МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»
Циклова комісія *комп'ютерних систем та мереж*
(повна назва циклової комісії)

Допустити до захисту
Голова випускової циклової комісії
комп'ютерних систем та мереж
(повна назва циклової комісії)
(підпис) *Ірина КРАВЧУК* (ім'я, ПРІЗВИЩЕ)
« 10 » 06 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬО-ПРОФЕСІЙНОГО СТУПЕНЯ ФАХОВИЙ МОЛОДШИЙ БАКАЛАВР

Тема: *Програмування логістичної системи для оптимізації поштової доставки*

Група: *3-011* Спеціальність: *123 «Комп'ютерна інженерія»*

Здобувач освіти (підпис) *Максим ГОРБАРУК* (ім'я, ПРІЗВИЩЕ)

Керівник роботи (підпис) *Роман МІНЕНКО* (ім'я, ПРІЗВИЩЕ)

Консультант з оформлення пояснювальної записки (підпис) *Оксана ОСАДЧА* (ім'я, ПРІЗВИЩЕ)

Кривий Ріг 2025 р.

КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

Відділення *комп'ютерної та програмної інженерії*
Циклова комісія *комп'ютерних систем та мереж*
Освітньо-професійний ступінь *фаховий молодший бакалавр*
Спеціальність *123 «Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ
Голова випускової циклової комісії
комп'ютерних систем та мереж
(повна назва циклової комісії)
(підпис) *Ірина КРАВЧУК*
(ім'я, ПРІЗВИЩЕ)
« *01* » *03* 2025 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧУ ОСВІТИ

Горбарук Максим Сергійович
(прізвище, ім'я, по батькові)

1.Тема роботи *Програмування логістичної системи для оптимізації поштової доставки*

Керівник роботи *Міненко Роман Вадимович, к.ф.-м.н.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по коледжу від « *04* » *04* 2025 року № *50-ст*

2. Строк подання здобувачем освіти роботи з *20.05.2025* по *16.06.2025*

3. Вихідні дані до роботи *Логістична система для оптимізації поштової доставки*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

Характеристика предметної області та постановка задачі

Проєктування архітектури системи

Програмна реалізація

Впровадження та експлуатація

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Презентація Microsoft PowerPoint

6. Консультанти розділів роботи (проекту)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання *13.04.2025*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	*Узгодження технічного завдання з керівником кваліфікаційної роботи*	*13.04.25*	*виконано*
2	*Підбір та вивчення науково-технічної літератури за темою кваліфікаційної роботи*	*19.04.25*	*виконано*
3	*Обґрунтування вибору програмних засобів*	*23.04.25*	*виконано*
4	*Опис компонентів. Обґрунтування їх вибору.*	*27.04.25*	*виконано*
5	*Розробка програмного забезпечення*	*02.05.25*	*виконано*
6	*Дослідження ефективності реалізованих методів.*	*05.05.25*	*виконано*
7	*Написання пояснювальної записки*	*13.05.25*	*виконано*
8	*Перевірка на плагіат пояснювальної записки*	*09.06.25*	*виконано*
9	*Попередній захист кваліфікаційної роботи*	*02.06.2025-06.06.2025*	*виконано*
10	*Захист кваліфікаційної роботи*		

Здобувач освіти _____ *Максим ГОРБАРУК*
(підпис) (ім'я, ПРІЗВИЩЕ)

Керівник роботи _____ *Роман МІНЕНКО*
(підпис) (ім'я, ПРІЗВИЩЕ)

StrikePlagiarism.com

Дата звіту 6/10/2025

Дата редагування ---

Документ прийнятий

Звіт подібності

метадані

Назва організації

Ukrainian national aviation university

Заголовок

Горбарук Максим_Сергійович_3-011_2025_123_ДР

Автор Науковий керівник / Експерт

ГорбарукМіненко Р

підрозділ

Криворізький Фаховий коледж

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

КП 1

КЦ

25
Довжина фрази для коефіцієнта подібності 2

8055
Кількість слів

65673
Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв	ß	1
Інтервали	A→	0
Мікропробіли	·	7
Білі знаки	ß	0
Парафрази (SmartMarks)	a	26

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	Сагайдак_Іван_Олександрович_3013_2025_123 [illegible] 6/9/2025 Ukrainian national aviation university (Криворізький Фаховий коледж)	60 0.74 %
2	Сагайдак_Іван_Олександрович_3013_2025_123 [illegible] 6/9/2025 Ukrainian national aviation university (Криворізький Фаховий коледж)	36 0.45 %

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Програмування логістичної системи для оптимізації поштової доставки»: 56 сторінок, 15 рисунків, 4 додатки, 8 використаних джерел.

МАРШРУТ, ПРОГРАМУВАННЯ, ГЕОКОДІНГ, ВІДСТАНІ, АВТОМАТИЗОВАНА СИСТЕМА, ПРОТОКОЛ, РОУТІНГ, КООРДИНАТА, API, OSRM, NOMINATIM, ПОШТОВА ДОСТАВКА, ЛОГІСТИКА.

Об'єктом дослідження є процес оптимізації логістичних операцій у сфері поштової доставки. Предметом проектування є інформаційна система для геокодування, обрахунку маршрутів та пошуку найближчих поштових відділень на основі координат або адреси користувача.

Метою кваліфікаційної роботи є розробка модуля програмного забезпечення, який дозволяє автоматизувати визначення геолокації користувача, здійснювати зворотний геокодінг, обчислювати оптимальні маршрути та знаходити найближчі відділення логістичної компанії Нова Пошта. Реалізація такого функціоналу сприяє покращенню користувацького досвіду та підвищенню ефективності обслуговування в логістичних сервісах.

Отриманий результат має інноваційний характер завдяки комплексному підходу до інтеграції відкритих геолокаційних сервісів у прикладне програмне забезпечення для логістичних компаній. Робота демонструє можливості оптимізації процесів доставки шляхом автоматизації прийняття рішень на основі географічних даних.

Значущість роботи полягає у створенні модульної та масштабованої основи, яку можна адаптувати до потреб інших логістичних компаній, кур’єрських сервісів або мобільних додатків, пов’язаних із картографією.

Результати роботи можуть бути впроваджені в реальні логістичні системи, де необхідна автоматизація пошуку найближчих об’єктів, маршрутизації та підвищення точності обслуговування клієнтів.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

API (Application Programming Interface) – інтерфейс прикладного програмування. Набір функцій, який дозволяє програмам взаємодіяти між собою або з зовнішніми сервісами.

HTTP (HyperText Transfer Protocol) – протокол передачі гіпертексту. Основний протокол для обміну даними між веб-браузером та сервером.

HTML (HyperText Markup Language) – мова розмітки гіпертексту, що використовується для створення структури веб-сторінок.

JSON (JavaScript Object Notation) – легкий формат обміну даними, зручний для читання людиною і обробки машиною.

UX (User Experience) – досвід користувача. Визначає, наскільки зручно, зрозуміло і приємно користуватися продуктом або веб-додатком. *.NET* – платформа від Microsoft для розробки і виконання програм, що працюють на Windows.

OSRM (Open Source Routing Machine) – система з відкритим кодом для побудови маршрутів на основі карт OpenStreetMap.

Nominatim – сервіс для геокодування (перетворення адреси в координати) та зворотного геокодування, що використовує дані OpenStreetMap. *Geocoding* – процес перетворення адреси або назви місця в географічні координати.

Frontend – частина додатку, з якою взаємодіє користувач.

Backend – частина додатку, що відповідає за логіку, зберігання і обробку даних на сервері.

Docker – платформа для контейнеризації додатків. Дозволяє створювати ізольовані середовища для запуску програм.

OpenStreetMap – відкритий проєкт зі створення вільної та доступної карти світу, яку може редагувати будь-хто.

ЗМІСТ

ВСТУП

Сучасне життя неможливо уявити без швидкої та надійної доставки поштових відправлень, яка є важливою складовою розвитку логістичних систем і сервісів. В умовах стрімкого зростання обсягів інтернет-торгівлі та загального підвищення мобільності населення значення автоматизованих систем пошуку найближчих поштових відділень та маршрутів доставки

постійно зростає. Існуючі рішення мають певні обмеження, пов’язані з якістю геокодування, інтеграцією різнорідних даних, а також ефективністю алгоритмів маршрутизації. Тому розробка надійної системи, яка забезпечить інтуїтивно зрозумілий інтерфейс для користувача та ефективну роботу бекенд-компонентів з інтеграцією зовнішніх API, є актуальним завданням для сучасних ІТ-рішень у сфері логістики та сервісів доставки.

Обрана тема відповідає пріоритетним напрямкам розвитку цифрових технологій та інтелектуальних транспортних систем, що затверджені державними програмами розвитку інфраструктури та підтримки ІТ-галузі. Крім того, робота узгоджується з планами розвитку локальних підприємств поштового зв’язку та логістичних компаній, які активно впроваджують цифрові рішення для оптимізації процесів доставки. Вона також відповідає актуальним науково-дослідним темам у галузі геоінформаційних систем (ГІС) та програмування веб-сервісів.

Метою кваліфікаційної роботи є створення програмного забезпечення, яке забезпечує пошук найближчих поштових відділень за заданими координатами або адресою, а також відображення оптимального маршруту доставки із використанням інтеграції з API зовнішніх сервісів. Для досягнення цієї мети необхідно виконати такі завдання:

Проаналізувати існуючі методи та інструменти геокодування, пошуку і маршрутизації;

Розробити архітектуру програмної системи, що включає інтеграцію з API Нової Пошти та OSRM;

Реалізувати користувацький інтерфейс для введення даних і відображення результатів;

Забезпечити надійну обробку запитів і коректне відображення найближчих відділень та маршрутів;

Провести тестування та оцінку якості роботи системи.

Об’єктом дослідження є процеси автоматизації пошуку найближчих поштових відділень і маршрутизації доставки в умовах інтеграції з відкритими та комерційними API.

Предметом дослідження є методи і алгоритми, що забезпечують геокодування, пошук найближчих точок та побудову маршруту, а також програмна реалізація цих методів у вигляді веб-сервісу з відповідним користувацьким інтерфейсом.

РОЗДІЛ 1
ХАРАКТЕРИСТИКА ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

У цьому розділі описується потреба у швидкому та точному пошуку найближчих об’єктів (наприклад, поштових відділень) за географічними координатами. Розкривається важливість геолокаційних сервісів у сучасних веб- і мобільних застосунках, особливо для логістики, торгівлі та обслуговування населення.

1.1. Аналіз предметної області доставки

У сучасному цифровому середовищі все більше користувачів надають перевагу онлайн-сервісам для замовлення товарів, що зумовлює зростання ролі логістичних компаній і, зокрема, служб доставки. Однією з найпоширеніших моделей доставки є доставка до відділення, що особливо характерно для українського ринку, де послуги компаній типу “Нова Пошта” є чи не основним способом отримання посилок. Однак попри наявність багатьох онлайн-платформ для замовлення товарів, у більшості з них процес вибору відділення залишається рутинним і незручним для користувача.

Проблематика полягає в тому, що в більшості вебсайтів користувач змушений вручну обирати населений пункт зі списку, після чого йому пропонується перелік усіх доступних відділень. Цей перелік часто є довгим і неструктурованим — без жодної візуальної прив’язки до географічного розташування. Користувачу, особливо якщо він перебуває в незнайомому місті або районі, складно зорієнтуватись, яке саме відділення є найближчим до нього. Це створює зайві кроки в процесі оформлення доставки, підвищує ризик помилок і знижує загальний рівень користувацького досвіду.

Особливо актуальною ця проблема є для мобільних користувачів, які очікують, що система “сама знає, де вони знаходяться” — адже переважна більшість сучасних пристроїв мають вбудовану підтримку геолокації через

браузер або мобільну операційну систему. Проте така функціональність майже ніколи не використовується при виборі відділення служби доставки. Це не лише створює незручності, але й суперечить загальним трендам у розвитку цифрових сервісів, де автоматизація та зниження бар’єрів у взаємодії з користувачем стають визначальними чинниками якості.

Окрім незручності, ручний вибір відділень також часто не враховує актуальні оновлення (наприклад, закриття або тимчасову недоступність відділення), відстань від поточного місцезнаходження користувача, транспортні шляхи або навіть реальний час маршруту. Як наслідок, користувачі часто обирають не найзручніше для себе відділення, що призводить до додаткових витрат часу, труднощів у навігації і, в окремих випадках, негативного досвіду користування сервісом.

У межах даного проєкту запропоновано рішення, яке враховує сучасні можливості геолокації та відкритих даних, і дозволяє автоматизувати вибір найближчого відділення до поточного розташування користувача. Реалізований вебзастосунок здатний отримувати координати користувача через браузер за допомогою інтерфейсу геолокації (Geolocation API), після чого звертатися до API логістичної компанії (зокрема, Нової Пошти) для отримання переліку доступних відділень у межах найближчого населеного пункту. На основі отриманих координат кожного відділення система визначає відстань до користувача, сортує їх і надає у зручному, лаконічному вигляді — першими відображаються найближчі варіанти.

Таким чином, користувач повністю звільняється від необхідності вводити адресу вручну, шукати відділення у великому списку або звіряти дані з картою. Весь процес відбувається автоматично — від геолокації до візуалізації відділень. У перспективі це дозволяє не лише зменшити кількість кліків, але й значно покращити загальний UX-сценарій роботи із замовленням доставки.

Впровадження такого функціоналу також відкриває нові можливості для оптимізації логістичних процесів: можна динамічно перенаправляти

клієнтів до менш завантажених або зручніших відділень, враховувати зміни в інфраструктурі, будувати оптимальні маршрути з урахуванням реального трафіку тощо. І все це — без необхідності втручання користувача в складні налаштування або додаткові дії.

Отже, існує чітко сформована потреба у створенні інструменту, який дозволяє в автоматизований спосіб визначати найближче відділення доставки, використовуючи сучасні відкриті сервіси геолокації, геокодування та маршрутизації. Це забезпечує не лише зручність, але й надає можливість інтеграції з іншими сервісами — від електронної комерції до логістичних платформ.

1.2. Аналіз існуючих рішень і джерел інформації

У процесі реалізації системи автоматизованого визначення найближчого відділення доставки надзвичайно важливим є вибір надійних, стабільних та функціонально придатних джерел інформації та програмних інтерфейсів. Враховуючи специфіку задачі — роботу з геоданими, логістичними точками та маршрутами — було проаналізовано як закриті комерційні рішення, так і відкриті публічні сервіси [2]. Основна увага приділялась рішенню, яке забезпечує баланс між точністю, швидкістю, простотою інтеграції, відкритістю ліцензій і безкоштовним використанням у невеликих проєктах.

1.2.1. API Нової Пошти

Одним із ключових джерел логістичної інформації став API Нової Пошти, що надає прямий доступ до актуальних даних про:

- населені пункти, у яких присутні відділення компанії;

- перелік доступних відділень з адресами, GPS-координатами та режимом роботи;

- типи відділень (поштове, поштомат, вантажне тощо).

Використання цього API дозволяє уникнути дублювання або ручного оновлення даних про відділення та забезпечує високу точність інформації. Однією з переваг цього сервісу є стабільна робота, регулярне оновлення даних на стороні постачальника і добре документований інтерфейс. Завдяки цьому стає можливим автоматичне завантаження інформації про відділення у реальному часі — що критично для побудови динамічних, контекстно-чутливих інтерфейсів користувача.

Однак API Нової Пошти не надає безпосередніх інструментів для роботи з координатами користувача (наприклад, побудови маршруту або переведення адреси в координати), тому для повноцінної інтеграції необхідно було доповнити систему зовнішніми сервісами геолокації та маршрутизації [1].

1.2.2. Nominatim (OpenStreetMap Geocoding)

Для виконання зворотного геокодування — тобто, переведення координат, отриманих із браузера користувача, у зрозумілу поштову адресу — було обрано Nominatim. Це відкритий геокодувальний сервіс, який базується на даних OpenStreetMap і дозволяє виконувати такі операції:

- перетворення географічних координат (широта, довгота) на текстову адресу;

- пошук координат за вказаною адресою;

- уточнення географічного розташування користувача в межах міста, вулиці або навіть будинку.

Перевагами Nominatim є:

- повна відкритість (є open source-реалізація);

- можливість розгортання власного сервера для уникнення обмежень API;

- хороша якість даних в межах України, зокрема великих міст;

- підтримка зворотного геокодування.

Саме використання Nominatim дозволило створити зручний користувацький сценарій: отримати поточне розташування користувача через API браузера, передати координати до Nominatim, отримати адресу та прив’язати її до даних з Нової Пошти.

1.2.3. OSRM (Open Source Routing Machine)

Для побудови маршруту до найближчого відділення було обрано OSRM — високопродуктивний маршрутний рушій, який працює на базі OpenStreetMap. OSRM дозволяє:

- розраховувати найкоротші маршрути між точками;

- враховувати типи доріг, заборони руху, напрямки, транспорт; - повертати точну геометрію маршруту, яка може бути візуалізована на мапі;

- інтегруватися у сторонні застосунки через HTTP API [7]. Використання OSRM дало змогу реалізувати ключову функцію — побудову шляху від поточного місця користувача до одного чи кількох найближчих відділень, що дозволяє автоматично визначити найзручніший варіант. Також OSRM

підтримує локальний запуск сервера, що є перевагою для автономної роботи, збереження контролю над картою та зменшення залежності від сторонніх сервісів [6].

Усі три інструменти — API Нової Пошти, Nominatim, OSRM — були обрані на основі таких критеріїв:

- відкритість: можливість використання у навчальних, дослідницьких та комерційних проєктах без значних ліцензійних обмежень; - гнучкість: можливість адаптації під конкретні потреби проєкту (наприклад, запуск локально або кастомізація запитів);

- актуальність даних: постійна синхронізація з офіційними джерелами (Нова Пошта), використання свіжих картографічних даних (OpenStreetMap);

- мінімальна залежність від закритих платформ: що важливо для контролю стабільності сервісу, безпеки та прогнозованості в довгостроковій перспективі.

Комбінація цих сервісів дала змогу створити повноцінну, ефективну та доступну систему, яка автоматизує складний процес логістичного вибору з урахуванням геопозиції користувача [6].

1.3. Постановка задачі

У сучасних умовах стрімкого розвитку електронної комерції, сервісів доставки та мобільних вебзастосунків постає гостра потреба у створенні інтелектуальних інструментів, що дозволяють спростити взаємодію користувача з логістичними системами. Зокрема, користувачі очікують швидкого, автоматизованого і зручного способу визначення найближчого відділення поштової служби без необхідності ручного перегляду списків міст і філій.

Традиційні підходи, що передбачають ручний вибір міста, вулиці та поштового відділення з випадаючих списків, часто виявляються незручними, особливо для мобільних користувачів або в незнайомих місцевостях. Тому

основною метою дипломної роботи є розробка сервісу, який дозволить значно підвищити зручність користування за рахунок автоматизації вибору та маршрутизації до найближчого відділення. Для реалізації цієї мети було сформульовано такі конкретні задачі [1].

Таким чином, підсумовуючи, основним завданням дипломної роботи є створення сучасного інтелектуального інструменту доставки, який автоматизує визначення найближчого відділення на основі геопозиції користувача, забезпечуючи повну інтеграцію з відкритими сервісами, маршрутизацію та геокодування, а також зручний користувацький інтерфейс для демонстрації роботи.

1.3.1. Розробка серверного API для взаємодії з Новою Поштою

Потрібно створити окремий програмний модуль (серверне API), який буде взаємодіяти з офіційним API Нової Пошти з метою:

- отримання списків доступних міст;
- отримання списку відділень конкретного міста;
- отримання координат та адрес кожного відділення.

Цей модуль буде служити проміжною ланкою між зовнішнім API Нової Пошти та клієнтською частиною системи, дозволяючи кешувати або фільтрувати дані, а також зменшувати залежність фронтенду від змін у структурі офіційного API [1].

1.3.2. Інтеграція геокодування через Nominatim

З метою покращення користувацького досвіду необхідно реалізувати функціонал двостороннього геокодування:

- зворотне геокодування — переведення координат користувача (отриманих через браузер) у текстову адресу;
- пряме геокодування — можливість користувачеві ввести адресу

вручну, яка буде конвертована у координати для подальшої обробки. Цей функціонал базуватиметься на відкритому сервісі Nominatim, що дозволяє інтегрувати його через HTTP API [6]. Отримані координати будуть використовуватись як відправна точка для визначення найближчого відділення і побудови маршруту.

1.3.3. Побудова маршруту до найближчого відділення за допомогою OSRM

Щоб побудувати оптимальний маршрут від поточного місця перебування користувача до найближчого поштового відділення, слід: - реалізувати виклики до локально піднятого сервера OSRM (Open Source Routing Machine);

- на основі координат користувача та координат усіх відділень поблизу розрахувати маршрути до кожного;

- знайти найкоротший маршрут (за відстанню або часом) і вивести його користувачеві на мапі;

- візуалізувати маршрут на основі повернутої геометрії [7]. OSRM дозволить досягти високої точності побудови маршруту без залучення комерційних сервісів, таких як Google Maps, що важливо для автономності та легальності навчального або внутрішнього корпоративного використання.

1.3.4. Створення клієнтського інтерфейсу на HTML/JavaScript Для демонстрації функціоналу буде створено клієнтську частину: - за допомогою JavaScript реалізується отримання геолокації користувача через браузер (Geolocation API);

- на основі координат викликається Nominatim для отримання адреси;

- дані передаються до серверного API, яке повертає список найближчих

відділень;

- обране відділення використовується для побудови маршруту через OSRM;

- результати виводяться на інтерактивну мапу (наприклад, з використанням Leaflet або MapLibre).

Цей інтерфейс не лише демонструє повноцінну роботу всіх сервісів у комплексі, але й забезпечує основу для потенційної інтеграції в реальні проєкти або мобільні додатки.

РОЗДІЛ 2
ПРОЄКТУВАННЯ АРХІТЕКТУРИ СИСТЕМИ

У другому розділі відбувається опис архітектурного стилю (наприклад, мікросервісна або REST-сервіс), де геолокаційний модуль працює як окремий сервіс, доступний іншим модулям через HTTP API. Розкривається технічна структура рішення, яка забезпечує ефективність, масштабованість і легкість інтеграції.

2.1. Опис предметної області та обґрунтування вибору моделей

У межах проєкту «GeoPost» була розроблена система, що дозволяє знаходити найближче поштове відділення (відділення чи поштомат) за координатами або адресою користувача. Основними доменними моделями, які відображають предметну область, є PostOffice, Address, Coordinate, Schedule, WorkingHours. Була побудована UML-діаграма класів предметної області сервісу геолокації, вона зображена на рисунку 2.1.

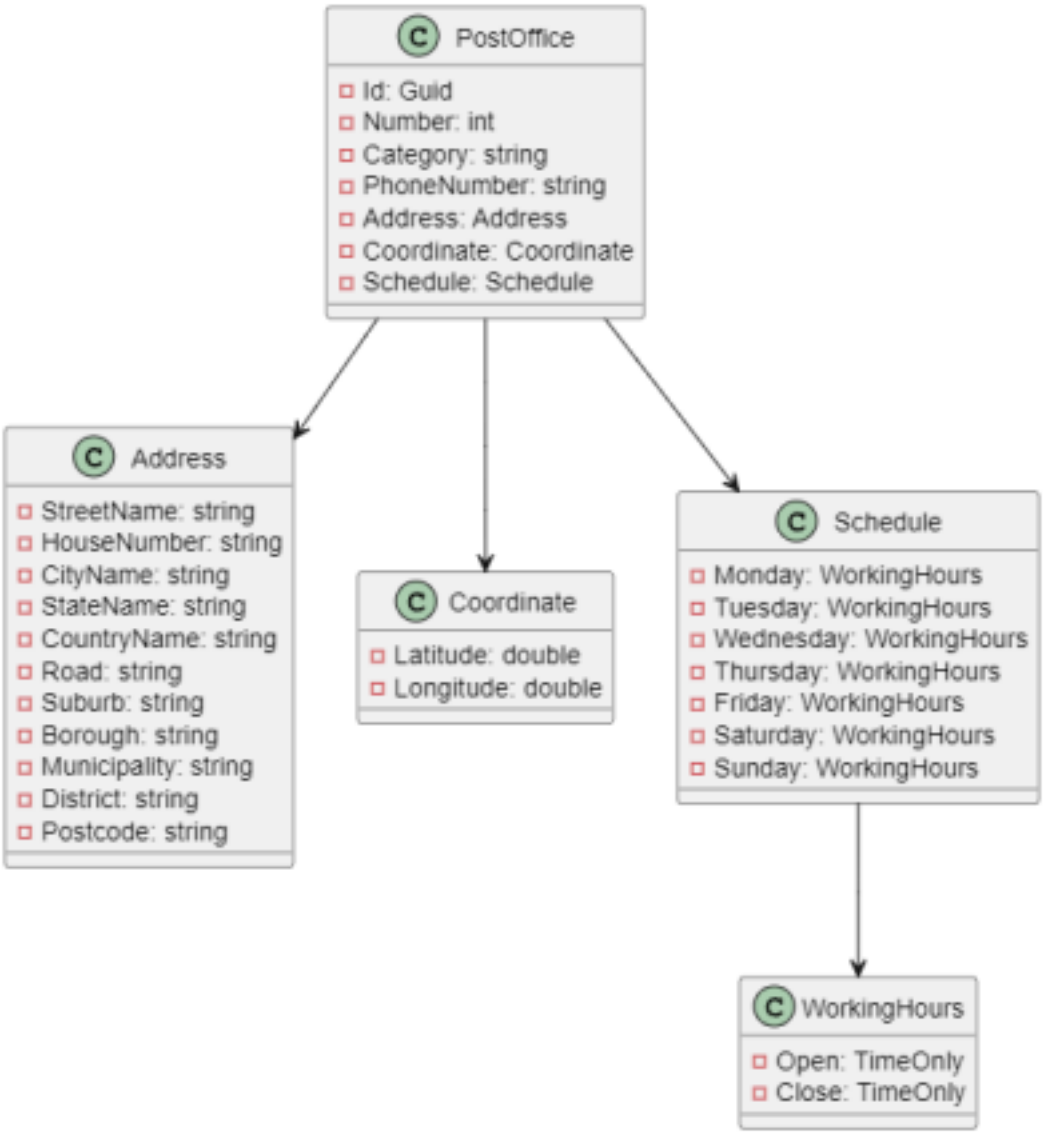


Рисунок 2.1 – UML-діаграма класів сервісу геолокації

Address — описує адресу поштового відділення. Містить як стандартні поля (вулиця, номер будинку, місто), так і додаткові поля, які можуть бути використані в міжнародних сценаріях: район, боро, муніципалітет тощо. Це дозволяє моделі бути адаптивною до різних джерел геокодування.

Coordinate — структура, що містить широту та довготу. Є ключовим елементом у пошуку маршруту та відстані.

PostOffice — головна сутність, яка представляє відділення пошти. Включає адресу, координати, графік роботи, категорію (відділення або поштомат) та інші метадані.

Schedule / WorkingHours — відображають графік роботи відділення. Застосування record гарантує незмінність значень, що особливо важливо у

контексті багатопотоковості або повторного використання.

Такі моделі обрано через їх високу чіткість, гнучкість і відповідність реальній предметній області. Вони дозволяють побудувати систему, яку легко розширити без порушення існуючої логіки.

2.2. Компонентна структура

У системі оптимізації доставки реалізовано кілька основних компонентів, які забезпечують злагоджену роботу:

Frontend (HTML/JavaScript) надає інтерфейс користувачу для введення адреси або отримання координат через геолокацію браузера. Відправляє запити до бекенду (API-сервер). Відображає результати — список найближчих відділень та маршрут.

Backend API-сервер (ASP.NET Core або Node.js/Express) обробляє запити від фронтенду. Інтегрується з трьома зовнішніми сервісами:

- Нова Пошта API — отримання списку міст, відділень; - Nominatim

(OpenStreetMap) — пряме і зворотне геокодування;

- OSRM — побудова маршрутів за координатами.

Визначає найближче відділення та надсилає інформацію назад клієнту. Нова Пошта API: надає структуровану інформацію про пункти доставки. Nominatim: трансформує адресу в координати та навпаки. Frontend ізольовано від логіки взаємодії з зовнішніми сервісами, це дозволяє спростити інтерфейс і мінімізувати обсяг коду, що виконується на клієнті.

Backend API сервер виконує роль посередника і агрегатора: він зводить дані з кількох джерел, виконує геообчислення, працює з координатами. Використання адаптерів дає змогу у майбутньому легко змінити провайдери

API без зміни логіки системи. Зовнішні сервіси не зберігають стану

користувача, тому система може працювати без реєстрації, з максимальною швидкістю і прозорістю.

На рисунку 2.2 продемонстрована схема системи з усіма її компонентами

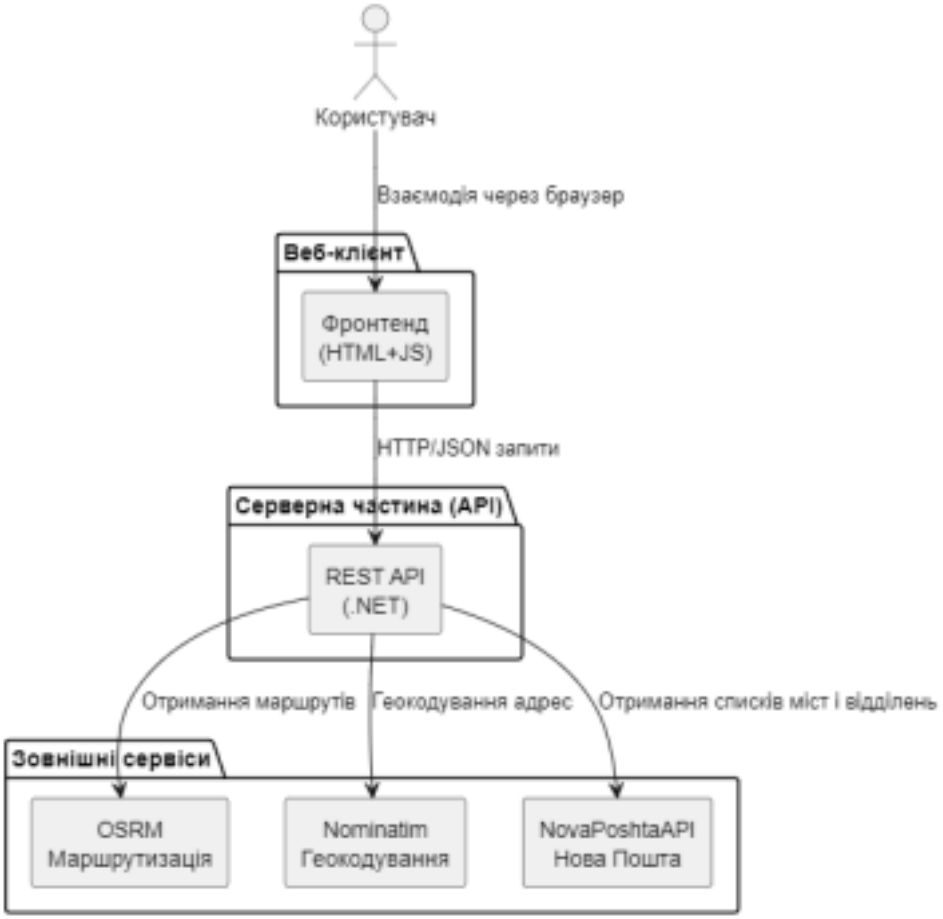


Рисунок 2.2 – Компонента схема системи геологістичного сервіса

2.3 Взаємодія сервісів

Система реалізує три основні сервіси:

- IGeocodingService — відповідає за перетворення між адресою та координатами. Є обгорткою над зовнішніми геокодувальними сервісами (наприклад, Nominatim чи Google Maps API);

- IRouteSearchService — розраховує маршрут між двома

координатами та надає додаткову інформацію (відстань, геометрію маршруту);

- IPostOfficeLocationService — надає список поштових відділень для заданого міста. Працює з базою даних або зовнішнім джерелом. У діаграмі послідовностей на рисунку 2.3 видно, як дані проходять через усі ключові компоненти системи: від дії користувача до відображення маршруту. Кожен крок обґрунтований з точки зору UX і технічної необхідності.

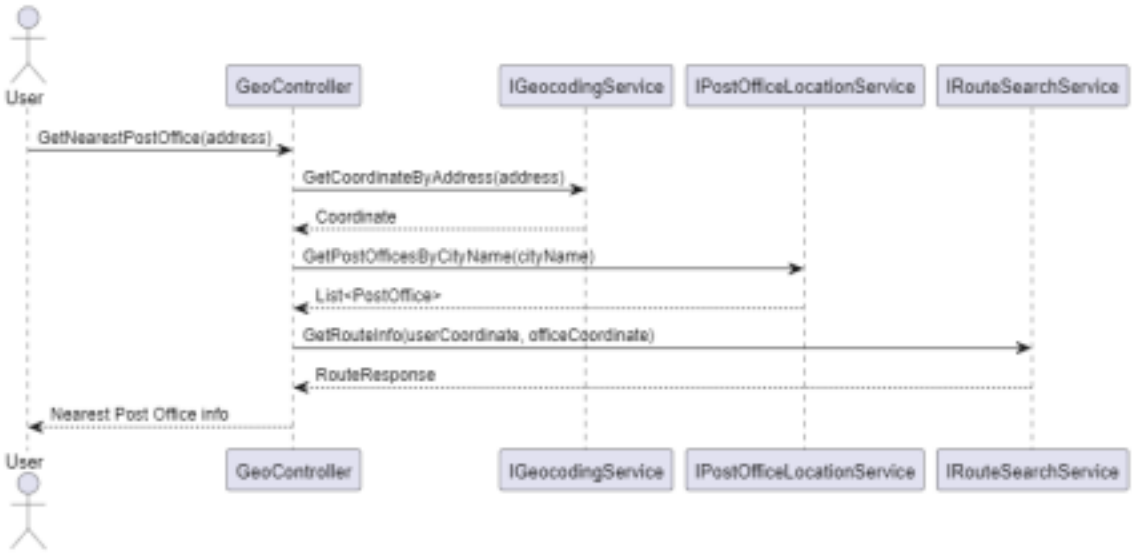


Рисунок 2.3 – UML-діаграма послідовностей сценарія отриамння інформації про найближче відділення

Всі сервіси реалізують єдиний підхід взаємодії — вони асинхронні та повертають чітко типізовані результати. Це спрощує тестування та інтеграцію.

2.3.1. Отримання геолокації або введення адреси користувачем

При першому завантаженні інтерфейсу користувачу одразу пропонується проста та інтуїтивно зрозуміла дія — натиснути кнопку "Знайти найближче відділення". Такий підхід знижує поріг входу, адже користувачеві не потрібно одразу заповнювати форми чи вводити адресу вручну. У відповідь на цю дію браузер за допомогою Web Geolocation API

ініціює запит на доступ до даних геолокації пристрою.

Сучасні мобільні та десктопні браузери (Chrome, Firefox, Safari, Edge тощо) підтримують стандартний механізм визначення місцезнаходження. Він дозволяє швидко отримати географічні координати (широта та довгота) користувача. Це значно зменшує ймовірність помилок, які можуть виникати при ручному введенні адреси: опечатки, різні формати, незрозумілий порядок заповнення полів тощо.

Таке рішення дозволяє:

- автоматизувати процес визначення місцезнаходження користувача;

- зекономити час і зробити досвід користування приємнішим;

- забезпечити високу точність стартової точки для подальшої маршрутизації до найближчого відділення;

- працювати без необхідності входу в акаунт чи реєстрації, що спрощує тестування й покращує UX.

Якщо користувач відмовляється надавати дозвіл на використання геолокації, що є цілком законним і розповсюдженим варіантом, система автоматично адаптується до цього сценарію. Замість повідомлення про помилку чи зупинки функціоналу, на екрані відображається форма ручного введення адреси, яка містить поля:

- назва міста;

- вулиця;

- номер будинку.

Це дає змогу користувачу все одно скористатися сервісом, але у менш автоматизований спосіб. Після введення адреси система використовує Nominatim для геокодування (перетворення адреси на координати), після чого процес пошуку найближчого відділення і побудова маршруту відбувається так само, як і у випадку з використанням геолокації.

Перевага адаптивного сценарію в тому, що забезпечується гнучкість інтерфейсу — незалежно від уподобань користувача або налаштувань браузера та підвищується довіра до сервісу. Користувач бачить, що йому не нав’язують геолокацію, а надають альтернативу [7].

Покращується доступність системи для користувачів зі старими пристроями або без GPS.

2.3.2. Геокодування через Nominatim

У цьому проєкті роль геокодування важко переоцінити. Це процес перетворення звичайної адресної інформації у географічні координати (широта та довгота) або навпаки, що дозволяє системі оперувати єдиним універсальним форматом даних. Для досягнення цієї мети було обрано сервіс Nominatim, реалізований на базі відкритих даних OpenStreetMap [5].

Що таке Nominatim? Nominatim — це відкритий геокодувальний сервіс, який забезпечує два головні інтерфейси HTTP:

- /search: дозволяє передавати текстову адресу і отримувати у відповідь перелік відповідних об’єктів з їх координатами. Наприклад, запит на https://nominatim.openstreetmap.org/search?q=Kyiv,+Khreshchatyk+22&format=json поверне масив записів із широтою, довготою та деталями адреси;

- /reverse: приймає координати й повертає найбільш релевантну адресу у зручному форматі. Виклик на кшталт https://nominatim.openstreetmap.org/reverse?lat=50.4501&lon=30.5234&format=json дасть об’єкт із полем display_name, яке містить повну текстову адресу.

Як звертатися до сервісу? Серверний модуль геокодування формує запит до Nominatim методом HTTP GET, зазначаючи такі обов’язкові параметри:

- format=json — щоб отримати результат у форматі JSON; - q

— текстовий рядок адреси для /search;

- lat, lon — координати для /reverse;

- додаткові параметри (addressdetails, limit), щоб отримати детальнішу інформацію або обмежити кількість відповідей. Після виконання запиту Nominatim повертає масив або об’єкт із поля display_name, а також властивості lat і lon. Ці значення обробляються модулем, і формується єдина доменна модель Location [5].

Уніфікація даних у єдиній моделі Location суттєво спрощує подальшу обробку геоданих. Всі наступні компоненти системи — від вибору найближчого відділення до побудови маршруту — працюють з керованою структурою, позбавленою неоднозначностей і залежностей від специфічних форматів зовнішніх API. Отримання списку відділень через API Нової Пошти. Сервер звертається до проксі-модуля Нової Пошти, передаючи city_id або координати для фільтрації по місту. Модуль API НП повертає перелік відділень з полями{ ref, name, latitude, longitude, type, work_time }.

Проксі дозволяє кешувати відповіді та обробляти помилки, захищаючи клієнтську частину від безпосереднього контакту з зовнішнім API.

2.3.3. Отримання списку відділень через API Нової Пошти

Одним із ключових джерел даних у даній системі є офіційне API Нової Пошти — популярної української логістичної компанії, що надає публічний доступ до широкого спектру сервісів через RESTful API. API дозволяє отримати актуальну інформацію про міста, вулиці, відділення, типи доставки, статуси відправлень тощо. У межах даного застосунку використовується, зокрема, метод getWarehouses — для отримання детального списку відділень

за певним населеним пунктом або з фільтрацією за іншими параметрами [1].

Для використання API необхідно зареєструватися на офіційному сайті Нової Пошти та отримати індивідуальний API-ключ (token), який ідентифікує розробника або застосунок. Цей ключ передається в тілі кожного запиту, і є обов’язковим для успішної авторизації на сервері Нової Пошти.

Запити до API формуються у форматі POST-запитів до єдиного endpoint'у: https://api.novaposhta.ua/v2.0/json/. У тілі запиту передається JSON-об’єкт з обов’язковими параметрами: apiKey, modelName, calledMethod, methodProperties. Наприклад, щоб отримати список відділень міста, запит матиме такий вигляд як зазначено на рисунку 2.4 [2].

```
POST /v2.0/json/
{
 "apiKey": "ВАШ_КЛЮЧ",
 "modelName": "Address",
 "calledMethod": "getWarehouses",
 "methodProperties": {
  "CityName": "Кривий Ріг"
 }
}
```

Рисунок 2.4 – Приклад оформлення запиту до API «Нова Пошта»

У відповіді сервер повертає масив об’єктів, кожен з яких містить дані про конкретне відділення: унікальний ідентифікатор (ref), назву (description),

геокоординати (longitude, latitude), тип (тип відділення: поштове, поштомат тощо), графік роботи, та інші метадані.

Для оптимізації взаємодії з API використовується проміжний сервер — проксі-модуль. Його основні завдання:

- кешування відповідей для уникнення повторних запитів; -

централізована обробка помилок або некоректних відповідей; - захист API-

ключа від публічного доступу (не розміщується на клієнті);

- стандартизація структури відповіді: проксі формує уніфіковану модель Branch, яку вже обробляє фронтенд [3].

Обґрунтування такої архітектури полягає в покращенні стабільності, безпеки та масштабованості системи. Прямий доступ клієнта до API Нової Пошти може призвести до компрометації ключа, перевантаження серверів та складнощів з обробкою помилок або змін у форматі відповіді. Через проксі всі ці ризики знижуються, а також відкривається можливість для локального кешування, що суттєво зменшує час відповіді системи [1].

2.3.4. Алгоритм пошуку найкращого маршруту до відділення Однією з ключових задач системи є визначення найближчого логістичного відділення до поточного розташування користувача [8]. Це завдання має не лише інтерфейсне, а й суттєве функціональне та технічне значення: вибір правильного відділення напряму впливає на зручність, швидкість та ефективність доставки.

У великих містах кількість відділень може сягати десятків або навіть сотень, і побудова маршруту до кожного з них за допомогою OSRM є ресурсоємною операцією. OSRM — хоч і високоефективний, однак потребує значного часу й обчислень при виконанні навіть одного запиту маршрутизації, особливо якщо у карті враховано дорожню інфраструктуру, тип покриття, дозволи на рух тощо [6].

Рішення – попереднє фільтрування за формулою Haversine. Щоб зменшити навантаження, перш ніж будувати маршрути, система застосовує географічне попереднє обрізання — фільтрацію списку відділень за простою та швидкою метрикою: геодезичною відстанню між координатами. Для цього використовується формула Haversine, яка дозволяє швидко оцінити "якби по прямій" (без урахування доріг), які відділення найближчі. Формула гаверсинуса виглядає наступним чином:

$$a = \sin^2(\Delta\varphi / 2) + \cos(\varphi 1) * \cos(\varphi 2) * \sin^2(\Delta\lambda / 2)$$

$$c = 2 * \operatorname{atan2}(\sqrt{a}, \sqrt{(1-a)}) \quad (2.1) \quad d = R * c$$

де:

$\varphi 1$, $\varphi 2$ — широти у радіанах;

$\Delta\varphi$ — різниця широт;

$\Delta\lambda$ — різниця довгот;

R — радіус Землі (~6371 км);

d — відстань між точками в кілометрах.

Ця формула враховує сферичну форму Землі, тому дає прийнятно точні результати навіть на масштабі міста [8].

Після отримання координат користувача, сервер бере весь список відділень у відповідному місті або регіоні.

До кожного відділення обчислюється приблизна географічна відстань до користувача через Haversine. Відділення, які знаходяться далі певного порогового значення (наприклад, 5–10 км), відкидаються. Залишається лише обмежена кількість "кандидатів" — 3–5 відділень, які потенційно є найближчими. Для цих кандидатів вже використовується OSRM, який будує реальний маршрут з урахуванням дорожньої мережі. Такий підхід дає оптимальний баланс між швидкодією і точністю:

- попередній фільтр працює дуже швидко та дозволяє відразу відсікти

непотрібні об'єкти;

- використання OSRM для невеликої кількості об'єктів гарантує,

що обраний маршрут буде справді зручним і неформальним (наприклад, уникатиме мостів, односторонніх вулиць тощо);

- кінцевий користувач отримує інтерфейс, який майже миттєво

видає найкраще відділення без потреби вручну вивчати списки або вводити назви.

Таким чином, формула Haversine в системі не є основним механізмом маршрутизації, але виступає критично важливою проміжною оптимізацією, яка зменшує обсяг роботи для більш "важкого" модуля — OSRM.

Завершальним і водночас ключовим етапом у логіці визначення зручного відділення є розрахунок маршруту від місцезнаходження користувача до обраного пункту призначення — відділення Нової Пошти. Цей процес реалізується за допомогою OSRM (Open Source Routing Machine) — одного з найпопулярніших сервісів маршрутизації на основі даних OpenStreetMap.

OSRM — це високопродуктивний рушій маршрутизації, який дозволяє будувати маршрути для автомобілів, велосипедів або пішоходів. Він працює на попередньо оброблених геоданих із відкритого проєкту OpenStreetMap, що дає змогу виконувати запити з дуже високою швидкістю. У рамках цього застосунку використовується локально розгорнутий OSRM-сервер, що дозволяє уникнути залежності від сторонніх сервісів, контролювати оновлення карти та не обмежуватися платними тарифами [6].

Після того як система визначає координати користувача (або введену вручну адресу, перетворену через Nominatim) та координати обраного відділення, формується спеціальний HTTP-запит до локального сервера OSRM у наступному вигляді

/route?coordinates=<user_lon>,<user_lat>;<office_lon>,<office_lat>&overview=full&geometries=geojson. Тут coordinates — набір координат точки старту та точки призначення, overview=full — запит на детальну інформацію про маршрут (у повному вигляді), geometries=geojson — формат геоданих, зручний для відображення на карті.

OSRM обробляє запит і повертає JSON-відповідь, у якій міститься:

- масив координат усієї траси у форматі GeoJSON;

- загальна довжина маршруту (у метрах);

- орієнтовна тривалість проходження (у секундах);

- інструкції навігації (turn-by-turn) — за потреби [7].

Після отримання відповіді клієнтська частина застосунку (HTML/JavaScript інтерфейс) здійснює візуалізацію маршруту на карті. Для цього використовується бібліотека Leaflet.js, яка підтримує накладання GeoJSON-маршрутів поверх інтерактивної мапи. Маршрут автоматично центрується на екрані, на початковій та кінцевій точках розміщуються маркери з відповідними підписами ("Ви тут", "Відділення №..."), а лінія маршруту відображається як ламана (polyline) з можливістю налаштування стилю.

Використання OSRM як автономного компонента в архітектурі системи дає низку переваг. Попередньо згенерований граф доріг дозволяє будувати маршрути за мілісекунди навіть для великої мапи. Локальний сервер гарантує стабільність, відсутність лімітів на запити та можливість кастомізації (наприклад, виключення певних типів доріг).

Сервер можна оновлювати з новими даними OpenStreetMap, що дозволяє адаптувати систему до змін у дорожній інфраструктурі. Крім того, розділення відповідальностей між компонентами (геокодування, вибір

відділення, маршрутизація) дозволяє легко замінювати окремі модулі або розширювати функціонал без втручання у всю систему загалом [5]. Побудова маршруту через OSRM завершує цикл інтерактивного пошуку найближчого відділення. Користувач бачить не лише, куди йому йти, але й як саме. Це суттєво підвищує юзабіліті сервісу та перетворює класичний ручний вибір відділення зі списку на сучасний, зручний і контекстний інструмент.

2.4. Вибір технологій

У процесі розробки програмного забезпечення для пошуку найближчого відділення Нової пошти на основі геолокації користувача було

прийнято рішення про використання низки перевірених і взаємодоповнюючих технологій. Вибір обґрунтовується вимогами до швидкої обробки запитів, надійності API, зручності розгортання і підтримки, а також необхідністю інтеграції з сторонніми геосервісами [1].

Як основну платформу для розробки серверної частини обрано ASP.NET Core. Це сучасний кросплатформний фреймворк від Microsoft, який дозволяє створювати високопродуктивні веб-додатки та REST API. Його використання дає змогу реалізувати логіку обробки запитів, виклики до сторонніх API, маршрутизацію та перевірку координат, геокодування й зворотне геокодування.

Підтримка архітектури REST, що ідеально підходить для побудови API-сервісів. Зручна підтримка впровадження залежностей (DI), що полегшує тестування та модульність. Вбудована система логування, конфігурацій, обробки помилок. Висока продуктивність, масштабованість та можливість хостингу на різних платформах. Серверна частина активно взаємодіє з трьома зовнішніми API:

- Nominatim API (OpenStreetMap) — для геокодування (адреса → координати) та зворотного геокодування (координати → адреса);

- OSRM (Open Source Routing Machine) — для обчислення маршруту та дистанції між точками, що використовується для знаходження найближчого відділення;

- Nova Poshta API — для отримання списку відділень у заданому місті, включаючи координати, розклад, тип (відділення / поштомат) та іншу корисну інформацію.

Ці сервіси забезпечують точність, актуальність і гнучкість геолокаційних запитів.

IGeocodingService — інкапсулює логіку запитів до Nominatim. IRouteSearchService — обробляє обчислення маршрутів через OSRM. IPostOfficeLocationService — відповідальний за завантаження відділень Нової пошти з офіційного API.

Кожен сервіс має чітко визначену відповідальність, що дозволяє легко їх розширювати, тестувати і змінювати без впливу на інші частини системи. Клієнтська частина реалізується з використанням HTML, JavaScript і за потреби Leaflet.js або аналогічної бібліотеки для відображення мап та маркерів.

Переваги вибору такого підходу:

- простота реалізації UI з базовими формами введення адреси, показу карти та відділень;

- взаємодія з бекендом через AJAX/Fetch API;

- зручність використання мапових бібліотек із відкритими даними OSM;

- не потребує складного фреймворку, достатньо легко інтегрується в будь-яке середовище.

Docker використовується для запуску локального інстансу OSRM-сервера, що дозволяє:

- працювати без зовнішніх обмежень (рейт-лімітів) сторонніх OSRM API;

- використовувати власні карти (наприклад, тільки для України чи окремого регіону);

- забезпечити ізольоване середовище для тестування та продакшену.

Переваги Docker:

- швидке розгортання OSRM-сервера з підключеною картою (наприклад, ukraine-latest.osm.pbf);

- незалежність від хостингу чи зовнішніх API (немає ризику втрати доступу);

- можливість масштабування та моніторингу.

Нижче на рисунку 2.5 продемонстрований приклад запущеного сервісу OSRM з завантаженою в нього картою України і налаштованої на порті 5000.

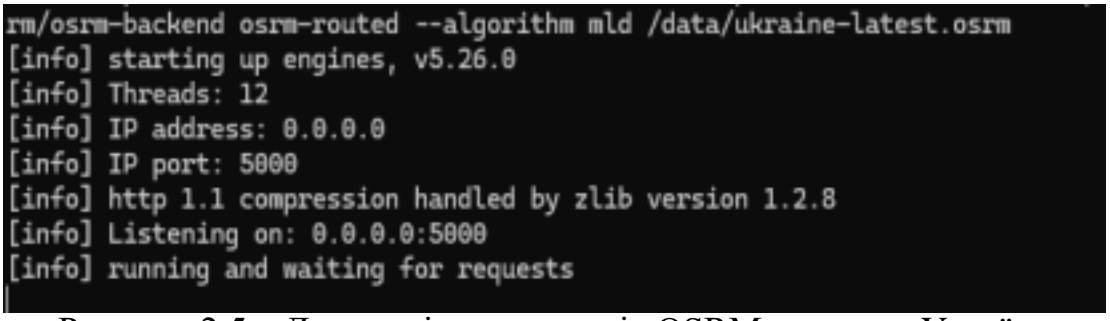


Рисунок 2.5 – Лог сервісу маршрутів OSRM з картою України

Обраний стек технологій повністю відповідає вимогам до створення надійного, модульного та продуктивного геолокаційного сервісу. Він дозволяє швидко інтегрувати відкриті й комерційні API, точно обчислювати

координати, маршрути й відстані, а також знаходити найближчі відділення Нової пошти для користувача [1]. Застосування ASP.NET Core на бекенді, разом із гнучким frontend-ом та контейнеризацією (Docker) створює надійну платформу для розширення функціоналу в майбутньому.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ

У проєкті реалізовано три ключові модулі, які відповідають за обробку геолокаційних запитів, взаємодію з Новою Поштою та побудову маршрутів. Їхня координація дозволяє знайти найближче відділення до заданих координат або адреси.

3.1 Модуль взаємодії з API Нової Пошти (кастомний проксі) Цей модуль виконує роль проксі-клієнта до API Нової Пошти. Він реалізований у вигляді сервісу PostOfficeLocationService, який дозволяє отримати список відділень у місті за назвою. Приклад методу клієнта, що отримує список всіх міст України через API Нової Пошти наведено на рисунку 3.1.

```
public async Task<List<NpCity>> GetAllCitiesAsync()
{
  var cities = new List<NpCity>();
  int page = 1;
  bool hasNextPage;

  do
  {
    var methodProperties = new
    {
      Page = page.ToString(),
      Limit = "100" // Максимум 100 записів на сторінку
    };

    var result = await SendRequestAsync<NpResponse<List<NpCity>>>("AddressGeneral", "getCities", methodProperties);
    cities.AddRange(result.Data);

    hasNextPage = result.Data.Count == 100;
    page++;
  } while (hasNextPage);

  return cities;
}
```

Рисунок 3.1– Приклад методу звернення до API Нової Пошти

Взаємодія з Новою Поштою здійснюється через клас NovaPoshtaHttpClient, який серіалізує запити в JSON і виконує їх через HTTP POST. Кожен запит формує окрему структуру із зазначенням modelName,

calledMethod та параметрів.

3.2 Модуль геокодування (Nominatim API)

Модуль геокодування забезпечує перетворення адреси у координати (і навпаки) з використанням публічного сервісу Nominatim. Сервіс GeocodingService викликає відповідні методи клієнта NominatimHttpClient. Використовуються два основні HTTP-запити:

- /search — для геокодування (адреса → координати);

- /reverse — для зворотного геокодування (координати → адреса).

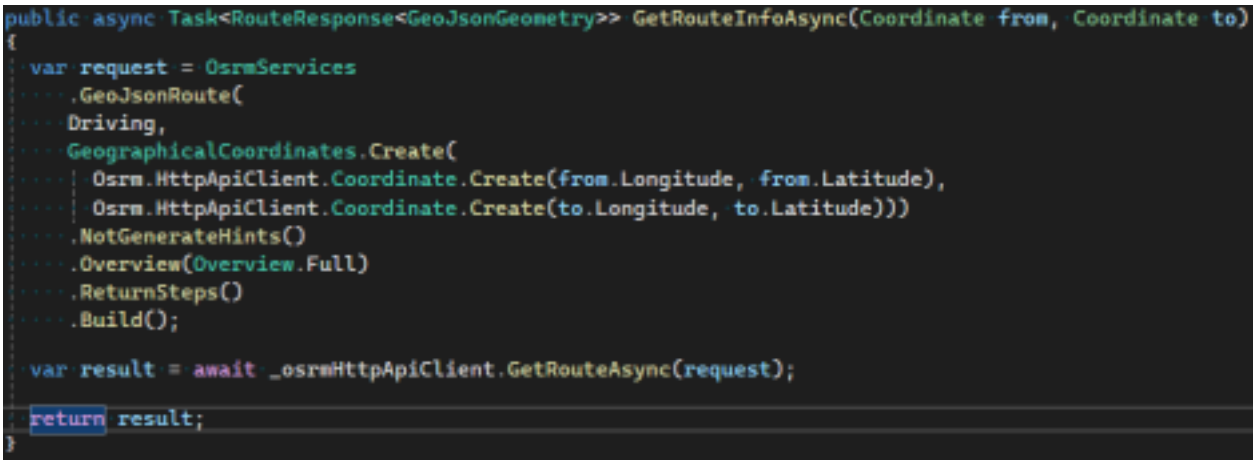
Приклад отримання адреси з координати продемонстровано на рисунку 3.2.

```
public async Task<NominatimReverseResponse> GetAddressByCoordinateAsync(double latitude, double longtitude)
{
  ///https://nominatim.openstreetmap.org/reverse?format=json&lat=47.9105&lon=33.3918&accept-language=uk_UA&zoom=18&address
  var lat = latitude.ToString(CultureInfo.InvariantCulture);
  var lon = longtitude.ToString(CultureInfo.InvariantCulture);

  var url = $"{BaseUrl}/reverse?format={Format}&lat={lat}&lon={lon}&accept-language={Accept_Language}&addressdetails={Addr
  using var request = new HttpRequestMessage(HttpMethod.Get, url);

  var response = await _httpClient.SendAsync(request);
  response.EnsureSuccessStatusCode();

  //var resString = await response.Content.ReadAsStringAsync();
  var result = await response.Content.ReadFromJsonAsync<NominatimReverseResponse>();

  return result ?? throw new Exception("Nominatim respoese not found");
}
```

Рисунок 3.2 – Метод отримання адреси з координати через зовнішній сервіс Nominatim

3.3 Модуль маршрутизації (OSRM)

Для обчислення відстані та побудови маршруту між поточним місцем користувача і відділенням використовується модуль маршрутизації, який реалізовано через клієнт до OSRM (Open Source Routing Machine). Сервіс OsrmRoutingService викликає API OSRM у режимі "driving". Приклад отримання детального маршруту вказано на рисунку 3.3. На ньому є метод що приймає дві координати та повертає GeoJson [6].



```
public async Task<RouteResponse<GeoJsonGeometry>> GetRouteInfoAsync(Coordinate from, Coordinate to)
{
    var request = OsrmServices
        .GeoJsonRoute(
        Driving,
        GeographicalCoordinates.Create(
            Osrm.HttpApiClient.Coordinate.Create(from.Longitude, from.Latitude),
            Osrm.HttpApiClient.Coordinate.Create(to.Longitude, to.Latitude)))
        .NotGenerateHints()
        .Overview(Overview.Full)
        .ReturnSteps()
        .Build();

    var result = await _osrmHttpApiClient.GetRouteAsync(request);

    return result;
}
```

Рисунок 3.3 – Приклад методу, що дозволяє побудувати маршрут між двома точками

Маршрут повертається у форматі GeoJSON, що дозволяє використовувати результат як у картографічних бібліотеках, так і для обчислення найкоротшого шляху. Ці три модулі взаємодіють для побудови повного циклу:

1. адреса;
2. координати;
3. список відділень;
4. вибір найближчого;
5. побудова маршруту.

Такий підхід забезпечує масштабованість і можливість підключення альтернативних сервісів у майбутньому [6].

3.4 Підключення автогенерованої документації OpenAPI

Для полегшення розробки, тестування та інтеграції зовнішніх систем із REST API був реалізований механізм автогенерації документації за допомогою OpenAPI (також відомого як Swagger). Це забезпечує візуальний інтерфейс для взаємодії з API, що дозволяє переглядати структуру запитів і відповідей, параметри, типи даних та статуси відповідей без необхідності додаткової ручної документації [1].

OpenAPI/Swagger дозволяє:

- автоматично документувати усі доступні маршрути API; - інтерактивно

тестувати запити безпосередньо в браузері; - генерувати клієнтські SDK для різних мов програмування; - Забезпечити єдине джерело правди (Single Source of Truth) щодо доступних кінцевих точок (endpoint'ів) [3].

У проєкті, побудованому на платформі .NET9, для реалізації OpenAPI-документації було використано стандартний пакет Swashbuckle.AspNetCore. Де кожен ендпоінт супроводжується описом, прикладами параметрів та відповідей, що значно покращує досвід взаємодії для користувачів, тестувальників і зовнішніх інтеграторів. [2]

Перейшовши за адресою https://localhost:5023/scalar потрапляємо на стрінку з документацією і інструментами тестування існуючих точок доступу. Її вигляд можна побачити на рисунку 3.4.

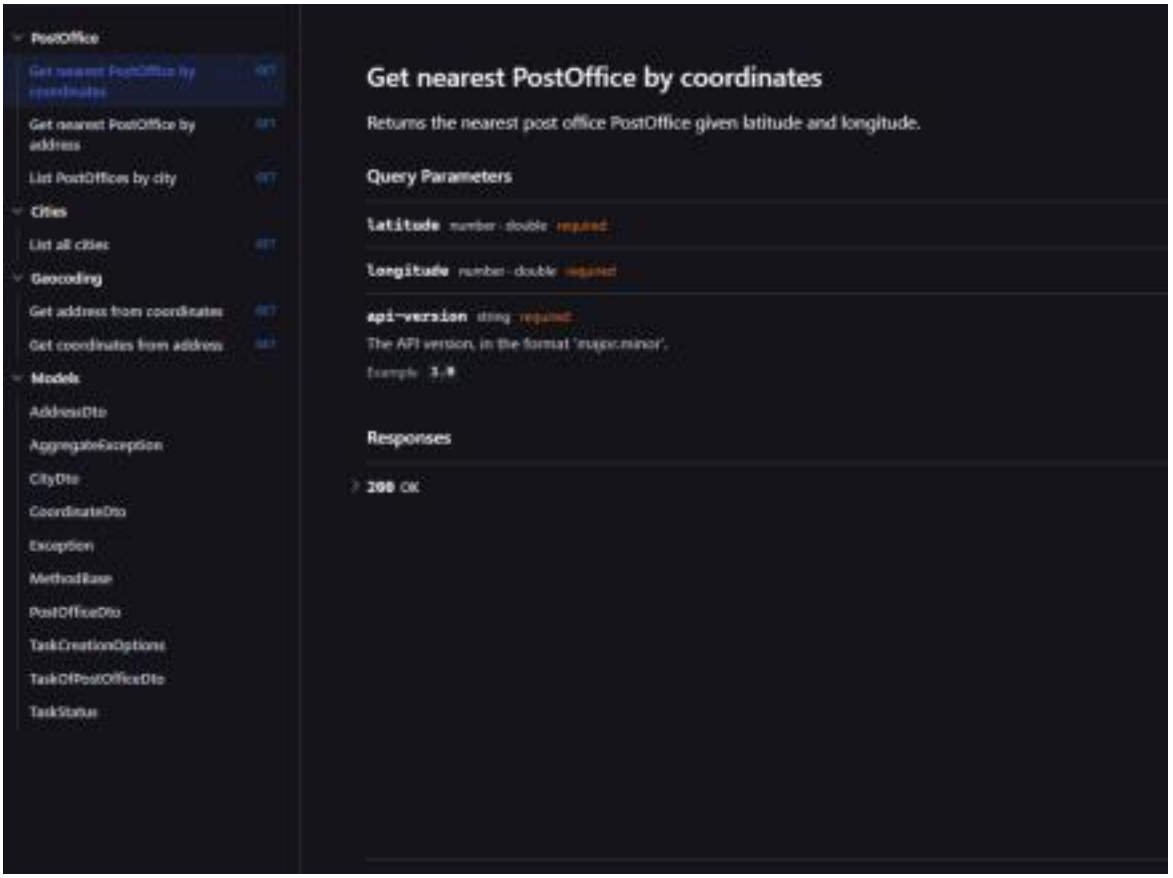


Рисунок 3.4 – Сторінка перегляду всіх доступних ендпоінтів програми

Також ця автогенерована документація дозволяє тестувати точки доступа, показує результат та можливі відповіді сервера. Приклад можна побачити на

рисунку 3.5.

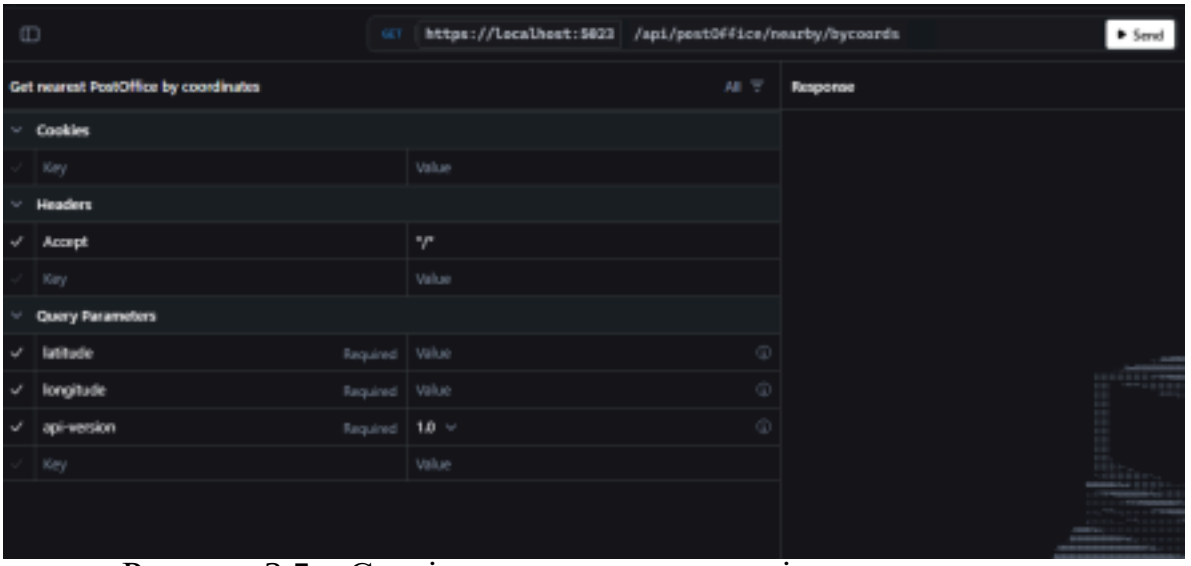


Рисунок 3.5 – Сторінка тестування ендпоінта з можливими параметрами

РОЗДІЛ 4
ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЯ

Останній четвертий розділ присвячений опису завершального етапу розробки системи, під час якого програмний продукт. Готується до розгортання — описуються підготовчі роботи, інсталяція, налаштування необхідних сервісів (наприклад, баз даних, Docker-контейнерів, хостів). Розгортається у робочому середовищі — показано, як саме відбувається запуск і конфігурація проєкту на сервері або у хмарному середовищі.

Надається інструкція для користувача — описано, як користуватися системою: пошук, фільтрація, авторизація, перегляд даних тощо. Розглядаються питання супроводу — що потрібно для стабільної експлуатації, резервного копіювання, оновлення, моніторингу.

Надаються рекомендації щодо подальшої підтримки, масштабування, безпеки та можливого розширення функціоналу. Цей розділ показує готовність системи до реального використання та її зручність як для користувачів, так і для адміністраторів.

4.1 Інструкція користувача

Для створення інтуїтивно зрозумілого та надійного сервісу доставки було спроєктовано логічний та простий флоу взаємодії користувача із системою. Нижче на рисунку 4.1 наведено приклад користувацького інтерфейсу, який ілюструє ці можливості.

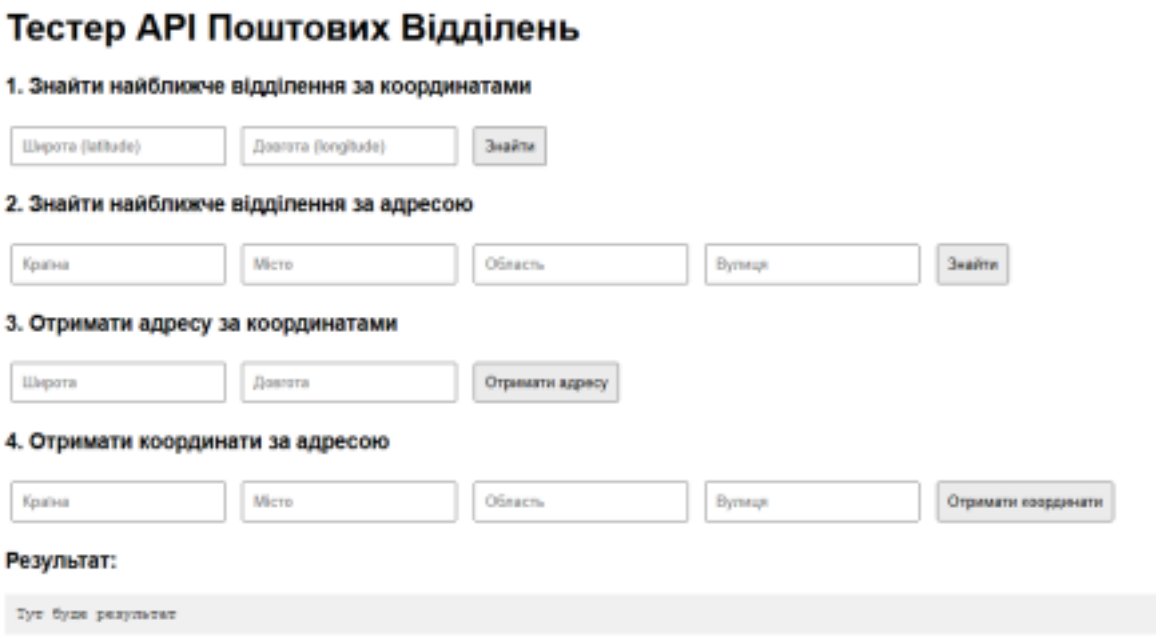


Рисунок 4.1 – HTML-сторінка тестера застосунку

Кожен крок має свою мотивацію та приносить конкретну користь для зручності користувача:

Як ввести адресу або координати? Користувач може ввести географічні координати (широту та довготу) або ж повну адресу, яка включає країну, місто, область та вулицю. Це дозволяє максимально точно визначити місце пошуку або доставки [7].

Як переглянути найкоротший маршрут? Після введення початкової та кінцевої точок у вигляді координат, система обчислює найкоротший маршрут, використовуючи сервіс OSRM. Це допомагає швидко оцінити відстань і час доставки [6].

Як дізнатись інформацію про відділення? Користувач може отримати детальний список найближчих відділень Нової Пошти за заданими координатами або адресою, що підвищує зручність вибору найбільш оптимального відділення для відправлення або отримання посилки.

На рисунку 4.2 показано приклад отримання інформації про найближче відділення. Ми вводимо координати 47.92481518020041, 33.325592800200276 що відповідають адресі м. Кривий Ріг, вул Туполєва 1.

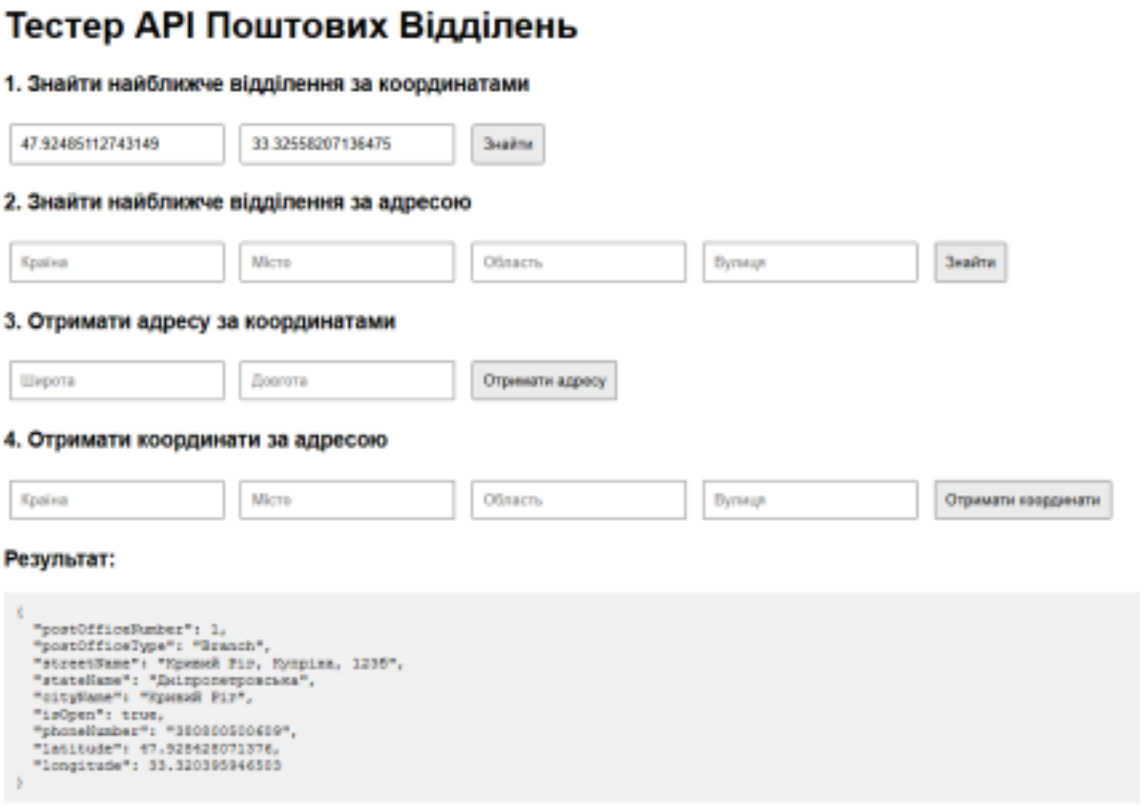


Рисунок 4.2 – Приклад отримання відповіді про найближче відділення

У відповіді можемо прочитати що найближчим відділенням є відділення №1 за адресою м. Кривий Ріг, Купріна 123б. За допомогою відкритих сервісів таких як Google Maps можемо впевнитись, що це відділення насправді є найближчим.

На рисунку 4.3 показано приклад отримання інформації про найближче відділення. Ми вводимо адресу Україна, м. Кривий Ріг, вул. Алмазна.

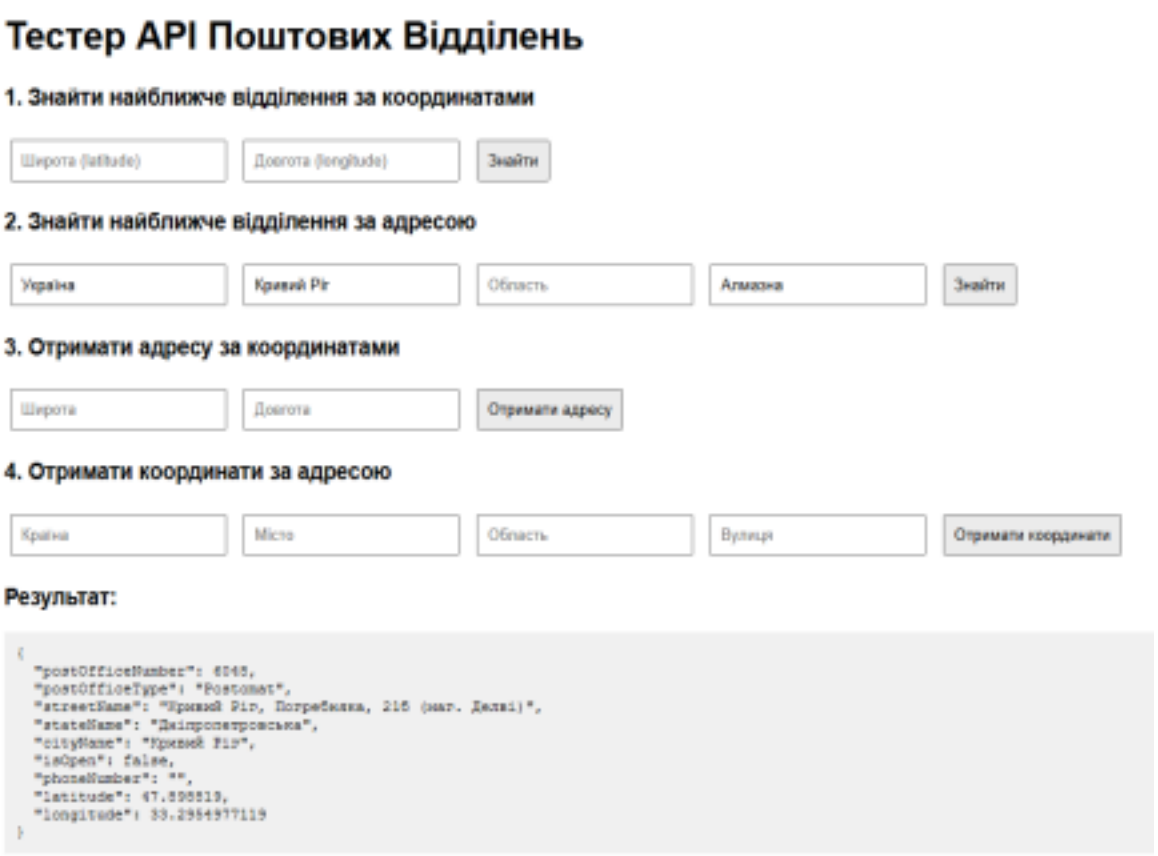


Рисунок 4.3 – Приклад отримання відповіді про найближче відділення за адресою

У відповіді можемо прочитати, що найближчим відділенням є поштомат №6048 за адресою Кривий Ріг, Погребняка, 21б (магазин Делві). За допомогою відкритих сервісів таких як Google Maps можемо впевнитись, що це відділення насправді є найближчим.

На рисунку 4.4 показано приклад отримання інформації про про координати за адресою. Ми вводимо адресу Україна, м. Кривий Ріг, вул. Алмазна.

Рисунок 4.4 – Приклад отримання відповіді про координати за адресою

В результаті отримані координати 47.8996, 33.29514, що є правильним і очікуваним результатом.

4.2 Інструкція адміністратора

Для забезпечення стабільної роботи та актуальності даних сервісу адміністратор повинен виконати наступні дії:

- налаштування доступу до API Нової Пошти. Вказати дійсний API-ключ у конфігурації сервісу, що дозволяє отримувати актуальні дані про міста та відділення;

- розгортання локального OSRM-сервера (за потреби). Для зниження залежності від зовнішніх сервісів та підвищення швидкодії можна налаштувати власний сервер маршрутизації OSRM із завантаженими картами.

Для використання публічного API сервісу доставки «Нова Пошта» необхідно попередньо зареєструватися в особистому кабінеті користувача та

отримати унікальний API-ключ. Цей ключ дозволяє автентифікувати запити та отримувати доступ до даних про відділення, адреси, тарифи, відправлення та інші сервіси.

API-ключ (API key) — це унікальний ідентифікатор, який використовується для автентифікації запитів до програмного інтерфейсу прикладного програмування (API). Його основна мета — забезпечити контроль доступу до функціоналу сервісу або зовнішньої системи. API-ключі виконують такі функції:

- Ідентифікація користувача або програми, яка надсилає запит;

- Забезпечення безпеки, обмежуючи доступ до API тільки авторизованим клієнтам;

- Ведення журналу активності, що дозволяє сервісам відслідковувати використання API;

- Обмеження кількості запитів (rate limiting) — для запобігання перевантаженню сервісу.

Нижче подано покрокову інструкцію щодо отримання API-ключа: -

перейдіть на офіційний сайт Нової Пошти: https://novaposhta.ua’

- у правому верхньому куті натисніть кнопку «Вхід»;

- авторизуйтесь за допомогою номера телефону або email, або зареєструйтесь, якщо ще не маєте облікового запису;

- після входу відкрийте особистий кабінет;

- регулярно перевіряти актуальність ключів та оновлювати їх у разі необхідності, а також оновлювати бази даних з інформацією про населені пункти і відділення;

- у меню зліва або зверху виберіть пункт «Налаштування»;

- перейдіть до вкладки «API» або «Інтеграції» (може змінюватися в залежності від версії сайту);

- натисніть кнопку «Створити API-ключ» або «Згенерувати ключ»;

- вкажіть назву або опис ключа (наприклад: «Для тестування логістичного модуля»);

- натисніть «Створити» — система згенерує унікальний API-ключ;

- збережіть ключ або скопіюйте його в безпечне місце — надалі він буде використовуватись у запитах до API.

Приклад коректно отриманих ключів для доступу до API зображено на рисунку 4.5.

Рисунок 4.5 – Меню отримання API ключа Нової Пошти

Потім отриманий ключ треба внести в конфігураційний файл appsettings.json в поле NovaPoshta.ApiKey, як це показано на рисунку 4.6.

Рисунок 4.6 – Приклад заповнення конфігураційного файлу appsettings.json

При розробці та підтримці системи важливо зосередити увагу на: - інтеграції API з різних джерел, таких як Nova Poshta API для відділень, Nominatim для геокодування та OSRM для маршрутизації; - логіці пошуку найближчого відділення, яка має бути швидкою і

точною, щоб користувачі отримували релевантні результати; - використанні відкритих сервісів (OpenStreetMap, Nominatim), що забезпечує безкоштовність і гнучкість, а також має широке покриття; - практичному застосуванні, оскільки всі ці рішення покращують зручність користувача, оптимізують логістику та підвищують якість доставки.

ВИСНОВКИ

Кваліфікаційна робота присвячена розробці системи для пошуку найближчих відділень поштової служби з використанням геокодування та маршрутизації. У ході виконання дипломної роботи було детально проаналізовано існуючі проблеми, пов’язані з оптимізацією пошуку та покращенням логістики доставки, що актуально в умовах стрімкого розвитку електронної комерції та зростання обсягів поштових відправлень.

Під час роботи були проведені моделювання ключових компонентів системи, а також реалізовано програмне забезпечення, що інтегрує API Нової Пошти, сервіси геокодування (Nominatim) та маршрутизації (OSRM). Це дозволило створити інтуїтивно зрозумілий та надійний інструмент для користувачів, який забезпечує:

- швидкий і точний пошук найближчих поштових відділень за введеними адресами або координатами;

- можливість отримувати інформацію про маршрути та найкоротші шляхи між точками, що значно спрощує планування доставки; - зручний інтерфейс користувача з можливістю вводити дані різними способами, що підвищує гнучкість і доступність сервісу.

Серед основних переваг розробленої системи варто виділити:

- високу інтеграцію з відкритими та комерційними API, що забезпечує актуальність і достовірність даних;

- можливість масштабування і налаштування під різні регіони завдяки використанню відкритих картографічних сервісів;

- автоматизацію рутинних процесів пошуку і маршрутизації, що підвищує ефективність логістики.

Водночас, в процесі розробки було виявлено і деякі недоліки, які потребують подальшого опрацювання:

- залежність від стабільності зовнішніх API, що може впливати на доступність та швидкодію сервісу;

- обмеження у роботі з геокодуванням для малих населених пунктів, де дані можуть бути неповними або застарілими;

- відсутність розширеної аналітики і персоналізації для користувачів, що могло б покращити якість рекомендацій.

У майбутньому планується розширити функціональність системи шляхом впровадження:

- підтримки роботи з додатковими поштовими сервісами і кур’єрськими

компаніями, що зробить систему більш універсальною; - розробки мобільного додатку з інтеграцією GPS-навігації для зручності користувачів у реальному часі;

- використання машинного навчання для прогнозування завантаженості відділень та оптимізації маршрутів;

- покращення інтерфейсу користувача з додаванням адаптивного дизайну і голосового введення.

Отже, реалізована система є ефективним інструментом для автоматизації процесів пошуку та маршрутизації у сфері поштових послуг, що сприяє підвищенню якості обслуговування і зручності для кінцевих користувачів. Подальший розвиток і вдосконалення цієї системи відкриває

широкі перспективи для впровадження інновацій у логістиці і сфері доставки.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Постановка задач і основи логістики доставки. [Електронний ресурс] Режим доступу: https://logisticsinfo.ua (дата звернення: 10.05.2025) 2. Нові технології у сфері геокодування та картографії / Іваненко О. П. – Київ: Наукова думка, 2020. – 256 с.

3. Все про електрику. [Електронний ресурс] Режим доступу: http://electricalschool.info (дата звернення: 12.04.2024)

4. Джексон М. «Arduino для всіх: Як створювати інтерактивні пристрої». Тернопіль: Навчальна книга - Богдан, 2021.

5. OpenStreetMap Wiki. [Електронний ресурс] Режим доступу: https://wiki.openstreetmap.org (дата звернення: 11.05.2025)

6. Routing Machine (OSRM) – офіційна документація. [Електронний ресурс] Режим доступу: http://project-osrm.org (дата звернення: 11.05.2025) 7. Інтеграція API поштових служб у сучасні IT-системи / Петренко В. С. – Харків: ХНУ, 2023. – 180 с.

8. Геокодування та зворотне геокодування: методи і практичне застосування / Смірнова Т. В. – Львів: Видавництво ЛНУ, 2019. – 142 с.

ДОДАТОК А

Код сервісу геокодінгу

```
public class GeocodingService(
 ILogger<GeocodingService> logger,
 NominatimHttpClient nominatimHttpClient,
 IMapper mapper)
 : IGeocodingService
 {
 private readonly ILogger<GeocodingService> _logger = logger;
 private readonly NominatimHttpClient _nominatimHttpClient = nominatimHttpClient;
private readonly IMapper _mapper = mapper;

 public async Task<Address> GetAddressByCoordinateAsync(Coordinate coordinate)
{
 var response = await
_nominatimHttpClient.GetAddressByCoordinateAsync(coordinate.Latitude,
coordinate.Longitude);

 var address = _mapper.Map<Address>(response.Address);

 return address;
 }

 public async Task<Coordinate> GetCoordinateByAddressAsync(Address address)
{
 var fullAddress = address.ToFullAddressString();
 var response = await _nominatimHttpClient.GetCoordinateByAddressAsync(fullAddress);
var coordinate = CoordinateExtensions.CreateCoordinate(response.Lat, response.Lon);
return coordinate;
 }
 }
```


ДОДАТОК Б

Код сервісу побудови маршрутів

```
using GeoPost.API.Domain.Interfaces;
using Osrm.HttpApiClient;

using Coordinate = GeoPost.API.Domain.Coordinate;

namespace GeoPost.API.Application.Services;

public class OsrmRoutingService(OsrmHttpApiClient osrmHttpApiClient) :
IRouteSearchService
 {
 private readonly OsrmHttpApiClient _osrmHttpApiClient = osrmHttpApiClient;

private const string Driving = "driving";

 public async Task<RouteResponse<GeoJsonGeometry>> GetRouteInfoAsync(Coordinate from,
Coordinate to)
 {
 var request = OsrmServices
 .GeoJsonRoute(
 Driving,
 GeographicalCoordinates.Create(
 Osrm.HttpApiClient.Coordinate.Create(from.Longitude, from.Latitude),
Osrm.HttpApiClient.Coordinate.Create(to.Longitude, to.Latitude)))
.NotGenerateHints()
 .Overview(Overview.Full)
 .ReturnSteps()
 .Build();

 var result = await _osrmHttpApiClient.GetRouteAsync(request);

 return result;
 }
}
```


ДОДАТОК В

Код сервісу побудови маршрутів

```
using GeoPost.API.Domain.Interfaces;
using Osrm.HttpApiClient;

using Coordinate = GeoPost.API.Domain.Coordinate;

namespace GeoPost.API.Application.Services;

public class OsrmRoutingService(OsrmHttpApiClient osrmHttpApiClient) :
IRouteSearchService
 {
 private readonly OsrmHttpApiClient _osrmHttpApiClient = osrmHttpApiClient;

private const string Driving = "driving";
```

```
 public async Task<RouteResponse<GeoJsonGeometry>> GetRouteInfoAsync(Coordinate from,
Coordinate to)
 {
 var request = OsrmServices
 .GeoJsonRoute(
 Driving,
 GeographicalCoordinates.Create(
 Osrm.HttpApiClient.Coordinate.Create(from.Longitude, from.Latitude),
Osrm.HttpApiClient.Coordinate.Create(to.Longitude, to.Latitude)))
.NotGenerateHints()
 .Overview(Overview.Full)
 .ReturnSteps()
 .Build();

 var result = await _osrmHttpApiClient.GetRouteAsync(request);

 return result;
 }
}
```


ДОДАТОК Г

Код сервісу інтеграції API «Нової-пошти»

```
public class PostOfficeLocationService(
 ILogger<PostOfficeLocationService> logger,
 NovaPoshtaCitiesRepository npCitiesRepository,
 IMapper mapper,
 NovaPoshtaHttpClient novaPoshtaClient)
 : IPostOfficeLocationService
 {
 private readonly ILogger<PostOfficeLocationService> _logger = logger;  private readonly
NovaPoshtaHttpClient _novaPoshtaClient = novaPoshtaClient;  private readonly
NovaPoshtaCitiesRepository _npCitiesRepository = npCitiesRepository;  private readonly
IMapper _mapper = mapper;

 public async Task<IEnumerable<PostOffice>> GetPostOfficesByCityNameAsync(string
cityName)
 {
 var npCity = _npCitiesRepository.GetNpCityByNameAsync(cityName);

 var npPostOffices = await _novaPoshtaClient.GetAllWarehousesByCityRefAsync(npCity.Ref);

 var postOffices = _mapper.Map<List<PostOffice>>(npPostOffices);

 return postOffices;
 }
}
public class NovaPoshtaHttpClient
 {
```

```
private readonly HttpClient _httpClient;
private readonly string _apiKey;
public NovaPoshtaHttpClient(HttpClient httpClient, string apiKey)
{
_httpClient = httpClient;
_apiKey = apiKey;
}

public async Task<List<NpArea>> GetAllAreasAsync()
{
var result = await SendRequestAsync<NpResponse<List<NpArea>>>("Address", "getAreas");
return result.Data;
}

#region Cities

public async Task<List<NpCity>> GetAllCitiesAsync()
```



```
{
var cities = new List<NpCity>();
int page = 1;
bool hasNextPage;

do
{
var methodProperties = new
{
Page = page.ToString(),
Limit = "100" // Максимум 100 записів на сторінку
};

var result = await SendRequestAsync<NpResponse<List<NpCity>>>("AddressGeneral",
"getCities", methodProperties);
cities.AddRange(result.Data);

hasNextPage = result.Data.Count == 100;
page++;
} while (hasNextPage);

return cities;
}

public async Task<NpCity> GetCityByNameAsync(string findByString = "")
{
var methodProperties = new
{
FindByString = findByString,
};

var result = await SendRequestAsync<NpResponse<List<NpCity>>>("AddressGeneral",
"getCities", methodProperties);
return result.Data.First();
}
```

```
public async Task<NpCity> GetCityByRefAsync(string findByRef = "")
{
var methodProperties = new
{
Ref = findByRef,
};

var result = await SendRequestAsync<NpResponse<List<NpCity>>>("AddressGeneral",
"getCities", methodProperties);
return result.Data.First();
}

#endregion Cities

#region Warehouses
```



```
public async Task<List<NpPostOffice>> GetAllWarehousesByCityRefAsync(string cityRef)
{
var warehouses = new List<NpPostOffice>();
int page = 1;
bool hasNextPage;

do
{
var methodProperties = new
{
CityRef = cityRef,
Page = page.ToString(),
Limit = "500",
Language = "UA"
};

var result = await SendRequestAsync<NpResponse<List<NpPostOffice>>>("AddressGeneral",
"getWarehouses", methodProperties);

if (result?.Data == null)
break;

warehouses.AddRange(result.Data);

hasNextPage = result.Data.Count == 500;
page++;

} while (hasNextPage);

return warehouses;
}

#endregion Warehouses

private async Task<TResponse> SendRequestAsync<TResponse>(string modelName, string
calledMethod, object? methodProperties = null)
{
```

```
var requestPayload = new
{
apiKey = _apiKey,
modelName,
calledMethod,
methodProperties = methodProperties ?? new { }
};

var json = JsonSerializer.Serialize(requestPayload);
var content = new StringContent(json, Encoding.UTF8, "application/json");

var response = await _httpClient.PostAsync("", content);
response.EnsureSuccessStatusCode();
```



```
var responseContent = await response.Content.ReadAsStringAsync();

return JsonSerializer.Deserialize<TResponse>(responseContent, new JsonSerializerOptions
{
PropertyNameCaseInsensitive = true
}) ?? throw new InvalidOperationException("Response content is null.");
}
}
```

КРИВОРІЗЬКИЙ ФАХОВИЙ КОЛЕДЖ
ДЕРЖАВНОГО НЕКОМЕРЦІЙНОГО ПІДПРИЄМСТВА
«ДЕРЖАВНИЙ УНІВЕРСИТЕТ «КИЇВСЬКИЙ АВІАЦІЙНИЙ ІНСТИТУТ»

РЕЦЕНЗІЯ
на кваліфікаційну роботу

випускника спеціальності: 123 «Комп'ютерна інженерія»
відділення: комп'ютерної та програмної інженерії
циклова комісія: комп'ютерних систем та мереж
Максим ГОРБАРУК
(ім'я, прізвище)

1. Обрана тема кваліфікаційної роботи, «Програмування логістичної системи для оптимізації поштової доставки», є актуальною.

2. Кваліфікаційна робота відповідає темі, затвердженій наказом.

3. Завдання на виконання кваліфікаційної роботи виконано у повному обсязі у встановлений термін.

4. В результаті виконання кваліфікаційної роботи було створено модуль програмного забезпечення, який обчислює оптимальні маршрути і може бути інтегрованим в транспортні компанії для покращення їх логістики.

5. Якість виконання пояснювальної записки та ілюстративного (графічного) матеріалу відповідає вимогам державних стандартів.

6. В кваліфікаційній роботі зроблений акцент на створенні модульної та масштабованої основи, яку можна адаптувати до потреб різних логістичних компаній.

7. Кваліфікаційна робота заслуговує оцінку «відмінно», а випускник присвоєння кваліфікації фахівця освітньо-професійного ступеня «Фаховий молодший бакалавр» спеціальності 123 «Комп'ютерна інженерія».

Рецензент _____
(науковий ступінь, посада)

«____»_____ 2025 р. _____ Марія КИСЛОВА
(підпис) (ім'я, прізвище)

З рецензією ознайомлений _____ Максим ГОРБАРУК
(підпис) (ім'я, прізвище)