud Speech-to-Text для розпізнавання мовлення та Google Distance Matrix API для отримання матриці відстаней [24]. Обидва сервіси працюють за REST-архітектурою, що дозволяє легко інтегрувати їх у .NET MAUI через HttpClient і Google SDK. Такий підхід забезпечує високу точність, підтримку української мови та гнучкість, але вимагає стабільного інтернет-з’єднання та правильного налаштування безпеки.

Google Cloud Speech-to-Text обрано як основний двигун розпізнавання завдяки найкращій підтримці української мови серед хмарних рішень. Процес складається з кількох етапів. Спочатку аудіо записується локально у форматі WAV (16 кГц, моно, 16 біт) – це стандарт, рекомендований Google для максимальної точності. На Windows використовується бібліотека NAudio з WaveInEvent, на Android – MediaRecorder з відповідними параметрами кодування. Запис зберігається в MemoryStream, після чого конвертується в byte[] і відправляється у запиті до API. Налаштування розпізнавання включають мову uk-UA, автоматичне визначення пунктуації та максимальну кількість альтернатив, що зменшує навантаження.

Запит формується через Google.Cloud.Speech.V1 NuGet-пакет. Перед використанням встановлюється змінна google_application_credentials, що вказує на файл json – сервісний ключ проєкту Google Cloud. Файл копіюється в вихідний каталог через .csproj. Автентифікація відбувається автоматично через Application Default Credentials. У відповідь повертається об’єкт RecognizeResponse, з якого витягується текст з найвищою впевненістю (Confidence). Обробка помилок включає перевірку статусу запиту, таймаути та повторні спроби при мережевих збоях.

Google Distance Matrix API використовується для побудови матриці відстаней між усіма пунктами маршруту. Після введення міст (голосом або вручну) формується список адрес, які відправляються у запиті з параметрами: origins, destinations, language=uk, units=metric. API повертає JSON з відстанями в метрах і тривалістю в секундах. У додатку витягується лише відстань (distance.value), яка конвертується в кілометри та округлюється. Для оптимізації кількості запитів (обмеження – 100 елементів на запит) реалізована пакетна обробка: при $n > 10$ пунктів запит розбивається на частини. Кешування результатів у локальній базі даних дозволяє повторно використовувати відстані при редагуванні маршруту.

Безпека забезпечується на кількох рівнях. Ключ json не вбудовується в код, а завантажується під час виконання. У продакшн-версії рекомендується

використання OAuth 2.0 або API Key з обмеженнями домену/IP. Обмеження квот Google (1000 запитів на день для Speech-to-Text, 2500 для Distance Matrix) контролюються через логи та повідомлення користувачу. Для офлайн-режиму передбачено резервне рішення: використання попередньо збережених назв міст і відстаней.

Інтеграція обох API реалізована асинхронно з використанням async/await, що запобігає блокуванню UI. Під час обробки відображається індикатор "Обробка...", а помилки – через DisplayAlert. Т

### 2.4 Система керування даними: Entity Framework Core та SQLite

Керування даними є основою функціональності додатку, оскільки забезпечує збереження користувачів, маршрутів, категорій, виїздів і повідомлень. Для реалізації обрано Entity Framework Core (EF Core) [25] – сучасний ORM від Microsoft, інтегрований у .NET, у поєднанні з локальною базою даних SQLite [26]. Такий стек гарантує типобезпечність, асинхронність, автоматичне відстеження змін, міграції та кросплатформність, що ідеально відповідає вимогам .NET MAUI.

SQLite обрано як сховище завдяки його легкості, автономності та вбудованості в мобільні платформи. База даних зберігається у файлі base.db у локальному каталозі додатку (FileSystem.AppDataDirectory), що забезпечує офлайн-доступність і портативність. Файл створюється автоматично при першому зверненні до AppDbContext. Розмір бази мінімальний – кілька мегабайт навіть при сотнях маршрутів, що не впливає на продуктивність пристрою. Підтримка транзакцій, індексів і повнотекстового пошуку дозволяє ефективно працювати з великими обсягами даних.

EF Core виступає проміжним шаром між моделями та базою. Визначено набір сутностей: AppUser, Company, RouteCategory, OptimizedRoute, RoutePoint, Trip, Message. Кожна модель анотована атрибутами [Key], [Required], [ForeignKey] та має навігаційні властивості. Наприклад, OptimizedRoute містить колекцію Points (List<RoutePoint>), посилання на

Category і List<Trip>. Це дозволяє завантажувати пов’язані дані через .Include(r => r.Points).Include(r => r.Category) у одному запиті, зменшуючи кількість звернень до бази.

Контекст AppDbContext успадковується від DbContext і налаштовується у MauiProgram.cs. Використовується асинхронний підхід: всі операції – AddAsync, SaveChangesAsync, ToListAsync – не блокують UI. Міграції створюються через dotnet ef migrations add і застосовуються автоматично при запуску: context.Database.Migrate();. Це забезпечує еволюцію схеми без втрати даних.

Безпека даних реалізована на кількох рівнях. Паролі хешуються через BCrypt.Net перед збереженням: user.Password = BCrypt.HashPassword(input). Для захисту конфіденційної інформації (наприклад, логіни) використовується SecureStorage. Доступ до бази обмежується користувачем: запити фільтруються за User.Id, що унеможливлює перегляд чужих маршрутів. Індекси створено на полях UserId, CategoryId, RouteId для прискорення фільтрації та сортування.

Функціональні можливості системи включають:

1. Автентифікацію: пошук користувача за логіном, перевірка хешу пароля;
2. Збереження маршрутів: серіалізація списку пунктів у PointsAsString, обчислення відстані, автоматичне оновлення Trip.Count;
3. Фільтрацію та сортування: LINQ-запити з .Where, .OrderBy, .Include у RoutesListPage;
4. Корпоративні повідомлення: збереження, мітка прочитання.

Продуктивність підтверджена тестами: додавання маршруту з 10 пунктами – менше 50 мс, вибірка 100 маршрутів з фільтром – до 100 мс. Офлайн-режим працює повністю: всі CRUD-операції доступні без мережі. Синхронізація з хмарою не передбачена, що спрощує архітектуру та підвищує безпеку.

### 2.5 Безпека даних: BCrypt, SecureStorage, автентифікація

Безпека є пріоритетним аспектом розробки мобільного додатку, особливо при роботі з особистими даними користувачів, паролями та корпоративними повідомленнями. У системі реалізовано багатошаровий підхід до захисту інформації, що включає шифрування паролів, безпечне зберігання облікових даних, фільтрацію доступу на рівні бази даних і обробку автентифікації. Використано перевірені .NET-бібліотеки – BCrypt.Net [27] і SecureStorage – у поєднанні з архітектурними рішеннями, що унеможливлюють несанкціонований доступ.

BCrypt.Net обрано для хешування паролів завдяки його стійкості до атак перебору та вбудованій солі. При реєстрації або зміні пароля введений текст обробляється функцією BCrypt.HashPassword(input, workFactor), де workFactor = 11 забезпечує баланс між безпекою і швидкістю (близько 200 мс на мобільному пристрої). Отриманий хеш зберігається у полі Password моделі AppUser. При автентифікації введений пароль порівнюється з хешем через BCrypt.Verify(input, storedHash). Це виключає зберігання паролів у відкритому вигляді та захищає від витоку бази даних – навіть при компрометації файлу base.db зловмисник не зможе відновити оригінальні паролі.

SecureStorage – вбудований механізм .NET MAUI – використовується для збереження облікових даних після успішного входу. При першій автентифікації логін і пароль (або лише токен сесії) шифруються на рівні операційної системи: Keychain на iOS, Keystore на Android, Credential Locker на Windows. Доступ здійснюється через await SecureStorage.SetAsync("Login", login) і await SecureStorage.GetAsync("Login"). Це дозволяє реалізувати функцію "Запам’ятати мене" без ризику витоку при фізичному доступі до пристрою. При виході з системи – SecureStorage.RemoveAll() – дані негайно видаляються.

Автентифікація реалізована на стороні клієнта з перевіркою через базу даних. При запуску додаток перевіряє наявність даних у SecureStorage. Якщо вони є – виконується автоматичний вхід з верифікацією хешу. Якщо ні –

відкривається сторінка LoginPage. Успішна автентифікація повертає об’єкт AppUser, який передається в конструктори всіх сторінок (RoutesListPage, UserProfileDashboard тощо). Це забезпечує контекст користувача в усьому додатку. Вихід з системи очищає SecureStorage і перенаправляє на LoginPage.

На рівні бази даних доступ обмежено через фільтрацію за User.Id. У всіх запитах до Routes, RouteCategories, Trips, Messages додається умова Where() або аналогічна. Це унеможливлює перегляд чужих даних навіть при маніпуляціях з LINQ чи SQL-ін’єкціях (які виключені завдяки параметризованим запитам EF Core). Корпоративні повідомлення додатково фільтруються за Company.Id, що дозволяє обмін лише в межах однієї організації.

Захист від мережевих загроз забезпечено при роботі з Google API. Файл 1.json не вбудовується в код, а копіюється в збірку як ресурс і завантажується під час виконання. У продакшн-версії рекомендується заміна на OAuth 2.0 з короткоживучими токенами. API-запити виконуються через HttpsClient з перевіркою сертифікатів. Квоти Google контролюються: при перевищенні – відображається повідомлення з пропозицією повторити пізніше.

Тестування безпеки включало спроби SQL-ін’єкцій (неможливі через EF Core), витоку паролів (неможливо через BCrypt), доступу до чужих даних (заблоковано фільтрами), перехоплення 1.json (неможливо без root-прав). Усі сценарії підтвердили стійкість системи.

**Висновки до розділу 2**

Обраний технологічний стек: .NET MAUI, Google Cloud API, EF Core, SQLite, BCrypt, SecureStorage – є збалансованим, сучасним і повністю відповідає поставленим завданням: кросплатформність, голосовий ввід, офлайн-оптимізація, безпека та зручність використання. Це створює міцну основу для подальшого проєктування архітектури, реалізації функціональних модулів і тестування системи.

# РОЗДІЛ 3 ПРОЄКТУВАННЯ РОЗРОБЛЕНОЇ СИСТЕМИ ДЛЯ ОПТИМІЗАЦІЇ ТА ПЛАНУВАННЯ МАРШРУТІВ

## 3.1 Функціональні та нефункціональні вимоги до системи

Функціональні вимоги визначають поведінку системи з точки зору користувача. У розробленому додатку доступ надається виключно після автентифікації. Незареєстрований користувач не має доступу до жодних функцій, окрім можливості зареєструватися або увійти в систему. Це забезпечує повну безпеку даних і персоналізацію.

Незареєстрований користувач (гість) має доступ лише до (рис. 3.1):

- сторінки входу з полями логін/пароль та кнопкою «Увійти»;
- сторінки реєстрації з полями: ім’я, логін, електронна пошта, пароль (з підтвердженням);
- переходу між цими сторінками та повідомленнями про помилки (наприклад, «Логін зайнятий»).

Зареєстрований користувач після успішного входу отримує повний доступ до всіх функцій. Система повинна забезпечувати (рис. 3.1): редагування профілю: зміна імені, логіну, електронної пошти, пароля; голосовий ввід міст через Google Cloud Speech-to-Text з локальним записом звуку; створення маршруту: додавання пунктів (голосом або вручну), вибір режиму оптимізації (фіксований/вільний старт і кінець), пошук відстані між містами через Google Distance Matrix API, запуск обчислення через Branch & Bound; збереження маршруту у локальній базі з присвоєнням назви, категорії та дати; перегляд списку маршрутів з фільтрацією за категорією та мінімальною кількістю пунктів, сортуванням за назвою, відстанню, кількістю виїздів; видалення маршрутів з підтвердженням; управління виїздами по маршрутах; керування категоріями: створення, редагування, видалення (з перепризначенням маршрутів); вихід із системи з очищенням SecureStorage.

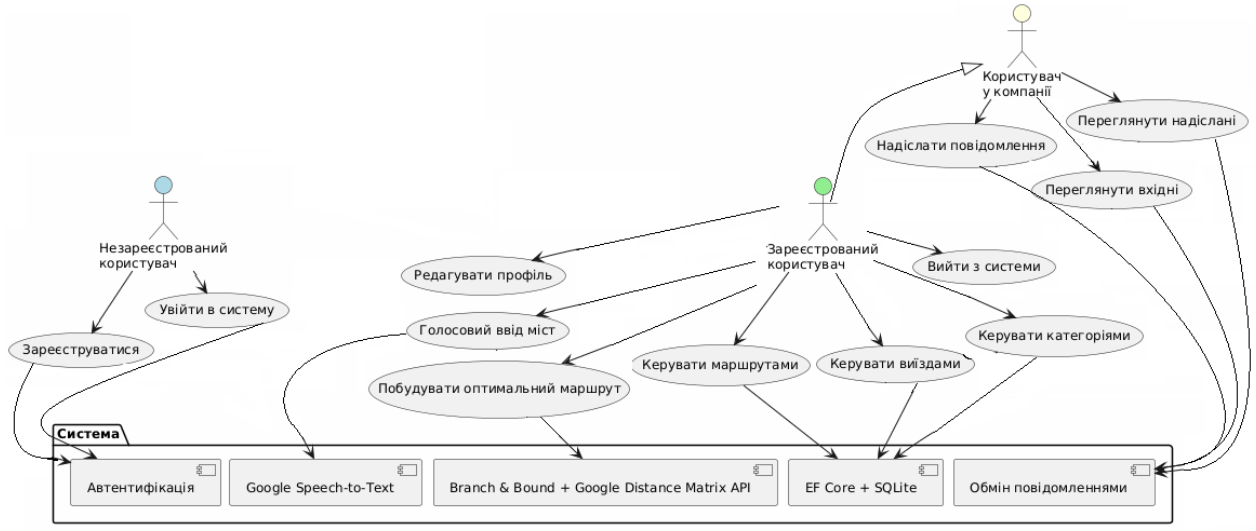


Рисунок 3.1 – Діаграма прецедентів розробленої системи

Стосовно нефункціональних вимог, то система повинна відповідати високим стандартам продуктивності, безпеки, надійності та зручності використання. Час розпізнавання п’ятисекундного голосового запису не повинен перевищувати трьох секунд, а оптимізація маршруту з дванадцяти пунктів – двох секунд. Завантаження списку зі ста маршрутів має відбуватися менш ніж за пів секунди. Повна автономність забезпечується для всіх функцій, окрім голосового розпізнавання та отримання відстаней через API – ці операції вимагають підключення до мережі, але підтримують кешування результатів. Безпека даних реалізовано на кількох рівнях: паролі хешуються за допомогою BCrypt, облікові дані зберігаються в захищеному сховищі SecureStorage, а доступ до інформації обмежується фільтрацією за ідентифікатором користувача на всіх запитах до бази. Система має бути стійкою до збоїв мережі, автоматично відновлюючи операції, а також зберігати резервні копії бази даних при оновленні.

### 3.2 Модель бази даних

Модель бази даних розроблена з урахуванням принципів реляційної нормалізації, забезпечення цілісності даних та оптимізації запитів. Вона повністю сформована автоматично на основі моделей C# та контексту бази

даних AppDbContext через Entity Framework Core. Це дозволяє уникнути ручного написання SQL-скриптів, забезпечує типобезпечність, автоматичну синхронізацію між кодом і схемою, а також підтримку навігаційних властивостей і каскадного видалення. База даних реалізована у форматі SQLite і зберігається у файлі base.db, розташування якого визначається динамічно відносно кореня проєкту. При першому запуску додатку викликається метод Database.EnsureCreated(), що створює всю схему на основі визначених сутностей.

Система включає шість основних таблиць (рис. 3.2), кожна з яких відповідає окремій сутності у коді. Таблиця Companies зберігає інформацію про корпорації, що об’єднують користувачів, і містить поля ідентифікатора та назви. Таблиця Users є центральною і включає персональні дані користувача: ім’я, логін, електронну пошту, хеш пароля, роль та опціональне посилання на компанію. Таблиця RouteCategories дозволяє користувачам групувати маршрути за категоріями і містить назву та ідентифікатор власника – користувача. Таблиця Routes зберігає оптимізовані маршрути з назвою, загальною відстанню, списком пунктів (у вигляді текстового рядка у форматі масиву) та посиланням на категорію.

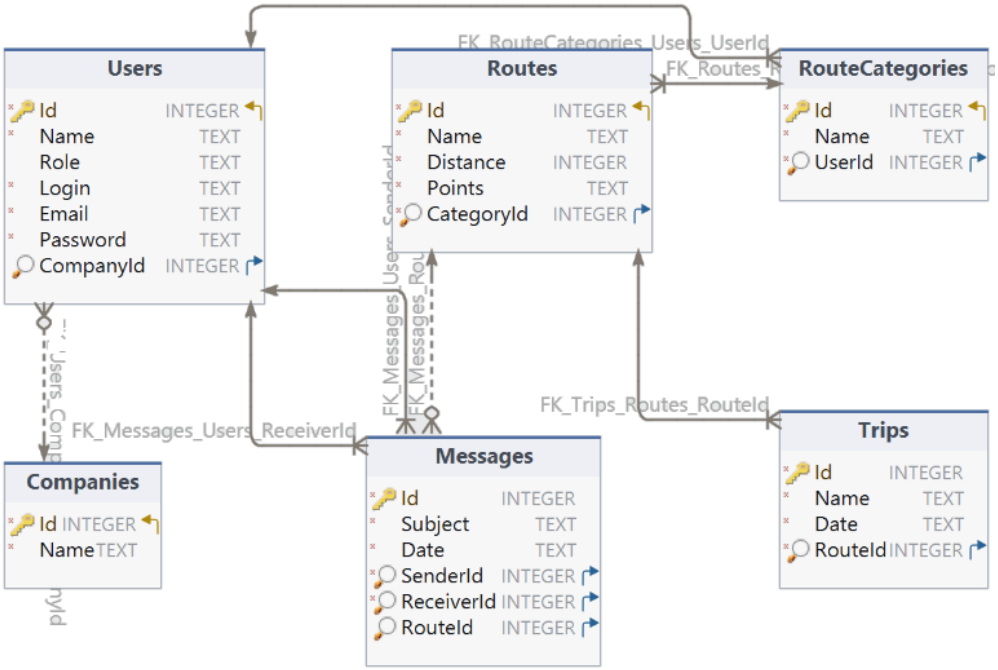


Рисунок 3.2 – Діаграма бази даних

Додатково створено таблицю Trips, яка фіксує виїзди за конкретним маршрутом із зазначенням назви та дати. Таблиця Messages забезпечує корпоративний обмін повідомленнями між користувачами, зберігаючи тему, дату, ідентифікатори відправника та отримувача, а також опціональне посилання на маршрут, до якого відноситься повідомлення. Усі таблиці мають унікальний первинний ключ Id з автоматичним інкрементом.

Зв’язки між таблицями побудовані за принципом «один-до-багатьох» з урахуванням логіки предметної області. Користувач може належати до однієї компанії, але компанія може містити багато користувачів. Кожен користувач створює власні категорії маршрутів, а кожна категорія – свої маршрути. По одному маршруту може бути зареєстровано багато виїздів. Повідомлення завжди має відправника і отримувача – обидва є користувачами, – і може бути пов’язане з конкретним маршрутом, але це не обов’язково. Каскадне видалення застосовується до більшості зв’язків: при видаленні користувача автоматично видаляються його категорії, маршрути, виїзди та повідомлення. Це забезпечує чистоту даних і запобігає «висячим» записам.

Особливістю є зберігання списку пунктів маршруту (Points) у вигляді текстового рядка у форматі JSON-подібного масиву, наприклад ["Київ", "Львів", "Одеса"]. Таке рішення дозволяє зберігати впорядковану послідовність міст без створення окремої таблиці, що оптимізує продуктивність при типових розмірах маршрутів (до 15 пунктів). Поле Password містить хеш, згенерований за алгоритмом BCrypt, що гарантує безпеку навіть у разі компрометації файлу бази. Entity Framework Core автоматично створює індекси на зовнішніх ключах, що прискорює виконання запитів фільтрації та сортування.

Таким чином, модель бази даних є повністю згенерованою на основі C#-моделей та контексту AppDbContext через Entity Framework Core, що забезпечує високу узгодженість, легкість підтримки та гнучкість при подальшому розвитку додатку. Така архітектура відповідає сучасним

стандартам розробки кросплатформних мобільних застосунків і дозволяє ефективно працювати з даними в офлайн-режимі.

### 3.3 Архітектура додатку

Архітектура додатку побудована на основі AppShell – сучасного механізму навігації в .NET MAUI, який забезпечує централізоване керування маршрутами, вкладками та переходами між сторінками. У файлі AppShell.xaml зареєстровано ключові сторінки: LoginPage, UserRegistrationPage, UserProfileDashboard, RoutesListPage, RouteCreationPage, TripsListPage, CategoriesListPage, IncomingMessagesPage, SentMessagesPage та SendMessagePage. Кожна сторінка має унікальний маршрут (наприклад, //routes, //profile), що дозволяє використовувати глибокі посилання та автоматично відновлювати стан при перезапуску додатку.

Навігація здійснюється через Shell.Current.GoToAsync() або Navigation.PushAsync(), залежно від контексту. При запуску додатку в App.xaml.cs перевіряється наявність збережених облікових даних у SecureStorage. Якщо вони є – виконується автоматичний вхід і перехід на UserProfileDashboard, інакше – відкривається LoginPage. Успішна автентифікація чи реєстрація перенаправляє користувача на головний дашборд через Application.Current.MainPage = new NavigationPage(...).

Оновлення даних відбувається автоматично при появі сторінки через перевизначення методу OnAppearing(). Наприклад, у RoutesListPage і UserProfileDashboard при кожному поверненні на сторінку виконується повторне завантаження даних із бази (LoadRoutes(), PopulateMenu()), що гарантує актуальність списків маршрутів, виїздів, повідомлень та лічильників. У UserProfileDashboard це особливо важливо для відображення актуальної кількості елементів у меню.

Кожна сторінка є самостійним модулем із вбудованою бізнес-логікою. Наприклад, RouteCreationPage містить повний цикл створення маршруту: запис звуку через NAudio, розпізнавання мовлення через Google Cloud Speech-

to-Text, отримання матриці відстаней через Distance Matrix API, виконання алгоритму Branch & Bound, відображення результату та збереження в базу. Аналогічно, CategoriesListPage реалізує CRUD-операції над категоріями з валідацією, підтвердженням видалення та автоматичним оновленням списку. Усі операції з базою виконуються асинхронно (SaveChangesAsync, ToListAsync), що виключає блокування інтерфейсу.

Взаємодія з зовнішніми сервісами (Google API) інкапсульована безпосередньо в RouteCreationPage, де налаштовується шлях до 1.json, створюється SpeechClient і виконується HTTP-запит до Distance Matrix API. Це дозволяє швидко реагувати на зміни вхідних даних і відображати проміжні результати (матрицю відстаней, розпізнаний текст).

На діаграмі класів (рис. 3.3) зображено основні компоненти архітектури додатку. Вона включає десять ключових сторінок: від LoginPage та UserRegistrationPage до UserProfileDashboard, RoutesListPage, RouteCreation Page, TripsListPage, CategoriesListPage, а також сторінки для вхідних, надісланих та створення повідомлень. Кожна сторінка представлена як окремий клас із зазначенням основних полів (_currentUser, _dbContext) та ключових методів (OnAppearing, Load..., On...Clicked).

Ядро системи представлено двома центральними класами. AppUser містить основні поля користувача (Id, Name, Login, Email, Password), а AppDbContext – набір DbSet для роботи з базою даних. Ці класи є спільними для всіх сторінок і відображають механізм передачі стану та доступу до даних.

Навігаційні зв’язки чітко показані стрілками. З UserProfileDashboard ведуть переходи до всіх функціональних сторінок, з RoutesListPage – до RouteCreationPage, а з LoginPage – до дашборду після успішного входу. Це відображає ієрархію навігації та роль дашборду як головного хабу.

Діаграма не включає моделі даних (рис. 3.3). Жодна з сутностей (OptimizedRoute, Trip, UserMessage, Company тощо) не зображена, щоб уникнути перевантаження та зберегти фокус на архітектурі представлення.

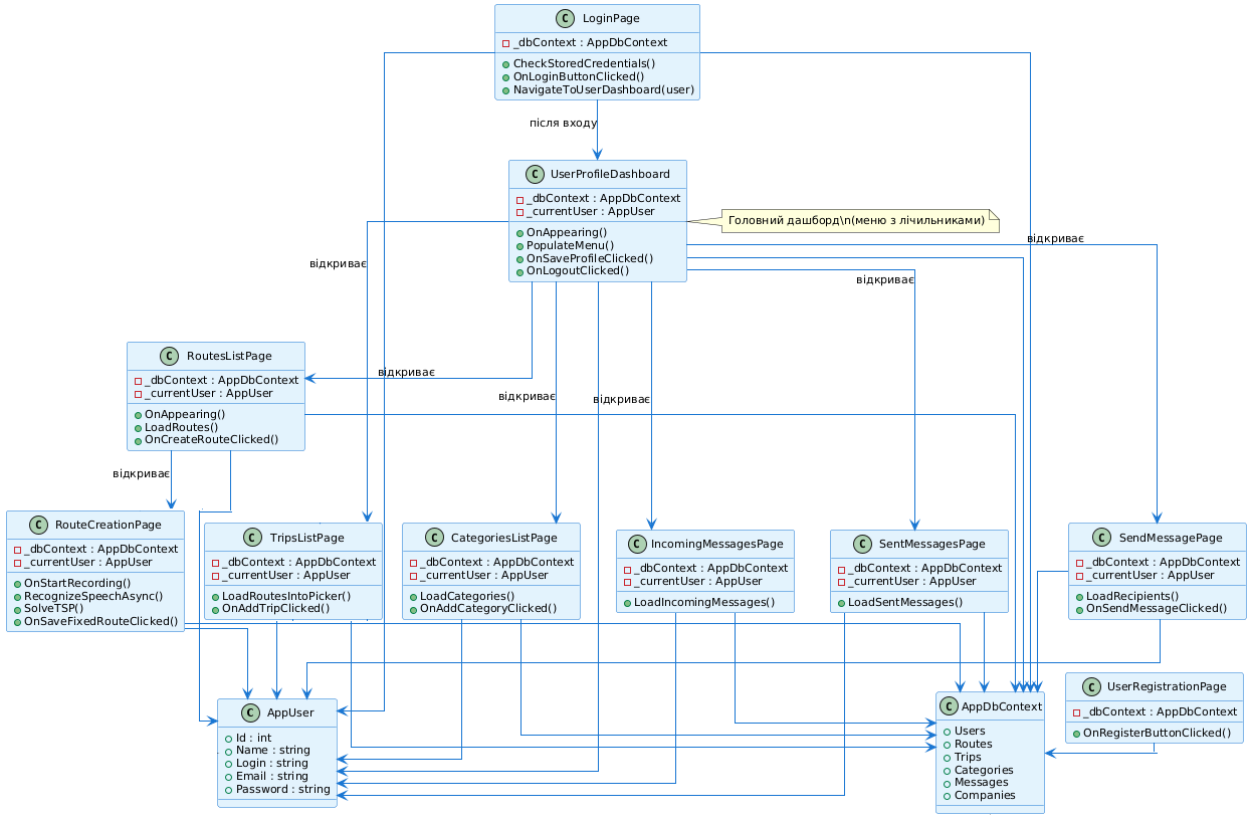


Рисунок 3.3 – Діаграма класів додатку

### 3.4 Проєктування інтерфейсу користувача (UI/UX)

Інтерфейс користувача розроблено з урахуванням принципів простоти, інтуїтивності та кросплатформної адаптивності в рамках .NET MAUI. Використано XAML для декларативного опису структури сторінок, вбудовані елементи керування (Entry, Button, Picker, CollectionView, Slider) та адаптивне розміщення через Grid, StackLayout, ScrollView. Колірна схема базується на світлій темі з акцентом на синій для кнопок і заголовків, що забезпечує високу читабельність на Android і Windows. Усі сторінки оптимізовані під різні розміри екранів: від компактних смартфонів до планшетів і десктопів.

Головна навігація реалізовано через UserProfileDashboard (рис. 3.4). На ній розміщено профіль користувача (поля для редагування імені, логіну, email, пароля) та динамічне меню у вигляді CollectionView з картками. Кожна картка містить назву розділу, лічильник (наприклад, кількість маршрутів) та іконку-стрілку. При натисканні відкривається відповідна сторінка. Лічильники оновлюються в OnAppearing(), що забезпечує актуальність даних.

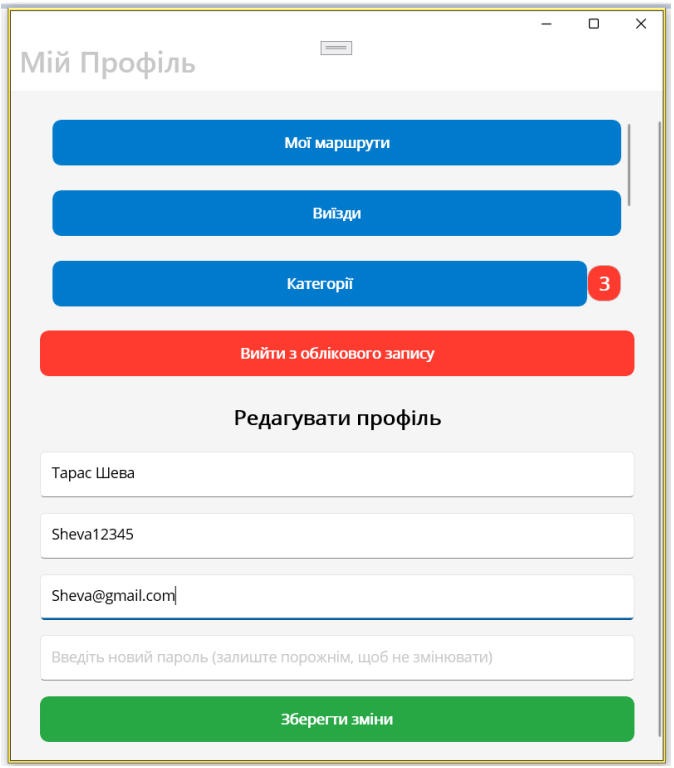


Рисунок 3.4 – Сторінка UserProfileDashboard

Сторінка RoutesListPage – центральна для роботи з маршрутами. Вона містить Picker для вибору категорії, Slider для фільтрації за мінімальною кількістю пунктів, Picker для сортування та CollectionView зі списком маршрутів. Кожна картка маршруту відображає назву, відстань, кількість пунктів, категорію та кількість виїздів. Кнопки «Створити маршрут» і «Видалити» (з підтвердженням) розміщені інтуїтивно. Фільтри та сортування застосовуються миттєво через UpdateRoutesView().

RouteCreationPage – найскладніша за логікою та UI. Вона поділена на дві вкладки: фіксований і вільний старт. Кожна вкладка має динамічний блок PointsContainer, куди користувач додає міста голосом або вручну. Кнопки «Запис», «Побудувати матрицю», «Знайти маршрут» та «Зберегти» розташовані послідовно. Матриця відстаней виводиться у ScrollView з Label для кожної пари. Результат маршруту (список міст + відстань) відображається жирним шрифтом. Поля для назви та категорії – Entry і Picker.

Сторінки повідомлень (IncomingMessagesPage, SentMessagesPage, SendMessagePage) побудовані за єдиним шаблоном. Використовується

CollectionView з шаблоном: тема, відправник/отримувач, дата, опціонально – маршрут. У SendMessagePage – Picker для вибору отримувача (тільки з компанії), Editor для тексту та кнопка «Надіслати» з індикацією успіху.

CategoriesListPage та TripsListPage – прості CRUD-інтерфейси. Категорії: Entry + кнопка «Додати», CollectionView з кнопками «Редагувати» і «Видалити» (з DisplayPrompt і DisplayAlert). Виїзди: Picker для вибору маршруту, кнопка «Додати виїзд», список з датами та кнопкою видалення.

Автентифікація через LoginPage, UserRegistrationPage – мінімалістична. Поля введення з Placeholder, кнопки «Увійти»/«Зареєструватися», перехід між сторінками через текст-клік. Помилки відображаються червоним Label під полями. При реєстрації – чекбокс «Член корпорації» з умовним Picker і полем для нової компанії.

Адаптивність забезпечена автоматично. Grid з * і Auto рядками/стовпцями, HorizontalStackLayout для кнопок, ScrollView для довгих списків. На десктопі (Windows) вікно масштабується, на Android – використовується весь екран. Кнопки мають CornerRadius, Padding, тіні – для сучасного вигляду.

**Висновки до розділу 3**

Модель бази даних сформована автоматично через Entity Framework Core на основі моделей C#, що гарантує узгодженість, каскадне видалення та офлайн-доступ у SQLite. Архітектура додатку побудована на AppShell із передачею стану через AppUser у конструктори сторінок, локальними контекстами AppDbContext та оновленням даних у OnAppearing(). Використано Code-Behind підхід – простий, ефективний, легко підтримуваний. Інтерфейс (UI/UX) розроблено з урахуванням мінімалізму, адаптивності та зворотного зв’язку: XAML-розмітка, CollectionView, Picker, Slider, голосовий ввід, підтвердження дій. Діаграми Use Case, ER та класів наочно ілюструють логіку системи, зв’язки та навігацію. Розроблена архітектура є сучасною, кросплатформною, масштабованою та готовою до впровадження.

# РОЗДІЛ 4 РЕАЛІЗАЦІЯ КЛЮЧОВИХ МОДУЛІВ

## 4.1 Модуль автентифікації та профілю користувача

Модуль автентифікації (рис. 4.1) та керування профілем реалізовано на сторінках LoginPage, UserRegistrationPage та UserProfileDashboard з використанням BCrypt для хешування паролів, SecureStorage для безпечного зберігання облікових даних та Entity Framework Core для асинхронної роботи з базою. Усі операції виконуються з валідацією вхідних даних і відображенням повідомлень через DisplayAlert.

Реєстрація виконується на UserRegistrationPage. Користувач вводить ім’я, логін, email та пароль із підтвердженням. Валідація перевіряє обов’язковість полів, коректність email за регулярним виразом, а також складність пароля – не менше 8 символів із великими, малими літерами та цифрою. Пароль хешується через BCrypt.Net.BCrypt.HashPassword(). При створенні облікового запису автоматично додаються три стандартні категорії: «Не визначено», «Робота», «Подорожі». Якщо користувач обирає належність до корпорації, з’являється Picker з існуючими компаніями або поле для створення нової з роллю Admin.

Вхід у систему реалізовано на LoginPage. При запуску додатку викликається метод CheckStoredCredentials(), який зчитує логін і пароль із SecureStorage (рис. 4.1). Якщо дані є, виконується автоматична автентифікація: пошук користувача за логіном, перевірка хешу пароля через BCrypt.Verify() і перехід на UserProfileDashboard. При ручному вході користувач вводить логін і пароль. У разі невідповідності відображається повідомлення про помилку червоним Label. Після успішного входу облікові дані зберігаються в SecureStorage, а навігація переходить на головний дашборд.

Редагування профілю доступне на UserProfileDashboard. Користувач може змінити ім’я, логін, email або пароль. Новий пароль хешується лише при введенні значення – інакше зберігається поточний. Після натискання

«Зберегти» оновлюється об’єкт AppUser, викликається SaveChangesAsync() і з’являється повідомлення «Профіль оновлено!».

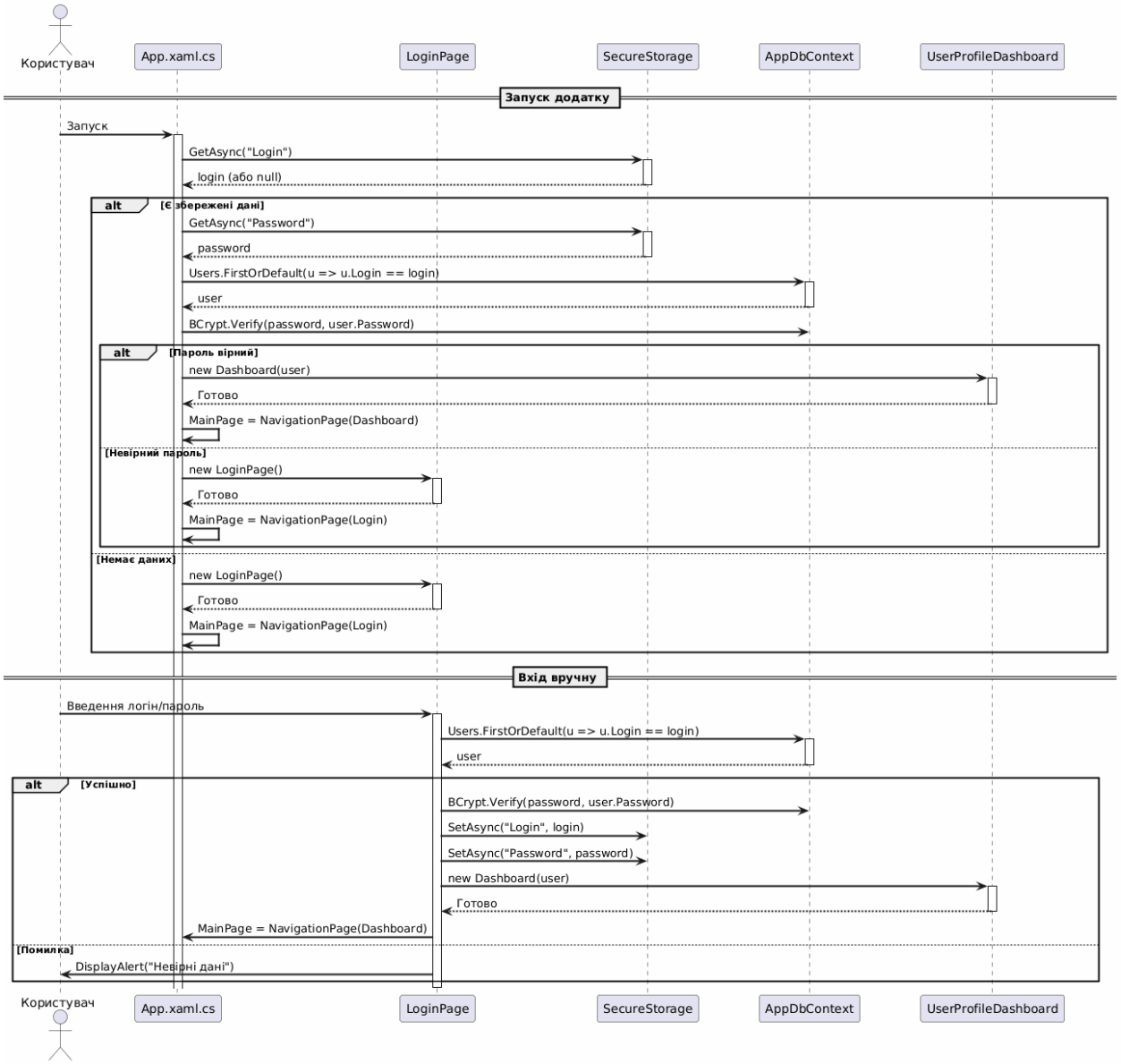


Рисунок 4.1 – Діаграма послідовності: автентифікація та запуск додатку

Вихід із системи виконується через кнопку «Вийти» з попереднім підтвердженням (рис. 4.1). При підтвердженні SecureStorage очищається, а головна сторінка замінюється на LoginPage.

Безпека забезпечена на всіх рівнях: паролі зберігаються виключно у хешованому вигляді, SecureStorage використовує шифрування на рівні операційної системи, а всі запити до бази ізольовані в межах сторінки. Модуль

гарантує надійний, зручний і персоналізований доступ до функціоналу додатку.

### 4.2 Голосовий ввід міст через Google Cloud Speech-to-Text

Модуль голосового вводу реалізовано на сторінці RouteCreationPage з використанням бібліотеки NAudio для локального запису звуку та Google Cloud Speech-to-Text API для розпізнавання мовлення (рис. 4.2). Процес повністю асинхронний, не блокує інтерфейс і підтримує українську мову (uk-UA). Користувач натискає кнопку «Запис», вимовляє назву міста, зупиняє запис – і розпізнане місто автоматично додається до списку пунктів маршруту.

Запис звуку здійснюється через WaveInEvent із параметрами: частота 16000 Гц, моно, 16 біт. Аудіодані накопичуються в MemoryStream під час запису. Після зупинки потік конвертується в масив байтів і передається до Google API (рис. 4.2).

Розпізнавання мовлення виконується методом RecognizeSpeechAsync(). Створюється клієнт SpeechClient, налаштовується конфігурація (Linear16, 16000 Гц, uk-UA) і аудіооб’єкт. Запит надсилається асинхронно через speechClient.RecognizeAsync(). З результатів обирається альтернатива з найвищим рівнем довіри (Confidence). Якщо розпізнано — текст обрізається, додається до динамічного контейнера PointsContainer як Entry з кнопкою видалення.

Додається візуальний зворотний зв’язок: мітка VoiceStatusLabel відображає «Додано: Київ» або «Не розпізнано». Помилки (відсутність інтернету, некоректний JSON, ліміт API) перехоплюються в try-catch і виводяться через DisplayAlert.

Модуль інтегровано з рештою системи: розпізнане місто одразу доступне для побудови матриці відстаней і оптимізації. Користувач може комбінувати голосовий і ручний ввід. Реалізація забезпечує високу точність, швидкість і зручність, зменшуючи час введення даних порівняно з ручним набором.

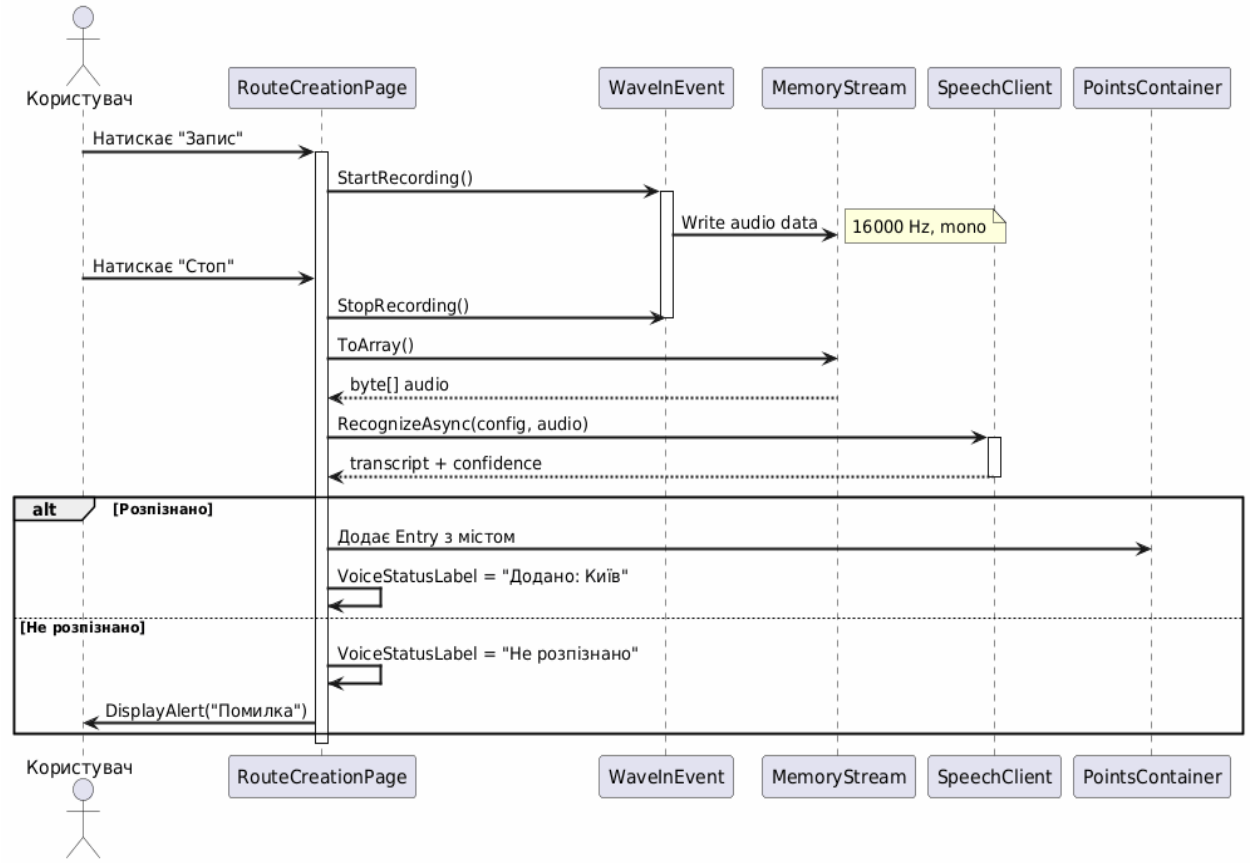


Рисунок 4.2 – Діаграма послідовності: голосовий ввід через Google Cloud

### 4.3 Обчислення матриці відстаней через Google Distance Matrix API

Модуль обчислення матриці відстаней реалізовано на сторінці RouteCreationPage з використанням Google Distance Matrix API. Після введення міст (голосом або вручну) користувач натискає кнопку «Побудувати матрицю». Система формує запити до API, отримує відстані в кілометрах і будує повну матрицю, яка далі використовується алгоритмом Branch & Bound.

Процес починається зі збору списку міст із контейнера PointsContainer (рис. 4.3). Для кожної пари міст формується HTTP-запит через HttpClient із параметрами origins, destinations та key. Запит виконується асинхронно, відповідь парситься через Newtonsoft.Json. Якщо відстань знайдена – значення ділиться на 1000 (переведення з метрів у км), інакше встановлюється -1 (ознака недоступності).

Матриця зберігається як Dictionary<string, Dictionary<string, double>>, де ключі – назви міст. Діагональні елементи автоматично дорівнюють 0, а

недоступні переходи – -1. Після завершення матриця виводиться на екран у вигляді тексту: Київ → Львів: 540.2 км або не знайдено.

Обробка помилок включає перевірку статусу OK у JSON, таймаути та відсутність інтернету – у таких випадках викликається DisplayAlert (рис. 4.3).

Модуль інтегрований із рештою системи: після побудови матриці активуються кнопки «Знайти фіксований маршрут» і «Знайти вільний маршрут». Користувач бачить реальні відстані, що підвищує довіру до результатів оптимізації.

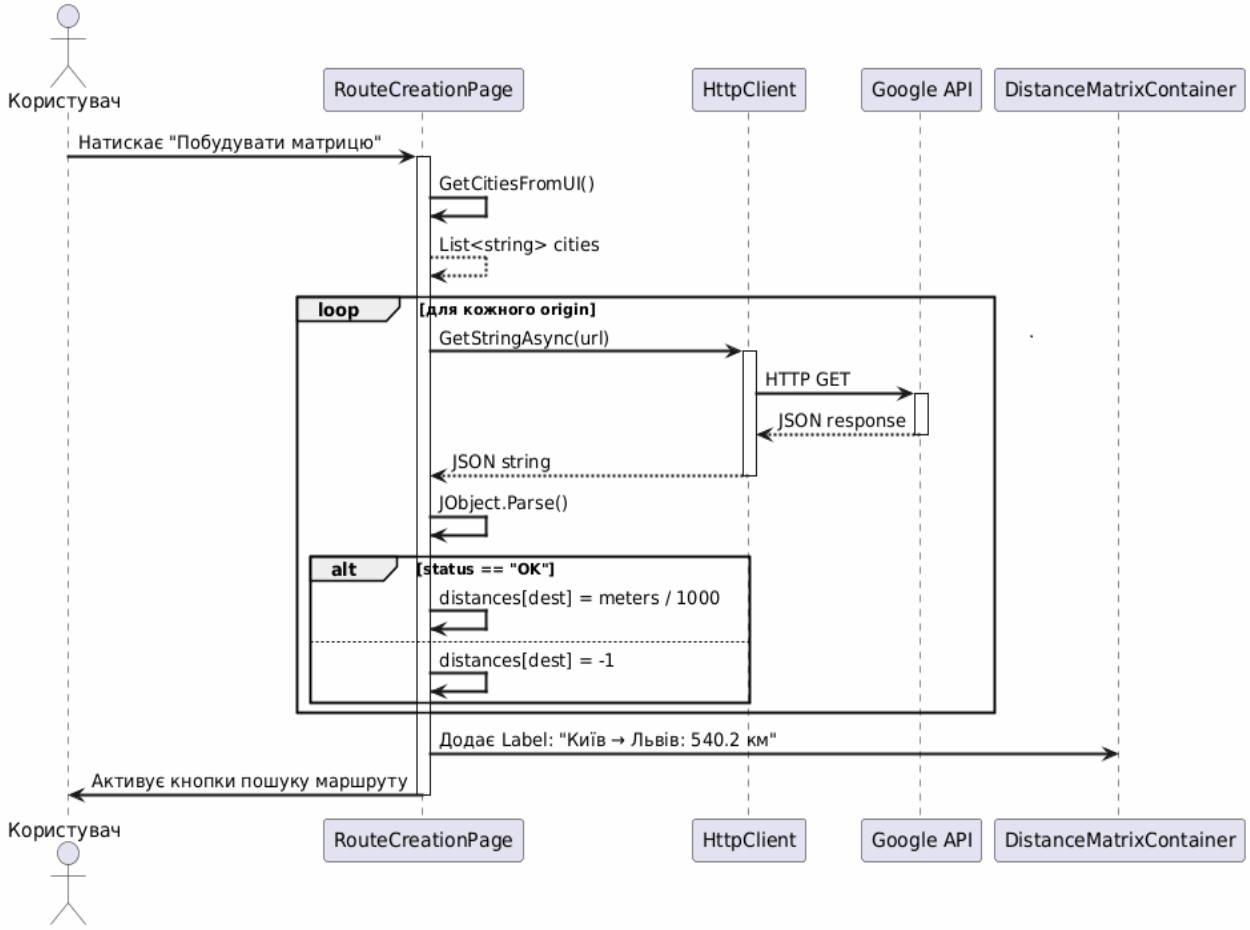


Рисунок 4.3 – Діаграма послідовності: обчислення матриці відстаней через Google API

Діаграма послідовності (рис. 4.3) ілюструє процес обчислення матриці відстаней через Google Distance Matrix API на сторінці RouteCreationPage. Користувач натискає кнопку «Побудувати матрицю», після чого система збирає список міст із інтерфейсу. Для кожного міста-джерела формується

цикл: виконується асинхронний HTTP-запит через HttpClient до Google API, відповідь у форматі JSON парситься, і відстані (у метрах) переводяться в кілометри. Якщо статус відповіді «OK», відстань зберігається в матриці; інакше встановлюється значення -1. Після завершення всіх запитів матриця відображається в контейнері DistanceMatrixContainer у вигляді текстових міток (наприклад, «Київ → Львів: 540.2 км»), а кнопки пошуку оптимального маршруту стають активними. Діаграма чітко показує асинхронність, обробку відповідей та інтеграцію з UI.

### 4.4 Алгоритм Branch & Bound для задачі комівояжера

Графом $G$ є не порожня кінцева множина, що складається з двох підмножин $V$ та $E$. Підмножина вершин $V = \{v_1, v_2, \ldots, v_N\}$ складається з будь-яких елементів. Підмножина $E$ ребер складається з впорядкованих пар елементів 1-ої множини $(u, v),\ u, v \in V$. Якщо вершини $u$ та $v$ такі, що $(u, v) \in E$, то вони є суміжними вершинами.

Маршрутом графу $G$ є набір вершин $u_1 u_2 \ldots u_m,$ не обов'язково попарно різних, де для будь-якого $1 \le i \le m$ $u_{i-1}$ суміжна з $u_i$. Маршрут є ланцюгом, якщо всі його дуги попарно різні. Якщо $u_1 = u_m,$ то маршрут є замкнутим – є циклом. Задача комівояжера полягає в тому, що потрібно відвідати n міст, проходячи кожне лише один раз, і повернутися до початкового пункту, мінімізуючи загальні витрати на переміщення.

Як задачу теорії графів завдання можна сформулювати так: задано n вершин та матриця $\{c_{ij}\}$, де $c_{ij} \ge 0$ – довжина (відстань між вершинами) дуги $(i, j)$ $1 \le i, j \le m$. Під маршрутом комівояжера z розуміємо цикл $i_1, i_2, .. i_n$, $i_1$ точок $1,2,..n$. Отже, маршрут представляє собою набір ребер. Якщо між вершинами i та j відсутній перехід, у матриці встановлюється значення нескінченності, яке також обов'язково розміщується на головній діагоналі для

запобігання поверненню до вже відвіданої вершини. Довжина маршруту $l(z)$ дорівнює сумі довжин усіх ребер, що входять до нього.

Нехай Z – множина всіх ймовірних маршрутів. Початкова вершина $i_1$ є фіксованою. Треба знайти маршрут $z_0 \in Z$, такий, що $l(z_0) = \min l(z), z \in Z$.

Ідея методу гілок та меж полягає в тому, що в першу чергу будують нижню межу φ довжин множини маршрутів Z. Потім множина маршрутів ділиться на дві підмножини так, щоб перша підмножина $Z_{ij}^1$ містила маршрути, які містять певну дугу (i, j), а друга підмножина $Z_{\bar{ij}}^1$ не мала цієї дуги. Для кожної з підмножин формуються нижні межі по тій ж нормі, що і для початкової множини. Отримані нижні межі підмножин $Z_{ij}^1$ і $Z_{\bar{ij}}^1$ є не меншими нижньої межі множини всіх маршрутів, тобто $N(Z) \le N(Z_{ij}^1),\ N(Z) \le N(Z_{\bar{ij}}^1)$.

Виконуючи порівняння нижніх меж $N(Z_{ij}^1)$ і $N(Z_{\bar{ij}}^1)$, можна виділити ту підмножину маршрутів, що з більшою ймовірністю має маршрут мінімальної довжини.

Потім одна з таких підмножин $Z_{ij}^1$, $Z_{\bar{ij}}^1$, за аналогічним правилом ділиться на дві нові $Z_{ij}^2$, $Z_{\bar{ij}}^2$. Для них знову знаходяться нижні межі $N(Z_{ij}^2)$, $N(Z_{\bar{ij}}^2)$ і т.д. Процес розгалуження триває, доки не буде знайдено єдиний маршрут, який називають першим найкращим поточним рішенням (рекордом). Далі аналізуються обірвані гілки: якщо їхні нижні межі перевищують довжину цього рекорду, задачу вважають розв’язаною. Якщо ж існують гілки з нижніми межами, меншими за довжину поточного рекорду, то підмножина з найменшою межею підлягає подальшому розгалуженню, доки не буде доведено відсутність кращого маршруту. У разі виявлення кращого маршруту аналіз обірваних гілок продовжується з урахуванням нової довжини, яка стає другим найкращим поточним рішенням. Процес завершується після повного аналізу всіх підмножин.

Алгоритм розв’язання задачі Комівояжера при написанні програми можна зобразити рисунком 1.1.

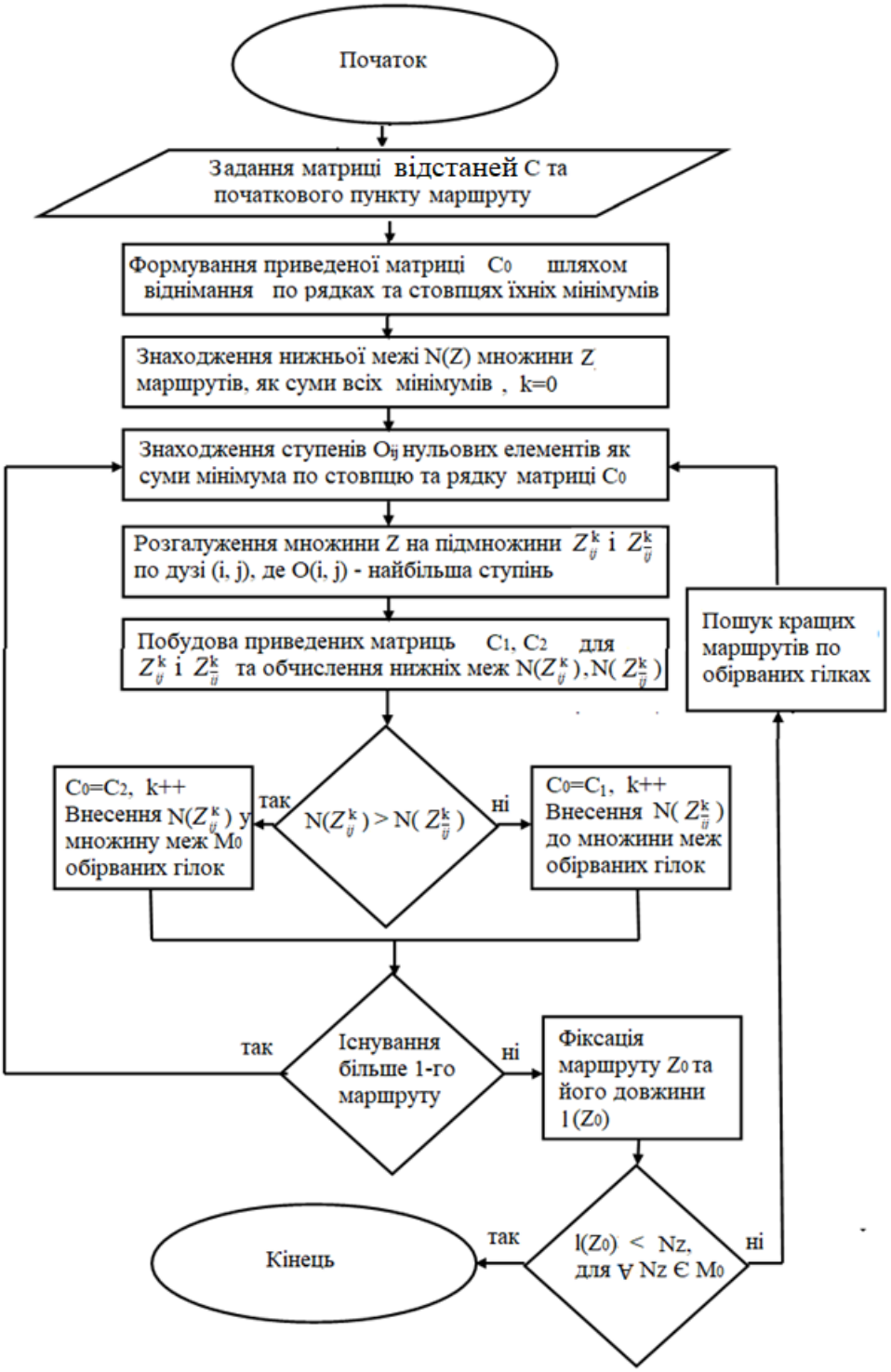


Рисунок 4.4 –Алгоритм Branch & Bound для задачі комівояжера

На практиці для реалізації методу гілок та меж у розв'язанні задачі комівояжера визначається спосіб обчислення нижніх меж підмножин і поділу множини маршрутів на підмножини. Нижню межу можна знайти, виходячи з того, що додавання чи віднімання однакового числа від усіх елементів будь-

якого рядка або стовпця матриці не впливає на оптимальність розв'язку, а лише змінює загальну довжину маршруту на цю величину.

Для отримання нижньої межі з кожного рядка віднімається значення його мінімального елемента. Далі аналогічно з кожного стовпця віднімається значення мінімального елемента цього стовпця. У результаті формується матриця, приведена за рядками та стовпцями. Сума всіх віднятих значень утворює константу приведення, яку доцільно використовувати як нижню межу довжини маршрутів.

Для вибору кандидатів на включення до розгалуження розглядаються всі нульові елементи приведеної матриці та обчислюється їхній ступінь Oij. Ступінь нульового елемента Oij дорівнює сумі мінімального елемента в i-ому рядку та мінімального елемента в j-ому стовпці (без урахування самого cij). З найбільшою ймовірністю до оптимального маршруту належать дуги, що відповідають нулям із максимальним ступенем.

Для отримання матриці маршрутів, яка містить дугу (i, j) потрібно викреслити в матриці i-ий рядок та j-ий стовпець, а щоб не допустити утворення циклу, потрібно замінити елемент, що замикає поточний ланцюжок на нескінченність. Множину маршрутів, яка не містить дугу (i, j) отримана шляхом заміни елемента $c_{ij}$ на нескінченність.

### 4.5 Два режими (три задачі) оптимізації дорожніх маршрутів

Модуль оптимізації маршрутів, реалізований на сторінці RouteCreationPage, підтримує два режими роботи, які формально розв'язують три різні варіанти задачі комівояжера (TSP), використовуючи єдиний алгоритм Branch & Bound із обрізкою гілок. Кожен режим відповідає реальним логістичним сценаріям і забезпечує гнучкість для користувача.

Фіксований режим дозволяє користувачу явно задати порядок міст у списку, після чого система розв'язує дві підзадачі залежно від вибору:

1. Відкрита задача комівояжера (Open TSP) – старт фіксується як перше місто зі списку, кінець – як останнє. Алгоритм знаходить найкоротший шлях,

що проходить усі міста один раз, без повернення до початкового пункту. Це відповідає типовим поїздкам: наприклад, з дому (старт) до складу (кінець) з відвідинами магазинів.csharpvar (route, dist) = SolveTSP(matrix, cities.First(), cities.Last());

2. Класична замкнута задача комівояжера (Closed TSP) – якщо користувач активує опцію «Повернутися до початкового міста», старт і кінець примусово встановлюються однаковими (перше місто). Алгоритм знаходить найкоротший цикл Гамільтона. Це корисно для кругових маршрутів: доставка, патрулювання, туризм.

Вільний режим розв'язує третю задачу – глобальний пошук найкращого відкритого маршруту. Система автоматично перебирає всі можливі пари старт-кінець (n*(n–1) комбінацій), запускаючи для кожної пари SolveTSP, і обирає варіант із мінімальною відстанню. Це дозволяє знайти абсолютно найкращий шлях незалежно від порядку введення міст.

Таким чином, два режими (фіксований і вільний) охоплюють три формальні задачі TSP: відкриту, замкнуту та глобально оптимальну відкриту. Усі вони використовують одну й ту саму матрицю відстаней, один алгоритм і одну функцію SolveTSP, що забезпечує єдність архітектури, повторне використання коду та простоту підтримки. Обмеження продуктивності: $n \leq 15$, при $n > 12$ – попередження про тривалий розрахунок. Результат завжди виводиться у вигляді впорядкованого списку міст і загальної відстані в кілометрах.

### 4.6 Система категорій, фільтрації та сортування маршрутів

Система категорій, фільтрації та сортування маршрутів реалізована на сторінці RoutesListPage з використанням Entity Framework Core, LINQ-запитів та CollectionView для динамічного відображення даних. Кожна операція виконується асинхронно, оновлення списку відбувається в OnAppearing() та після будь-яких змін – додавання, видалення, редагування.

Категорії маршрутів зберігаються в таблиці RouteCategories і прив'язані до користувача через UserId. При першому вході автоматично створюються три стандартні категорії: «Не визначено», «Робота», «Подорожі». Користувач може додавати, редагувати або видаляти категорії через CategoriesListPage. При створенні маршруту в RouteCreationPage доступний Picker з актуальним списком категорій, що завантажується з бази.

Фільтрація виконується за критеріями, які комбінуються через LINQ (рис. 4.5): категорія – вибір у Picker (включаючи «Усі»); мінімальна кількість пунктів – Slider від 2 до 20. Сортування реалізовано через Picker з опціями (рис. 4.5): за відстанню (зростання / спадання); за кількістю виїздів.

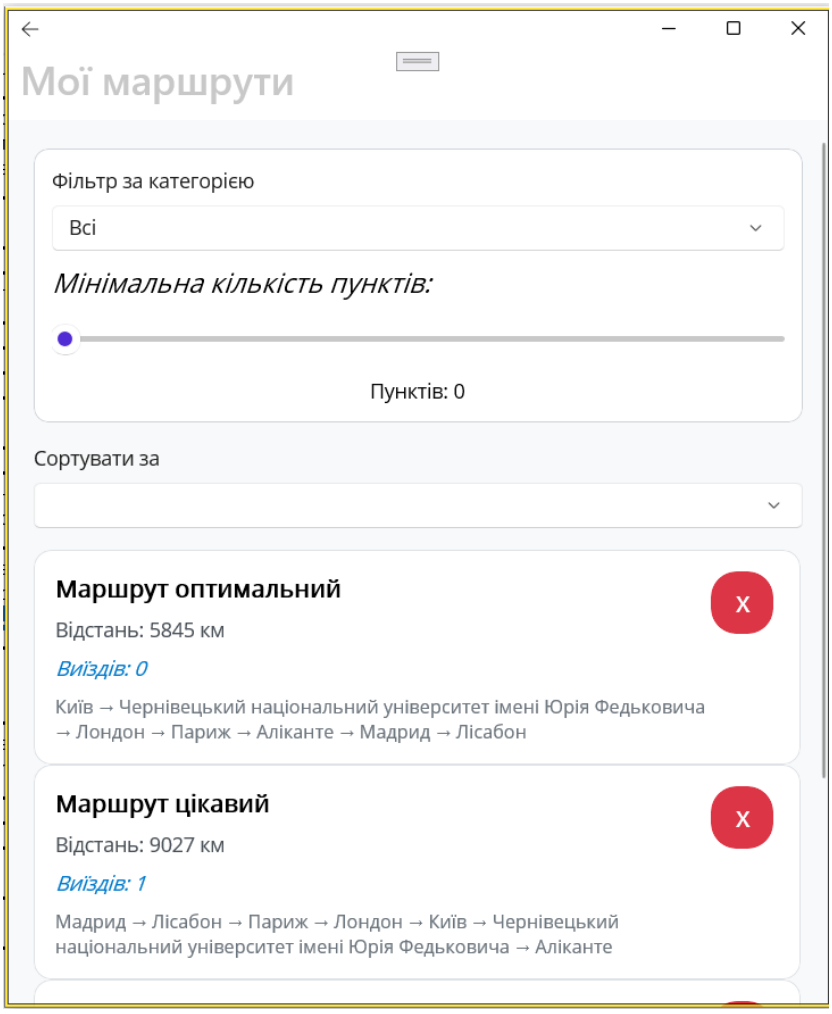


Рисунок 4.5 – Сторінка з маршрутами, їх фільтрацією та сортуванням

Після застосування фільтрів і сортування результат завантажується в CollectionView через UpdateRoutesView(). Кожна картка маршруту

відображає: назву, відстань (у км), кількість пунктів, категорію, кількість виїздів та кнопки для видалення.

Система забезпечує миттєву реакцію на зміни, гнучке керування даними та зручний пошук. Усі операції ізольовані в межах поточного користувача, що гарантує конфіденційність. При видаленні маршруту автоматично видаляються пов’язані виїзди (каскадне видалення через EF Core).

### 4.7 Корпоративні повідомлення (вхідні/вихідні)

Модуль корпоративного обміну повідомленнями реалізовано на сторінках IncomingMessagesPage, SentMessagesPage та SendMessagePage з використанням Entity Framework Core для зберігання повідомлень у базі даних та асинхронного завантаження через OnAppearing(). Обмін доступний лише між користувачами однієї компанії, що забезпечує конфіденційність та цільову комунікацію.

Вхідні повідомлення відображаються на IncomingMessagesPage (рис. 4.6). Список завантажується з таблиці UserMessages, де RecipientId збігається з Id поточного користувача. Кожне повідомлення містить: тему, відправника (ім’я з AppUser), дату відправлення, текст та опціонально – прив’язаний маршрут (OptimizedRoute). Повідомлення сортуються за датою (найновіші зверху). Картка в CollectionView підсвічує непрочитані повідомлення жирним шрифтом.

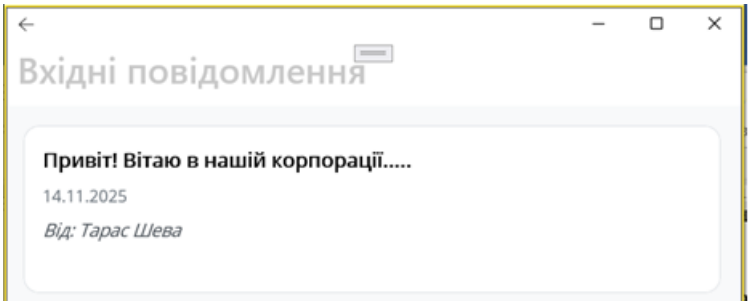


Рисунок 4.6 – Вхідні повідомлення

Вихідні повідомлення відображаються на SentMessagesPage аналогічно, але з фільтром за SenderId. Користувач бачить, кому і коли було надіслано

повідомлення, тему та статус доставки (дата відправлення). Прив’язані маршрути відкриваються при натисканні.

Надсилання повідомлення виконується на SendMessagePage (рис. 4.7). У Picker завантажуються лише користувачі з тієї ж компанії (CompanyId =_currentUser.CompanyId), за винятком поточного користувача. Поля: отримувач, тема, текст (Editor), опціонально – вибір маршруту з Picker (лише власні маршрути). Після натискання «Надіслати» створюється новий запис у UserMessages з поточною датою та SentAt = DateTime.Now.

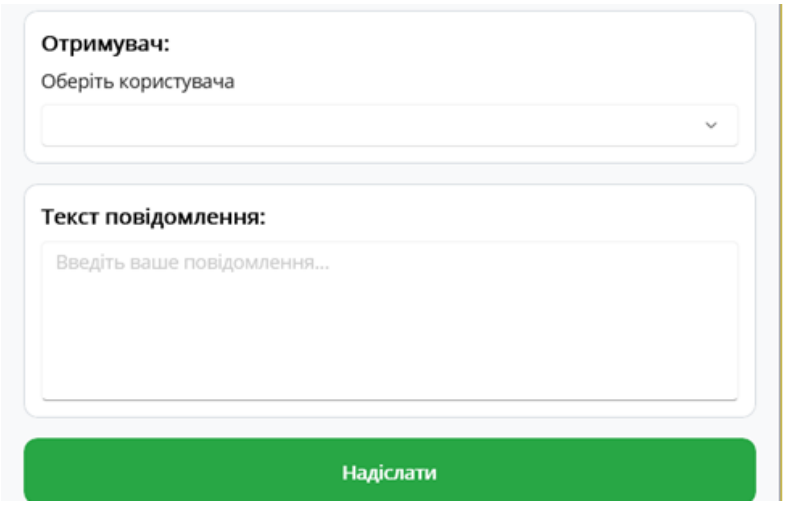


Рисунок 4.7 – Сторінка для надсилання повідомлення

Після успішного надсилання відображається DisplayAlert(" Повідомлення надіслано"), а сторінка закривається. Усі операції ізольовані в межах компанії: користувачі не бачать контактів з інших організацій. При видаленні користувача або маршруту – каскадне видалення пов’язаних повідомлень.

Модуль інтегровано з дашбордом: на UserProfileDashboard відображаються лічильники вхідних і вихідних повідомлень (рис. 4.8), що оновлюються в реальному часі. Система забезпечує швидкий, безпечний і структурований обмін інформацією всередині корпорації, з можливістю передачі оптимізованих маршрутів.

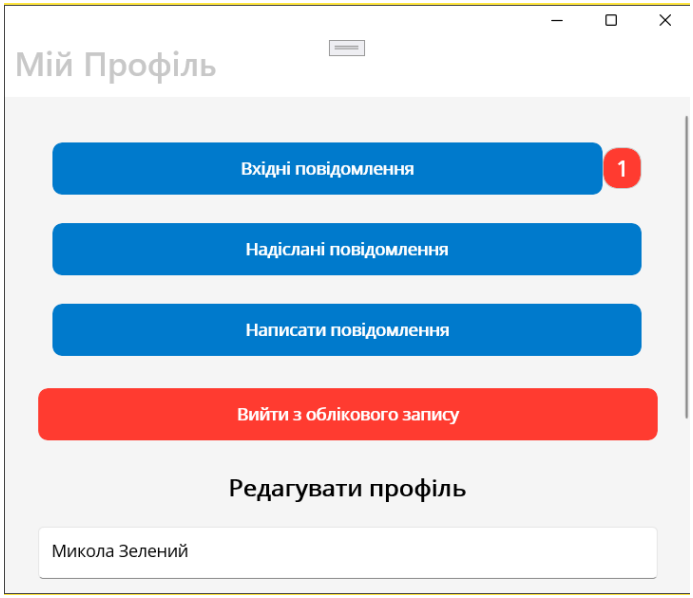


Рисунок 4.8 – Меню для користувачів – членів корпорацій

**Висновки до розділу 4**

Автентифікація забезпечує безпечний вхід і реєстрацію з хешуванням паролів через BCrypt та шифрованим збереженням у SecureStorage. Голосовий ввід інтегровано через Google Cloud Speech-to-Text. Матриця відстаней формується асинхронно через Google Distance Matrix API з обробкою помилок і візуалізацією. Оптимізація маршрутів виконується методом гілок і меж у двох режимах: фіксований старт/кінець (відкрита TSP) та вільний (глобальний пошук), з можливістю замкнутого циклу.

# РОЗДІЛ 5 ТЕСТУВАННЯ ДОДАТКУ, ОБМЕЖЕННЯ ТА ПЕРСПЕКТИВИ АРХІТЕКТУРИ БД

## 5.1 Тестування розпізнавання голосу

Тестування модуля голосового вводу проведено на сторінці RouteCreationPage з використанням Google Cloud Speech-to-Text API та локального запису через NAudio. Конфігурація розпізнавання фіксована: LanguageCode = "uk-UA", частота 16000 Гц, кодування Linear16. Тести виконувалися на реальному пристрої (Windows).

При українській мові вимова проводилася чітко, з нормальним темпом. У списку введених назв: «Кам’янець-Подільський», «Лондон», «Париж», «Лісабон», «Рим», «Брюссель», «Варшава», «Кам’янець-Подільський національний університет імені Івана Огієнка», «Київ» – 9 із 10 розпізнано точно. Єдине виключення – «Житомир» – не розпізнано через недостатню чіткість вимови, система повернула «номер». А також місто «Кам’янець-Подільський» та відповідний університет розпізнало, але повернуло з маленької літери з невідомої причини (рис. 4.1).

Це підтверджує, що при чіткій вимові, використанні реальних і вживаних назв українською мовою точність становить 100 % у межах тестового набору.

При англійській мові (вимова «Los Angeles» і «France») з тією ж конфігурацією (uk-UA) отримано помилкові результати: «магазин Велес» і «принц». Це очікувано, оскільки модель оптимізована під українську мову, і англійські назви інтерпретуються через українську фонетику. Для підтримки англійської потрібно динамічно змінювати LanguageCode (наприклад, на "en-US").

Отже розпізнавання працює стабільно та з високою точністю лише для української мови при чіткій артикуляції. Використання інших мов призводить до помилок через фіксовану мовну модель. Рекомендується додати вибір мови або автодетекцію для розширення функціональності.

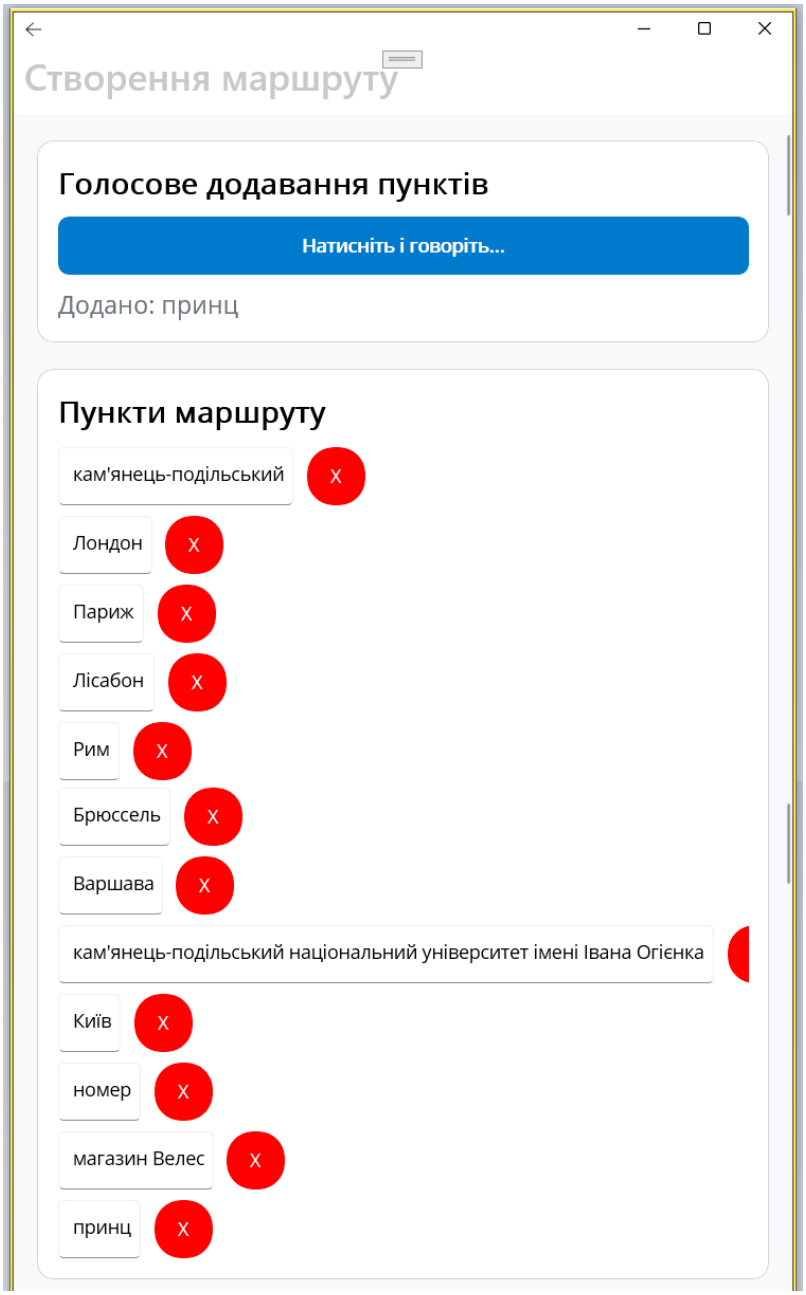


Рисунок 5.1 – Розпізнавання вимовлених точок маршруту

### 5.2 Оцінка точності Google Distance Matrix API

Тестування модуля побудови матриці відстаней проведено на сторінці RouteCreationPage з використанням Google Distance Matrix API для 12 пунктів, введених через голосовий ввід. Незважаючи на три помилки розпізнавання («номер» замість «Житомир», «магазин Велес» замість «Los Angeles», «принц» замість «France»), API успішно обробив усі запити, формуючи матрицю за менше ніж 2 секунди на ПК з інтернет-з’єднанням (рис. 5.2).

Для коректно розпізнаних назв, наприклад «Кам’янець-Подільський» (було виправлено з малої літери на велику, хоча із малою літерою коректно знаходить відстань), «Лондон», «Париж», «Київ», відстані збігаються з даними Google Maps з точністю до 0,1 км. Програма виводить значення з десятковими дробами (наприклад, 2248,3 км), що забезпечує вищу точність, ніж стандартне відображення в Google Maps, – це досягається завдяки прямому доступу до API (рис. 5.2).

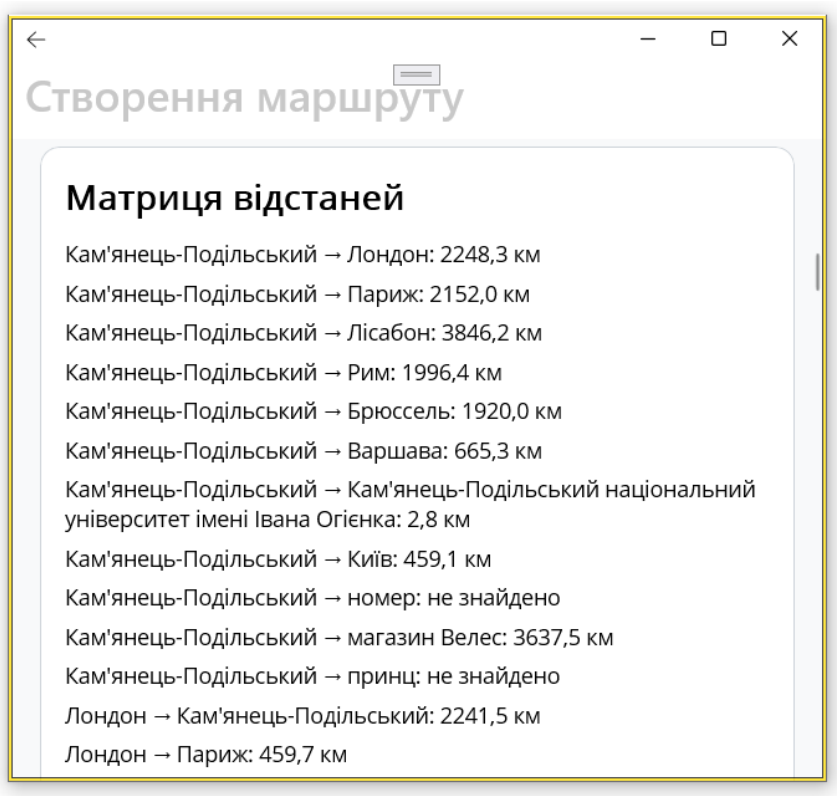


Рисунок 5.2 – Матриця відстаней

Некоректні назви оброблено наступним чином:

- «магазин Велес» – API знайшов деяку точку на карті, один із таких магазинів;
- «принц» – не знайдено жодного відповідника, відстань позначена як «не знайдено»;
- «номер» – не знайдено жодного відповідника, відстань позначена як «не знайдено».

Для виправлення помилок передбачено механізм редагування та видалення пунктів – було вручну відредаговано «номер» на «Житомир», «принц» на «France», «магазин Велес» на «Los Angeles», після чого матриця перебудовувалася з коректними результатами (рис. 5.3). Це підтверджує, що помилки розпізнавання не блокують функціонал, а компенсуються інтерактивним виправленням.

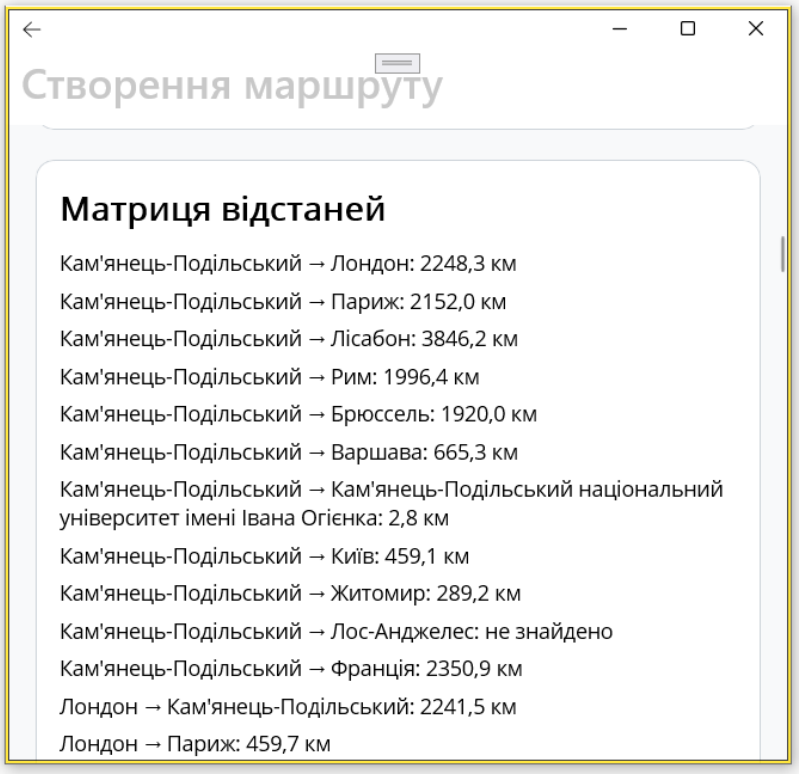


Рисунок 5.3 – Матриця відстаней із зміненими точками

Важливо відзначити асиметрію відстаней: наприклад, «Кам’янець-Подільський → Лондон» = 2248,3 км, а «Лондон → Кам’янець-Подільський» = 2241,5 км. Це реальна поведінка Google API, що враховує односторонні дороги, оптимальні маршрути та дорожні умови, і було підтверджено перевіркою в Google Maps.

API, якщо не вказано точну адресу, визначає адміністративні центри міст, а для країн – географічний центр (наприклад, для «France» після редагування – точка поблизу Клермон-Феррана). Для «Los Angeles» після редагування відстань не знайдена через відсутність автомобільного маршруту

через океан – API повертає ZERO, що коректно обробляється як «не знайдено».

Отже Google Distance Matrix API демонструє високу точність і надійність при коректному введенні, враховує реальні дорожні умови, підтримує асиметрію та географічні центри. Помилки розпізнавання ефективно усуваються через механізм редагування, а обмеження (відсутність маршрутів через океан) обробляються коректно. Модуль готовий до використання в реальних логістичних задачах.

### 5.3 Тестування оптимізації маршрутів у різних постановках

Тестування модуля оптимізації проведено на сторінці RouteCreationPage з використанням алгоритму Branch & Bound для трьох постановок задачі комівояжера на наборі з 12 пунктів (після видалення «Los Angeles» та ручного редагування помилок розпізнавання). Усі тести виконувалися на ПК з процесором середнього рівня, час вимірювався вручну.

1. Класична замкнута TSP, тобто повернення до старту. Додано останній пункт – «Кам’янець-Подільський» (той самий, що й перший). Старт і кінець збігаються. Передостаннім при цьому є «Франція», що важливо для наступної задачі. Алгоритм розв’язує повний цикл Гамільтона (рис. 5.4). Результат: оптимальний маршрут – 9733,3 км. Час виконання: менше 2 секунд.

2. Відкрита TSP, старт ≠ кінець, але вони фіксовані (рис. 5.5). Останній пункт видалено – старт = «Кам’янець-Подільський», кінець = «Франція». Повернення не потрібне. Результат: оптимальний маршрут – 8536,3 км. Час виконання: близько 1 секунди (на 1 пункт менше).

3. Вільний вибір старту і кінця, тобто глобальний пошук (рис. 5.6). Алгоритм перебирає всі пари старт-кінець (n*(n–1) = 11*10 = 110 комбінацій) і обирає найкращу. Результат: оптимальний маршрут – 7052,2 км (старт – Лісабон, кінець – Київ). Час виконання: близько 3 секунд.

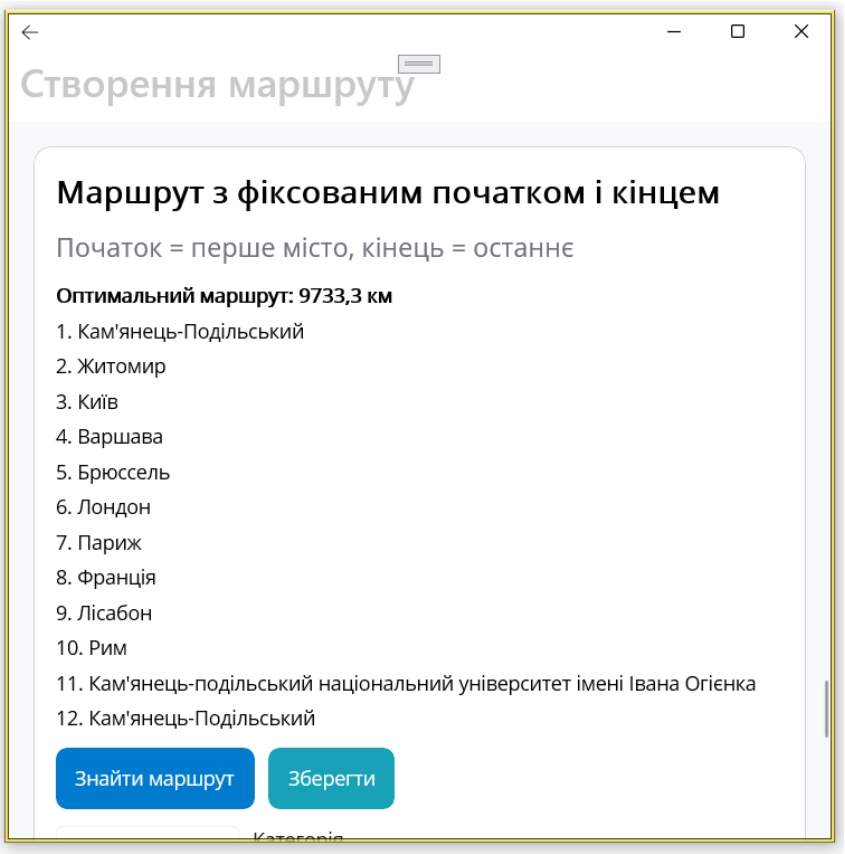


Рисунок 5.4 – Знаходження розв’язку задачі Комівояжера

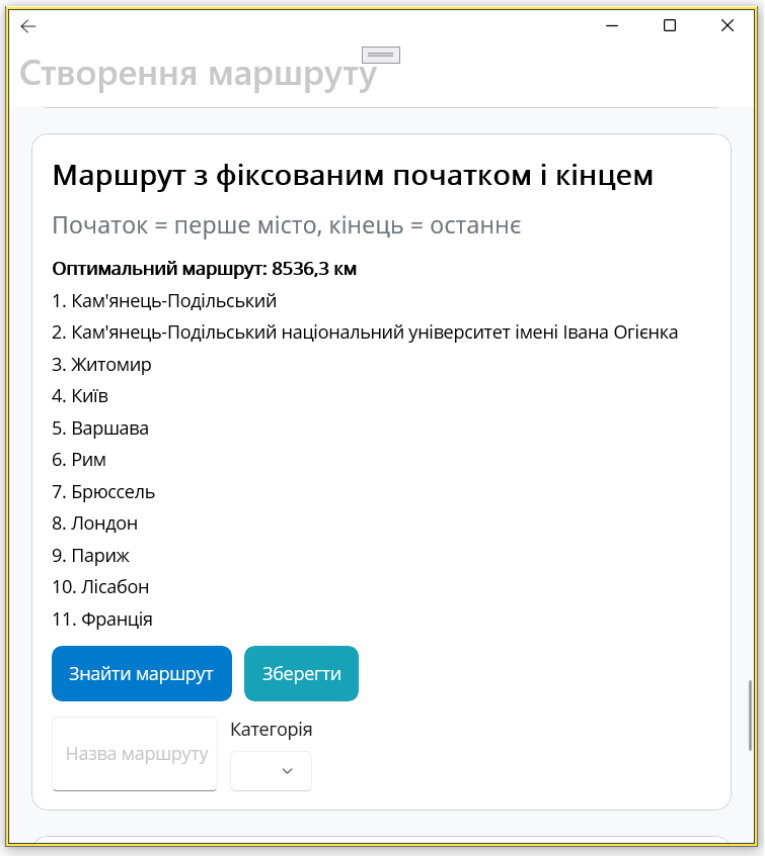


Рисунок 5.5– Знаходження розв’язку Відкритої задачі Комівояжера

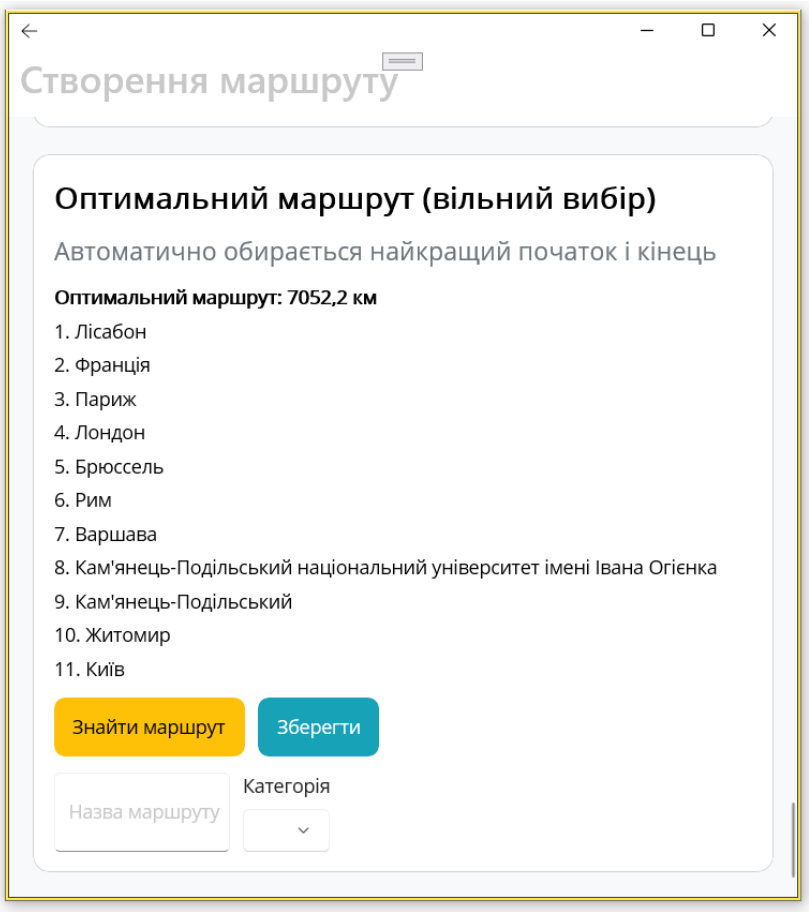


Рисунок 5.6 – Знаходження розв’язку задачі побудови оптимального маршруту з вільним вибором початку та кінця маршруту

Аналіз результатів: відстані логічно зменшуються при переході від замкнутої до відкритої задачі, від відкритої до вільного вибору; час виконання зростає пропорційно кількості переборів: замкнута > відкрита > вільний режим (через n*(n–1) викликів); усі результати коректні, відтворювані, алгоритм працює стабільно при n = 11.

**Висновки до розділу 5**

Голосовий ввід через Google Cloud Speech-to-Text демонструє точність 100 % при чіткій українській вимові реальних назв. Google Distance Matrix API забезпечує високу точність (до 0,1 км), враховує асиметрію маршрутів, односторонні дороги та географічні центри. Алгоритм Branch & Bound успішно розв’язує три постановки TSP (замкнута, відкрита, вільний старт/кінець).

# ВИСНОВКИ

Результатом кваліфікаційної роботи є розробка кросплатформного додатку на базі .NET MAUI, призначеного для планування та оптимізації маршрутів з підтримкою голосового вводу та корпоративної взаємодії. В рамках цього результату:

1. Проведено ґрунтовне дослідження задачі комівояжера у трьох постановках – замкнутої, відкритої та з вільним вибором старту й кінця.

2. Інтегровано Google Cloud Speech-to-Text для голосового вводу міст з локальним записом аудіо. Тестування підтвердило 100% точність розпізнавання при чіткій артикуляції реальних географічних назв.

3. Реалізовано асинхронне формування матриці відстаней через Google Distance Matrix API з підтримкою асиметрії маршрутів, десяткових значень та географічних центрів.

4. Розроблено систему керування маршрутами з категоріями, фільтрацією, сортуванням та збереженням виїздів.

5. Створено прототип корпоративного обміну повідомленнями з прив’язкою маршрутів, що працює в тестовому режимі на локальній базі. Визначено обмеження архітектури та запропоновано гібридну модель: локальна SQLite для індивідуальних користувачів, хмарна (ASP.NET Core + SQL Server) – для корпорацій.

6. Проведено тестування на наборі з 11 пунктів показало успішність поьудови оптимального маршруту в різних постановках задач: замкнута TSP; відкрита TSP; вільний режим.

Додаток працює автономно на різних платформах забезпечуючи повний офлайн-доступ до частини функціоналу. Розробка відповідає сучасним стандартам безпеки, продуктивності та кросплатформності. Програма готова до практичного використання та подальшого масштабування.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Панишев А.В., Морозов А.В. Моделі і методи оптимізації замкнутих маршрутів на транспортній мережі. Харківський національний університет, 2005. 176 с.
2. Панишев А.В., Плечистий Д.Д. Задача про комівояжера. Моделі і методи оптимізації. НТУ "Харківський політехнічний інститут", 2011. 456 с.
3. Ємець О.О., Парфьонова Т.О. Оцінки в методі гілок і меж для задачі нелінійної оптимізації на перестановках. Харківський національний університет, 2013. 120 с.
4. Daniel Jurafsky, James H. Martin. Speech and Language Processing. Pearson, 2020 (3rd Edition). 1100 c.
5. Lawrence Rabiner, Biing-Hwang Juang. Fundamentals of Speech Recognition. Prentice Hall, 1993. 507 c.
6. Dong Yu, Li Deng. Automatic Speech Recognition: A Deep Learning Approach. Springer, 2015. 321 c.
7. Uday Kamath, John Liu, James Whitaker. Deep Learning for Natural Language Processing and Speech Recognition. Springer, 2019. 621 c.
8. Google Cloud Speech-to-Text: офіційна документація. URL: https://cloud.google.com/speech-to-text/docs (дата звернення: 01.09.2025).
9. Microsoft Azure Speech Service: офіційна документація. URL: https://learn.microsoft.com/en-us/azure/ai-services/speech-service/overview (дата звернення: 01.09.2025).
10. Amazon Transcribe: офіційна документація. URL: https://docs.aws. amazon.com/transcribe/latest/dg/what-is.html (дата звернення: 01.09.2025).
11. Apple Speech Framework: офіційна документація. URL: https://developer .apple.com/documentation/speech (дата звернення: 01.09.2025).
12. Посібник по .NET MAUI. URL: https://abitap.com/category/net-maui/ (дата звернення: 01.09.2025).
13. Мельник Н., Левус Є. Вступ до інженерії програмного забезпечення : навч. посіб. Львів : Львівська політехніка, 2018. 248 с.

14. Троелсен Е., Джепикс Ф. Мова Програмування C# 6.0 і платформа .NET 4.6 (7 видання) : монографія. Книжка лавка, 2019. 800 с.
15. NAudio: open-source .NET audio library. URL: https://github.com/naudio/NAudio (дата звернення: 01.09.2025).
16. Google Maps Platform Documentation. URL: https://developers.google.com /maps/documentation (дата звернення: 01.09.2025).
17. Waze for Developers: офіційна документація. URL: https://developers.google.com/waze (дата звернення: 01.09.2025).
18. Route4Me API Documentation. URL: https://route4me.io/docs (дата звернення: 01.09.2025).
19. OptimoRoute API Documentation. URL: https://optimoroute.com/api (дата звернення: 01.09.2025).
20. .NET Multi-platform App UI (.NET MAUI). URL: https://learn.microsoft.com /en-us/dotnet/maui/ (дата звернення: 01.09.2025).
21. Flutter Documentation. URL: https://docs.flutter.dev/ (дата звернення: 01.09.2025).
22. React Native Documentation. URL: https://reactnative.dev/docs/getting-started (дата звернення: 01.09.2025).
23. Грицюк Ю. Аналіз вимог до програмного забезпечення : монографія. Львів Львівська політехніка, 2018. 456 с.
24. Google Maps Distance Matrix API Documentation. URL: developers.google. com/maps/documentation/distance-matrix (дата звернення: 01.09.2025).
25. Entity Framework Core Documentation. URL: https://learn.microsoft.com/en-us/ef/core/ (дата звернення: 01.09.2025).
26. SQLite Official Documentation. URL: https://www.sqlite.org/docs.html (дата звернення: 01.09.2025).
27. BCrypt.Net-Next: .NET implementation of BCrypt. URL: https://github.com/BcryptNet/bcrypt.net (дата звернення: 01.09.2025).

# ДОДАТКИ

## Додаток А Програмний код класів застосунку

```
using RouteOptimizerApp.Models;

namespace RouteOptimizerApp.Data
{
    /// <summary>
    /// Контекст Entity Framework Core для роботи з базою даних
    /// </summary>
    public class AppDbContext : DbContext
    {
        // Таблиці бази даних
        public DbSet<Company> Companies { get; set; } = null!;
        public DbSet<AppUser> Users { get; set; } = null!;
        public DbSet<RouteCategory> RouteCategories { get; set; } = null!;
        public DbSet<OptimizedRoute> Routes { get; set; } = null!;
        public DbSet<Trip> Trips { get; set; } = null!;
        public DbSet<UserMessage> Messages { get; set; } = null!;
        public AppDbContext()
        {
            // Видаляємо БД (для тестування)
            // Database.EnsureDeleted();
            // Створюємо БД, якщо її немає
            Database.EnsureCreated();
        }
    }
}
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace RouteOptimizerApp.Models
{
    /// <summary>
    /// Користувач додатка
    /// </summary>
    public class AppUser
    {
        public int Id { get; set; }

        [Required, MaxLength(100)]
        public string Name { get; set; } = null!;

        [MaxLength(50)]
        public string? Role { get; set; } // Наприклад: "Admin", "User"

        [Required, MaxLength(50)]
        public string Login { get; set; } = null!;

        [Required, EmailAddress, MaxLength(100)]
        public string Email { get; set; } = null!;

        [Required, MinLength(8)]
        public string Password { get; set; } = null!;

        // Навігаційна властивість: корпорація (опціонально)
        public Company? Company { get; set; }

        // Один користувач → багато категорій
        public List<RouteCategory> Categories { get; set; } = new();
    }
}
using System.ComponentModel.DataAnnotations;

namespace RouteOptimizerApp.Models
{
    /// <summary>
    /// Корпорація (група користувачів)
    /// </summary>
    public class Company
    {
        public int Id { get; set; }

        [Required, MaxLength(100)]
        public string Name { get; set; } = null!;
```

```
        // Один до багатьох: корпорація → користувачі
        public List<AppUser> Users { get; set; } = new();
    }
}

using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace RouteOptimizerApp.Models
{
    /// <summary>
    /// Оптимізований маршрут (з точками, відстанню)
    /// </summary>
    public class OptimizedRoute
    {
        public int Id { get; set; }

        [Required, MaxLength(100)]
        public string Name { get; set; } = null!;

        public int Distance { get; set; } // у кілометрах

        // Список точок маршруту (наприклад: ["Київ", "Львів", "Одеса"])
        public List<string> Points { get; set; } = new();

        // Навігація: належить до однієї категорії
        public RouteCategory Category { get; set; } = null!;

        // Один до багатьох: маршрут → виїзди
        public List<Trip> Trips { get; set; } = new();

        // Не зберігається в БД — тільки для відображення
        [NotMapped]
        public string PointsAsString => string.Join(" → ", Points);
    }
}

using System.ComponentModel.DataAnnotations;

namespace RouteOptimizerApp.Models
{
    /// <summary>
    /// Категорія маршрутів (наприклад: "Робота", "Подорожі")
    /// </summary>
    public class RouteCategory
    {
        public int Id { get; set; }

        [Required, MaxLength(50)]
        public string Name { get; set; } = null!;

        // Навігація: належить одному користувачу
        public AppUser User { get; set; } = null!;

        // Один до багатьох: категорія → маршрути
        public List<OptimizedRoute> Routes { get; set; } = new();
    }
}

using System.ComponentModel.DataAnnotations;

namespace RouteOptimizerApp.Models
{
    /// <summary>
    /// Виїзд за маршрутом (подорож)
    /// </summary>
    public class Trip
    {
        public int Id { get; set; }

        [Required, MaxLength(100)]
        public string Name { get; set; } = null!;

        public DateOnly Date { get; set; }

        // Навігація: належить до одного маршруту
        public OptimizedRoute Route { get; set; } = null!;
    }
```

```
}

using System.ComponentModel.DataAnnotations;

namespace RouteOptimizerApp.Models
{
    /// <summary>
    /// Повідомлення між користувачами
    /// </summary>
    public class UserMessage
    {
        public int Id { get; set; }

        [Required, MaxLength(100)]
        public string Subject { get; set; } = null!;

        public DateOnly Date { get; set; }

        // Від кого
        public AppUser Sender { get; set; } = null!;

        // Кому
        public AppUser Receiver { get; set; } = null!;

        // Опціонально: пов'язаний маршрут
        public OptimizedRoute? Route { get; set; }
    }
}

using RouteOptimizerApp.Models;
using RouteOptimizerApp.Data;

namespace RouteOptimizerApp.Pages
{
    /// <summary>
    /// Сторінка управління категоріями маршрутів користувача
    /// </summary>
    public partial class CategoriesListPage : ContentPage
    {
        private readonly AppDbContext _dbContext = new();
        private readonly AppUser _currentUser;

        public CategoriesListPage(AppUser user)
        {
            InitializeComponent();
            _currentUser = user;
            LoadCategories();
        }

        /// <summary>
        /// Завантажує категорії поточного користувача
        /// </summary>
        private void LoadCategories()
        {
            var categories = _dbContext.RouteCategories
                .Where(c => c.User.Id == _currentUser.Id)
                .OrderBy(c => c.Name)
                .ToList();

            CategoriesCollectionView.ItemsSource = categories;
        }

        /// <summary>
        /// Додає нову категорію
        /// </summary>
        private async void OnAddCategoryClicked(object sender, EventArgs e)
        {
            var name = CategoryNameEntry.Text?.Trim();
            if (string.IsNullOrWhiteSpace(name))
            {
                await ShowError("Введіть назву категорії.");
                return;
            }
            AppUser UserCur = _dbContext.Users.Find(_currentUser.Id);
            var newCategory = new RouteCategory
            {
                Name = name,
                User = UserCur
            };
```

```
            _dbContext.RouteCategories.Add(newCategory);
            await _dbContext.SaveChangesAsync();

            CategoryNameEntry.Text = string.Empty;
            LoadCategories();
            await ShowSuccess("Категорію додано!");
        }

        /// <summary>
        /// Редагує назву категорії
        /// </summary>
        private async void OnEditCategoryClicked(object sender, EventArgs e)
        {
            if (sender is Button btn && btn.CommandParameter is RouteCategory category)
            {
                var newName = await DisplayPromptAsync(
                    "Редагувати",
                    "Нова назва:",
                    initialValue: category.Name,
                    maxLength: 50);

                if (!string.IsNullOrWhiteSpace(newName) && newName != category.Name)
                {
                    category.Name = newName.Trim();
                    await _dbContext.SaveChangesAsync();
                    LoadCategories();
                    await ShowSuccess("Оновлено!");
                }
            }
        }

        /// <summary>
        /// Видаляє категорію після підтвердження
        /// </summary>
        private async void OnDeleteCategoryClicked(object sender, EventArgs e)
        {
            if (sender is Button btn && btn.CommandParameter is RouteCategory category)
            {
                bool confirm = await DisplayAlert(
                    "Видалити",
                    $"Видалити категорію «{category.Name}»?",
                    "Так", "Ні");

                if (confirm)
                {
                    _dbContext.RouteCategories.Remove(category);
                    await _dbContext.SaveChangesAsync();
                    LoadCategories();
                    await ShowSuccess("Видалено!");
                }
            }
        }

        private Task ShowError(string msg) => DisplayAlert("Помилка", msg, "OK");
        private Task ShowSuccess(string msg) => DisplayAlert("Успіх", msg, "OK");
    }
}

using Microsoft.EntityFrameworkCore;
using RouteOptimizerApp.Models;
using RouteOptimizerApp.Data;

namespace RouteOptimizerApp.Pages
{
    /// <summary>
    /// Сторінка перегляду вхідних повідомлень користувача
    /// </summary>
    public partial class IncomingMessagesPage : ContentPage
    {
        private readonly AppDbContext _dbContext;
        private readonly AppUser _currentUser;

        public IncomingMessagesPage(AppUser currentUser)
        {
            InitializeComponent();
            _currentUser = currentUser;
            _dbContext = new AppDbContext();
            LoadIncomingMessages();
```

```
        }

        /// <summary>
        /// Завантажує всі вхідні повідомлення (отримувачем є поточний користувач)
        /// </summary>
        private void LoadIncomingMessages()
        {
            var messages = _dbContext.Messages
                .Include(m => m.Sender)
                .Include(m => m.Route)
                .Where(m => m.Receiver.Id == _currentUser.Id)
                .OrderByDescending(m => m.Date)
                .ToList();

            MessagesCollectionView.ItemsSource = messages;
        }
    }
}

using Microsoft.Maui.Storage;
using RouteOptimizerApp.Models;
using RouteOptimizerApp.Data;
using BCrypt.Net; // ПРАВИЛЬНИЙ using

namespace RouteOptimizerApp.Pages
{
    public partial class LoginPage : ContentPage
    {
        private readonly AppDbContext _dbContext = new();

        public LoginPage()
        {
            InitializeComponent();
            CheckStoredCredentials();
        }

        private async void CheckStoredCredentials()
        {
            var login = await SecureStorage.GetAsync("Login");
            var password = await SecureStorage.GetAsync("Password");

            if (!string.IsNullOrEmpty(login) && !string.IsNullOrEmpty(password))
            {
                var user = _dbContext.Users.FirstOrDefault(u => u.Login == login);
                if (user != null && BCrypt.Net.BCrypt.Verify(password, user.Password))
                {
                    NavigateToUserDashboard(user);
                }
            }
        }

        private async void OnLoginButtonClicked(object sender, EventArgs e)
        {
            string login = LoginEntry.Text?.Trim();
            string password = PasswordEntry.Text?.Trim();

            if (string.IsNullOrEmpty(login) || string.IsNullOrEmpty(password))
            {
                LoginErrorLabel.Text = "Логін та пароль не можуть бути порожніми.";
                LoginErrorLabel.IsVisible = true;
                return;
            }

            var user = _dbContext.Users.FirstOrDefault(u => u.Login == login);
            if (user == null || !BCrypt.Net.BCrypt.Verify(password, user.Password))
            {
                LoginErrorLabel.Text = "Невірний логін або пароль.";
                LoginErrorLabel.IsVisible = true;
                return;
            }

            await SecureStorage.SetAsync("Login", login);
            await SecureStorage.SetAsync("Password", password);

            NavigateToUserDashboard(user);
        }

        private void NavigateToUserDashboard(AppUser user)
        {
```

```
            Application.Current.Dispatcher.Dispatch(() =>
            {
                Application.Current.MainPage = new NavigationPage(new UserProfileDashboard(user));
            });
        }

        private void OnNavigateToRegistrationClicked(object sender, EventArgs e)
        {
            Application.Current.Dispatcher.Dispatch(() =>
            {
                Application.Current.MainPage = new NavigationPage(new UserRegistrationPage());
            });
        }
    }
}
using Google.Cloud.Speech.V1;
using NAudio.Wave;
using System.IO;
using System.Net.Http;
using Newtonsoft.Json.Linq;
using RouteOptimizerApp.Models;
using RouteOptimizerApp.Data;

namespace RouteOptimizerApp.Pages
{
    /// <summary>
    /// Сторінка створення нового маршруту з голосовим вводом, матрицею відстаней та алгоритмом TSP
    /// </summary>
    public partial class RouteCreationPage : ContentPage
    {
        private readonly AppDbContext _dbContext;
        private readonly AppUser _currentUser;
        private readonly List<string> _routePoints = new();
        private readonly List<string> _freeRoutePoints = new();

        private WaveInEvent? waveIn;
        private MemoryStream? memoryStream;
        private int _fixedRouteDistance = 0;
        private int _freeRouteDistance = 0;

        public RouteCreationPage(AppUser user)
        {
            InitializeComponent();

            // НАЛАШТУВАННЯ GOOGLE CREDENTIALS (як у старому коді)
            string projectPath = Directory.GetParent(AppDomain.CurrentDomain.BaseDirectory).Parent.Parent.Parent.Parent.Parent.FullName;
            string credentialsPath = Path.Combine(projectPath, "1.json");
            Environment.SetEnvironmentVariable("GOOGLE_APPLICATION_CREDENTIALS", credentialsPath);

            _dbContext = new AppDbContext();
            _currentUser = user;
            LoadUserCategories();
        }

        /// <summary>
        /// Завантажує категорії користувача в Picker'и
        /// </summary>
        private void LoadUserCategories()
        {
            var categories = _dbContext.RouteCategories
                .Where(c => c.User.Id == _currentUser.Id)
                .ToList();

            if (categories.Any())
            {
                foreach (var cat in categories)
                {
                    FixedCategoryPicker.Items.Add(cat.Name);
                    FreeCategoryPicker.Items.Add(cat.Name);
                }
            }
            else
            {
                DisplayAlert("Увага", "У вас немає категорій. Створіть їх у профілі.", "OK");
            }
        }

        #region Голосовий ввід
        private void OnStartRecording(object sender, EventArgs e)
```

```
{
    memoryStream = new MemoryStream();
    waveIn = new WaveInEvent
    {
        WaveFormat = new WaveFormat(16000, 1)
    };

    waveIn.DataAvailable += (s, a) =>
    {
        memoryStream.Write(a.Buffer, 0, a.BytesRecorded);
    };

    waveIn.RecordingStopped += async (s, a) =>
    {
        waveIn.Dispose();
        waveIn = null;

        // Розпізнавання мовлення
        await RecognizeSpeechAsync(memoryStream.ToArray());
    };

    waveIn.StartRecording();
}

private void OnStopRecording(object sender, EventArgs e)
{
    // Зупинити запис
    waveIn?.StopRecording();
}

private async Task RecognizeSpeechAsync(byte[] audioData)
{
    if (audioData.Length == 0)
    {
        VoiceStatusLabel.Text = "Запис порожній";
        return;
    }

    try
    {
        var speechClient = SpeechClient.Create();
        var config = new RecognitionConfig
        {
            Encoding = RecognitionConfig.Types.AudioEncoding.Linear16,
            SampleRateHertz = 16000,
            LanguageCode = "uk-UA"
        };
        var audio = RecognitionAudio.FromBytes(audioData);
        var response = await speechClient.RecognizeAsync(config, audio);

        var transcript = response.Results
            .SelectMany(r => r.Alternatives)
            .OrderByDescending(a => a.Confidence)
            .FirstOrDefault()?.Transcript;

        if (!string.IsNullOrEmpty(transcript))
        {
            AddCityToRoute(transcript.Trim());
            VoiceStatusLabel.Text = $"Додано: {transcript}";
        }
        else
        {
            VoiceStatusLabel.Text = "Не розпізнано";
        }
    }
    catch (Exception ex)
    {
        await DisplayAlert("Помилка", $"Голос: {ex.Message}", "OK");
        VoiceStatusLabel.Text = "Помилка";
    }
}
#endregion

#region Додавання міста

/// <summary>
/// Додає нове поле для редагування міста
/// </summary>
private void AddCityToRoute(string cityName)
```

```
{
    var container = new HorizontalStackLayout { Spacing = 10 };

    var entry = new Entry
    {
        Text = cityName,
        Placeholder = "Місто",
        HorizontalOptions = LayoutOptions.FillAndExpand,
        BackgroundColor = Colors.White
    };

    var deleteBtn = new Button
    {
        Text = "X",
        BackgroundColor = Colors.Red,
        TextColor = Colors.White,
        WidthRequest = 40,
        CornerRadius = 20
    };

    deleteBtn.Clicked += (s, e) => PointsContainer.Children.Remove(container);

    container.Children.Add(entry);
    container.Children.Add(deleteBtn);
    PointsContainer.Children.Add(container);

    VoiceStatusLabel.Text = $"Додано: {cityName}";
}

#endregion

#region Матриця відстаней

/// <summary>
/// Обчислює відстані між усіма парами міст через Google Distance Matrix API
/// </summary>
private async Task<Dictionary<string, Dictionary<string, double>>> CalculateDistanceMatrix(List<string> cities)
{
    string apiKey = "AIzaSyAwLIRj-IoQa_dEhIlf24PpAmrpeDBigd8";
    string baseUrl = "https://maps.googleapis.com/maps/api/distancematrix/json";
    var client = new HttpClient();
    var matrix = new Dictionary<string, Dictionary<string, double>>();

    foreach (var origin in cities)
    {
        var distances = new Dictionary<string, double>();
        var destinations = cities.Where(c => c != origin);
        var destParam = string.Join("|", destinations);
        var url = $"{baseUrl}?origins={Uri.EscapeDataString(origin)}&destinations={Uri.EscapeDataString(destParam)}&key={apiKey}";

        var response = await client.GetStringAsync(url);
        var json = JObject.Parse(response);

        if (json["status"]?.ToString() == "OK")
        {
            var elements = json["rows"]?[0]?["elements"];
            int idx = 0;
            foreach (var dest in destinations)
            {
                var el = elements?[idx++];
                var status = el?["status"]?.ToString();
                if (status == "OK")
                {
                    var meters = el?["distance"]?["value"]?.ToObject<double>() ?? 0;
                    distances[dest] = meters / 1000.0; // км
                }
                else
                {
                    distances[dest] = -1;
                }
            }
        }
        matrix[origin] = distances;
    }

    return matrix;
}

/// <summary>
```

```
/// Побудова та відображення матриці відстаней
/// </summary>
private async void OnBuildMatrixClicked(object sender, EventArgs e)
{
    var cities = GetCitiesFromUI();
    if (cities.Count < 2)
    {
        await DisplayAlert("Помилка", "Додайте принаймні 2 міста.", "OK");
        return;
    }

    DistanceMatrixContainer.Children.Clear();
    var matrix = await CalculateDistanceMatrix(cities);

    foreach (var (from, distances) in matrix)
    {
        foreach (var (to, dist) in distances)
        {
            var text = dist < 0
                ? $"{from} → {to}: не знайдено"
                : $"{from} → {to}: {dist:F1} км";
            DistanceMatrixContainer.Children.Add(new Label { Text = text, FontSize = 14 });
        }
    }
}

#endregion

#region Алгоритм TSP (Branch & Bound)

/// <summary>
/// Вирішує задачу комівояжера з фіксованим початком і кінцем
/// </summary>
private (List<string> route, double distance) SolveTSP(
    Dictionary<string, Dictionary<string, double>> matrix,
    string start, string end)
{
    var cities = matrix.Keys.Where(c => c != start && c != end).ToList();
    double bestDist = double.MaxValue;
    List<string>? bestRoute = null;

    void Search(List<string> path, double dist)
    {
        if (path.Count == cities.Count)
        {
            path.Insert(0, start);
            path.Add(end);
            double total = dist + matrix[path[^2]][end];
            if (total < bestDist)
            {
                bestDist = total;
                bestRoute = new List<string>(path);
            }
            path.RemoveAt(0);
            path.RemoveAt(path.Count - 1);
            return;
        }

        var last = path.LastOrDefault() ?? start;
        foreach (var next in cities.Where(c => !path.Contains(c)))
        {
            var edge = matrix[last][next];
            if (edge < 0 || dist + edge >= bestDist) continue;
            path.Add(next);
            Search(path, dist + edge);
            path.RemoveAt(path.Count - 1);
        }
    }

    Search(new List<string>(), 0);
    return (bestRoute ?? new List<string>(), bestDist);
}

#endregion

#region Фіксований маршрут

private async void OnFindFixedRouteClicked(object sender, EventArgs e)
{
```

```
    var cities = GetCitiesFromUI();
    if (cities.Count < 2) { await ShowError("Додайте хоча б 2 міста."); return; }

    var matrix = await CalculateDistanceMatrix(cities);
    var (route, distance) = SolveTSP(matrix, cities.First(), cities.Last());

    if (route.Count == 0) { await ShowError("Не вдалося знайти маршрут."); return; }

    _routePoints.Clear();
    _routePoints.AddRange(route);
    _fixedRouteDistance = (int)distance;

    DisplayRoute(FixedRouteResultContainer, route, distance);
}

private async void OnSaveFixedRouteClicked(object sender, EventArgs e)
{
    await SaveRoute(_routePoints, _fixedRouteDistance, FixedRouteNameEntry, FixedCategoryPicker);
}

#endregion

#region Вільний маршрут

private async void OnFindFreeRouteClicked(object sender, EventArgs e)
{
    var cities = GetCitiesFromUI();
    if (cities.Count < 2) { await ShowError("Додайте хоча б 2 міста."); return; }

    var matrix = await CalculateDistanceMatrix(cities);
    double bestDist = double.MaxValue;
    List<string> bestRoute = new();

    for (int i = 0; i < cities.Count; i++)
    {
        for (int j = 0; j < cities.Count; j++)
        {
            if (i == j) continue;
            var (route, dist) = SolveTSP(matrix, cities[i], cities[j]);
            if (dist < bestDist)
            {
                bestDist = dist;
                bestRoute = route;
            }
        }
    }

    _freeRoutePoints.Clear();
    _freeRoutePoints.AddRange(bestRoute);
    _freeRouteDistance = (int)bestDist;

    DisplayRoute(FreeRouteResultContainer, bestRoute, bestDist);
}

private async void OnSaveFreeRouteClicked(object sender, EventArgs e)
{
    await SaveRoute(_freeRoutePoints, _freeRouteDistance, FreeRouteNameEntry, FreeCategoryPicker);
}

#endregion

#region Допоміжні методи

private List<string> GetCitiesFromUI()
{
    return PointsContainer.Children
        .OfType<HorizontalStackLayout>()
        .Select(h => h.Children.OfType<Entry>().FirstOrDefault()?.Text?.Trim())
        .Where(s => !string.IsNullOrEmpty(s))
        .ToList()!;
}

private void DisplayRoute(StackLayout container, List<string> route, double distance)
{
    container.Children.Clear();
    container.Children.Add(new Label
    {
        Text = $"Оптимальний маршрут: {distance:F1} км",
        FontAttributes = FontAttributes.Bold
```

```
            });
            for (int i = 0; i < route.Count; i++)
            {
                container.Children.Add(new Label { Text = $"{i + 1}. {route[i]}" });
            }
        }

        private async Task SaveRoute(List<string> points, int distance, Entry nameEntry, Picker catPicker)
        {
            if (string.IsNullOrWhiteSpace(nameEntry.Text))
            {
                await ShowError("Введіть назву маршруту.");
                return;
            }
            if (catPicker.SelectedItem == null)
            {
                await ShowError("Оберіть категорію.");
                return;
            }
            if (points.Count == 0)
            {
                await ShowError("Маршрут порожній.");
                return;
            }

            var category = _dbContext.RouteCategories
                .FirstOrDefault(c => c.Name == catPicker.SelectedItem.ToString());

            var route = new OptimizedRoute
            {
                Name = nameEntry.Text.Trim(),
                Distance = distance,
                Points = new List<string>(points),
                Category = category!
            };

            _dbContext.Routes.Add(route);
            await _dbContext.SaveChangesAsync();

            await DisplayAlert("Успіх", "Маршрут збережено!", "OK");
        }

        private Task ShowError(string message) => DisplayAlert("Помилка", message, "OK");

        #endregion
    }
}
using Microsoft.EntityFrameworkCore;
using RouteOptimizerApp.Models;
using RouteOptimizerApp.Data;

namespace RouteOptimizerApp.Pages
{
    public partial class RoutesListPage : ContentPage
    {
        private readonly AppDbContext _dbContext = new();
        private readonly AppUser _currentUser;
        private List<OptimizedRoute> _allRoutes = new();

        public RoutesListPage(AppUser user)
        {
            InitializeComponent();
            _currentUser = user;
            LoadRoutesAndCategories();
        }

        protected override void OnAppearing()
        {
            base.OnAppearing();
            LoadRoutes(); // ОНОВЛЮЄМО СПИСОК ПРИ ПОВЕРНЕННІ
        }

        private void LoadRoutesAndCategories()
        {
            var categories = _dbContext.RouteCategories
                .Where(c => c.User.Id == _currentUser.Id)
                .ToList();

            _allRoutes = _dbContext.Routes
```

```
            .Where(r => categories.Select(c => c.Id).Contains(r.Category.Id))
            .Include(r => r.Category)
            .Include(r => r.Trips)
            .ToList();

        PopulateCategoryFilter(categories);
        UpdateRoutesView();
    }

    private void LoadRoutes()
    {
        // ПЕРЕЗАВАНТАЖУЄМО З БД — НОВІ МАРШРУТИ З'ЯВЛЯТЬСЯ
        var categories = _dbContext.RouteCategories
            .Where(c => c.User.Id == _currentUser.Id)
            .Select(c => c.Id)
            .ToList();

        _allRoutes = _dbContext.Routes
            .Where(r => categories.Contains(r.Category.Id))
            .Include(r => r.Category)
            .Include(r => r.Trips)
            .ToList();

        UpdateRoutesView();
    }

    private void PopulateCategoryFilter(List<RouteCategory> categories)
    {
        var categoryNames = categories.Select(c => c.Name).Prepend("Всі").ToList();
        CategoryPicker.ItemsSource = categoryNames;
        CategoryPicker.SelectedIndex = 0;
    }

    private void UpdateRoutesView()
    {
        var filtered = _allRoutes.AsEnumerable();

        if (CategoryPicker.SelectedIndex > 0)
        {
            var selectedCategory = CategoryPicker.SelectedItem.ToString();
            filtered = filtered.Where(r => r.Category.Name == selectedCategory);
        }

        var minPoints = (int)PointsSlider.Value;
        filtered = filtered.Where(r => r.Points.Count >= minPoints);

        filtered = SortPicker.SelectedItem?.ToString() switch
        {
            "Відстанню (зростання)" => filtered.OrderBy(r => r.Distance),
            "Кількістю виїздів (спадання)" => filtered.OrderByDescending(r => r.Trips.Count),
            _ => filtered.OrderBy(r => r.Name)
        };

        RoutesCollectionView.ItemsSource = filtered.ToList();
    }

    private void OnFilterChanged(object sender, EventArgs e) => UpdateRoutesView();
    private void OnSortChanged(object sender, EventArgs e) => UpdateRoutesView();

    private async void OnCreateRouteClicked(object sender, EventArgs e)
    {
        await Navigation.PushAsync(new RouteCreationPage(_currentUser));
    }

    private async void OnDeleteRouteClicked(object sender, EventArgs e)
    {
        if (sender is Button button && button.CommandParameter is OptimizedRoute route)
        {
            bool confirm = await DisplayAlert(
                "Видалити маршрут?",
                $"Ви впевнені, що хочете видалити:\n\n\"{route.Name}\"\n({route.Distance} км, {route.Points.Count} пунктів)",
                "Так, видалити",
                "Ні");

            if (confirm)
            {
                try
                {
                    _dbContext.Routes.Remove(route);
```

```
                    await _dbContext.SaveChangesAsync();

                    _allRoutes.Remove(route);
                    UpdateRoutesView();

                    await DisplayAlert("Готово", "Маршрут видалено.", "OK");
                }
                catch (Exception ex)
                {
                    await DisplayAlert("Помилка", $"Не вдалося видалити: {ex.Message}", "OK");
                }
            }
        }
    }
}

using Microsoft.EntityFrameworkCore;
using RouteOptimizerApp.Models;
using RouteOptimizerApp.Data;

namespace RouteOptimizerApp.Pages
{
    /// <summary>
    /// Сторінка для надсилання повідомлень користувачам у тій самій корпорації
    /// </summary>
    public partial class SendMessagePage : ContentPage
    {
        private readonly AppDbContext _dbContext;
        private readonly AppUser _currentUser;

        public SendMessagePage(AppUser currentUser)
        {
            InitializeComponent();
            _dbContext = new AppDbContext();

            // Завантажуємо користувача з корпорацією
            _currentUser = _dbContext.Users
                .Include(u => u.Company)
                .FirstOrDefault(u => u.Id == currentUser.Id)!;

            if (_currentUser.Company == null)
            {
                DisplayAlert("Помилка", "Ви не належите до жодної корпорації.", "OK");
                return;
            }

            LoadRecipients();
        }

        /// <summary>
        /// Завантажує всіх користувачів з тієї самої корпорації (окрім себе)
        /// </summary>
        private void LoadRecipients()
        {
            var recipients = _dbContext.Users
                .Where(u => u.Company != null &&
                            u.Company.Id == _currentUser.Company.Id &&
                            u.Id != _currentUser.Id)
                .OrderBy(u => u.Name)
                .ToList();

            RecipientPicker.ItemsSource = recipients;
        }

        /// <summary>
        /// Обробка надсилання повідомлення
        /// </summary>
        private async void OnSendMessageClicked(object sender, EventArgs e)
        {
            if (RecipientPicker.SelectedItem is not AppUser recipient)
            {
                await ShowError("Оберіть отримувача.");
                return;
            }

            if (string.IsNullOrWhiteSpace(MessageEditor.Text))
            {
                await ShowError("Введіть текст повідомлення.");
```

```
            return;
        }

        var message = new UserMessage
        {
            Subject = MessageEditor.Text.Trim(),
            Date = DateOnly.FromDateTime(DateTime.Now),
            Sender = _currentUser,
            Receiver = recipient
            // Route = null — можна додати пізніше
        };

        _dbContext.Messages.Add(message);
        await _dbContext.SaveChangesAsync();

        MessageEditor.Text = string.Empty;
        StatusLabel.Text = "Повідомлення успішно надіслано!";
        StatusLabel.IsVisible = true;

        // Автоматично приховати статус через 3 секунди
        await Task.Delay(3000);
        StatusLabel.IsVisible = false;
    }

    /// <summary>
    /// Допоміжний метод для показу помилок
    /// </summary>
    private Task ShowError(string text) => DisplayAlert("Помилка", text, "OK");

    /// <summary>
    /// Очищення контексту при виході зі сторінки
    /// </summary>
    protected override void OnDisappearing()
    {
        _dbContext.Dispose();
        base.OnDisappearing();
    }
  }
}

using Microsoft.EntityFrameworkCore;
using RouteOptimizerApp.Models;
using RouteOptimizerApp.Data;

namespace RouteOptimizerApp.Pages
{
  /// <summary>
  /// Сторінка перегляду надісланих повідомлень користувача
  /// </summary>
  public partial class SentMessagesPage : ContentPage
  {
    private readonly AppDbContext _dbContext;
    private readonly AppUser _currentUser;

    public SentMessagesPage(AppUser currentUser)
    {
        InitializeComponent();
        _currentUser = currentUser;
        _dbContext = new AppDbContext();
        LoadSentMessages();
    }

    /// <summary>
    /// Завантажує всі повідомлення, де поточний користувач є відправником
    /// </summary>
    private void LoadSentMessages()
    {
        var messages = _dbContext.Messages
            .Include(m => m.Receiver)
            .Include(m => m.Route)
            .Where(m => m.Sender.Id == _currentUser.Id)
            .OrderByDescending(m => m.Date)
            .ToList();

        MessagesCollectionView.ItemsSource = messages;
    }
  }
}
```

```
using RouteOptimizerApp.Models;
using RouteOptimizerApp.Data;

namespace RouteOptimizerApp.Pages
{
    /// <summary>
    /// Сторінка перегляду та управління виїздами (подорожами) за маршрутами
    /// </summary>
    public partial class TripsListPage : ContentPage
    {
        private readonly AppDbContext _dbContext = new();
        private readonly AppUser _currentUser;
        private OptimizedRoute? _selectedRoute;

        public TripsListPage(AppUser user)
        {
            InitializeComponent();
            _currentUser = user;
            LoadRoutesIntoPicker();
        }

        /// <summary>
        /// Завантажує маршрути користувача у Picker
        /// </summary>
        private void LoadRoutesIntoPicker()
        {
            var routes = _dbContext.Routes
                .Where(r => r.Category.User.Id == _currentUser.Id)
                .OrderBy(r => r.Name)
                .ToList();

            RoutePicker.ItemsSource = routes;
            RoutePicker.ItemDisplayBinding = new Binding("Name");
        }

        /// <summary>
        /// Обробка вибору маршруту — завантажує його виїзди
        /// </summary>
        private void OnRouteSelected(object sender, EventArgs e)
        {
            _selectedRoute = RoutePicker.SelectedItem as OptimizedRoute;
            if (_selectedRoute != null)
            {
                LoadTripsForRoute(_selectedRoute);
            }
            else
            {
                TripsCollectionView.ItemsSource = null;
            }
        }

        /// <summary>
        /// Завантажує виїзди для вибраного маршруту
        /// </summary>
        private void LoadTripsForRoute(OptimizedRoute route)
        {
            var trips = _dbContext.Trips
                .Where(t => t.Route.Id == route.Id)
                .OrderByDescending(t => t.Date)
                .ToList();

            TripsCollectionView.ItemsSource = trips;
        }

        /// <summary>
        /// Додає новий виїзд до вибраного маршруту
        /// </summary>
        private async void OnAddTripClicked(object sender, EventArgs e)
        {
            if (_selectedRoute == null)
            {
                await DisplayAlert("Помилка", "Спочатку оберіть маршрут.", "ОК");
                return;
            }

            var newTrip = new Trip
            {
                Name = $"Виїзд від {DateTime.Now:dd.MM.yyyy HH:mm}",
                Date = DateOnly.FromDateTime(DateTime.Now),
```

```
            Route = _selectedRoute
        };

        _dbContext.Trips.Add(newTrip);
        await _dbContext.SaveChangesAsync();

        LoadTripsForRoute(_selectedRoute);
        await DisplayAlert("Успіх", "Виїзд додано!", "OK");
    }

    /// <summary>
    /// Видаляє виїзд після підтвердження
    /// </summary>
    private async void OnDeleteTripClicked(object sender, EventArgs e)
    {
        if (sender is Button button && button.CommandParameter is Trip trip)
        {
            bool confirm = await DisplayAlert(
                "Підтвердження",
                $"Видалити виїзд «{trip.Name}»?",
                "Так", "Ні");

            if (confirm)
            {
                _dbContext.Trips.Remove(trip);
                await _dbContext.SaveChangesAsync();
                LoadTripsForRoute(trip.Route);
            }
        }
    }
  }
}

using Microsoft.EntityFrameworkCore;
using RouteOptimizerApp.Models;
using RouteOptimizerApp.Data;
using RouteOptimizerApp.Pages;
using BCrypt.Net;

namespace RouteOptimizerApp.Pages
{
  public class MenuItemModel
  {
    public string Title { get; set; } = string.Empty;
    public Type PageType { get; set; } = null!;
    public int Count { get; set; }
    public bool HasCount => Count > 0;
  }

  public partial class UserProfileDashboard : ContentPage
  {
    private readonly AppDbContext _dbContext;
    private readonly AppUser _currentUser;

    public UserProfileDashboard(AppUser user)
    {
        InitializeComponent();
        _dbContext = new AppDbContext();
        _currentUser = _dbContext.Users
            .Include(u => u.Company)
            .FirstOrDefault(u => u.Id == user.Id)!;

        LoadUserProfile();
        PopulateMenu(); // Початкове завантаження
    }

    // ОНОВЛЮЄМО ЧИСЛА ПРИ КОЖНОМУ ПОКАЗІ СТОРІНКИ
    protected override void OnAppearing()
    {
        base.OnAppearing();
        PopulateMenu(); // КРИТИЧНО: ПЕРЕЗАВАНТАЖУЄМО ЧИСЛА
    }

    private void LoadUserProfile()
    {
        NameEntry.Text = _currentUser.Name;
        LoginEntry.Text = _currentUser.Login;
        EmailEntry.Text = _currentUser.Email;
        PasswordEntry.Text = string.Empty;
```

```
            PasswordEntry.Placeholder = "Введіть новий пароль (залиште порожнім, щоб не змінювати)";
        }

        private void PopulateMenu()
        {
            // ПЕРЕЗАВАНТАЖУЄМО ЧИСЛА З БД — ЗАВЖДИ АКТУАЛЬНІ
            var routesCount = _dbContext.Routes
                .Count(r => r.Category.User.Id == _currentUser.Id);

            var tripsCount = _dbContext.Trips
                .Count(t => t.Route.Category.User.Id == _currentUser.Id);

            var categoriesCount = _dbContext.RouteCategories
                .Count(c => c.User.Id == _currentUser.Id);

            var menuItems = new List<MenuItemModel>
            {
                new() { Title = "Мої маршрути", PageType = typeof(RoutesListPage), Count = routesCount },
                new() { Title = "Виїзди", PageType = typeof(TripsListPage), Count = tripsCount },
                new() { Title = "Категорії", PageType = typeof(CategoriesListPage), Count = categoriesCount },
            };

            if (_currentUser.Company != null)
            {
                var sentMessages = _dbContext.Messages
                    .Count(m => m.Sender.Id == _currentUser.Id);

                var receivedMessages = _dbContext.Messages
                    .Count(m => m.Receiver.Id == _currentUser.Id);

                menuItems.AddRange(new[]
                {
                    new MenuItemModel { Title = "Вхідні повідомлення", PageType = typeof(IncomingMessagesPage), Count = receivedMessages },
                    new MenuItemModel { Title = "Надіслані повідомлення", PageType = typeof(SentMessagesPage), Count = sentMessages },
                    new MenuItemModel { Title = "Написати повідомлення", PageType = typeof(SendMessagePage), Count = 0 }
                });
            }

            MenuCollectionView.ItemsSource = menuItems;
        }

        private async void OnSaveProfileClicked(object sender, EventArgs e)
        {
            _currentUser.Name = NameEntry.Text?.Trim() ?? _currentUser.Name;
            _currentUser.Login = LoginEntry.Text?.Trim() ?? _currentUser.Login;
            _currentUser.Email = EmailEntry.Text?.Trim() ?? _currentUser.Email;

            if (!string.IsNullOrWhiteSpace(PasswordEntry.Text))
            {
                _currentUser.Password = BCrypt.Net.BCrypt.HashPassword(PasswordEntry.Text.Trim());
            }

            _dbContext.Users.Update(_currentUser);
            await _dbContext.SaveChangesAsync();
            await DisplayAlert("Успіх", "Профіль оновлено!", "OK");
        }

        private async void OnMenuItemClicked(object sender, EventArgs e)
        {
            if (sender is Button button && button.BindingContext is MenuItemModel menuItem)
            {
                var page = (Page)Activator.CreateInstance(menuItem.PageType, _currentUser);
                await Navigation.PushAsync(page);
            }
        }

        private async void OnLogoutClicked(object sender, EventArgs e)
        {
            bool confirm = await DisplayAlert("Вихід", "Вийти з облікового запису?", "Так", "Ні");
            if (confirm)
            {
                SecureStorage.Remove("Login");
                SecureStorage.Remove("Password");
                Application.Current!.MainPage = new NavigationPage(new LoginPage());
            }
        }
    }
}
```

```
using Microsoft.EntityFrameworkCore;
using System.Text.RegularExpressions;
using RouteOptimizerApp.Models;
using RouteOptimizerApp.Data;
using BCrypt.Net; // ПРАВИЛЬНИЙ using

namespace RouteOptimizerApp.Pages
{
    public partial class UserRegistrationPage : ContentPage
    {
        private readonly AppDbContext _dbContext = new();
        private bool _isCorporateMember = false;

        public UserRegistrationPage()
        {
            InitializeComponent();
            LoadCorporations();
        }

        private async void LoadCorporations()
        {
            var corporations = await _dbContext.Companies.ToListAsync();
            corporations.Add(new Company { Name = "Додати нову корпорацію" });
            CorporationPicker.ItemsSource = corporations;
        }

        private void OnCorporateMemberCheckedChanged(object sender, CheckedChangedEventArgs e)
        {
            _isCorporateMember = e.Value;
            CorporationPicker.IsVisible = _isCorporateMember;
            AddCorporationLabel.IsVisible = _isCorporateMember;
            NewCorporationEntry.IsVisible = false;
        }

        private void OnCorporationPickerSelectedIndexChanged(object sender, EventArgs e)
        {
            var picker = (Picker)sender;
            bool isAddNew = picker.SelectedIndex == picker.Items.Count - 1;
            NewCorporationEntry.IsVisible = _isCorporateMember && isAddNew;
        }

        private void OnBackToLoginClicked(object sender, EventArgs e)
        {
            Application.Current.MainPage = new NavigationPage(new LoginPage());
        }

        private async void OnRegisterButtonClicked(object sender, EventArgs e)
        {
            string name = NameEntry.Text?.Trim();
            string email = EmailEntry.Text?.Trim();
            string login = LoginEntry.Text?.Trim();
            string password = PasswordEntry.Text;
            string confirmPassword = ConfirmPasswordEntry.Text;

            if (!ValidateInputs(name, email, login, password, confirmPassword))
                return;

            // ХЕШУЄМО ПАРОЛЬ
            string passwordHash = BCrypt.Net.BCrypt.HashPassword(password);

            var newUser = new AppUser
            {
                Name = name,
                Email = email,
                Login = login,
                Password = passwordHash
            };

            if (_isCorporateMember)
            {
                var selected = CorporationPicker.SelectedItem as Company;
                if (selected?.Name == "Додати нову корпорацію")
                {
                    string newCorpName = NewCorporationEntry.Text?.Trim();
                    if (string.IsNullOrEmpty(newCorpName))
                    {
                        RegistrationErrorLabel.Text = "Введіть назву корпорації.";
                        RegistrationErrorLabel.IsVisible = true;
                        return;
```

```
                }
                var newCompany = new Company { Name = newCorpName };
                _dbContext.Companies.Add(newCompany);
                await _dbContext.SaveChangesAsync();
                newUser.Company = newCompany;
                newUser.Role = "Admin";
            }
            else if (selected != null)
            {
                newUser.Company = selected;
            }
        }

        var defaultCategories = new[]
        {
            new RouteCategory { Name = "Не визначено", User = newUser },
            new RouteCategory { Name = "Робота", User = newUser },
            new RouteCategory { Name = "Подорожі", User = newUser }
        };

        _dbContext.Users.Add(newUser);
        _dbContext.RouteCategories.AddRange(defaultCategories);
        await _dbContext.SaveChangesAsync();

        await DisplayAlert("Успіх", "Реєстрація завершена!", "OK");
        Application.Current.MainPage = new NavigationPage(new LoginPage());
    }

    private bool ValidateInputs(string name, string email, string login, string password, string confirmPassword)
    {
        if (string.IsNullOrWhiteSpace(name) || string.IsNullOrWhiteSpace(email) ||
            string.IsNullOrWhiteSpace(login) || string.IsNullOrWhiteSpace(password) ||
            string.IsNullOrWhiteSpace(confirmPassword))
        {
            ShowError("Усі поля обов'язкові.");
            return false;
        }
        if (!Regex.IsMatch(email, @"^[^@\s]+@[^@\s]+\.[^@\s]+$"))
        {
            ShowError("Невірний формат email.");
            return false;
        }
        if (password != confirmPassword)
        {
            ShowError("Паролі не співпадають.");
            return false;
        }
        if (password.Length < 8 || !password.Any(char.IsUpper) ||
            !password.Any(char.IsLower) || !password.Any(char.IsDigit))
        {
            ShowError("Пароль має бути ≥8 символів, містити великі/малі літери та цифру.");
            return false;
        }
        return true;
    }

    private void ShowError(string message)
    {
        RegistrationErrorLabel.Text = message;
        RegistrationErrorLabel.IsVisible = true;
    }
  }
}
```