НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО **Декан факультету**	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ** **Завідувач кафедри комп'ютерних наук**
_____ Ігор БОЛБОТ **(підпис)**	**_____ Белла Голуб** **(підпис)**
«___»_____ 2026 р.	**«___» _____ 2026 р.**

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

На тему: «Геоінформаційна система оптимізації маршрутів міського транспорту на основі просторових даних»

Спеціальність «122 Комп'ютерні науки»
Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми д.е.н., професор	_____ (підпис)	**Роман РУДЕНСЬКИЙ**
Керівник бакалаврської кваліфікаційної роботи (к.т.н., доцент)	_____ (підпис)	**Віталій СВАТКО**
Виконав	_____ (підпис)	**Богдан ЯКУБЕНКО**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук

к.т.н., доцент _____ Белла ГОЛУБ
(підпис)

«___» _____ 2026 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ

Якубенку Богдану Вячеславовичу
(прізвище, ім'я, по батькові)

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи

«Геоінформаційна система оптимізації маршрутів міського транспорту на основі просторових даних»

затверджена наказом від «_06_» ___січня__ 2026 р. № _46_«С»
Термін подання завершеної роботи на кафедру «25» _травня_ 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту): Геодані OpenStreetMap (м. Київ); стек: Python (FastAPI, PostGIS, OSMnx), React (Leaflet.js).

Перелік питань, що підлягають дослідженню:

1. Провести системний аналіз предметної області й сформулювати вимоги до системи.
2. Охарактеризувати теоретичні основи представлення транспортної інфраструктури у вигляді графів і обґрунтувати вибір технологічного стеку.
3. Спроєктувати інформаційну модель предметної області та логіко-фізичну схему просторової бази даних.
4. Реалізувати серверну частину веб-застосунку з API та алгоритмами обробки просторових даних (у тому числі маршрутизацію по графу доріг для відображення лінії маршруту, аналітичні процедури оптимізації мережі та зміну складу маршрутів за підтвердженням користувача).
5. Розробити клієнтський інтерфейс для карти, табличних панелей і графіків аналітики.

6. Описати реалізовані алгоритми, виконати тестування та оцінити коректність ключових сценаріїв.

Перелік графічних документів (за необхідності).

_____

Дата видачі завдання «06» ___січня____ 2026 р.

Керівник бакалаврської кваліфікаційної роботи	_____ (підпис)	**Віталій СВАТКО**
Завдання взяв до виконання	_____ (підпис)	**Богдан ЯКУБЕНКО**

ЗМІСТ

ВСТУП

Актуальність дослідження. Розвиток великих міст супроводжується зростанням мобільності, ущільненням транспортних коридорів і тиском на наявну інфраструктуру громадського транспорту. У парадигмі «розумного» міста та цифровізації міського середовища одних лише табличних зведень уже недостатньо: рішення доводиться перевіряти на карті, в метрах і в топології мережі, а не лише в рядках реєстру. Відстань між житловими масивами й зонами зосередження попиту, прогалини в обслуговуванні та накладення маршрутів на одні й ті самі вулиці впливають на якість обслуговування відчутно сильніше, ніж це видно з агрегованих показників без просторового контексту.

Практичні запити транспортної аналітики часто «роз'їжджаються» між довідниками зупинок, схемами руху й таблицями довжин; зіставити їх із реальною геометрією вулиць без ГІС-інструменту важко. Геоінформаційні технології дають спільну основу для збереження просторових об'єктів і атрибутів (тип руху, ідентифікатори, районні показники) та для виконання операцій буферизації, оцінки покриття, ізохронної доступності й сценарного порівняння. Відкриті дані OpenStreetMap у зв'язці з веб-архітектурою знижують поріг відтворюваності: методику обчислень легше пояснити й повторити, ніж у закритих комерційних стеках. У такому контексті розроблення веб-ГІС для просторового аналізу маршрутної мережі Києва виглядає інженерно доречним: місто має густу мережу ОТ і водночас — відкриті геодані, придатні для навчального та дослідницького прототипу.

Мета дослідження — розробити та програмно реалізувати веб-геоінформаційну систему для просторового аналізу маршрутної мережі міського пасажирського транспорту (на матеріалах Києва), що інтегрує відкриті геодані, просторову базу даних і серверні обчислення на дорожньому графі, а також забезпечує інтерактивну візуалізацію результатів у браузері.

Завдання дослідження:

1. Провести системний аналіз предметної області й сформулювати вимоги до системи.
2. Охарактеризувати теоретичні основи представлення транспортної інфраструктури у вигляді графів і обґрунтувати вибір технологічного стеку.
3. Спроєктувати інформаційну модель предметної області та логіко-фізичну схему просторової бази даних.
4. Реалізувати серверну частину веб-застосунку з API та алгоритмами обробки просторових даних (у тому числі маршрутизацію по графу доріг для відображення лінії маршруту, аналітичні процедури оптимізації мережі та зміну складу маршрутів за підтвердженням користувача).
5. Розробити клієнтський інтерфейс для карти, табличних панелей і графіків аналітики.
6. Описати реалізовані алгоритми, виконати тестування та оцінити коректність ключових сценаріїв.

Об’єкт дослідження — процеси збору, зберігання, просторового аналізу та візуалізації інформації про маршрутну мережу міського пасажирського транспорту.

Предмет дослідження — методи, моделі та програмні засоби побудови веб-ГІС для аналізу транспортних коридорів і пов’язаних показників доступності та покриття.

Методи та технології дослідження: системний аналіз (декомпозиція предметної області); аналітико-синтетичний підхід при опрацюванні стандартів і практик ГІС; елементи теорії графів для роботи з дорожнім графом і пошуку найкоротших шляхів на його підмножині; об’єктно-орієнтоване проєктування архітектури застосунку; моделювання даних для схеми PostgreSQL/PostGIS.

Наукова новизна полягає в узгодженому поєднанні в одній веб-архітектурі: просторового сховища й запитів PostGIS; серверних обчислень на графі доріг OSM (OSMnx/NetworkX, Python/Flask); інтерактивної картографії та діаграм у

браузері (Leaflet, Chart.js); автоматизованого формування PDF-звіту (ReportLab). Реалізовані сценарії включають оцінку покриття буферами, пошук «прогалин», аналіз потенційного дублювання коридорів, вузлові зупинки, рекомендації щодо мережі, злиття пари маршрутів із контролем правил моделі та показом агрегованих показників «до/після», а також сценарне порівняння гіпотетичних маршрутів із оцінкою приросту покриття з наближеним урахуванням демографії районів — без введення окремого модуля «перестановки зупинок», який не входить у поточну версію прототипу.

Практична цінність — у наявності працездатного прототипу, який демонструє повний цикл від даних OSM і БД до карти й звіту й може слугувати навчальним зразком або відправною точкою для подальшої адаптації під відомчі вимоги (за наявності інтеграції офіційних реєстрів і політик безпеки).

Апробація результатів. Основні положення та практичні результати бакалаврської кваліфікаційної роботи пройшли апробацію шляхом публікації тез доповіді у матеріалах VII Всеукраїнської науково-практичної конференції студентів та аспірантів «Теоретичні та прикладні аспекти розробки комп'ютерних систем «2026» (м. Київ, 2026 р.), а також були перевірені шляхом функціонального тестування реалізованого прототипу веб-ГІС.

Структура пояснювальної записки — вступ, три розділи, висновки, список використаних джерел (23 найменувань), додатки; загальний обсяг 82 сторінок. У першому розділі — теоретичні засади ГІС у транспорті; у другому — системний аналіз і проєктування БД; У третьому — реалізація, алгоритми та тестування; додатки, що містять лістинг коду (додаток А), керівництво користувача (додаток Б) та декларацію про академічну доброчесність і використання інструментів ШІ (додаток В).

РОЗДІЛ 1 ТЕОРЕТИКО-МЕТОДОЛОГІЧНІ ЗАСАДИ ЗАСТОСУВАННЯ ГЕОІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ У ТРАНСПОРТНИХ ДОСЛІДЖЕННЯХ

1.1 Концептуальні основи застосування геоінформаційних технологій у задачах оптимізації маршрутів міського транспорту на основі просторових даних

Зростання мегаполісів та концентрація населення детермінують громадський транспорт як критичну підсистему життєдіяльності міста [1]. Якість обслуговування території визначається не лише наявністю рухомого складу, а й просторовою впорядкованістю інфраструктури: топологією дорожньої мережі, взаємним розташуванням забудови та вузлових зупинок [22]. Ця «географічність» транспортних систем зумовлює необхідність залучення методів роботи з просторовими даними — узгодженим у системі координат описом об'єктів (точок, ліній, полігонів) та їхніх атрибутів [16].

У межах даної роботи оптимізація маршрутів розглядається як набір технічно відтворюваних операцій над просторовими сутностями: оцінка зон впливу зупинок, аналіз доступності, пошук дублювань та сценарне порівняння траєкторій на графі дорожньої мережі [8]. Такий підхід наближає поняття оптимізації до інженерної реальності програмної системи, що використовує конкретні алгоритми та джерела геопросторової інформації.

Геоінформаційні технології (ГІС) виступають аналітичним шаром, що дозволяє формалізувати просторові запити щодо взаємного розташування об’єктів та аналізу топології вулично-дорожньої мережі (вузлів, ребер та альтернативних шляхів) [9, 16]. Просторові дані тут є операційною основою, на якій базуються всі оптимізаційні процедури.

Окрему роль відіграють відкриті дані проєкту OpenStreetMap (OSM), які формують масив геоінформації про транспортну інфраструктуру. Використання

OSM дозволяє відтворити повний цикл підготовки графа мережі без залучення пропрієтарних сервісів [14]. Ефективне зберігання та обробка такої інформації реалізується через розширення PostGIS для СУБД PostgreSQL. Воно підтримує геометричні типи та просторові оператори згідно зі стандартами OGC, забезпечуючи цілісність транспортної логіки та територіальних показників [19].

Веб-орієнтована архітектура системи змінює характер аналітичної роботи: користувач отримує доступ до інструментів через браузер, тоді як сервер забезпечує виконання обчислень та взаємводію з базою даних **[13]**. Візуалізація на інтерактивній карті дозволяє оперативно порівнювати сценарні варіанти, забезпечуючи стратегічну підтримку прийняття рішень при оцінці структури мережі та плануванні нових маршрутів [18].

1.2 Математичні моделі транспортної мережі та методи просторового аналізу в задачах оптимізації маршрутів

Міський громадський транспорт у межах містобудівного та інфраструктурного планування розглядається як складна сукупність маршрутних ліній та зупинкової інфраструктури, що базуються на вулично-дорожній мережі. Дорожній каркас міста виступає фундаментальним обмежувачем: саме він визначає, які переміщення є фізично реалізованими, а які, попри геометричну близькість точок, залишаються топологічно недосяжними. Таким чином, наукове обґрунтування «оптимальності» будь-якого маршруту вимагає попередньої фіксації адекватної математичної моделі середовища руху.

Найбільш універсальним апаратом для опису таких систем є теорія графів. У межах даного дослідження транспортна мережа інтерпретується як зважений граф:

$$G = (V, E, W) \quad (1.1)$$

де:

- V — множина вузлів (перехрестя, зупинки);
- E — множина ребер (фізичні сегменти доріг);

- W — вагові коефіцієнти ребер.

Орієнтованість графа та його внутрішня топологія задаються правилами побудови конкретного середовища: для автомобільних доріг враховується напрямок руху, тоді як для пішохідних моделей граф частіше є неорієнтованим. У межах програмної реалізації роботи за базову метрику («вагу» ребра) приймається фізична довжина ділянки L у метрах. Такий підхід забезпечує стабільність розрахунків у статичній постановці задачі [9].

Центральною задачею при роботі з транспортним графом є знаходження найкоротшого шляху. Класичні методи, зокрема алгоритм Дейкстри, дозволяють знайти глобальний оптимум для графів із невід'ємними вагами ребер [7]. В інженерних ГІС-проєктах ефективність пошуку тісно пов'язана з коректністю підготовки вхідних даних. Топологічна зв'язність, відсутність розривів у місцях перехресть та уніфікація систем координат є обов'язковими умовами для того, щоб теоретично вірний алгоритм не став джерелом нереалістичних маршрутів. Відтак, етап валідації графа (аналіз орієнтації ребер та обробка дубльованих сегментів) стає невід'ємною частиною розробки [9].

Окрім дискретних моделей на графах, транспортний аналіз широко використовує операції безперервної геометрії. Типовим прийомом є буферизація — побудова зони заданого радіуса R навколо точкового об'єкта (зупинки) як метричного аналога його «зони впливу». При роботі з просторовими даними у форматі координат (широта/довгота) принципово використовувати типи даних geography у середовищі PostGIS, щоб забезпечити фізичну точність розрахунків відстаней на поверхні еліпсоїда [19].

Більш складним інструментом є побудова ізохрон. На відміну від радіального буфера, ізохрона на графі відображає реальну доступність території з урахуванням конфігурації вулиць. Множина вузлів, що входять до такої зони, визначається наступним чином:

$$V_{reach} = \{v \in V \mid d(start, v) \leq R_{limit}\} \quad (1.2)$$

Отримана множина ребер згодом апроксимується полігоном для візуалізації та подальшого аналізу покриття території [16].

Коли задача потребує впорядкованого відвідування множини точок, вона стає спорідненою із задачею комівояжера (TSP). Враховуючи високу обчислювальну складність точних методів, у програмній системі застосовуються евристичні стратегії, які забезпечують раціональний компроміс між якістю маршруту та швидкістю обробки запитів [3].

Для узагальненої оцінки схожості наборів зупинок двох маршрутів A та B (наприклад, для виявлення дублювання коридорів) використовується індекс Жаккара:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \quad (1.3)$$

Коефіцієнт, що наближається до одиниці, свідчить про значний ступінь схожості складів зупинок, що дозволяє приймати рішення про оптимізацію мережі [15].

Для прогнозування наслідків змін у мережі застосовується метод сценарного порівняння зон покриття. Приріст ефективної площі обслуговування ΔS (у км²) обчислюється як різниця полігональних об’єднань буферних зон «до» та «після» впровадження змін. Для наближеної оцінки кількості охопленого населення N (осіб) використовується середня щільність населення ρ_pop (осіб/км²) у межах району:

$$N = \Delta S \times \rho_{pop} \quad (1.4)$$

Така модель базується на припущенні про рівномірний розподіл населення, що є прийнятним рівнем абстракції для порівняльного аналізу різних сценаріїв трасування маршрутів за їхнім соціальним внеском [18].

1.3 Огляд сучасного програмного забезпечення для обробки просторової інформації

Розробка транспортної веб-ГІС вимагає інтеграції трьох рівнів: просторового сховища, модулів мережевих обчислень та клієнтського веб-інтерфейсу. Вибір технологічного стеку базується на критеріях функціональної

повноти, вартості ліцензування, відтворюваності алгоритмів та можливостей інтеграції.

Комерційні універсальні ГІС: ArcGIS Екосистема ArcGIS (Esri) (рис. 1.1) є галузевим еталоном для геоаналізу, що забезпечує потужний мережевий аналіз, топологічний контроль та хмарну інтеграцію. Однак висока вартість ліцензій, високі вимоги до апаратних ресурсів і закритість пропрієтарних алгоритмів роблять її недоцільною для незалежних дослідницьких проєктів [11]. У межах даної роботи ArcGIS розглядається виключно як функціональний орієнтир, а не інструмент реалізації.

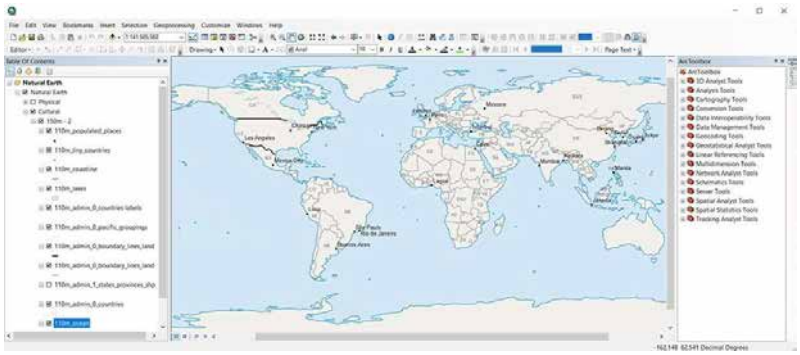

Рис. 1.1 — Приклад інтерфейсу системи ArcGIS (комерційна ГІС-платформа)

Відкриті універсальні ГІС: QGIS

QGIS (рис. 1.2) посідає провідне місце серед відкритих настільних ГІС завдяки підтримці багатьох форматів даних та розвиненій екосистемі плагінів [20]. Для транспортних досліджень її цінність полягає у можливості візуалізації векторних шарів, перевірки топологічних аномалій та підготовки картографічних макетів.

Суттєвою перевагою є прямий зв'язок із PostgreSQL/PostGIS, що дозволяє працювати з просторовими таблицями безпосередньо в середовищі QGIS та узгоджувати дані між середовищами розробки й аналізу [19]. Автоматизація обробки реалізується через PyQGIS. У межах створення веб-ГІС настільна

система залишається інструментом контролю та верифікації даних, тоді як основна логіка виноситься на серверний рівень.

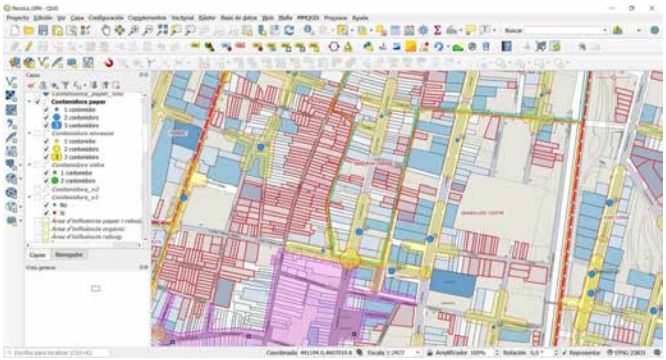

Рис. 1.2 — Приклад інтерфейсу системи QGIS (відкрита ГІС-платформа)

Спеціалізоване транспортне ПЗ: TransCAD та Maptitude

Для професійного транспортного планування сформувався клас продуктів, які будують аналітику навколо матриць поїздок, транспортних зон і рівноважного розподілу попиту. TransCAD (рис 1.3) є представником потужного транспортного моделювання, орієнтованого на інженерні масштаби: моделювання чотирьохетапного попиту, оптимізація розкладів і аналіз завантаженості мережі [5]. Maptitude (рис. 1.4) займає суміжну нішу — бізнес-аналіз, швидкі карти доступності та робота з типовими логістичними сценаріями.

Обидва продукти корисні для ілюстрації того, як у галузі формально описуються транспортні мережі та показники ефективності. Проте в контексті розробки власного продукту на відкритих компонентах необхідно враховувати монолітність і закритість частини інженерних сценаріїв, ліцензійні обмеження та залежність від вбудованих наборів постачальника.

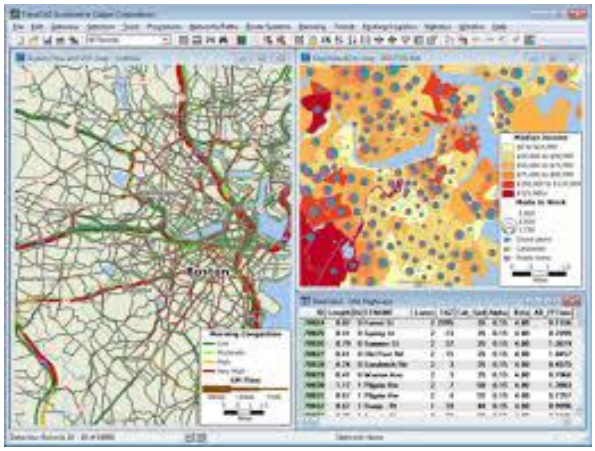

Рис. 1.3 — Приклад інтерфейсу системи TransCAD (спеціалізована ГІС для транспортного моделювання)

Commented [1]: Не має бути жирним

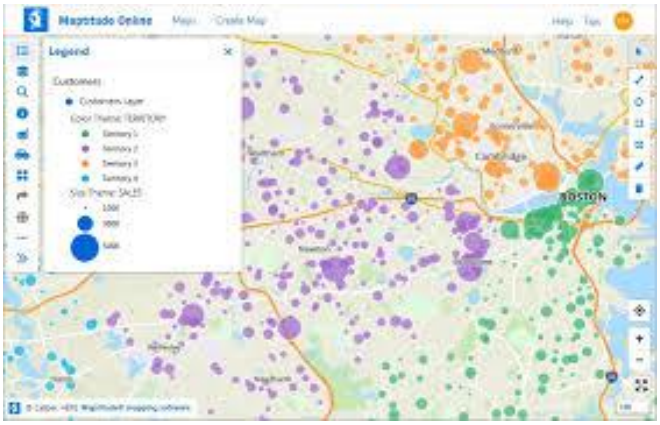

Рис. 1.4 — Приклад інтерфейсу системи Maptitude (комерційна платформа просторового планування)

Commented [2]: Немає бути жирним

Просторові бази даних і маршрутизація Сховище даних реалізується на базі PostgreSQL із розширенням PostGIS, що забезпечує підтримку геометричних типів, просторової індексації (GiST) та агрегуючих запитів на рівні СУБД [19]. Для обчислення найкоротших шляхів на графі замість внутрішньобазового

pgRouting застосовуються Python-бібліотеки NetworkX та OSMnx [4], що забезпечує гнучку кастомізацію алгоритмів та безшовну інтеграцію із серверним ядром на базі REST-фреймворку Flask.

Веб-інтерфейс та звітність Клієнтська картографія реалізується бібліотекою Leaflet [2], яка дозволяє інтерактивно відображати просторові шари (GeoJSON) та ізохрони у браузері. Візуальна статистика забезпечується бібліотекою Chart.js, а автоматизована генерація підсумкових документів — ReportLab.

Порівняльне узагальнення Переваги обраного спеціалізованого веб-орієнтованого стеку відносно інших класів рішень узагальнено в табл. 1.1.

Таблиця 1.1 – Порівняння ГІС

Commented [3]: Не виділяйте жирним

Критерій	ArcGIS	QGIS	TransCAD	PostgreSQL/PostGIS + Flask + Leaflet
Ліцензія	Комерційна	Відкрита (GPL)	Комерційна	Відкрита (FOSS)
Вартість	Висока	Безкоштовно	Висока	Безкоштовно
Мережевий аналіз	Network Analyst	Плагіни	Вбудований	OSMnx / NetworkX
Транспортні моделі	Обмежено	Обмежено	Повне	Кастомні (Python)
Веб-доступ	ArcGIS Online	Немає	Немає	Нативний (браузер)
Візуалізація / Звіти	Вбудована	Макети	Вбудована	Leaflet, Chart.js, ReportLab
Кастомізація	Обмежена	PyQGIS	Обмежена	Повна (через код)

Висновки до розділу

Проведений огляд показує, що жоден «універсальний пакет» не знімає необхідності осмисленого поєднання інструментів. Для реалізації системи оптимізації маршрутів обрано стек: PostgreSQL/PostGIS як просторове сховище, серверний застосунок на Flask, графові бібліотеки OSMnx та NetworkX, клієнтська картографія Leaflet, діаграми Chart.js та PDF-звітність через ReportLab. Така конфігурація відповідає інженерним компромісам між функціональною повнотою настільних ГІС і необхідністю предметної логіки, доступної користувачу через веб-інтерфейс.

РОЗДІЛ 2 ПРОЄКТУВАННЯ ТА РОЗРОБКА АРХІТЕКТУРИ СИСТЕМИ ОПТИМІЗАЦІЇ МІСЬКОГО ТРАНСПОРТУ

2.1 Системний аналіз предметної області та обґрунтування функціональних вимог до системи

Постановка задачі

Ефективне планування транспортної мережі потребує узгодження топології маршрутів, зон попиту та параметрів покриття території. Проблема полягає у неефективності неінтегрованого аналізу, зумовленій відсутністю єдиного сховища просторових об’єктів, складнощами сценарного моделювання та неможливістю автоматизованого контролю цілісності під час структурних змін мережі.

Метою роботи є розробка веб-орієнтованої геоінформаційної системи для автоматизації аналітичного циклу обслуговування транспортної мережі. Система повинна забезпечувати:

- централізоване зберігання даних у СУБД PostgreSQL/PostGIS;
- мережевий аналіз на основі графа OpenStreetMap;
- обчислення показників покриття, побудову ізохрон та виявлення дублювань;
- інструментарій для сценарного моделювання (зокрема злиття маршрутів) та формування звітності у форматі PDF.

Опис предметної області

Предметною областю дослідження є структура міської пасажирської мережі (зупинки, маршрути, районування) у взаємозв’язку з дорожньою інфраструктурою та демографічними показниками [22].

Організаційна модель експлуатації системи передбачає виконання трьох ключових функцій:

1. Планування: керування складом маршрутів, ініціювання структурних змін та сценарний аналіз.
2. Аналітика: оцінка територіального покриття, дослідження вузлових зупинок та демографічного контексту районів.
3. Контроль: моніторинг узагальнених показників мережі через систему звітності.

Просторову основу системи становить дорожній граф, сформований на базі даних OpenStreetMap. Граф імпортується у локальні файли та інтегрується із серверною частиною при запуску. Розроблений прототип орієнтований на режим дослідницької демонстрації, тому не містить підсистеми розмежування прав доступу, надаючи повний функціонал через єдиний веб-інтерфейс.

Функціональні вимоги

Функціональні вимоги сформульовано за фактично реалізованими можливостями системи.

Картографічне відображення зупинок громадського транспорту з фільтрацією за типом рухомого складу; відображення на карті наявних маршрутів та їхніх атрибутів.

Ведення довідника маршрутів і складу маршрутів: перегляд, створення, редагування та видалення маршрутів; додавання та виключення зупинок з урахуванням порядку проходження.

Побудова траєкторії маршруту на дорожній мережі між послідовними зупинками за критерієм найкоротшого шляху на підготовленому графі вулиць; розрахунок довжини маршруту.

Зведена статистика по мережі: кількість зупинок і маршрутів, сукупна та середня довжина, розподіл за типами транспорту; вибіркові рейтинги для аналізу (зокрема найдовші маршрути).

Ізохрона пішохідної доступності від заданої точки на основі фрагмента дорожньої мережі з використанням пішохідного графа або, за його відсутності, автомобільного графа з відповідним коригуванням швидкості переміщення.

Аналіз територіального покриття зупинками: об’єднання зон обслуговування у вигляді буферів, оцінка площі покриття; за потреби — відображення територій із нижчим покриттям відносно обмежувального контуру міста.

Районна аналітика: показники по адміністративних одиницях (кількість зупинок і маршрутів, площа тощо); тематичне відображення демографії та щільності населення на основі наявних довідкових даних.

Аналітика маршрутної мережі для виявлення вузьких місць: пошук пар маршрутів із високою схожістю коридорів за перетином множин зупинок; ранжування зупинок за кількістю маршрутів, що їх обслуговують; зведені рекомендації з урахуванням дублювань та районів із низькою забезпеченістю зупинками у перерахунку на населення.

Злиття двох маршрутів у моделі мережі за підтвердженням користувача: перевірка коректності запиту та допустимості операції за правилами моделі; збереження одного маршруту з об’єднаною послідовністю зупинок і видалення другого; відображення агрегованих показників впливу на мережу (зокрема довжини) до та після операції.

Евристичне уточнення порядку зупинок у межах одного маршруту з оцінкою зміни сумарної відстані між послідовними зупинками без претензії на глобально оптимальний розв’язок для всієї мережі.

Сценарний аналіз гіпотетичних маршрутів: створення, перегляд і видалення тимчасових конфігурацій; оцінка приросту площі покриття та наближеної зміни «охоплення» населення за моделлю районів.

Нефункціональні вимоги

Архітектура «клієнт–сервер»: основна взаємодія через браузер; обчислення та доступ до бази даних зосереджені на сервері [13].

Реалізація сервера: мова Python 3, веб-фреймворк Flask; обмін структурованими даними з клієнтом по протоколу HTTP у форматах JSON та GeoJSON.

Зберігання просторових даних: PostgreSQL із розширенням PostGIS; використання просторових індексів та предикатів для запитів щодо зупинок і районів.

Відкриті просторові дані: дорожній граф формується на основі OpenStreetMap через локальні файли графа, що забезпечує відтворюваність схеми підготовки даних.

Прозорість алгоритмів на графі: маршрутизація та побудова ізохрон виконуються бібліотеками NetworkX та OSMnx у коді застосунку.

Ресурси та час відгуку: система розрахована на локальне розгортання та демонстраційний режим; при старті виконується завантаження великих файлів графа. Для важких запитів припустимий інтерактивний режим очікування результату в клієнті; формальні договірні показники часу відгуку для прототипу не нормуються.

Клієнтська частина: HTML, CSS, JavaScript; бібліотека Leaflet для інтерактивної карти, Chart.js для діаграм; відсутність вимоги встановлення настільного ГІС на робочу станцію користувача.

Клас системи

Розроблюваний програмний комплекс належить до класу веб-геоінформаційних систем: просторові об’єкти відображаються на інтерактивній карті, дані зберігаються у просторовій СУБД, результати аналізу передаються у вигляді структур GeoJSON та зведених таблиць [16].

За характером підтримуваних процесів система є аналітичною системою підтримки прийняття рішень орієнтованого типу: вона не призначена для оперативного диспетчерування руху в реальному часі; натомість забезпечує обчислення та візуалізацію показників, корисних для оцінки покриття, виявлення потенційного дублювання коридорів, порівняння сценаріїв гіпотетичних маршрутів і застосування обмеженого набору структурних змін до моделі маршрутної мережі (злиття маршрутів) з поясненням наслідків для узагальнених метрик, а також формування звіту.

За рівнем автоматизації це інформаційно-аналітична система з елементами сценарного моделювання гіпотетичних маршрутів. За своїм профілем вона не є універсальною інформаційно-пошуковою системою довільного призначення: добір і аналіз даних підпорядковані сутностям предметної області (зупинка, маршрут, район).

2.2 Моделювання архітектури програмного забезпечення з використанням засобів UML

Сучасна інженерна практика розроблення програмних систем передбачає не лише реалізацію коду, а й поетапну формалізацію знань про систему для учасників проєкту та для подальшого супроводу й експлуатації. Одним із загально прийнятих інструментів такої формалізації є UML (Unified Modeling Language) — набір стандартизованих графічних нотацій, що описують статику й динаміку, структури й поведінку, а також конфігурацію програмних артефактів відносно вузлів виконання. [17]

Моделювання архітектури програмного забезпечення за допомогою UML дозволяє наочно зафіксувати статичні та динамічні аспекти системи: межі відповідальності, поведінкові сценарії, послідовність взаємодій між частинами ПЗ та конфігурацію розгортання. У межах кваліфікаційної роботи побудова UML-моделей узгоджується з результатами системного аналізу (розділ 2.1) і слугує проміжним рівнем між формулюванням вимог та описом програмної реалізації: на діаграмах відображаються ті самі функціональні можливості, але вже у вигляді взаємодій, процесів або структурних елементів. Це зменшує ймовірність розбіжностей між текстовим описом вимог, архітектурними схемами та фактичною організацією коду й сценарії клієнт–сервер.

Діаграма прецедентів відображає функціональні вимоги до системи та взаємодію акторів із її основними функціями (рис. 2.1). Діаграма діяльності описує логіку виконання ключових процесів у вигляді послідовності дій і

умовних переходів (рис. 2.3). Діаграма послідовності ілюструє взаємодію компонентів системи в часі при обробці конкретних запитів (рис. 2.2). Організаційна структура програмного забезпечення та розподіл компонентів за пакетами представлені на діаграмі пакетів (рис. 2.4). Діаграма розміщення визначає фізичну топологію системи — вузли розгортання та розподіл програмних компонентів між ними (рис. 2.5).

Діаграма прецедентів

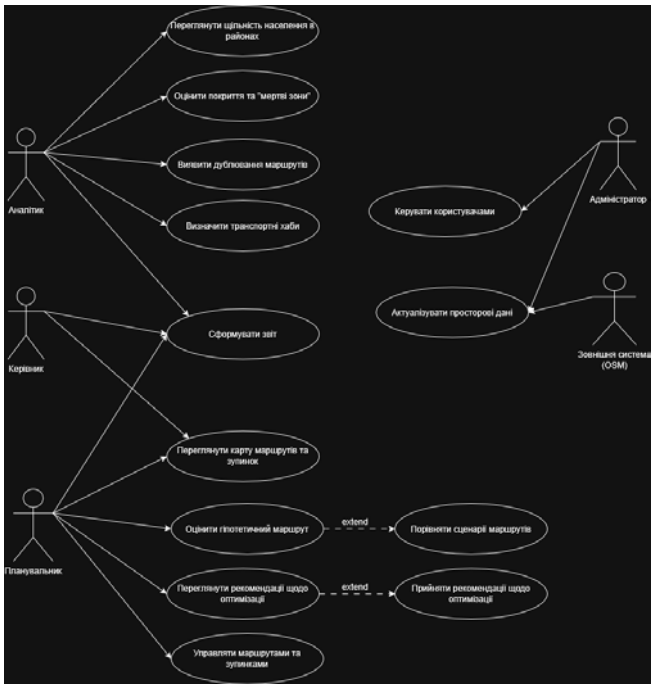

Рис. 2.1 – Діаграма прецедентів

Діаграма прецедентів (Use Case Diagram) відображає функціональні можливості веб-геоінформаційної системи з позиції її користувачів і зовнішніх акторів взаємодії [17]. Межа системи відокремлює реалізовану програмну поведінку від зовнішнього організаційного та технологічного середовища. Нижче наведено

структурований опис акторів, прецедентів і зв’язків між ними, узгоджений із результатами системного аналізу та обмеженнями навчального прототипу.

Мета діаграми та склад акторів

Метою моделі прецедентів є формалізація зовнішнього контуру функціональності системи без деталізації реалізації. Виділено чотири ролі користувачів: аналітик, планувальник, керівник, адміністратор, а також зовнішній актор OpenStreetMap (OSM) як джерело просторових даних. Ролі відображають функціональний розподіл у предметній області та не обов’язково відповідають окремим обліковим записам у реалізації.

Прецеденти аналітика

Аналітик виконує діагностичні функції:

- аналіз щільності населення в районах;
- оцінка покриття та «мертвих зон»;
- виявлення дублювання маршрутів;
- визначення транспортних хабів;
- формування звітів.

Прецеденти планувальника

Планувальник відповідає за моделювання та коригування маршрутів:

- управління маршрутами та зупинками;
- перегляд карти маршрутів;
- перегляд рекомендацій щодо оптимізації;
- оцінка гіпотетичних маршрутів.

Прецедент «оцінка гіпотетичного маршруту» має зв’язок <> з «порівнянням сценаріїв маршрутів», що означає опціональне розширення функціональності [17]. Аналогічно, «перегляд рекомендацій щодо оптимізації» може бути розширений прецедентом «прийняття рекомендацій щодо оптимізації».

Прецеденти керівника

Керівник виконує узагальнюючі та контрольні функції:

- перегляд карти маршрутів та зупинок;
- формування звітів.

Спільні прецеденти

Прецедент формування звіту є спільним для аналітика, планувальника та керівника, що відображає універсальність звітності як загального результату роботи системи.

Прецеденти адміністратора та взаємодія з OSM

Адміністратор забезпечує інфраструктурні функції:

- керування користувачами;
- актуалізація просторових даних.

Зовнішній актор OSM використовується як джерело відкритих просторових даних для побудови та оновлення дорожнього графа [14].

Узгодження з реалізованим прототипом

У навчальному прототипі відсутні підсистема автентифікації та розмежування ролей, тому всі функції доступні через єдиний веб-інтерфейс. Прецедент керування користувачами розглядається як елемент промислового розширення.

Актуалізація даних OSM реалізована не як інтерактивний процес, а як офлайн-процедура підготовки та оновлення графових даних. Таким чином, діаграма відображає цільову (узагальнену) модель системи, а не лише її спрощену навчальну реалізацію.

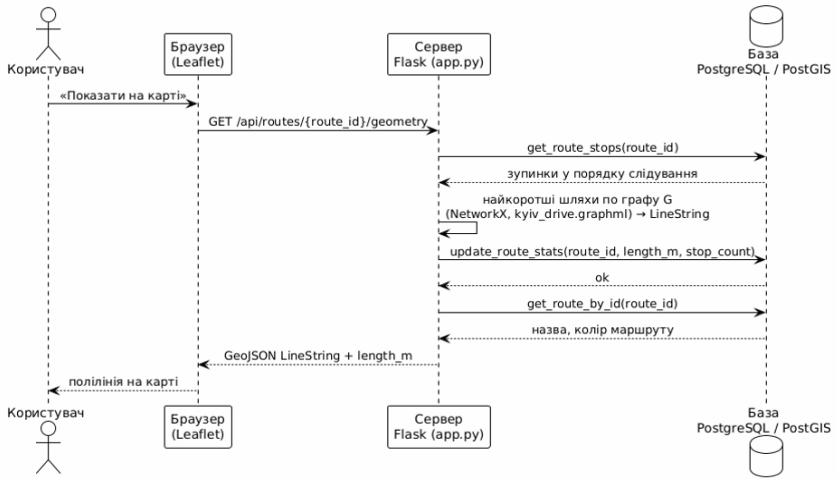

Рис. 2.2 – Діаграма послідовності побудови геометрії маршруту

На діаграмі наведено сценарій візуалізації лінії маршруту громадського транспорту. Учасниками взаємодії є користувач, клієнтський веб-інтерфейс (бібліотека Leaflet), сервер застосунку (Flask) та сервер СУБД (PostgreSQL/PostGIS).

Процес ініціюється вибором маршруту в інтерфейсі, після чого браузер надсилає HTTP-запит до сервера для отримання геометрії. Серверна логіка зчитує з бази даних упорядкований список зупинок, на основі якого виконується розрахунок ділянок шляху на графі дорожньої мережі міста. Отримані сегменти зшиваються в єдину просторову лінію. Паралельно сервер оновлює в СУБД похідні показники маршруту (загальну довжину та кількість зупинок) відповідно до актуальної траєкторії.

Результат обчислень передається клієнту у структурованому форматі, що містить лінійну геометрію та атрибути стилізації (назву, колір тощо). Браузер накладає об’єкт на картографічну основу, забезпечуючи візуалізацію маршруту вздовж вулично-дорожньої мережі. Діаграма демонструє повний цикл обробки запиту: від інтерфейсної події до верифікації даних у сховищі та фінальної візуалізації.

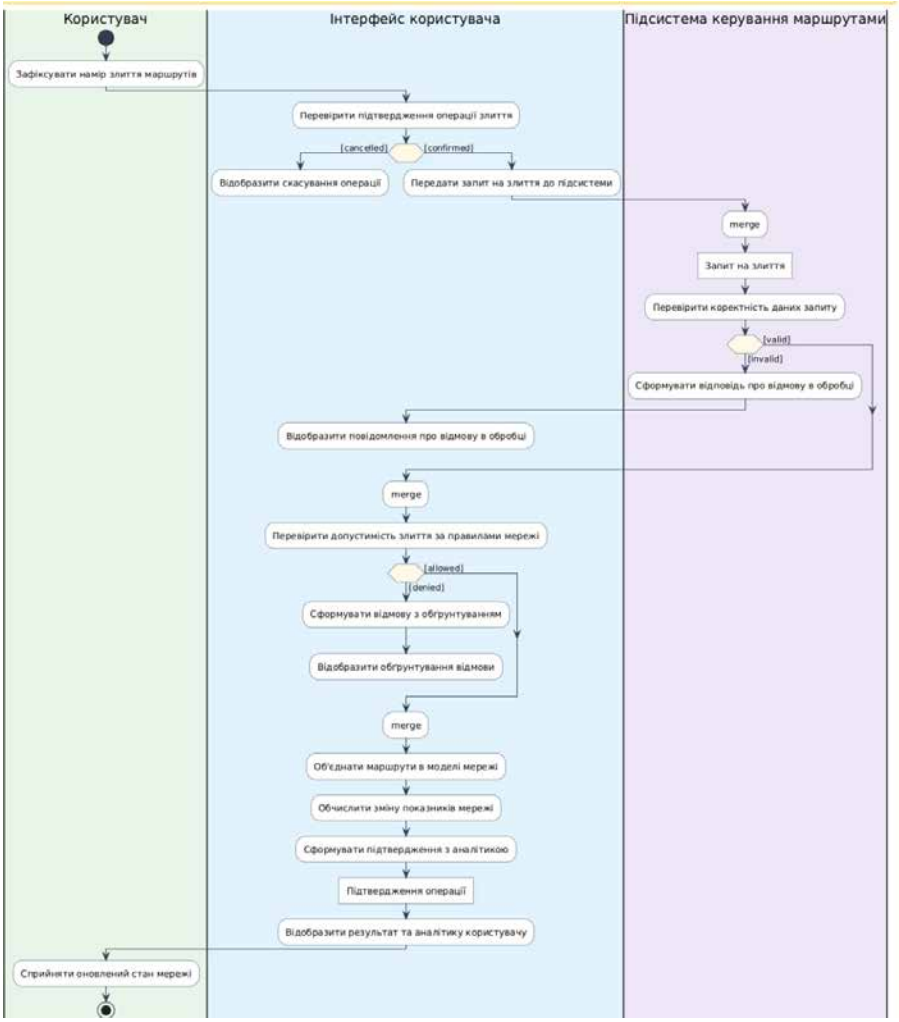

Рис. 2.3 – Діаграма діяльності: Алгоритм злиття схожих маршрутів

Опис сценарію злиття маршрутів (Activity Diagram)

На рис. 2.3 представлено діаграму діяльності, що моделює процес злиття двох маршрутів у мережі громадського транспорту. Сценарій декомпоновано на три зони відповідальності (swimlanes) [17]: «Користувач», «Інтерфейс користувача» та «Підсистема керування маршрутами».

Процес розпочинається з фіксації наміру користувача та обов'язкового підтвердження операції через вузол прийняття рішення (DecisionNode). У разі скасування потік завершується локально (Flow Final) із відповідним сповіщенням; при підтвердженні — інтерфейс передає об'єкт «Запит на злиття» (ObjectNode) до серверної частини.

Підсистема керування маршрутами виконує дворівневу верифікацію: спочатку технічну валідацію ідентифікаторів, потім перевірку бізнес-обмежень через вузол злиття (MergeNode). Будь-яка невідповідність на будь-якому рівні формує обґрунтовану відмову, що відображається в інтерфейсі.

Після проходження перевірок система виконує об'єднання маршрутів та обчислює похідні метрики. Результат інкапсулюється в об'єкт «Підтвердження операції» (ObjectNode) із агрегованою аналітикою впливу злиття на показники мережі. Єдиний вузол завершення діяльності (ActivityFinalNode) відповідає виключно успішному сценарію; відмови та скасування позначені як локальні зупинки потоку, що забезпечує чітке розмежування цільового сценарію та виняткових ситуацій.

Також на рис. 2.4 наведено діаграму пакетів.

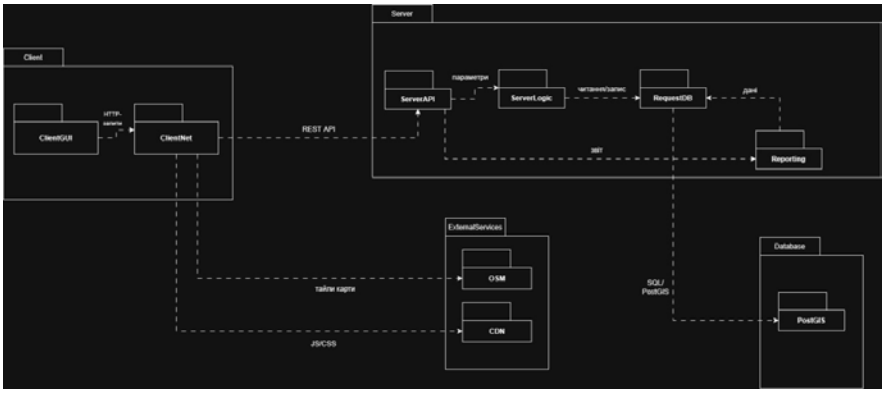

Рис 2.4 – Діаграма пакетів

Залежності на діаграмі пакетів визначають ієрархію взаємодії: клієнтська частина звертається до серверного API за протоколом REST (12), тоді як

серверна логіка взаємодіє з базою даних через виділений шар доступу. Зовнішні пакети забезпечують отримання картографічних тайлів та статичних ресурсів. Діаграма пакетів фіксує статичну організацію програмних елементів, доповнюючи динамічні моделі алгоритмів.

Перед деталізацією програмних модулів визначено розподіл функцій між вузлами системи. У межах геоінформаційного застосунку чітко розмежовано рівні просторових обчислень, серверної логіки та візуалізації. Зовнішнє джерело OpenStreetMap використовується виключно для формування графової основи, не замінюючи внутрішні довідники маршрутів.

Діаграма розміщення (рис. 2.5) узагальнює системну конфігурацію, що включає веб-браузер, сервер застосунку (модулі обробки та звітності), сервер СУБД та зовнішній сервіс геоданих. На діаграмі визначено протоколи взаємодії (HTTP/HTTPS, доступ до БД) та ключові компоненти всередині вузлів. Дана модель є архітектурним орієнтиром для подальшого опису програмної реалізації та інтерфейсів обміну даними.

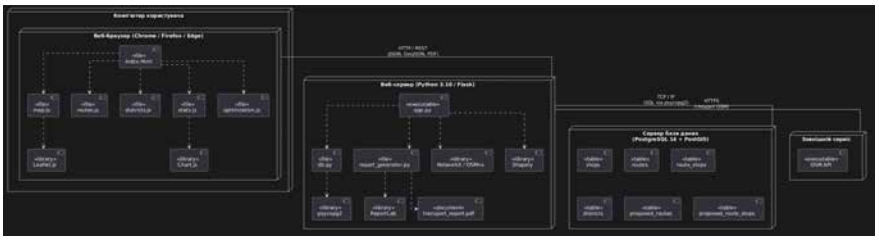

Рис. 2.5 – Діаграма розміщення компонентів веб-орієнтованої ГІС

Діаграма розміщення відображає архітектурний розподіл системи: клієнтська частина забезпечує візуалізацію та введення даних, тоді як сервер застосунку виконує основний обсяг обчислень. Такий підхід мінімізує вимоги до клієнтського середовища та гарантує цілісність бізнес-логіки. Сервер СУБД із розширенням PostGIS реалізує стійке зберігання та обробку просторових

об'єктів, а інтеграція з OpenStreetMap дозволяє оновлювати графову основу незалежно від внутрішньої моделі даних.

Застосований комплекс UML-моделювання дозволив формалізувати перехід від вимог до технічної реалізації через чотири взаємодоповнювані подання:

діаграма прецедентів визначає функціональні межі системи та ролі користувачів;

діаграма послідовності деталізує взаємодію компонентів під час побудови геометрії маршрутів;

діаграма діяльності описує алгоритмічний сценарій злиття маршрутів з урахуванням перевірок коректності;

діаграма розміщення фіксує топологію вузлів (браузер, Flask, PostgreSQL) та канали обміну даними.

Систематизація цих моделей забезпечує цілісність архітектурних рішень і створює підґрунтя для опису програмної реалізації та структури бази даних у наступних розділах.

2.3 Проєктування логічної та фізичної моделей бази просторових даних

Розробка програмного комплексу для картографічного аналізу транспортних мереж потребує чіткої структуризації даних. У роботі реалізовано розподіл на логічний рівень (сутності, зв’язки та бізнес-обмеження) та фізичний рівень (таблиці, індекси, тригери та специфічні типи PostGIS). Логічна модель розроблена відповідно до постановки задачі та візуалізована у формі ER-діаграми [6], тоді як фізична модель втілена у схемі transport СУБД PostgreSQL.

Логічна модель даних

На логічному рівні систему описують базові поняття предметної області: маршрут, зупинка, послідовність обходу зупинок, територіальні райони та сценарні моделі маршрутів. Атрибутивний склад подано узагальненими типами

даних (ідентифікатори, рядки, числові значення та просторові об'єкти «точка» і «мультиполігон») без прив’язки до специфікацій конкретної СУБД.

Ключова сутність «Маршрут» агрегує технічні та візуальні характеристики лінії руху: найменування, режим роботи, довжину та кількість зупинок. Ці показники є критичними для оперативного доступу в аналітичних інтерфейсах та автоматизованій звітності. Логічні атрибути сутності систематизовано у табл. 2.1.

Таблиця 2.1 — Логічні атрибути сутності «Маршрут»

Атрибут	Логічний тип	Призначення та обмеження
Ідентифікатор маршруту	Унікальний ідентифікатор	Первинний ключ сутності.
Назва	Рядок	Обов’язкове поле; відображається користувачу.
Код режиму руху	Перелік допустимих значень	Обмеження предметної області: автобус, тролейбус, трамвай, метро.
Колір відображення	Рядок	Необов’язкове поле для картографічного шару.
Опис	Рядок	Необов’язкове довідкове поле.
Загальна довжина	Ціле число (метри)	Похідний показник довжини траєкторії маршруту.
Кількість зупинок	Ціле число	Похідний показник кількості зупинок у маршруті.
Мітка часу створення	Дата й час	Фіксує момент занесення запису.

Сутність «Зупинка» є просторовим довідником зупинок із координатами та геометрією точки в географічній системі координат; передбачено опційне групування через посилання на «батьківську» зупинку. Склад атрибутів наведено у табл. 2.2.

Таблиця 2.2 — Логічні атрибути сутності «Зупинка»

Атрибут	Логічний тип	Призначення та обмеження
Ідентифікатор зупинки	Унікальний ідентифікатор	Первинний ключ сутності.
Назва	Рядок	Обов’язкове поле.
Ідентифікатор об’єкта в OSM	Ціле число	Необов’язкове поле; має бути унікальним, якщо задане.
Широта, довгота	Дійсне число	Обов’язкові координати; погоджені з геометрією.
Геометрія точки	Просторовий тип «точка»	Обов’язкове поле для просторових операцій.
Ідентифікатор батьківської зупинки	Унікальний ідентифікатор	Необов’язкове поле; ієрархія або узгодження дублікатів.

Асоціативна сутність «Зупинка в маршруті» розв'язує зв'язок типу багато-до-багатьох між маршрутом і зупинкою та зберігає порядок обходу зупинок (табл. 2.3).

Таблиця 2.3 — Логічні атрибути асоціативної сутності «Зупинка в маршруті»

Атрибут	Логічний тип	Призначення та обмеження
Ідентифікатор запису	Унікальний ідентифікатор	Первинний ключ асоціації.
Маршрут	Унікальний ідентифікатор	Зв’язок із сутністю «Маршрут».
Зупинка	Унікальний ідентифікатор	Зв’язок із сутністю «Зупинка».
Порядковий номер	Ціле число	Порядок зупинки в маршруті; має бути визначеним у межах маршруту.

Сутність «Район» є територіальним довідником із полігональною геометрією та показниками для аналітики; її атрибути наведено у табл. 2.4.

Таблиця 2.4 — Логічні атрибути сутності «Район»

Атрибут	Логічний тип	Призначення та обмеження
Ідентифікатор району	Унікальний ідентифікатор	Первинний ключ сутності.
Назва	Рядок	Обов’язкове поле; унікальне в межах довідника.
Населення	Ціле число	Необов’язковий показник.
Площа	Дійсне число	Необов’язковий показник (наприклад, км²).

Таблиця 2.4 (продовження)

Атрибут	Логічний тип	Призначення та обмеження
Геометрія меж	Просторовий тип «мультиполігон»	Обов’язкове поле для просторових запитів.

Сутність «Гіпотетичний маршрут» призначена для опису альтернативного або проєктного маршруту окремо від обліку діючих маршрутів; послідовність зупинок задається тією ж логікою, що й для діючого маршруту, з окремими обмеженнями унікальності. Атрибути сутності наведено у табл. 2.5, атрибути асоціації зупинок гіпотетичного маршруту — у табл. 2.6.

Таблиця 2.5 — Логічні атрибути сутності «Гіпотетичний маршрут»

Атрибут	Логічний тип	Призначення та обмеження
Ідентифікатор гіпотетичного маршруту	Унікальний ідентифікатор	Первинний ключ сутності.
Назва	Рядок	Обов’язкове поле.
Код режиму руху	Перелік допустимих значень	Як для сутності «Маршрут».
Примітки	Рядок	Необов’язкове поле.
Мітка часу створення	Дата й час	Фіксує момент занесення запису.

Таблиця 2.6 — Логічні атрибути асоціації «Зупинка в гіпотетичному мар шруті»

Атрибут	Логічний тип	Призначення та обмеження
Ідентифікатор запису	Унікальний ідентифікатор	Первинний ключ асоціації.
Гіпотетичний маршрут	Унікальний ідентифікатор	Зв’язок із сутністю «Гіпотетичний маршрут».
Зупинка	Унікальний ідентифікатор	Зв’язок із довідником «Зупинка».
Порядковий номер	Ціле число	Порядок зупинки; унікальний у межах одного гіпотетичного маршру ту.

Зв’язки між сутностями

Між маршрутом і зупинкою реалізується зв’язок M:N через асоціативну с утність із атрибутом порядку. Для зупинки передбачено опційний зв’язок типу рекурсивний (посилання зупинки на батьківську зупинку) з семантикою групув ання або узгодження записів довідника. Район є самостійним просторовим дові дником; зв’язок маршрутів і зупинок із районами в логічній моделі встановлюєт ься переважно через просторове накладання геометрій, а не через обов’язковий зовнішній ключ. Гіпотетичний маршрут пов’язаний із зупинками асоціацією M: N за тим самим принципом, що й діючий маршрут, що дозволяє зберігати проєк тні варіанти без зміни основного обліку.

Структуру бази ілюструє ER діаграма (рис. 2.6), на якій наведено сутності та кардинальність зв’язків без деталізації фізичн их типів полів СУБД

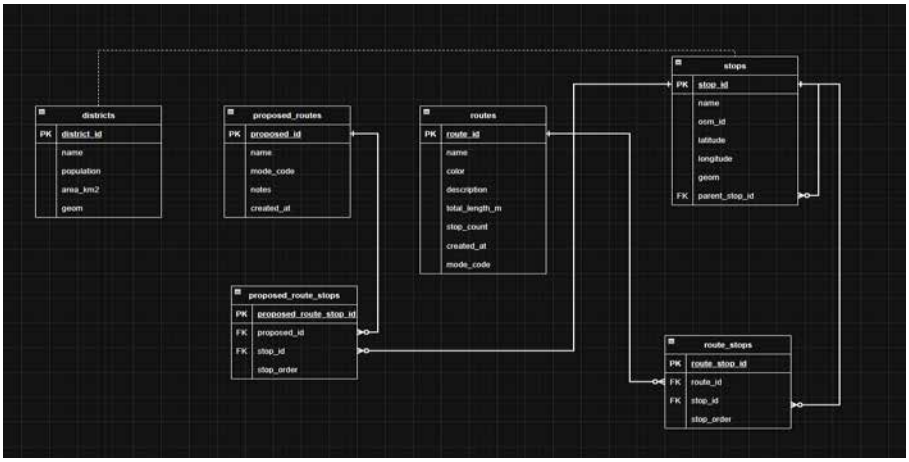

Рис. 2.6 – Логічна модель бази даних транспортної системи

Логічна модель системи узгоджена з функціоналом прототипу та охоплює облік маршрутів, зупинок, територіальних меж районів і сценарних моделей. Для оптимізації просторових запитів PostGIS координати зупинок зберігаються дубльовано: як числові значення та як геометричні об’єкти в SRID 4326. Це виключає постійний перерахунок геометрії під час звернень.

Фізична модель реалізована в PostgreSQL з використанням розширення PostGIS. Первинні та зовнішні ключі представлені цілочисельними типами (SERIAL), а для режимів руху встановлено обмеження CHECK. Просторові поля типу geometry супроводжуються індексами GiST на геометріях зупинок і районів [19]. Кількість зупинок у маршруті підтримується тригерами при зміні таблиць зв'язку [21], тоді як показник загальної довжини оновлюється на рівні застосунку після побудови фактичної траєкторії на графі. Специфікацію центральної таблиці ядра сховища transport.routes наведено в табл. 2.6.

Таблиця 2.6 — Фізична структура таблиці transport.routes

Фізичне ім’я поля	Тип даних СУБД	Обмеження	Призначення
route_id	SERIAL	PRIMARY KEY	Первинний ключ маршруту
name	TEXT	NOT NULL	Назва маршруту
mode_code	TEXT	NOT NULL, DEFAULT 'bus', CHECK (bus/trolleybus/tram /metro)	Код виду рухомого складу
color	TEXT	—	Колір лінії на карті
description	TEXT	—	Довідковий опис
total_length_m	INTEGER	DEFAULT 0	Довжина маршруту, м (похідний показник)
stop_count	INTEGER	DEFAULT 0	Кількість зупинок (похідний показник)
created_at	TIMESTAMP	NOT NULL, DEFAULT NOW()	Час створення запису

Просторовий довідник зупинок реалізовано у таблиці transport.stops. Поле geom типу geometry(Point, 4326) є обов'язковим і використовується у всіх просторових запитах; поле parent_stop_id забезпечує ієрархічне групування пов'язаних зупинок (табл. 2.7).

Таблиця 2.7 — Фізична структура таблиці transport.stops

Фізичне ім’я поля	Тип даних СУБД	Обмеження	Призначення
stop_id	SERIAL	PRIMARY KEY	Первинний ключ зупинки
name	TEXT	NOT NULL	Назва зупинки
osm_id	BIGINT	UNIQUE	Ідентифікатор вузла OSM (якщо задано)
latitude	DOUBLE PRECISION	NOT NULL	Широта
longitude	DOUBLE PRECISION	NOT NULL	Довгота
geom	geometry(Point,4326)	NOT NULL	Точка в WGS 84 для PostGIS
parent_stop_id	INTEGER	FK→ stops(stop_id) ON DELETE SET NULL	Групування/узгодження пов’язаних зупинок

Зв'язок між маршрутами та зупинками реалізовано через асоціативну таблицю transport.route_stops. Вона зберігає порядок обходу зупинок і забороняє

дублювання однієї зупинки в межах одного маршруту через обмеження унікальності на пару (route_id, stop_id), як показано у табл. 2.8.

Таблиця 2.8 — Фізична структура таблиці transport.route_stops

Фізичне ім’я поля	Тип даних СУБД	Обмеження	Призначення
route_stop_id	SERIAL	PRIMARY KEY	Первинний ключ запису зв’язку
route_id	INTEGER	NOT NULL, FK → routes, ON DELETE CASCADE	Маршрут
stop_id	INTEGER	NOT NULL, FK → stops, ON DELETE CASCADE	Зупинка
stop_order	INTEGER	NOT NULL, DEFAULT 1	Порядковий номер у маршруті
—	—	UNIQUE (route_id, stop_id)	Заборона дублювання зупинки в одному маршруті

Територіальний довідник районів зберігається у таблиці transport.districts. Геометрія меж типу geometry(MultiPolygon, 4326) є обов'язковим полем і

використовується у просторових запитах ST_Within при аналізі покриття по районах (табл. 2.9).

Таблиця 2.9 — Фізична структура таблиці transport.districts

Фізичне ім’я поля	Тип даних СУБД	Обмеження	Призначення
district_id	SERIAL	PRIMARY KEY	Первинний ключ району
name	TEXT	NOT NULL, UNIQUE	Назва району
population	INTEGER	—	Населення (довідково)
area_km2	DOUBLE PRECISION	—	Площа (довідково)
geom	geometry(MultiPolygon,4326)	NOT NULL	Межі району

Для підтримки сценарного аналізу передбачено окрему таблицю transport.proposed_routes, яка зберігає гіпотетичні маршрути незалежно від діючих. Структура таблиці наведена у табл. 2.10.

Таблиця 2.10 — Фізична структура таблиці transport.proposed_routes

Commented [4]: Розриви таблиці зробіть згідно вимог
Продовження табл.2.10

Фізичне ім’я поля	Тип даних СУБД	Обмеження	Призначення
proposed_id	SERIAL	PRIMARY KEY	Первинний ключ гіпотетичного маршруту

Таблица 2.10 (продовження)

Фізичне ім’я поля	Тип даних СУБД	Обмеження	Призначення
name	TEXT	NOT NULL	Назва сценарію
mode_code	TEXT	NOT NULL, DEFAULT 'bus', CHECK	Режим руху
notes	TEXT	—	Примітки
created_at	TIMESTAMP	NOT NULL, DEFAULT NOW()	Час створення

Послідовність зупинок гіпотетичного маршруту зберігається у таблиці transport.proposed_route_stops. На відміну від основної таблиці зв'язку, тут передбачено два обмеження унікальності — на пару (proposed_id, stop_order) та на пару (proposed_id, stop_id), що гарантує як унікальність позиції, так і відсутність дублювання зупинок у межах одного сценарію (табл. 2.11).

Таблиця 2.11 — Фізична структура таблиці transport.proposed_route_stops

Фізичне ім’я поля	Тип даних СУБД	Обмеження	Призначення
proposed_route_stop_id	SERIAL	PRIMARY KEY	Первинний ключ запису
proposed_id	INTEGER	NOT NULL, FK → proposed_routes, ON DELETE CASCADE	Гіпотетичний маршрут

Таблиця 2.11 (продовження)

Фізичне ім’я поля	Тип даних СУБД	Обмеження	Призначення
stop_id	INTEGER	NOT NULL, FK → stops, ON DELETE CASCADE	Зупинка з довідника
stop_order	INTEGER	NOT NULL	Порядок у сценарії
—	—	UNIQUE (proposed_id, stop_order), UNIQUE (proposed_id, stop_id)	Цілісність послідовності

Для наочності нижче наведено фізичну ER-діаграму (рис. 2.7), на якій відображено таблиці схеми transport, типи полів і напрямки зовнішніх ключів. Каскадне видалення для залежних записів (route_stops, proposed_route_stops) зменшує ризик осиротілих рядків при очищенні маршрутів або сценарних конфігурацій; для parent_stop_id обрано ON DELETE SET NULL, щоб не

руйнувати довідник зупинок при вибірковому прибиранні «батьківських» записів.

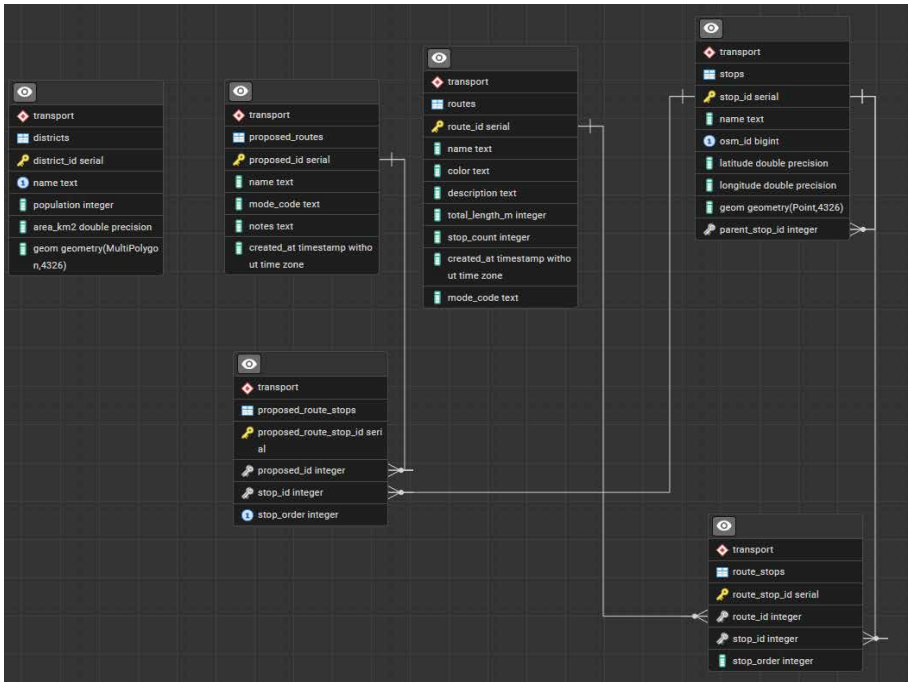

Рис 2.7 — Фізична модель бази даних транспортної системи

Завершення опису фізичної моделі даних означає, що структура сховища узгоджена з прийнятими рішеннями щодо сутностей, ключів і просторових типів і може бути безпосередньо реалізована в PostgreSQL/PostGIS. Довідникові таблиці маршрутів, зупинок і районів та асоціативні таблиці послідовностей зупинок побудовані так, щоб уникнути зайвого дублювання змістових даних: координати й геометрія зупинки задаються в одному записі таблиці stops, а участь зупинки в маршруті відображається окремими рядками зв’язку. Разом із тим у таблиці routes наявні похідні показники довжини та кількості зупинок; вони свідомо денормалізовані для прискорення типових вибірок і відображення в інтерфейсі, тому формулювання про «ідеальну» третю нормальну форму

для всієї схеми було б некоректним — коректніше говорити про узгодженість основної частини структури з принципами нормалізації та про цілеспрямовані відхилення там, де це обґрунтовано вимогами продуктивності.

Числові поля широти й довготи визначено з підвищеною точністю (DOUBLE PRECISION), додатково використовується поле geometry(Point, 4326), що забезпечує узгодженість із PostGIS і подальшими просторовими операціями (буфери, перетини з полігонами районів тощо). Розрахунки відстаней і побудова траєкторій між зупинками виконуються в програмному модулі серверної частини на основі графа дорожньої мережі; база даних при цьому постачає упорядковані координати та довідникові атрибути, необхідні для цих обчислень, — їхній порядок і деталізація розглядаються в наступних розділах записки разом із описом реалізації алгоритмів.

РОЗДІЛ 3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ГІС

3.1Реалізація серверних алгоритмів просторової обробки та аналітики маршрутної мережі

Реалізація веб-геоінформаційної системи передбачає перехід від моделей предметної області та UML-діаграм до виконуваного коду з узгодженими форматами обміну даними та детермінованою послідовністю обчислень. У розробленому прототипі серверне обчислювальне ядро побудовано мовою Python 3 на базі фреймворку Flask (модуль app.py) із шаром доступу до PostgreSQL і розширення PostGIS у модулі db.py. Такий стек дозволяє поєднати транзакційну цілісність довідників маршрутів і зупинок, виконання просторових предикатів і метрик (буфери, площі, відстані в метрах через тип geography) та інтеграцію з бібліотеками NetworkX і OSMnx для обчислень на дорожньому графі, отриманому з OpenStreetMap.

У межах підрозділу 3.1 розглянуто два взаємодоповнювані серверні алгоритми, які безпосередньо змінюють або уточнюють модель маршрутної мережі та її просторове відображення. Перший забезпечує побудову геометрії маршруту вздовж вулично-дорожньої мережі за впорядкованим списком зупинок і синхронізацію збереженої довжини маршруту з фактичною полілінією на карті. Другий реалізує злиття двох маршрутів у єдину послідовність зупинок із перевірками допустимості, транзакційним оновленням таблиць і формуванням агрегованих показників впливу на мережу; його детальний опис і блок-схема наведені після викладу першого алгоритму.

Commented [5]: Нащо тут частина слів виділених жирним?

Дорожня мережа подається як орієнтований зважений граф G = (V, E), який завантажується при старті застосунку з файлу kyiv_drive.graphml. Для ребер задана вага length — довжина фрагмента дороги в метрах. Щоб відобразити рух громадського транспорту не «по прямій» між координатами зупинок, а вздовж

доступних зв'язків графа, для кожної пари послідовних зупинок після прив'язки координат до вузлів o та d шукається шлях, що мінімізує суму ваг на ребрах:

$$L(o,d) = \sum_{e \in P} w(e), \quad w(e) = length(e) \geq 0 \quad (3.1)$$

За невід'ємних ваг це класична задача найкоротшого шляху; у коді використовуються функції NetworkX shortest_path та shortest_path_length із параметром weight="length". Окремо для швидкої оцінки довжини ламаної між зупинками «по поверхні сфери» (допоміжні метрики, зокрема при аналізі наслідків злиття) у db.py застосовується формула Гаверсина: для кутів φ (широта) та λ (довгота) у радіанах

$$hav(\theta) = \frac{\theta}{2} \quad (3.2)$$

$$hav(\Theta) = hav(\varphi_2 - \varphi_1) + cos\ cos\ \varphi_1\ cos\ cos\ \varphi_2\ \ hav(\lambda_2 - \lambda_1) \quad (3.3)$$

$$d = 2R\ arcsin\ arcsin\ \sqrt{hav(\Theta)} \quad (3.4)$$

Кути переводяться з градусів множенням на $\pi/180$. Основна довжина, що відображається на карті й записується в routes.total_length_m після побудови геометрії, формується з графових сегментів; наближення за Гаверсином не замінює вуличну геометрію, а доповнює обчислення там, де потрібна легка оцінка по послідовності точок.

Побудова геометрії інкапсульована в обробнику GET /api/routes/<route_id>/geometry. Спочатку з бази зчитується послідовність зупинок (get_route_stops); якщо їх менше двох, обчислення не виконується. Для прискорення на великому місті будується підграф у прямокутнику навколо зупинок (truncate_graph_bbox у OSMnx), після чого для кожної сусідньої пари визначаються найближчі вузли (safe_nearest_node з обробкою випадків, коли найближчий вузол «випадає» з обрізаного графа), обчислюється найкоротший шлях, координати ребер зшиваються в GeoJSON типу LineString. Якщо шляху на підграфі немає, виконується повтор на повному G; якщо шляху немає й там —

сегмент заноситься в missing_segments, а побудова триває для наступних пар. Після циклу викликається update_route_stats, що оновлює total_length_m і кількість зупинок у transport.routes.

Наведена нижче блок-схема формалізує послідовність рішень і гілок цього алгоритму — від перевірки кількості зупинок до повернення геометрії клієнту.

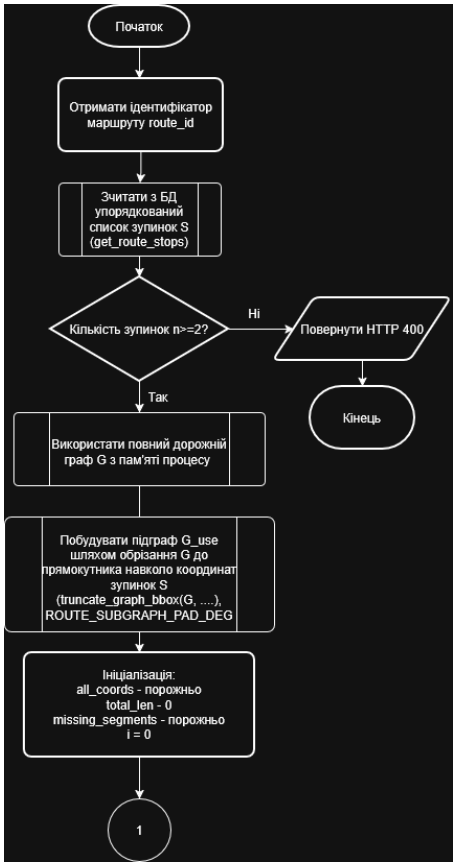

Рис. 3.1, а – Блок-схема алгоритму побудови геометрії маршруту на дорожньому графі OpenStreetMap

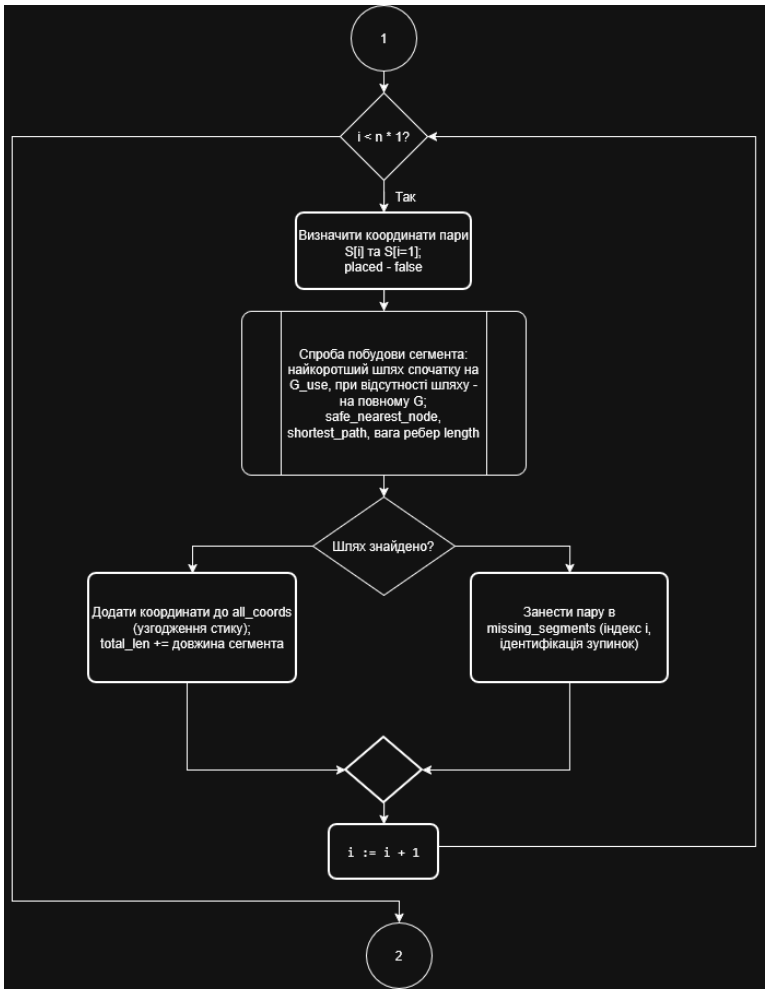

Рис. 3.1, б – Блок-схема алгоритму побудови геометрії маршруту на дорожньому графі OpenStreetMap

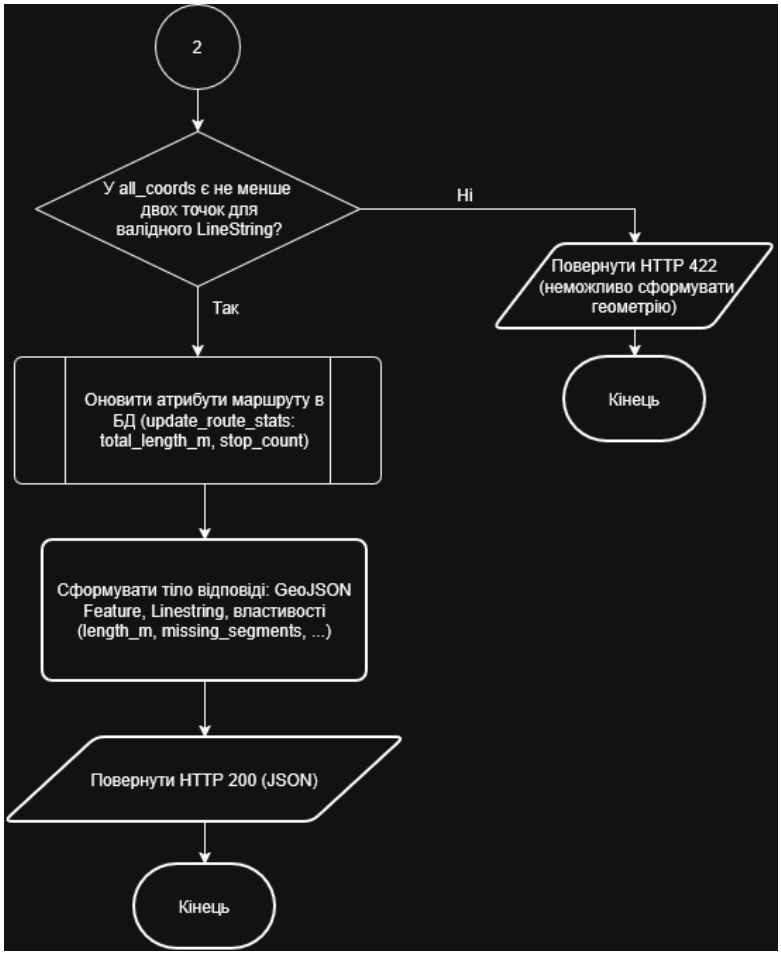

Рис. 3.1, в – Блок-схема алгоритму побудови геометрії маршруту на дорожньому графі OpenStreetMap

Ініціалізація та валідація. Процес розпочинається із запиту GET /api/routes/<route_id>/geometry, де route_id — ідентифікатор маршруту в transport.routes. Функція get_route_stops (db.py) повертає впорядкований список зупинок із transport.route_stops і transport.stops. Якщо зупинок менше двох — побудова LineString неможлива, сервер повертає HTTP 400.

Підготовка графа. Використовується граф G, завантажений з kyiv_drive.graphml (OpenStreetMap). Для оптимізації формується підграф G_use шляхом обрізки області навколо зупинок (OSMnx truncate_graph_bbox); за потреби виконується відкат до повного графа.

Ітеративний розрахунок. Для кожної пари зупинок S[i] → S[i+1] визначаються найближчі вузли (safe_nearest_node) та будується найкоротший шлях (NetworkX shortest_path, вага length). Координати додаються до all_coords без дублювання, довжина — до total_len. При відсутності шляху на G_use виконується спроба на повному графі; у разі невдачі сегмент фіксується у missing_segments, але розрахунок продовжується.

Агрегація та завершення. Після циклу перевіряється достатність координат: менше двох — повертається HTTP 422. Інакше оновлюються total_length_m і кількість зупинок через update_route_stats. Успішна відповідь (HTTP 200) містить GeoJSON LineString із довжиною, кількістю зупинок та missing_segments; клієнт (Leaflet) відображає маршрут на карті.

Після відтворення просторової топології маршруту на дорожньому графі система переходить до аналітичних процедур виявлення структурних недоліків моделі мережі. Основним завданням є детермінований аналіз дублювання транспортних коридорів на основі обчислення перетину множин зупинок різних маршрутів. Додатково реалізовано алгоритм злиття двох маршрутів з автоматизованою перевіркою допустимості операції та оцінкою її впливу на агреговані показники мережі.

Запропоновані підходи розмежовують дві принципово різні постановки задач. Геометрично-транспортна задача — пошук найкоротшого шляху на дорожньому графі методом Дейкстри [10] — спрямована на мінімізацію сумарної метричної довжини між заданими точками. Комбінаторно-множинна задача — виявлення дублювання маршрутів — базується на аналізі множин зупинок із логічним групуванням фізично наближених об'єктів через parent_stop_id, що усуває надмірність при обробці просторових даних.

З архітектурної точки зору, злиття маршрутів є транзакційною операцією над сутностями routes та route_stops. Це дозволяє модифікувати логічну структуру мережі без перебудови геометрії вуличної інфраструктури, забезпечуючи високу швидкість обробки та цілісність бази даних.

Математичне підґрунтя відбору дубльованих коридорів

Нехай для маршруту А множина логічних зупинок S_A, для маршруту В — S_B, а $|\cdot|$ позначає потужність множини.

Введемо:

$shared = |\ S_A \cap S_B\ |$ — кількість спільних логічних зупинок;

$total_a = |\ S_A\ |, total_b = |\ S_B\ |$ — кількість логічних зупинок на кожному маршруті.

Тоді коефіцієнт схожості коридорів (коефіцієнт Жаккара для множин зупинок) визначається як

$$J(A,B) = \frac{|S_A \cap S_B|}{|S_A \cup S_B|} = \frac{shared}{total_a + total_b - shared} \quad (3.5)$$

Процедура ідентифікації дубльованих коридорів базується на обчисленні коефіцієнта схожості: маршрут вважається кандидатом на дублювання, якщо розрахований показник подібності дорівнює або перевищує граничне значення min_similarity, що передається як параметр запиту до API.

Для підвищення точності вибірки застосовується евристичний аналіз назв: при збігу символьних послідовностей до розділювача «:» система класифікує пару як зустрічні напрямки однієї лінії та автоматично виключає її з переліку дублікатів. Додатково виконується колапсування множинних записів — у фінальному звіті залишається лише варіант із найвищим рівнем схожості.

Алгоритм злиття маршрутів реалізовано в обробнику POST /api/optimization/merge-routes з обов'язковим контролем цілісності: перевіркою існування об'єктів та мінімально допустимої кількості зупинок (не менше двох) у результуючій структурі. Технологічний цикл включає три етапи: перезапис послідовності зупинок у route_stops для цільового маршруту; видалення надлишкового запису з transport.routes; автоматичне оновлення метрик та

формування об'єкта impact із показниками «до» та «після» операції. Логічна послідовність виконання цих процедур представлена у вигляді блок-схеми (рис. 3.4).

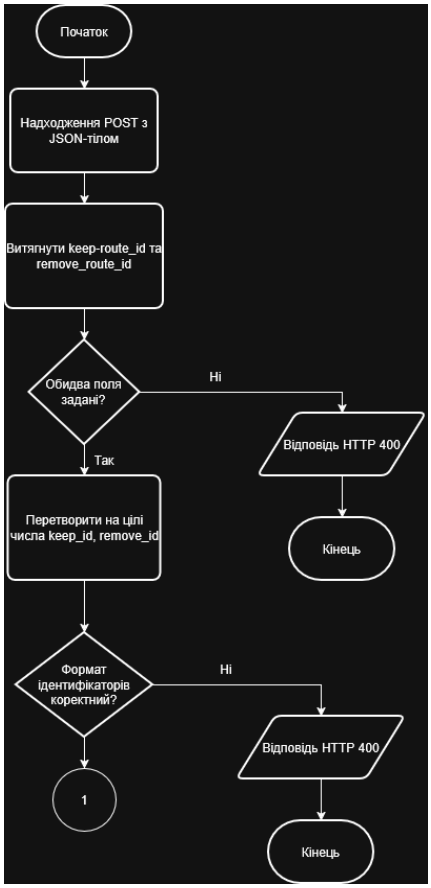

Рис. 3.2, а – Блок-схема алгоритму злиття двох маршрутів (Частина 1)

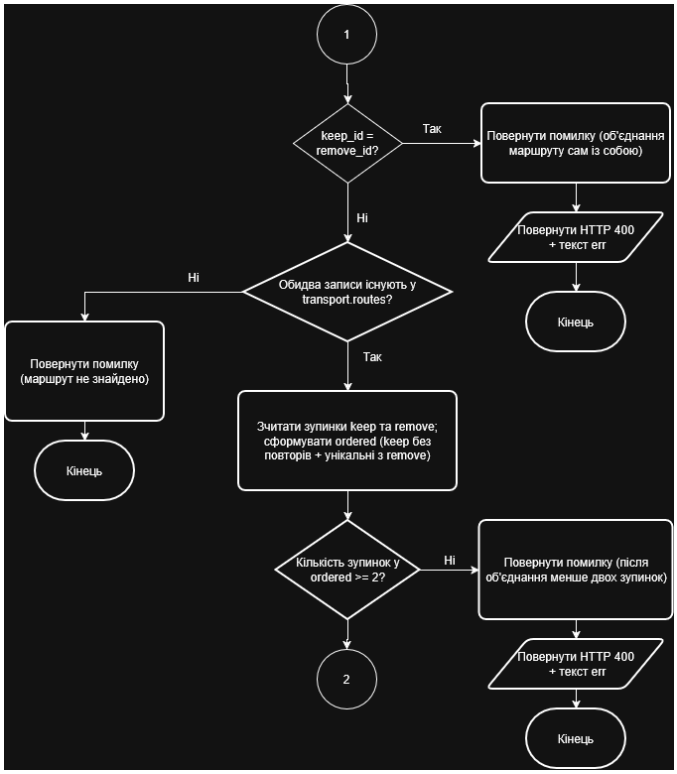

Рис. 3.2, б – Блок-схема алгоритму злиття двох маршрутів (Частина 2)

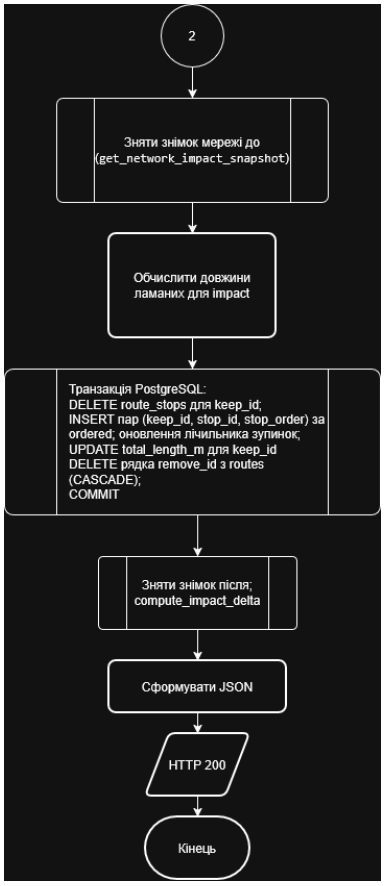

Рис. 3.2, в – Блок-схема алгоритму злиття двох маршрутів (Частина 3)

Детальний опис алгоритмічних кроків злиття двох маршрутів

1. Отримання та підготовка даних. Сервер приймає POST /api/optimization/merge-routes із JSON: keep_route_id, remove_route_id — який маршрут лишається й який знімається з моделі. Доступ до PostgreSQL (db.py, psycopg2) без ORM: перевіряється наявність обох записів у transport.routes, з transport.route_stops зчитуються зупинки в порядку stop_order.

2. Ініціалізація базису та контроль цілісності. Формується список ordered: спочатку унікальні зупинки keep, потім з remove, яких ще немає. Перевірки: keep ≠ remove, обидва маршрути існують, після об'єднання ≥ 2 зупинок. Для аналітики знімається знімок мережі «до» (get_network_impact_snapshot) і обчислюються довжини ламаних по зупинках для полів impact.

3. Ядро — атомарна зміна в БД. У одній транзакції: DELETE route_stops для keep_id; INSERT нових пар у порядку ordered; оновлення лічильника зупинок і total_length_m для keep; DELETE рядка remove з transport.routes (каскадно знімаються його route_stops); COMMIT.

4. Синтез метрик після зміни. Знімок мережі «після», compute_impact_delta між «до» та «після» (кількість маршрутів, обслуговувані зупинки, сума довжин, наближене покриття). У impact додаються також довжини «до» для кожного маршруту та для об'єднаної послідовності.

5. Формування відповіді. У JSON повертаються ok, ідентифікатори й назви маршрутів, stop_count, вкладений impact. Клієнт оновлює списки та карту (Leaflet); графіки (Chart.js) підтягують зведені дані з інших запитів за потреби.

Перевірки виконуються до запису, зміна моделі — транзакційно, відповідь містить готові метрики впливу, що зручно для інтерактивного прототипу без ручного перерахунку в таблицях.

Фрагменти вихідного коду серверної частини (модулі просторової обробки на Python) наведено в додатку А.

3.2 Розробка інтерфейсу користувача та засобів візуалізації просторових даних

Клієнтську частину побудовано на HTML5, CSS3 та Vanilla JS — це забезпечує легкість застосунку, пряме керування DOM-деревом і відсутність складних процедур збірки. Логіку декомпоновано за модульним принципом: модулі state.js, map.js, routes.js, districts.js, stats.js, optimization.js інтегруються в шаблон Jinja2; обмін даними з Flask реалізовано через Fetch API.

Картографічну візуалізацію реалізовано засобами Leaflet.js (CDN) із підкладом OpenStreetMap без залежності від комерційних API. Рендеринг організовано за багатошаровим принципом: базовий шар тайлів OSM; шар зупинок з ендпоінта /api/stops у вигляді L.circleMarker із кольоровою індикацією типу рухомого складу та інтерактивними tooltip і popup; аналітичні шари геометрії маршрутів, ізохрон і буферних зон у вигляді полілінійі і полігонів. Інтерфейс реалізовано за концепцією односторінкового застосунку (рис. 3.3).

Рис. 3.3 – Головний екран геоінформаційної системи: карта OSM, бічна панель із вкладками та елементами керування аналізом маршрутної мережі.

Інтерфейс побудовано на CSS Flexbox: ліва панель (300 px) містить навігаційні вкладки та інструменти керування, права — картографічну область. Зведена статистика відображається через Chart.js, звітність експортується у PDF через /api/report/pdf.

Створення маршруту відбувається через модальне вікно (рис. 3.4): форма атрибутів надсилає POST /api/routes, після чого стає доступна панель керування зупинками. Зупинки додаються пошуком за назвою або кліком по карті (findNearest) — обидва способи використовують POST /api/routes/<route_id>/stops (рис. 3.5).

Геометрія маршруту будується окремо: GET /api/routes/<route_id>/geometry повертає GeoJSON LineString по графу OSM і

оновлює total_length_m; Leaflet відображає полілінію на карті. Такий підхід розмежовує логічний порядок зупинок у БД та геометрію вуличної мережі.

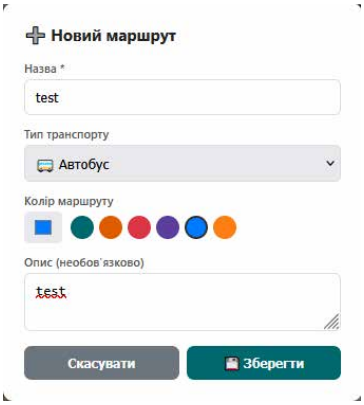

Рис. 3.4 – Процес створення нового маршруту: модальне вікно атрибутів

Рис 3.5 – Додавання нових зупинок до маршруту кліками по зупинкам на карті

Модуль аналітики мережі на вкладці «Оптим.» забезпечує комплексний аналіз транспортної інфраструктури: результати серверних процедур відображаються у вигляді графіків і таблиць в інформаційній панелі (рис. 3.6).

Функція виявлення дублювання формує картки для пар маршрутів із коефіцієнтом схожості Жаккара вище порогу min_similarity; кожна картка містить відсоток збігу, кількість спільних зупинок і рекомендацію щодо злиття. Аналіз вузлових навантажень відображає дані GeoJSON на підкладці Leaflet: радіус маркера пропорційний кількості маршрутів через зупинку, паралельно формується рейтинг перевантажених вузлів.

Модуль рекомендацій після операції злиття або видалення відображає порівняльну таблицю показників до та після: кількість маршрутів, унікальні зупинки, сумарна довжина та охоплення буферними зонами. Відстані між зупинками супроводжуються поясненням, що розраховані за формулою Гаверсина, а не на основі дорожнього графа. Функція симуляції гіпотетичного маршруту повертає полігон приросту покриття з площею зони обслуговування та наближеною чисельністю населення в її межах.

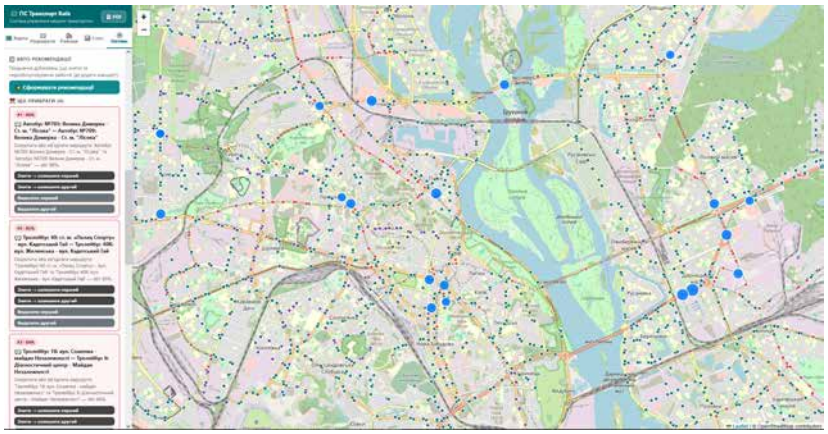

Рис. 3.6 — Вкладка «Оптим.» приклад відображення результатів аналізу дублювання, шару хабів на карті

На відміну від топологічного аналізу мережі, блок «Райони» спрямований на дослідження територіальної нерівномірності покриття та кореляції між чисельністю населення і щільністю зупинок. Аналіз транспортного забезпечення

базується на просторових запитах PostGIS: для кожного полігона району ідентифікуються зупинки в його межах [19], обчислюється кількість маршрутів та їх розподіл за видами транспорту . Щільність покриття розраховується як відношення кількості зупинок до площі району. Результати візуалізуються в інформаційній панелі у вигляді порівняльних статистик, накопичувальних діаграм та рейтингових таблиць щільності.

Для оцінки демографічного навантаження використовуються геопросторові дані у форматі GeoJSON. Ключовим метричним показником є транспортна доступність — кількість зупинок на тисячу жителів, що дозволяє об'єктивно порівнювати райони незалежно від їхнього розміру [1]. Візуалізація результатів реалізована за допомогою хороплетної карти з фіксованими класами щільності населення та відповідною легендою. Такий підхід дозволяє виявляти зони з високою густотою населення та дефіцитом транспортної інфраструктури (рис. 3.7) .

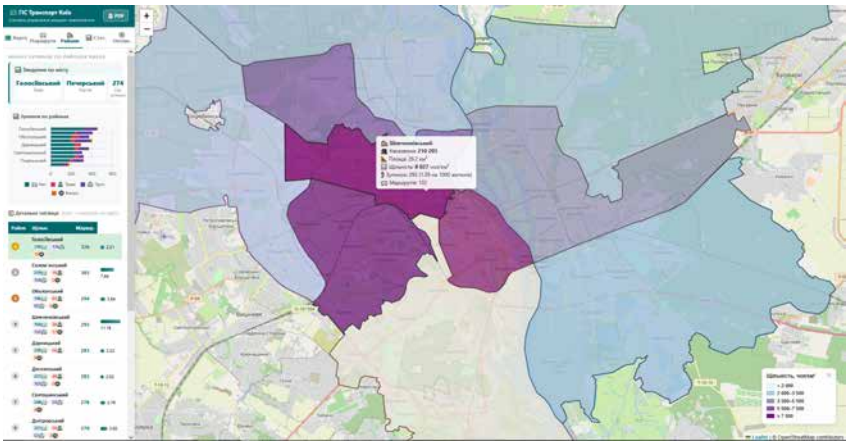

Рис 3.7 — Аналіз районів Києва та щільності населення

Геоінформаційна система потребує не лише карти, а й зведених показників для оцінки масштабу і структури транспортної мережі. Якщо картографічний модуль Leaflet дає просторове уявлення про розташування зупинок і траєкторій маршрутів, то для порівняння районів, частки видів транспорту та найбільш

протяжних ліній зручніше використовувати діаграми та таблиці. У клієнтській частині прототипу для цього застосовано бібліотеку Chart.js [13], яка будує інтерактивні графіки на основі елемента canvas [13].

Візуальна аналітика розподілена між вкладками бічної панелі відповідно до змісту даних. Загальні показники мережі виводяться на вкладці «Стат.»: при її відкритті завантажуються агреговані дані про кількість зупинок і маршрутів, сумарний та середній пробіг ліній, розподіл маршрутів за видами руху. У верхній частині панелі відображаються чотири ключові індикатори (KPI), нижче — кільцева діаграма розподілу за видами транспорту та таблиця десяти найдовших маршрутів із назвою, типом, довжиною та кількістю зупинок (рис. 3.8).

Територіальний зріз реалізовано на вкладці «Райони»: для кожного адміністративного району обчислюється кількість зупинок та їх розподіл за видами транспорту. Результат подається у вигляді горизонтальної стовпчастої діаграми з накопиченням, що доповнює таблицю щільності зупинок і кольоровий шар меж районів на карті.

При повторному оновленні графіків попередні екземпляри діаграм коректно знищуються [13], щоб не перевантажувати пам'ять браузера — це особливо важливо після змін у мережі, коли зведення потрібно перерахувати.

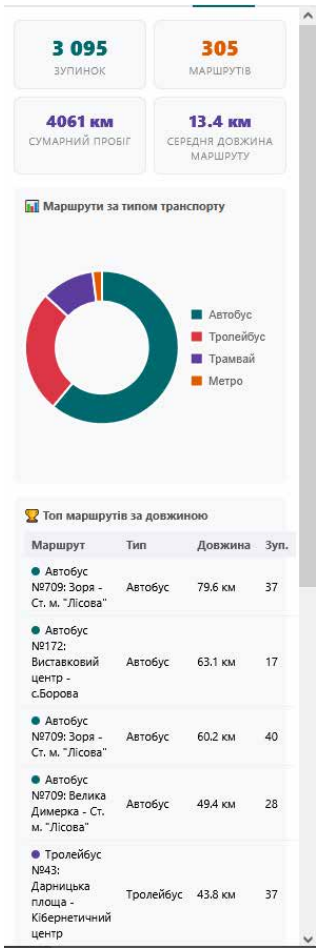

Рис. 3.8 — Вкладка «Статистика»

Розроблена веб-ГІС адаптована для хмарного розгортання, що відповідає сучасним вимогам до доступу та масштабованості. Серверну частину (Python/Flask) та просторову базу даних (PostgreSQL/PostGIS) розгорнуто на віддаленому віртуальному сервері (VPS/хмарному хостингу). Це забезпечує цілодобовий доступ до аналітичних інструментів без необхідності встановлення

ПЗ на локальні комп'ютери диспетчерів. Для роботи з системою клієнту потрібен лише сучасний веб-браузер та доступ до мережі Інтернет.

Таблиця 3.1 – Апаратні вимоги для хмарного розгортання сервера

Компонент	Мінімальні	Рекомендовані	Обґрунтування
CPU	2 ядра, 1,5 GHz	2 ядра, 2,0 GHz	Граф доріг, запити PostGIS, карта в браузері.
ОЗП (RAM)	4 GB	6 GB	Flask, PostgreSQL, завантажений GraphML, вкладка з Leaflet.
Диск	2 GB вільно	4 GB (бажано SSD)	Залежності Python, GraphML, БД.
Мережа	Доступ в інтернет	Стабільний канал	Тайли OSM; перша установка — pip install.

Програмні вимоги. Для запуску прототипу необхідні: Python 3.10+ (серверна частина Flask); PostgreSQL 15+ із розширенням PostGIS (зберігання зупинок, маршрутів, районів та просторові обчислення); сучасний веб-браузер із підтримкою ECMAScript 6 та HTML5 Canvas. Окремі контейнери або промислові веб-сервери не обов'язкові; потрібен інтернет для тайлів OpenStreetMap та завантаження пакетів Python при першій установці.

Пакет розгортання. Інсталяційним пакетом є ZIP-архів із вихідним кодом: app.py, db.py, каталоги Templates/ та static/, SQL-скрипти схеми (sql/install/), файл залежностей requirements.txt та дорожні графи (kyiv_drive.graphml). Каталог віртуального середовища venv до архіву не включається — залежності відновлюються командою pip install -r requirements.txt. Дані користувача

зберігаються в PostgreSQL, тому для перенесення маршрутів слід додатково передати дамп бази.

Сценарій А — первинна інсталяція:

Розпакувати архів, наприклад до C:\GIS-Transport\

Встановити PostgreSQL із PostGIS, створити базу та виконати схему: psql -d gis_kyiv -f sql\install\install.sql

Налаштувати змінні середовища: DB_NAME, DB_USER, DB_PASSWORD, DB_HOST, DB_PORT

Створити віртуальне середовище та встановити залежності: python -m venv venv && venv\Scripts\activate && pip install -r requirements.txt

Імпортувати дані (зупинки, райони, графи) скриптами підготовки

Запустити сервер: python app.py

Відкрити у браузері: http://127.0.0.1:5000/

Сценарій Б — передача розгорнутої системи: створити ZIP без venv, за потреби додати дамп PostgreSQL (pg_dump). Отримувач розпаковує архів, виконує pip install -r requirements.txt, відновлює дамп або повторює імпорт даних, запускає python app.py.

Щоденний запуск після інсталяції: активувати venv → python app.py → браузер за тією ж адресою (PostgreSQL має бути запущений як служба ОС). Система не потребує окремої команди DevOps і є автономним інструментом для муніципального підприємства за умови офісного ПК з 4 ГБ ОЗП (див. табл. 3.1).

3.3 Тестування системи та аналіз результатів оптимізації

Завершальним етапом розробки є комплексне тестування програмних модулів та аналіз ефективності реалізованих алгоритмів. Мета — підтвердження працездатності архітектури «клієнт–сервер», перевірка коректності просторових обчислень і емпірична оцінка результатів мережевої оптимізації. Тестування проводилося на реальних даних: зупинки та маршрути Києва з OpenStreetMap, полігони адміністративних районів, дорожній граф у форматі GraphML.

Інтеграційне тестування REST API виконувалося засобами Postman. Перевірялися основні групи ендпоінтів: отримання зупинок і маршрутів, створення маршруту та додавання зупинок, побудова геометрії, аналіз покриття, модулі оптимізації, оцінка гіпотетичного маршруту, зведена статистика та PDF-звіт. Особлива увага приділялася валідації вхідних даних і граничним випадкам.

У ході тестування підтверджено коректну обробку виняткових ситуацій: запит геометрії з однією зупинкою повертає HTTP 400; злиття з некоректними ідентифікаторами — HTTP 400; неповне покриття дорожнім графом — HTTP 422 з переліком проблемних сегментів без оновлення довжини в БД. При коректному складі маршруту транзакція завершується успішно, клієнт отримує GeoJSON-лінію для відображення на карті. Фрагмент журналу подій наведено нижче:

[10:14:02] INFO: Завантаження kyiv_drive.graphml... готово (близько 16 с)

[10:14:18] GET /api/stops?type=all — 200 OK

[10:14:25] POST /api/routes — 201 Created («Тестовий-62»)

[10:14:31] POST /api/routes/41/stops — 200 OK (3 зупинки)

[10:14:38] GET /api/routes/41/geometry — 200 OK (total_length_m: 6840)

[10:15:02] GET /api/optimization/duplicates?min_similarity=0.6 — 200 OK (6 пар)

[10:15:41] POST /api/optimization/merge-routes — 200 OK (impact: before/after)

[10:16:05] GET /api/statistics — 200 OK

Основна довжина маршруту обчислюється по дорожньому графу OSM (найкоротший шлях між вузлами). Допоміжний розрахунок за формулою Гаверсина (R = 6371 км) використовується для ламаної «по зупинках» у зведеннях після злиття. Коректність реалізації перевірено порівнянням з еталонними значеннями PostGIS (ST_Distance, тип geography). Критерій прийнятності для внутрішньоміських перегонів до 4 км — відносна похибка не більше 0,5%. Результати наведено у табл. 3.2.

Таблиця 3.2 – Перевірка розрахунку відстаней за формулою Гаверсина між зупинками

Сегмент (від → до)	Розрахунок ГІС, км	Еталон PostGIS, км	Абсолютна похибка, м	Відносна похибка, %
Хрещатик → Майдан Незалежності	0,291	0,292	1	0,34
Арсенальна → Палац Спорту	1,845	1,848	3	0,16
Либідська → Олімпійська	2,120	2,124	4	0,18
Почайна → Контрактова площа	3,450	3,456	6	0,17
Середнє по вибірці	—	—	**3,5**	**0,21**

Як свідчать результати табл. 3.2, максимальна розбіжність на перегоні понад 3 км становить 6 м; середня відносна похибка — 0,21%, що не перевищує встановленого критерію. Для допоміжних зведень по зупинках така точність є достатньою.

Порівняння довжини по дорожньому графу та суми сегментів Гаверсина для п'яти маршрутів показало середню різницю 28,4% (маршрут № 62: 11,40 км по графу проти 8,95 км «по прямій»; маршрут № 455: 14,75 км проти 11,20 км). Це підтверджує доцільність використання дорожнього графа для побудови реальної траєкторії руху.

Аналіз надлишковості мережі виконано за коефіцієнтом Жаккара на множинах логічних зупинок: при порозі схожості 0,6 виявлено 6 пар; при 0,8 — 2 пари зі схожістю понад 80%. Проведено контрольне злиття маршрутів № 62 та № 62А (схожість 0,84) зі збереженням першого маршруту. Результати наведено у табл. 3.3.

Таблиця 3.3 – Результати аналізу дублікатів і злиття маршрутів

Пара / маршрут	Схожість (Жаккар)	Довжина 1-го, км	Довжина 2-го, км	Дія	Ефект для мережі
№ 62 ↔ № 62А	0,84	11,40	10,90	Злиття (залишено № 62)	Мінус 1 маршрут; мінус 10,9 км у сумарній довжині
№ 114 ↔ № 114к	0,79	9,85	9,10	Рекомендовано; не виконувалось	—
№ 24 ↔ № 24А	0,76	7,30	6,95	Рекомендовано; не виконувалось	—
№ 318 ↔ № 47	0,68	16,20	14,05	Нижче порогу 0,8	—
№ 455	—	14,75	—	Без пари в топ-2	—
Агрегати після злиття	—	142,6 (до)	128,4 (після)	Злиття однієї пари	Мінус 14,2 км; покриття 285,3 → 284,9 км²

Скорочення сумарної довжини мережі на 14,2 км пояснюється видаленням маршруту-дубліката та об'єднанням зупинок в одній лінії. Площа покриття змінилася незначно (−0,4 км²), оскільки набір обслуговуваних зупинок практично не змінився. Злиття ініціюється виключно рішенням диспетчера через інтерфейс системи. Після операції на вкладці «Оптим.» відобразився блок

порівняння показників до та після; на карті — одна оновлена траса замість двох збіжних. Симуляція гіпотетичного маршруту з п'яти зупинок у недообслуговуваному районі показала приріст покриття 0,38 км² та близько 4200 осіб за наближеною оцінкою.

Оцінка продуктивності на локальному комп'ютері (4 ГБ RAM), середній час виконання запитів (50 звернень після прогріву): GET /api/statistics — 45 мс; GET /api/optimization/duplicates — 180 мс; GET /api/routes/<id>/geometry — 620 мс; перший запуск із завантаженням GraphML — 12–18 с. Споживання пам'яті: Python із графом — 380–450 МБ; PostgreSQL — 90–140 МБ; браузер — 160–220 МБ.

Тестування підтвердило життєздатність архітектури «браузер — Flask — PostgreSQL/PostGIS» та коректність обробки граничних випадків. Похибка формули Гаверсина відносно PostGIS становить близько 0,2%; алгоритми виявлення дублікатів і злиття дозволяють зменшити надлишковість мережі без суттєвої втрати покриття. Апаратні вимоги залишаються помірними для розгортання на диспетчерському пункті.

ВИСНОВКИ

У результаті виконання бакалаврської кваліфікаційної роботи було успішно вирішено актуальне завдання — спроєктовано та програмно реалізовано веб-геоінформаційну систему для просторового аналізу та оптимізації маршрутної мережі міського пасажирського транспорту (на прикладі м. Києва).

Відповідно до поставлених завдань отримано такі основні результати:

1. Проведено системний аналіз предметної області та сформульовано функціональні й технічні вимоги до веб-ГІС. Обґрунтовано доцільність використання геоінформаційних технологій для транспортної логістики з метою підвищення ефективності планування міської мобільності.
2. Охарактеризовано теоретичні основи інтерпретації транспортної інфраструктури у вигляді зважених просторових графів. На основі цього обґрунтовано вибір оптимального стеку відкритих технологій (Python, PostgreSQL/PostGIS, OpenStreetMap), що забезпечує незалежність від пропрієтарних комерційних ГІС-платформ та дозволяє гнучко реалізувати специфічну транспортну бізнес-логику.
3. Спроєктовано інформаційну модель предметної області та розроблено логіко-фізичну схему просторової бази даних, реалізовану в середовищі PostgreSQL із розширенням PostGIS. За допомогою засобів UML формалізовано поведінку системи. Запропонована структура ефективно зберігає координати зупинок (у форматі geometry), забезпечує ієрархічне групування об'єктів та підтримує зв'язки з районами без надлишкового дублювання даних.
4. Реалізовано серверну частину веб-застосунку на базі фреймворку Flask з розробкою відповідного REST API. Успішно впроваджено алгоритми обробки просторових даних, зокрема: маршрутизацію по реальному графу дорожньої мережі (NetworkX, OSMnx) для точного відображення ліній маршрутів, аналітичний модуль пошуку дубльованих коридорів на основі

коефіцієнта Жаккара, а також транзакційний алгоритм злиття маршрутів за підтвердженням користувача з автоматичним перерахунком агрегованих показників мережі.

5. Розроблено клієнтський інтерфейс користувача за концепцією односторінкового застосунку (Vanilla JS, Leaflet, Chart.js). Це забезпечило швидкий рендеринг просторових шарів (зупинки, лінії маршрутів, полігони сигнальних районів, ізохрони) та зручну інтерактивну візуалізацію аналітичних показників у вигляді динамічних діаграм і рейтингових таблиць.
6. Детально описано реалізовані алгоритми та проведено комплексне тестування системи. Емпірично доведено точність допоміжних розрахунків (похибка формули Гаверсина склала близько 0,2 % відносно еталону PostGIS), а також підтверджено працездатність ключових сценаріїв. Зокрема, сценарне тестування модуля оптимізації показало, що злиття лише однієї пари маршрутів-дублікатів (№ 62 та № 62А) дозволяє скоротити сумарний пробіг мережі на 14,2 км без суттєвої втрати територіального покриття.

Практичне значення отриманих результатів полягає у створенні повністю автономного та працездатного прототипу ГІС. Розроблена система може бути використана як навчально-дослідницький комплекс для аналізу урбаністичних даних, а також слугувати базою для впровадження у муніципальних транспортних підприємствах та диспетчерських центрах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Антонюк Л. Л., Мельниченко О. А., Шевченко М. М. Урбаністика : навч. посіб. Київ : КНЕУ, 2018. 360 с.
2. Agafonkin V. Leaflet — a JavaScript library for interactive maps. 2024. URL: https://leafletjs.com (дата звернення: 14.05.2024).
3. Applegate D. L., Bixby R. E., Chvátal V., Cook W. J. The Traveling Salesman Problem: A Computational Study. Princeton : Princeton University Press, 2006. 606 p.
4. Boeing G. Modeling and Analyzing Urban Networks and Amenities with OSMnx. *arXiv*. 2025. arXiv:2505.00736. URL: https://arxiv.org/abs/2505.00736 (дата звернення: 14.05.2024).
5. Caliper Corporation. TransCAD Transportation Planning Software. 2024. URL: https://www.caliper.com/tcovu.htm (дата звернення: 14.05.2024).
6. Connolly T., Begg C. Database Systems: A Practical Approach to Design, Implementation and Management. 6th ed. Harlow : Pearson, 2015. 1440 p.
7. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 4th ed. Cambridge : MIT Press, 2022. 1312 p.
8. de Smith M. J., Goodchild M. F., Longley P. A. Geospatial Analysis: A Comprehensive Guide to Principles, Techniques and Software Tools. 6th ed. Winchelsea Press, 2018. URL: https://www.spatialanalysisonline.com (дата звернення: 14.05.2024).
9. Diestel R. Graph Theory. 5th ed. Berlin : Springer, 2017. 428 p.
10. Dijkstra E. W. A note on two problems in connexion with graphs. *Numerische Mathematik*. 1959. Vol. 1, no. 1. P. 83–89. DOI: 10.1007/BF01386390.
11. Esri. ArcGIS Platform Overview. 2024. URL: https://www.esri.com/en-us/arcgis/about-arcgis/overview (дата звернення: 14.05.2024).
12. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures : PhD thesis. University of California, Irvine, 2000. URL:

https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm (дата звернення: 14.05.2024).

13. Fu P., Sun J. Web GIS: Principles and Applications. Redlands : ESRI Press, 2011. 312 p.
14. Haklay M., Weber P. OpenStreetMap: User-Generated Street Maps. *IEEE Pervasive Computing*. 2008. Vol. 7, no. 4. P. 12–18. DOI: 10.1109/MPRV.2008.80.
15. Jaccard P. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin de la Société Vaudoise des Sciences Naturelles*. 1901. Vol. 37. P. 547–579.
16. Longley P. A., Goodchild M. F., Maguire D. J., Rhind D. W. Geographic Information Science and Systems. 4th ed. Hoboken : John Wiley & Sons, 2015. 496 p.
17. Object Management Group. Unified Modeling Language Specification, Version 2.5.1. 2017. URL: https://www.omg.org/spec/UML/2.5.1 (дата звернення: 14.05.2024).
18. Ortúzar J. de D., Willumsen L. G. Modelling Transport. 4th ed. Chichester : John Wiley & Sons, 2011. 607 p.
19. PostGIS Project Steering Committee. PostGIS 3.5 Manual. 2024. URL: https://postgis.net/docs/manual-3.5/ (дата звернення: 14.05.2024).
20. QGIS Development Team. QGIS Geographic Information System. Open Source Geospatial Foundation Project. 2024. URL: https://qgis.org (дата звернення: 14.05.2024).
21. The PostgreSQL Global Development Group. PostgreSQL 15 Documentation. 2023. URL: https://www.postgresql.org/docs/15/ (дата звернення: 14.05.2024).
22. Vuchic V. R. Urban Transit Systems and Technology. Hoboken : John Wiley & Sons, 2007. 624 p.

ДОДАТКИ

Додаток А

Лістинг ключових програмних модулів системи

Лістинг А.1 — Функція api_route_geometry (модуль app.py): побудова GeoJSON LineString за найкоротшими шляхами між парами зупинок на графі доріг (вага ребра length)

```
@app.route("/api/routes/<int:route_id>/geometry")
def api_route_geometry(route_id):
    try:
        stops = get_route_stops(route_id)
        if len(stops) < 2:
            return jsonify({"error": "Потрібно мінімум 2 зупинки"}),
400

        G_use = drive_subgraph_for_route_stops(stops)
        all_coords = []
        total_len = 0
        missing_segments = []

        for i in range(len(stops) - 1):
            oLat, oLon = stops[i][2], stops[i][3]
            dLat, dLon = stops[i+1][2], stops[i+1][3]
            placed = False

            for graph_try in (G_use, G):
                try:
                    o = safe_nearest_node(graph_try, oLon, oLat)
                    d = safe_nearest_node(graph_try, dLon, dLat)
                    seg   =   nx.shortest_path(graph_try,   o,   d,
weight="length")
                    seg_l  =  nx.shortest_path_length(graph_try,  o,
d, weight="length")
                    coords = route_to_coords(graph_try, seg)
                    total_len += seg_l
```

```
                    if all_coords and coords:
                        all_coords.extend(coords[1:])
                    else:
                        all_coords.extend(coords)
                    placed = True
                    break
                except nx.NetworkXNoPath:
                    continue

            if not placed:
                missing_segments.append({
                    "from": stops[i][1], "to": stops[i+1][1],
                    "index": i
                })
                log.warning(
                    "route_geometry: немає шляху %s -> %s
(route_id=%s)",
                    stops[i][1], stops[i+1][1], route_id
                )

        if len(all_coords) < 2:
            return jsonify({
                "error": (
                    "Не вдалося побудувати лінію маршруту:
недостатньо точок для LineString "
                    "(усі сегменти між зупинками відсутні на графі
доріг). Див. missing_segments."
                ),
                "missing_segments": missing_segments,
            }), 422

        update_route_stats(route_id, round(total_len), len(stops))
        route_info = get_route_by_id(route_id)
```

```
        return jsonify({
            "type": "Feature",
            "geometry":   {"type":   "LineString",   "coordinates":
all_coords},
            "properties": {
                "route_id": route_id,
                "name": route_info[1] if route_info else "",
                "color": route_info[3] if route_info else "#01696f",
                "length_m": round(total_len),
                "stops": len(stops),
                "missing_segments": missing_segments
            }
        })
    except Exception as e:
        traceback.print_exc()
        return jsonify({"error": str(e)}), 500
```

Лістинг А.2 — Функція merge_routes_into (модуль db.py): формування об’єднаної послідовності зупинок і транзакція в PostgreSQL (оновлення route_stops, total_length_m, видалення другого маршруту)

```
def merge_routes_into(keep_route_id, remove_route_id):
    """
    Зупинки remove додаються після ланцюга keep (унікальні stop_id без дублікатів).
    Рядок remove видаляється з transport.routes (зупинки зв'язку CASCADE).
    """
    if keep_route_id == remove_route_id:
        return None, "Неможливо об'єднати маршрут сам із собою"

    info_k = get_route_by_id(keep_route_id)
    info_r = get_route_by_id(remove_route_id)

    if not info_k or not info_r:
        return None, "Один або обидва маршрути не знайдено"
```

```
stops_k = get_route_stops(keep_route_id)
stops_r = get_route_stops(remove_route_id)

ordered = []
seen = set()

for row in stops_k:
    sid = int(row[0])
    if sid not in seen:
        ordered.append(sid)
        seen.add(sid)

for row in stops_r:
    sid = int(row[0])
    if sid not in seen:
        ordered.append(sid)
        seen.add(sid)

if len(ordered) < 2:
    return None, "Після об'єднання потрібно щонайменше 2 зупинки"

before = get_network_impact_snapshot(500)
kept_len = route_polyline_length_m(stops_k)
removed_len = route_polyline_length_m(stops_r)

coord_by_id = {}
for row in stops_k:
    coord_by_id[int(row[0])] = (float(row[2]), float(row[3]))
for row in stops_r:
    coord_by_id[int(row[0])] = (float(row[2]), float(row[3]))

merged_rows = []
for sid in ordered:
    lat, lon = coord_by_id[sid]
```

```
    merged_rows.append((sid, None, lat, lon, None))

merged_len = route_polyline_length_m(merged_rows)

conn = get_connection()
cur = conn.cursor()

cur.execute(
    "DELETE FROM transport.route_stops WHERE route_id = %s;",
    (keep_route_id,),
)

for order, sid in enumerate(ordered, start=1):
    cur.execute(
        """
        INSERT INTO transport.route_stops (route_id, stop_id, stop_order)
        VALUES (%s, %s, %s);
        """,
        (keep_route_id, sid, order),
    )

_refresh_stop_count(cur, keep_route_id)

cur.execute(
    """
    UPDATE transport.routes
       SET total_length_m = %s
     WHERE route_id = %s;
    """,
    (merged_len, keep_route_id),
)

cur.execute(
    "DELETE FROM transport.routes WHERE route_id = %s;",
    (remove_route_id,),
```

```
)

conn.commit()
cur.close()
conn.close()

after = get_network_impact_snapshot(500)
delta = compute_impact_delta(before, after)

return {
    "keep_route_id": keep_route_id,
    "keep_name": info_k[1],
    "removed_route_id": remove_route_id,
    "removed_name": info_r[1],
    "stop_count": len(ordered),
    "impact": {
        "before": before,
        "after": after,
        "delta": delta,
        "length_by_stops_m": {
            "kept_route_before": kept_len,
            "removed_route": removed_len,
            "sum_two_routes": kept_len + removed_len,
            "merged_route": merged_len,
        },
    },
}, None
```

ДОДАТОК Б

Керівництво користувача (диспетчера)

Б.1 Призначення та загальна структура інтерфейсу

Веб-ГІС «ГІС Транспорт Київ» призначена для аналізу, моделювання та оптимізації міської транспортної мережі. Для роботи системи потрібен сучасний веббраузер та підключення до мережі Інтернет.

Інтерфейс системи (рис. Б.1) складається з інтерактивної карти (права частина) та бічної панелі керування (ліва частина). Панель має п'ять основних вкладок: «Карта» (базовий просторовий аналіз), «Маршрути» (управління геометрією), «Райони» (демографічний аналіз), «Стат.» (ключові показники) та «Оптим.» (автоматичні рекомендації).

Commented [6]: Вирівнювання по ширині далі по тексту

Рис. Б.1 — Головне вікно геоінформаційної системи

Б.2 Основні сценарії роботи диспетчера

Б.2.1 Перегляд та аналіз існуючої мережі Для відображення зупинок необхідно перейти на вкладку «Карта» та натиснути кнопку відображення (маркери розфарбовуються за типами транспорту). Для аналізу просторової доступності користувач може:

- Побудувати ізохрону (зону пішохідної доступності), обравши часовий ліміт (5, 10 або 15 хв) та клікнувши на будь-яку точку карти.

- Використати інструмент «Аналіз покриття» для виявлення покритих територій та «мертвих зон» у заданому радіусі (наприклад, 500 м) від існуючих зупинок.

Б.2.2 Управління маршрутами На вкладці «Маршрути» диспетчер може створювати нові або редагувати існуючі лінії. Додавання зупинок до маршруту здійснюється через текстовий пошук або інструментом «Додати кліком» безпосередньо на карті. Після формування послідовності зупинок система автоматично прокладає оптимізовану лінію маршруту вздовж вуличної мережі (дорожнього графа OpenStreetMap) та розраховує його загальну довжину.

Б.2.3 Автоматична оптимізація та злиття маршрутів Це ключовий інструмент аналітика для усунення надмірності мережі, що знаходиться на вкладці «Оптим.».

1. Для пошуку дублюючих маршрутів диспетчер встановлює поріг схожості (наприклад, 80%) та ініціює пошук.
2. У блоці «Сформувати рекомендації» система пропонує пари маршрутів для злиття.
3. За допомогою кнопок «Злити» або «Видалити» застосовуються зміни. У блоці результатів система миттєво розраховує зміну ефективності мережі (показники «До / Після»).

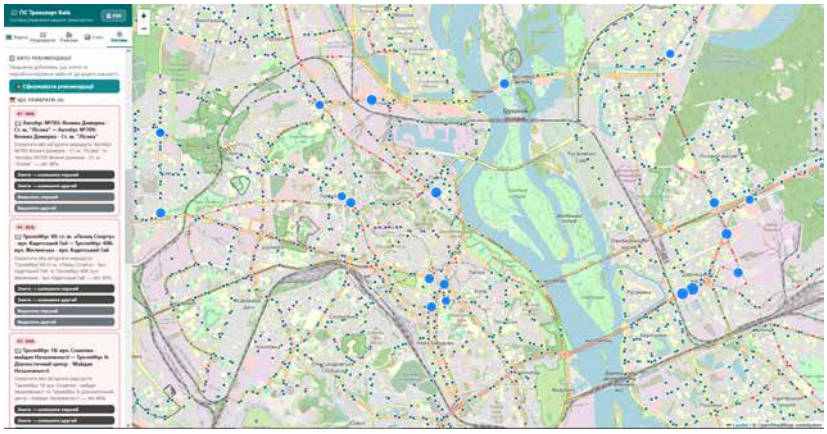

Рис. Б.2 — Інтерфейс автоматичної оптимізації маршрутів

Б.2.4 Симуляція гіпотетичного маршруту Для перевірки доцільності запуску нового транспортного сполучення диспетчер вводить його назву та позначає пропоновані зупинки на карті. Інструмент «Оцінити покриття» розраховує площу нової зони обслуговування та орієнтовну кількість населення, що отримає доступ до транспорту, без внесення змін до основної бази даних.

Б.3 Формування звітності У будь-який момент роботи користувач може ініціювати створення звіту про поточний стан мережі, натиснувши кнопку «PDF» у верхній частині панелі. Система автоматично згенерує документ із загальними KPI мережі, розподілом за типами транспорту та аналітикою по районах, який можна зберегти або роздрукувати.

Б.4 Важливі обмеження та застереження

- Незворотність дій: Операції автоматичного злиття та видалення маршрутів у базі даних є незворотними на рівні користувацького інтерфейсу.
- Залежність від графа: Геометрія маршруту будується за наявним дорожнім графом. Якщо алгоритм не може прокласти шлях між двома точками, необхідно перевірити правильність послідовності зупинок.
- Точність демографічних даних: Оцінка охоплення населення під час симуляції є наближеною і базується на усередненій щільності населення відповідних районів міста.

ДОДАТОК В

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

Декларація про академічну доброчесність та використання ШІ

Я, Якубенко Богдан Вячеславович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Геоінформаційна система оптимізації маршрутів міського транспорту на основі просторових даних» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):

- [V] інструменти генеративного ШІ не використовувалися.
- [V] інструменти ШІ (вказати назву: Gemini, Composer) інші були використані для:
- [ ] перекладу іншомовних джерел;
- [V] стилістичного редагування та перевірки граматики;
- [V] допомоги у написанні/відлагодженні програмного коду;
- [ ] інше: .

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«__»_____ 2026 р. _____ Богдан Якубенко