ХЕРСОНСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
(повне найменування вищого навчального закладу)
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА ДИЗАЙНУ
(повне найменування інституту, назва факультету (відділення))
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ І ТЕХНОЛОГІЙ
(повна назва кафедри (предметної, циклової комісії))

Пояснювальна записка

до кваліфікаційної роботи
магістра

на тему:

«Дослідження методів створення ефективних маршрутів у web-сервісах для виконання завдань»

Виконав: здобувач 6 курсу, групи 6ПР2
спеціальності
121 – «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

Івелов Владислав Максимович
(прізвище та ініціали)
Керівник к.т.н, Величко Ю. І.
(прізвище та ініціали)
Рецензент к.т.н. Григорова А.А.
(прізвище та ініціали)

Хмельницький – 2025

Херсонський національний технічний університет
(повне найменування вищого навчального закладу)

Інститут, факультет, відділення Інформаційних технологій та дизайну
Кафедра, циклова комісія Програмних засобів і технологій
Освітньо-кваліфікаційний рівень магістр
Напрям підготовки ОПП - Програмне забезпечення систем
(шифр і назва)
Спеціальність 121 - «Інженерія програмного забезпечення»
(шифр і назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри Програмних засобів і технологій
д.т.н. доц. О.Є. Огнєва
«____»_____2025 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТУ

Івелов Владислав Максимович
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) Дослідження методів створення ефективних маршрутів у web-сервісах для виконання завдань

керівник проекту (роботи) к.т.н, Величко Ю. І.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від «___»___2025 року №___
2. Строк подання здобувачем роботи 16.12.2025
3. Вихідні дані до роботи ДСТУ з обробки інформації, літературні та періодичні джерела
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної галузі задачі.
2. Системний аналіз і обгрунтування проблеми.
3. Методи та засоби вирішення проблеми.
4. Практична реалізація.
5. Висновки.
6. Список використаної літератури.
7. Додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
1. Існуючі реалізації; 2. Архітектура системи; 3. Структура БД; 4. Екранні форми

6. Консультанти розділів роботи

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

7. Дата видачі завдання 12.09.2025

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів виконання роботи	Термін виконання етапів роботи	Примітки
1.	Отримання завдання	12.09.2025	Виконано
2.	Підбір літератури	20.09.2025	Виконано
3.	Аналіз предметної області	10.10.2025	Виконано
4.	Розробка та обґрунтування завдання	22.10.2025	Виконано
5.	Вибір алгоритму	06.11.2025	Виконано
6.	Розробка алгоритму	13.11.2025	Виконано
7.	Розробка додатку	20.11.2025	Виконано
8.	Тестування алгоритму	02.12.2025	Виконано
9.	Оформлення пояснювальної записки	06.12.2025	Виконано
10.	Захист кваліфікаційної роботи	16.12.2025	Виконано

Студент _____ Івелов В.М.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Величко Ю. І.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка: 81 сторінка, 31 рисунок, 3 таблиці, 11 джерел.

Мета роботи – алгоритм створення ефективних маршрутів у веб-сервісах для виконання завдань.

Об’єкт дослідження – процес формування та оптимізації маршрутів виконання завдань.

Предмет дослідження – методи та алгоритми оптимізації маршрутів виконання мікрозавдань у веб-сервісах, які дозволяють підвищити ефективність вибору послідовності завдань виконавцем.

Методи дослідження – аналіз і порівняння існуючих алгоритмів, модифікація та використання алгоритму для створення веб-додатку.

Результат роботи – розроблено алгоритм та веб-додаток на його основі, що дозволяє знаходити оптимальні шляхи для виконання задач.

Ключові слова: алгоритм, маршрут, шлях, оптимізація, сайт, NestJs, NextJs, задачі.

АНОТАЦІЯ

Перший розділ «Аналіз предметної області задачі» містить формулювання мети та задач магістерського проекту. Проведено аналіз існуючих аналогів, їх функціональних можливостей, переваг та недоліків. Сформульовано постановку задачі та обґрунтовано вибір способів її реалізації.

Другий розділ «Системний аналіз і обґрунтування проблеми» присвячений аналізу та постановці проблеми. В розділі проаналізовано об’єкт дослідження; класифіковано, поставлено та обґрунтовано проблему; наведено вхідні та вихідні дані; а також вимоги до програмного продукту.

Третій розділ - «Методи та засоби вирішення проблеми». В розділі проведено експериментальний аналіз ефективності алгоритмів та обрано найкращий для подальшого використання.

Четвертий розділ «Практична реалізація» містить безпосередній опис створення веб-додатку з використанням розробленого алгоритму. Розглянуто реалізацію основних алгоритмів, структуру проекту. Показано логіку роботи окремих компонентів та наведено екранні форми.

ANNOTATION

The first section "Analysis of the subject area of the problem" contains the formulation of the goal and objectives of the master's project. An analysis of existing analogues, their functionality, advantages and disadvantages is carried out. The statement of the problem is formulated and the choice of methods for its implementation is justified.

The second section "System analysis and justification of the problem" is devoted to the analysis and formulation of the problem. The section analyzes the object of research; classifies, poses and justifies the problem; provides input and output data; as well as requirements for the software product.

The third section - "Methods and means of solving the problem". The section conducts an experimental analysis of the effectiveness of algorithms and selects the best one for further use.

The fourth section "Practical implementation" contains a direct description of creating a web application using the developed algorithm. The implementation of the main algorithms, the structure of the project are considered. The logic of the operation of individual components is shown and screen forms are provided.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

АПІ (API) - програмний інтерфейс програми

БД - база даних

СУБД - система управління базами даних

DOM - Об'єктна модель документа

SQL - структурована мова запитів

ORM - Об'єктно-реляційна проекція

ВСТУП

У сучасному світі інформаційних технологій управління завданнями та проектами стало невід’ємною складовою ефективної роботи як в особистій, так і в командній діяльності. Актуальність теми курсового проекту полягає в постійному зростанні потреби в цифрових інструментах для організації роботи, відстеження прогресу, делегування обов’язків та контролю за виконанням задач. Розробка алгоритму дозволяє використовувати його для оптимізації маршрутів у реальних веб-сервісах мікрозавдань для підвищення ефективності роботи виконавців і зменшення часу виконання замовлень.

Метою проекту є розробка алгоритму створення ефективних маршрутів у веб-сервісах для виконання завдань. Цей застосунок має забезпечити простий та інтуїтивно зрозумілий інтерфейс для користувачів, можливість фільтрації, сортування задач, а також гнучке управління правами доступу залежно від ролі користувача.

Програмний продукт дозволяє:

- Вводити початкові дані про задачі (координати, час, винагорода);
- Задавати часові обмеження користувача;
- Використовувати один із алгоритмів оптимізації (Greedy, Genetic Algorithm, Ant Colony Optimization);
- Отримувати оптимальний або близький до оптимального маршрут;
- Аналізувати отримані результати.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ЗАДАЧІ

В умовах стрімкого розвитку цифрових технологій та зростання популярності моделей економіки спільного користування - за останні 10 років кількість затраченого часу в соціальних мережах зросла на 60% [1], дедалі більшої ваги набувають сервіси, що дозволяють користувачам обмінюватися послугами чи допомогою на локальному рівні. Одним із перспективних напрямів таких систем є веб-сервіси для виконання мікрозавдань, де користувачі можуть створювати прості завдання, а інші - виконувати їх за винагороду.

Цю тематику в останні роки досліджували Necula R., Konovalenko A., Mardešić N., Larsen, A. та інші, про що ще буде згадано нижче.

1.1 Актуальність теми

Актуальність дослідження зумовлена потребою в підвищенні ефективності логістичних процесів у дрібномасштабних міських середовищах, де щодня виникають десятки короткотривалих завдань, розподілених у просторі. Оптимізація маршрутів у таких умовах може дати не лише економічний ефект, а й соціальний - сприяти залученню молоді до активної діяльності, допомозі літнім людям, розвитку місцевих спільнот.

З наукового погляду, дослідження орієнтоване на модифікацію та адаптацію класичних задач комбінаторної оптимізації (зокрема, задачі комівояжера та задачі маршрутизації транспортних засобів - VRP) до специфіки мікрозавдань у міському середовищі. Існуючі алгоритми оптимізації часто не враховують специфіку коротких відстаней, невеликої тривалості виконання, мінливості локацій та потреб у реальному часі. Тому розроблення адаптивних, евристичних або гібридних методів є актуальним науковим завданням.

Для України тема має особливу практичну цінність. З одного боку, багато населених пунктів стикаються з проблемою маломобільності літніх людей та недостатньої кількості дрібних сервісів побутової допомоги. З іншого - молодь потребує доступних способів підробітку та соціальної активності. Розвиток вітчизняних систем мікрозавдань, оптимізованих під українські реалії (щільність забудови, рівень цифровізації, ментальні особливості користувачів), сприятиме підвищенню цифрової інклюзії, розвитку локальних ініціатив і малого підприємництва. [2]

Таким чином, актуальність теми дослідження обумовлена:

- наявністю незаповненої наукової ніші у сфері маршрутної оптимізації для систем мікрозавдань;
- потребою у подальшому розвитку методів адаптивної оптимізації в умовах динамічних даних;
- необхідністю узагальнення та вдосконалення наявного досвіду побудови веб-платформ типу Kabanchik чи TaskRabbit;
- суспільною значущістю задачі для підвищення рівня соціальної взаємодії, залучення молоді та підтримки маломобільних груп населення;
- практичною доцільністю розробки українського рішення, орієнтованого на місцевий контекст і цифрову трансформацію громад.

Отже, дослідження методів створення ефективних маршрутів у веб-сервісах для виконання завдань є актуальним як у науковому, так і в соціально-економічному аспекті, оскільки сприяє розвитку сучасних IT-підходів до оптимізації, підвищенню ефективності цифрових сервісів та формуванню нових моделей міської взаємодії в Україні.

1.2 Мета і задачі дослідження

Метою дослідження є алгоритм створення ефективних маршрутів у веб-сервісах для виконання завдань.

Для досягнення поставленої мети необхідно виконати ряд задач:

- Проаналізувати предметну область та проблему;
- Охарактеризувати існуючі альтернативи;
- Обрати/створити декілька підходящих алоритмів;
- Обрати найкращий алгоритм за допомогою теорії та експериментів;
- Створити прототип веб-додатку на базі обранного алгоритму.

1.3 Об’єкт і предмет дослідження

Об’єктом дослідження є процес формування та оптимізації маршрутів виконання завдань.

Предметом дослідження є методи та алгоритми оптимізації маршрутів виконання мікрозавдань у веб-сервісах, які дозволяють підвищити ефективність вибору послідовності завдань виконавцем.

Зокрема, досліджуються способи формалізації задачі оптимізації, вибору критерію ефективності (час, відстань, винагорода) та застосування сучасних алгоритмів комбінаторної оптимізації (евристичних, стохастичних або гібридних) для динамічних середовищ.

1.4 Наукова новизна

Наукова новизна одержаних результатів полягає у розробленні та теоретичному обґрунтуванні підходу до оптимізації маршрутів у веб-сервісах мікрозавдань, який враховує динамічний характер завдань і комбіновані критерії ефективності (час, відстань, винагорода).

У роботі:

- Вперше здійснено комплексне дослідження застосування методів комбінаторної оптимізації до систем локальних мікрозадач у міському середовищі;
- Формалізовано математичну модель маршрутизації виконавців з урахуванням пріоритетності завдань і часових обмежень;
- Запропоновано адаптивний алгоритм підбору завдань на основі модифікації задачі комівояжера, який забезпечує покращення показників ефективності порівняно з традиційними методами;
- Визначено залежності між характеристиками завдань (час, відстань, винагорода) та якістю побудованого маршруту.

Отримані результати поглиблюють наукові уявлення про застосування методів оптимізації в динамічних багатокритеріальних середовищах і можуть бути використані при розробці сучасних інтелектуальних систем планування маршрутів.

1.5 Практичне і теоретичне значення результатів

Практичне значення полягає у можливості використання запропонованого алгоритму оптимізації маршрутів у реальних веб-сервісах мікрозавдань для підвищення ефективності роботи виконавців і зменшення часу виконання замовлень.

Розроблений підхід може бути інтегрований у системи типу Kabanchik.ua або аналогічні локальні сервіси, а також у платформи громадської допомоги, муніципальні або волонтерські проекти.

Впровадження таких алгоритмів сприятиме розвитку українських IT-рішень, оптимізації логістики у міських середовищах та залученню молоді до соціально корисної діяльності.

Теоретичне значення роботи полягає у розвитку підходів до формалізації та розв’язання задач комбінаторної оптимізації в контексті систем динамічного призначення мікрозавдань.

Результати дослідження доповнюють наукову базу знань у галузі інтелектуальних інформаційних систем, оптимізаційних методів і геоінформаційних технологій.

Отримані теоретичні положення можуть бути використані як основа для подальших досліджень у сфері адаптивних систем маршрутизації, а також при викладанні дисциплін з оптимізації, штучного інтелекту та розробки веб-сервісів.

1.6 Аналіз процесу формування маршрутів у веб-сервісах мікрозавдань

У сучасних веб-сервісах, орієнтованих на виконання мікрозавдань, ключовим компонентом ефективності є алгоритм формування маршрутів виконавців. Такі системи поєднують риси класичної задачі комівояжера (TSP) і задачі маршрутизації транспортних засобів (VRP), але з урахуванням особливостей динамічного середовища, де завдання постійно з’являються, змінюються або скасовуються в режимі реального часу.

Об’єктом моделювання в даному контексті є динамічне маршрутування виконавців, яке передбачає оптимальний вибір послідовності завдань у просторі та часі з метою мінімізації витрат або максимізації винагороди. У класичному VRP усі завдання відомі наперед, однак у реальних умовах сервісів мікрозавдань це припущення не виконується. Саме тому традиційні методи втрачають ефективність, що обґрунтовує потребу у спеціальних алгоритмах для динамічної задачі маршрутизації. [3]

Як зазначає Allan Larsen, у DVRP з’являється додатковий вимір - часова еволюція даних, тобто надходження нової інформації після початку побудови маршруту. Це робить задачу не лише складнішою обчислювально, а й більш

прикладною для сучасних веб-систем, де виконавці рухаються в реальному просторі, а запити створюються користувачами постійно.

Подальші дослідження підтверджують, що для таких сценаріїв ефективними є реактивні та адаптивні підходи, здатні оновлювати маршрут у міру зміни середовища. Огляд сучасних методів у сфері стохастичного DVRP, проведений Mardešić та іншими [4], показує, що актуальні підходи ґрунтуються на поєднанні евристичних алгоритмів із методами машинного навчання. Це дозволяє не лише оптимізувати маршрут, а й прогнозувати появу нових завдань у певних локаціях.

Особливої уваги заслуговує підхід на основі глибокого підсиленого навчання (Deep Reinforcement Learning, DRL), який запропонували Konovalenko та Hvattum 2024. [5]

Їхній алгоритм демонструє здатність адаптувати рішення в реальному часі, з урахуванням появи нових точок завдань і часових обмежень. Для веб-сервісів мікрозавдань цей підхід є надзвичайно перспективним, оскільки дозволяє динамічно переобчислювати оптимальні маршрути без повного перезапуску моделі.

Інший напрям досліджень - евристичні та метаевристичні методи, такі як мурашині алгоритми (Ant Colony Optimization) [6] чи гібридні двоетапні схеми оптимізації [7]. Такі методи особливо ефективні в умовах обмежених ресурсів обчислення та неповної інформації - типової ситуації для мобільних виконавців. Вони дають змогу швидко отримати близьке до оптимального рішення, що має практичну цінність для задач планування маршрутів у міських умовах.

Дослідження Andersen [8] акцентує на тому, що в умовах просторової залежності (тобто коли нові завдання з’являються в радіусі дії виконавця) ефективнішими є переоптимізуючі моделі, які періодично перераховують оптимальний маршрут на основі оновлених даних. Це безпосередньо корелює з

поведінкою користувачів у мікрозадачних сервісах, де завдання мають геоприв’язку й короткий життєвий цикл.

Horváth і Tamási [9] запропонували загальну симуляційну платформу для моделювання процесу динамічного маршрутування, яка може бути використана як основа для тестування оптимізаційних стратегій у подібних веб-сервісах.

Узагальнюючи, можна зазначити, що аналіз процесу формування маршрутів у веб-сервісах мікрозавдань демонструє перехід від статичних моделей до динамічно-адаптивних алгоритмів, здатних реагувати на зміни в режимі реального часу. Це відкриває перспективи для створення інтелектуальних систем, які підвищують ефективність виконання завдань і рівень задоволеності користувачів - як замовників, так і виконавців.

1.7 Особливості формування маршрутів у системах мікрозавдань

Формування маршрутів у системах мікрозавдань має низку специфічних особливостей, що відрізняють його від класичних задач логістики та транспортного маршрутування. Головною відмінністю є динамічний і децентралізований характер середовища, у якому маршрути не задаються наперед, а формуються в процесі взаємодії користувачів - виконавців і замовників.

На відміну від традиційних логістичних систем, де замовлення надходять централізовано і плануються заздалегідь, в даному випадку нові завдання можуть з’являтися будь-коли, мати різну пріоритетність, тривалість виконання, складність і винагороду. Це робить задачу не лише динамічною, але й стохастичною, адже неможливо точно передбачити ні кількість завдань, ні їхнє просторове розташування в конкретний момент часу.

Особливістю є також короткий життєвий цикл завдань. Наприклад, якщо літня людина створює задачу «принести воду» або «прибрати у дворі», вона має бути виконана протягом короткого проміжку часу, після чого стає

неактуальною. Це накладає обмеження на час реакції системи, тому алгоритм маршрутизації має забезпечувати реальновчасну переоптимізацію маршруту при появі нових даних.

У таких умовах важливими параметрами оптимізації є не лише мінімізація відстані чи часу пересування, а й максимізація винагороди за одиницю часу. На відміну від логістики, де мета - мінімізувати витрати, у системах мікрозавдань виконавець прагне максимізувати прибуток, враховуючи баланс між відстанню, кількістю завдань та винагородою за кожне з них. Таким чином, задача набуває багатокритеріального характеру, де критерії можуть включати час, відстань, енергозатрати, рейтинг виконавця або навіть соціальну користь завдання.

Ще однією особливістю є людський фактор - маршрути формуються не для автономних транспортних засобів, а для людей. Це означає, що алгоритм має враховувати поведінкові особливості виконавців: переваги щодо типів завдань, звичні маршрути пересування, рівень фізичної підготовки або навіть погодні умови. Як зазначено у дослідженні Andersen [8], моделі, які не враховують локальні особливості або «проксімальні залежності» (тобто ймовірність появи завдань поблизу виконавця), втрачають значну частину ефективності.

Додатковою проблемою є некерованість появи завдань у просторі, оскільки користувачі створюють їх спонтанно. Тому система повинна бути здатною до онлайн-оновлення карти завдань і адаптивного формування маршрутів із використанням методів машинного навчання. Для таких випадків ефективно зарекомендували себе алгоритми на основі підсиленого навчання, які дозволяють навчати модель приймати рішення в реальному часі, обираючи наступне завдання, що принесе найбільший сукупний виграш.

У практичних реалізаціях веб-сервісів мікрозавдань маршрути часто не будуються централізовано - кожен виконавець приймає рішення самостійно. Тому доцільно застосовувати децентралізовані алгоритми, зокрема мурашині

колонії або генетичні алгоритми, які дозволяють одночасно обробляти кілька локальних рішень та поступово покращувати глобальний результат. Такі підходи також знижують навантаження на центральний сервер, що критично для мобільних застосунків з великою кількістю користувачів.

Важливим є й соціальний аспект: система мікрозавдань спрямована не лише на ефективність, а й на соціальну взаємодію та розвиток локальних спільнот. Формування маршруту тут може враховувати не лише географічні дані, а й рейтинг довіри, історію виконанн або навіть рекомендації користувачів.

Таким чином, формування маршрутів у веб-сервісах мікрозавдань характеризується:

- Високим ступенем динамічності середовища;
- Необхідністю реальновчасного оновлення даних;
- Багатокритеріальністю (час, винагорода, зручність, рейтинг);
- Урахуванням людських та поведінкових факторів;
- Децентралізованістю прийняття рішень;
- Просторовою стохастичністю розподілу завдань.

Ці особливості формують новий підклас задач маршрутизації, який можна визначити як DVRP-LH (Dynamic Vehicle Routing Problem with Local Human Factors) - динамічну задачу маршрутизації з урахуванням локальних людських обмежень. Її вивчення є основою для побудови інтелектуальних веб-сервісів нового покоління, здатних ефективно поєднувати технологічну оптимізацію та соціальну корисність.

1.8 Існуючі моделі та методи оптимізації маршрутів у динамічних системах

1.8.1. Класифікація моделей маршрутизації

Перш ніж розглядати конкретні методи, корисно подати класифікацію моделей:

- Статичні моделі - усі завдання відомі наперед, маршрути прокладаються «одним махом». (Класичний VRP, TSP, VRP з часовими вікнами);
- Динамічні моделі - нові завдання надходять під час виконання маршруту, або умови змінюються (DVRP);
- Стохастичні / ймовірнісні моделі - враховують невизначеність у надходженні завдань або змінність параметрів;
- Гібридні / адаптивні моделі - поєднують реактивну оптимізацію з прогнозуванням або алгоритмами навчання (наприклад, машинне навчання, reinforcement learning).

1.8.2 Точні й комбінаційні методи

Точні алгоритми забезпечують ідеальне рішення, але їх складність росте експоненційно, тому їх застосовність обмежена малими масштабами:

- Методи галуження й меж (branch-and-bound / branch-and-cut / branch-and-price) застосовуються у VRP/VRPTW і можуть бути використані в DVRP (особливо для невеликих підзадач) для частин маршруту, коли потік нових задач не надто інтенсивний;
- Динамічне програмування, особливо з обмеженим простором станів, може використовуватися для малих інстанцій. Наприклад, у роботі «Deep Policy Dynamic Programming» запропоновано комбінований підхід, де нейронна мережа керує скороченням простору станів у динамічному програмуванні для задач VRP / VRP з часовими вікнами. [10]

Такий підхід може бути застосований у системах, де кількість завдань одночасно невелика, або як підзадача для локальної переоптимізації маршруту.

1.8.3 Евристики та метаевристики

Для практичних задач великого масштабу зазвичай використовують евристичні та метаевристичні підходи, які дають добрі наближені рішення:

- Мурашині алгоритми - добре підходять для динамічних задач, оскільки «феромонні шляхи» можуть оновлюватись у режимі реального часу при зміні даних. Наприклад, у задачі DVRP з часовими вікнами застосований ACO із механізмами вставки й локального пошуку для динамічного оновлення маршруту;
- Генетичні алгоритми, еволюційні стратегії - змінюють послідовності завдань, комбінують маршрути, додають варіанти «скремблів», адаптують під нові дані.
- Розроблені гібридні алгоритми, наприклад, двоетапні - спочатку конструктивний алгоритм формує початковий маршрут, потім алгоритм покращення перебудовує його з урахуванням динаміки;
- Гіперевристики, які вибирають найкращий метод оптимізації на льоту (наприклад, Particle Swarm Optimization Hyper-heuristic) в залежності від поточного стану задачі. [11]

Сфери застосування цих методів: служби доставки (last-mile delivery), кур’єрські мережі, логістика «останньої милі», сервіси таксі з попутним маршрутом, платформи спільного перевезення.

1.8.4 Алгоритми машинного навчання / підсилене навчання

З розвитком обчислювальних ресурсів і доступністю великих даних, стали популярними підходи на базі ML:

- Reinforcement Learning (RL) - агент навчається вибирати наступне завдання/маршрут із урахуванням винагороди, змін у середовищі, штрафів. Наприклад, Konovalenko і Hvattum використовували DRL для адаптивного вибору маршруту в динамічному середовищі;

- Гібридні моделі «ML + оптимізація», коли ML використовується для оцінки потенційної вигоди чи пріоритету завдань, після чого класичний алгоритм будує маршрут;
- Нейронні політики, комбіновані з динамічним програмуванням або локальним пошуком - як у Deep Policy Dynamic Programming, де політика мережі спрямовує DP або обмежує пошук простору станів.

Ці алгоритми вже почали застосовувати в сучасних системах доставки, смарт-містах, автономних логістичних рішеннях.

Висновки до розділу

У цьому розділі було виконано аналіз предметної області проекту, визначено мету та задачі дослідження, а також проведено огляд теоретичних джерел й існуючих алгоритмів для пошуку шляху.

РОЗДІЛ 2. СИСТЕМНИЙ АНАЛІЗ І ОБҐРУНТУВАННЯ ПРОБЛЕМИ

2.1 Системний аналіз об’єкта дослідження і предметної області

У рамках даної магістерської кваліфікаційної роботи об’єктом дослідження є процес планування та оптимізації маршруту виконання множини просторово розподілених завдань з урахуванням часових обмежень та величини винагороди за їх виконання.

Предметом дослідження є методи, алгоритми та інформаційні технології, що використовуються для формування оптимального або наближеного до оптимального маршруту виконання завдань за заданий інтервал часу з метою максимізації сумарної винагороди.

Необхідність проведення системного аналізу обумовлена складністю досліджуваної проблеми, яка характеризується наявністю великої кількості взаємопов’язаних факторів: просторове розташування точок, тривалість виконання кожного завдання, часові обмеження користувача, величина винагороди та характер маршруту, який необхідно побудувати. У таких умовах виникають труднощі з однозначним формулюванням проблеми, вибором адекватних засобів розв’язання та оцінкою ефективності отриманих результатів без використання системного підходу.

2.2 Класифікація проблеми

Досліджувана проблема належить до класу складних оптимізаційних задач, оскільки:

- Має багато вхідних параметрів і обмежень;
- Характеризується високим рівнем невизначеності (зміна умов, випадковий характер початкових даних);
- Не має однозначного аналітичного розв’язання;

- Належить до класу NP-складних задач, подібних до задачі комівояжера (Travelling Salesman Problem) і задачі рюкзака (Knapsack Problem).

Для формалізації цієї проблеми та її подальшого вирішення доцільно використовувати ієрархічну модель системи, в якій можна виділити три основні рівні:

Верхній рівень (рівень управління) - користувач, що задає параметри задачі (часовий інтервал, тип алгоритму, початкова точка, обмеження).

Середній рівень (рівень обробки) - алгоритмічний модуль, який реалізує вибраний метод оптимізації (Greedy, Genetic Algorithm, Ant Colony Optimization) та формує оптимальний маршрут.

Нижній рівень (рівень даних) - набір задач (точок) з координатами, часом виконання та величиною винагороди, між якими формується маршрут.

В даній ієрархічній структурі прямий зв’язок реалізується у вигляді керуючих впливів (вибір параметрів алгоритму, запуск обчислень), а зворотний зв’язок - у вигляді отриманих результатів (маршрут, сумарна винагорода, загальна відстань, час роботи алгоритму). Такий підхід дозволяє усунути нечіткість у трактуванні цілей завдання і відокремити засоби досягнення результату від самих цілей.

Одним з ключових методів отримання нових знань у межах даного дослідження є системне моделювання, яке полягає у побудові формальної моделі реального процесу планування маршруту. При цьому кожна задача розглядається як елемент системи, що має власні характеристики (місце, час виконання, винагорода), а маршрут виступає як впорядкована послідовність цих елементів, з’єднаних транспортними переходами.

У роботі застосовується декілька типів моделей:

- Математична модель, яка описує задачу у вигляді оптимізаційної функції (максимізація винагороди при часовому обмеженні);

- Інформаційна модель, що визначає структуру вхідних та вихідних даних, а також їх зв’язки;
- Алгоритмічна модель, в якій реалізується логіка роботи методів оптимізації.

Для реалізації системного аналізу та проєктування структури програмного продукту доцільно використати парадигму «методологія – метод – нотація – засіб».

Таким чином, системний аналіз дає змогу чітко структурувати досліджуваний об’єкт, усунути неоднозначності, виявити ключові змінні та взаємозв’язки між ними і створити передумови для коректної постановки задачі й ефективного вибору методів її розв’язання.

В результаті системного аналізу визначено, що задача оптимізації маршруту виконання просторово розподілених завдань з урахуванням часових обмежень та винагороди належить до складних багатокритеріальних задач, розв’язання яких доцільно здійснювати з використанням евристичних та метаевристичних алгоритмів. Отримані результати визначають подальший напрям дослідження та формують основу для формулювання і обґрунтування основної задачі даної магістерської роботи.

2.2 Постановка й обґрунтування проблеми

У сучасних умовах розвитку цифрових технологій усе більшого поширення набувають задачі, пов’язані з оптимальним плануванням діяльності в умовах обмеженого часу та ресурсів. Особливо актуальними є задачі, у яких необхідно сформувати ефективний маршрут відвідування ряду просторово розподілених об’єктів для виконання певних дій, отримання послуг або виконання завдань, з метою максимізації загальної винагороди або ефективності.

У рамках даної магістерської кваліфікаційної роботи під проблемою дослідження розуміється відсутність універсального ефективного алгоритмічного підходу до формування оптимального маршруту виконання множини задач, кожна з яких характеризується:

- Місцем розташування;
- Часом, необхідним для її виконання;
- Величиною винагороди;
- Обмеженнями загального часу, доступного користувачеві.

Таким чином, формується багатокритеріальна оптимізаційна задача, яка поєднує властивості задачі комівояжера (TSP) та задачі рюкзака (Knapsack Problem). Необхідно не лише знайти оптимальну послідовність проходження точок, але й обрати підмножину завдань таким чином, щоб сумарний час їх виконання разом з часом на переміщення не перевищував заданої межі, а загальна винагорода була максимальною.

Розроблений програмний продукт призначений для підтримки прийняття рішень у процесі планування маршруту виконання множини задач із урахуванням часових обмежень та величини винагороди.

Програмний продукт дозволяє:

- Вводити початкові дані про задачі (координати, час, винагорода);
- Задавати часові обмеження користувача;
- Використовувати один із алгоритмів оптимізації (Greedy, Genetic Algorithm, Ant Colony Optimization);
- Отримувати оптимальний або близький до оптимального маршрут;
- Аналізувати отримані результати.

На сьогодні існують різноманітні програмні комплекси навігації та планування маршрутів (Google Maps, Waze, спеціалізовані логістичні платформи), однак більшість з них:

- Не враховує показник винагороди за виконання задачі;

- Не здійснює автоматичний відбір підмножини завдань у межах заданого часу;
- Орієнтована переважно на скорочення відстані або часу, а не на максимізацію вигоди.

Тому створення спеціалізованого алгоритмічного програмного засобу, що поєднує в собі елементи кількох задач оптимізації та забезпечує гнучке налаштування під конкретні умови користувача, є обґрунтованим і доцільним.

2.3 Вхідні та вихідні дані

Вхідними даними для програмного продукту є:

- Координати задач (широта, довгота або x та y);
- Час, необхідний для виконання кожної задачі;
- Величина винагороди за виконання задачі;
- Початкова точка маршруту;
- Максимально доступний час користувача.

Вихідними даними програмного продукту є:

- Список задач, обраних до виконання;
- Порядок їх відвідування;
- Візуалізація маршруту;
- Сумарна винагорода;
- Загальна довжина маршруту.

2.4 Вимоги до програмного продукту

До розробленого програмного продукту висуваються такі вимоги:

Функціональні:

- Коректне формування маршруту;
- Реєстрація та авторизація;

- Додавання задач;
- Коректний підрахунок винагороди та часу.

Технічні:

- Реалізація у вигляді браузерного застосунку на React;
- Сумісність із сучасними браузерами;
- Низькі обчислювальні витрати.

Ергономічні:

- Інтуїтивно зрозумілий інтерфейс;
- Візуалізація маршруту та результатів.

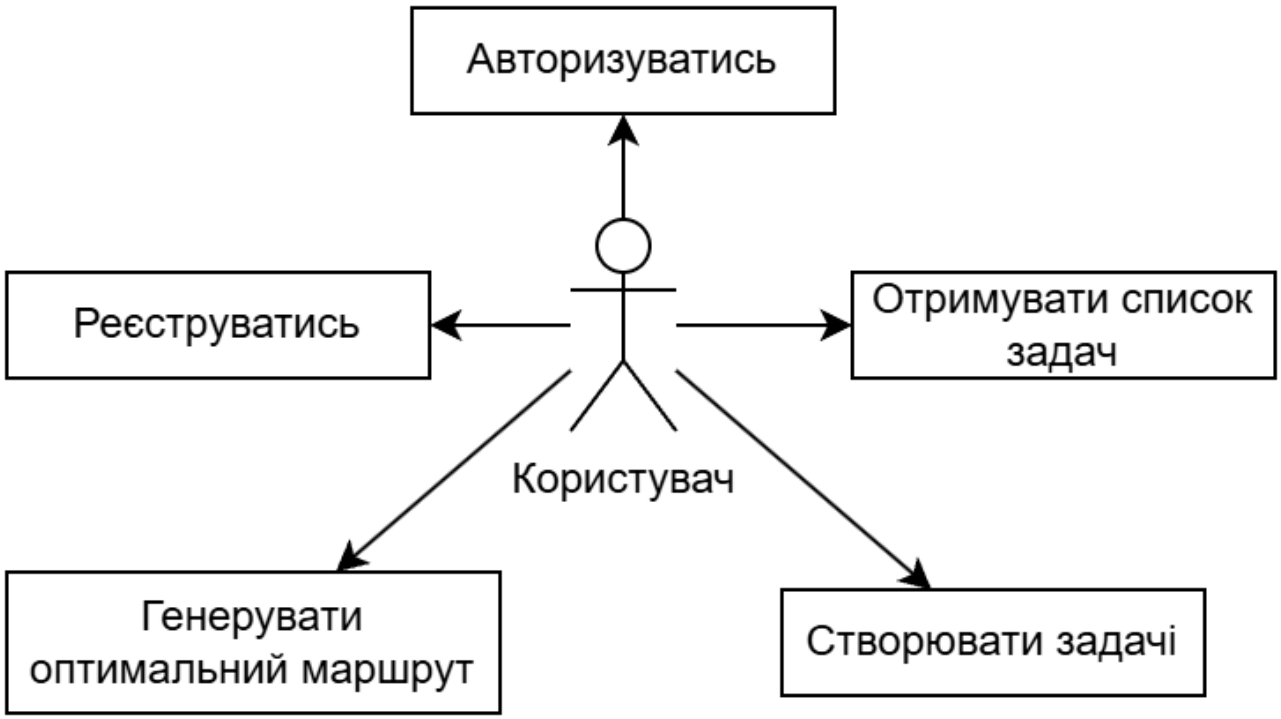

Рисунок 2.1 – Можливості користувача

Висновки до розділу

У цьому розділі було проаналізовано об’єкт дослідження та предметну область, класифіковано та обгрунтовано проблему, перелічено вхідні та вихідні дані а також вимогу до продукту.

РОЗДІЛ 3. МЕТОДИ ТА ЗАСОБИ ВИРІШЕННЯ ПРОБЛЕМИ

3.1 Технології та засоби побудови баз даних

У процесі розроблення програмного продукту для оптимізації маршруту та максимізації винагороди за обмежений проміжок часу виникає необхідність у надійному, масштабованому та високопродуктивному середовищі збереження й обробки даних. У якості системи управління базами даних у проєкті використовується PostgreSQL, що є однією з найпотужніших реляційних СУБД з відкритим кодом.

3.1.1 Особливості створення бази даних для програмного продукту

База даних проєкту призначена для збереження та обробки наступних основних сутностей:

- Задач (точок виконання із координатами);
- Характеристик задач (винагорода, час виконання);
- Користувацьких налаштувань;
- Результатів роботи алгоритмів (маршрути, загальна відстань, сумарна винагорода, час виконання);
- Допоміжних даних (тип алгоритму, кількість ітерацій, обмеження часу тощо).

Особливості побудови БД:

- Використовується реляційна модель даних;
- Структура бази даних нормалізована, що зменшує дублювання даних;
- Застосовується зовнішні ключі (FOREIGN KEY) для забезпечення цілісності;
- Використовуються індекси для прискорення пошуку за координатами, ідентифікаторами маршрутів та користувачів;

- Передбачена можливість легкого розширення структури (додавання нових типів алгоритмів, характеристик задач тощо).

Таким чином, структура БД спроєктована з урахуванням масштабування та можливості обробки великих обсягів просторових і обчислювальних даних.

3.1.2 Кількісні та якісні характеристики бази даних

База даних у межах програмного продукту здатна обробляти:

- Від тисяч до мільйонів записів задач (в залежності від масштабів використання);
- Десятки тисяч маршрутів, збережених в історії обчислень;
- Величини винагороди та часу з плаваючою точкою;
- Координати у форматі FLOAT/DOUBLE PRECISION.

Основні якісні характеристики:

- Висока надійність (ACID-властивості транзакцій);
- Точність обчислень завдяки використанню числових типів postgresql;
- Стійкість до помилок при паралельній обробці запитів;
- Швидкість доступу до даних за рахунок індексування;
- Можливість роботи з геоданими (при підключенні postgis);
- Висока масштабованість (вертикальна і горизонтальна при кластеризації).

3.1.3 Технологічні вимоги до середовища опрацювання даних

Для ефективного функціонування бази даних визначені такі вимоги:

Сумісність з іншими системами:

- Повна інтеграція з nestjs через ORM (typeorm/Prisma);
- Сумісність із Next.js для візуалізації даних;

- Підтримка стандартних SQL-запитів та REST API;
- Можливість роботи у Docker-середовищі.

Мови програмування:

- TypeScript (основна мова клієнтської та серверної частини);
- SQL (опис структури, індекси, оптимізаційні запити).

Інтерфейси:

- Restful API для обміну даними між фронтендом та бекендом;
- Внутрішній API для алгоритмічних модулів;
- Pgadmin / dbeaver для адміністрування БД.

Продуктивність:

- Підтримка великої кількості одночасних користувачів;
- Обробка складних аналітичних запитів;
- Оптимізоване виконання JOIN-операцій;
- Можливість кешування результатів.

Мережні можливості:

- Робота у локальній та хмарній мережі;
- Підтримка TCP/IP-з’єднань;
- Можливість використання SSL-з’єднання;
- Масштабування через реплікацію.

Обсяг даних:

- Підтримка до декількох терабайтів інформації;
- Ефективне управління великими таблицями.

3.1.4 Характеристика програмного продукту управління базами даних

У проєкті використовується PostgreSQL, який має такі технологічні переваги:

- Безкоштовне та стабільне програмне забезпечення з відкритим кодом;

- Підтримка складних типів даних: json, масиви, геодані;
- Висока швидкість обробки транзакцій;
- Безпека доступу на рівні ролей;
- Підтримка тригерів, процедур і функцій;
- Зручна інтеграція з node.js / nestjs;
- Активна спільнота та довгострокова підтримка.

Для управління БД використовуються:

- ORM (Prisma або TypeORM) - для зручної роботи з даними через класи та моделі;
- pgAdmin / DBeaver - для візуального управління та адміністрування;
- Docker - для стандартизованого розгортання в різних середовищах.

Обрана технологія бази даних повністю відповідає вимогам даного програмного продукту, оскільки дозволяє швидко обробляти великі обсяги даних, зберігати результати обчислень та забезпечувати стабільну роботу алгоритмів оптимізації.

3.2 Вибір та обґрунтування засобів вирішення проблеми

Ефективність реалізації програмного продукту для оптимізації маршрутів та прийняття рішень значною мірою залежить від правильного вибору програмних, системних і допоміжних засобів, що забезпечують накопичення, обробку, зберігання та візуалізацію даних. У межах даної магістерської роботи обрані засоби повинні відповідати вимогам продуктивності, надійності, гнучкості та масштабованості.

3.2.1 Вимоги до платформи, на якій реалізується розроблений продукт

Обрана платформа повинна забезпечувати:

- Кросплатформність (можливість роботи на різних операційних системах);
- Високу продуктивність для виконання евристичних та метаевристичних алгоритмів (жадібний, генетичний, мурашиний);
- Стабільну роботу з великою кількістю даних;
- Можливість швидкої візуалізації результатів на клієнтській стороні;
- Підтримку асинхронних обчислень;
- Роботу з реляційними базами даних;
- Масштабованість і можливість хмарного розгортання;
- Підтримку сучасних веб-інтерфейсів і REST API.

Усім зазначеним вимогам відповідає технологічна платформа Node.js + TypeScript, що лежить в основі як фронтенду (Next.js), так і бекенду (NestJS).

3.2.2 Обґрунтування вибору операційної системи та середовища реалізації

У якості основного операційного середовища для розробки й розгортання програмного продукту обрана Ubuntu (Linux) та/або Windows 10/11 з підтримкою WSL, а також можливість розміщення у хмарних середовищах (типу AWS, Vercel, Railway, DigitalOcean).

Переваги обраного середовища:

- Сумісність із базовими технологіями (Node.js, postgresql, Docker);
- Стабільність і безпека ОС Linux;
- Підтримка контейнеризації через Docker;
- Безкоштовність і відкритість програмного забезпечення;
- Зручність автоматизації розгортання (CI/CD);
- Перевага над застарілими середовищами (наприклад, Windows Server без контейнеризації або локальні монолітні установки).

Середовище Node.js дозволяє запускати обчислювальні модулі незалежно від платформи, що забезпечує гнучкість і масштабованість системи.

Характеристика вибраного операційного середовища з точки зору функціонування програмного продукту

Операційне середовище забезпечує:

- Багатозадачність та асинхронну обробку запитів;
- Стабільну роботу серверних процесів;
- Швидку обробку даних з бд;
- Виконання алгоритмів оптимізації у режимі реального часу;
- Мережну взаємодію клієнт-сервер;
- Запуск у контейнеризованому середовищі без втрати продуктивності.

Ubuntu/Windows+WSL дозволяє:

- Використовувати сучасні системи керування пакетами;
- Масштабувати обчислення;
- Розгортати середовище як локально, так і на віддалених серверах;
- Забезпечувати стабільну роботу PostgreSQL.

Таким чином, обране системне середовище повністю відповідає особливостям та вимогам програмного продукту.

3.2.3 Вимоги, вибір та обґрунтування застосування додаткових сервісних, допоміжних та інтерфейсних засобів

Для забезпечення повноцінного функціонування розробленого програмного продукту використовуються додаткові інструменти:

Сервісні засоби:

- Docker – для контейнеризації сервісів (postgresql, бекенд).
- Git + github/gitlab – для контролю версій та командної розробки.
- Postman / Insomnia – для тестування API.

- Pgadmin / dbeaver – для адміністрування postgresql.
- Інтерфейсні засоби:
- Next.js + React – для створення інтерактивного інтерфейсу користувача;
- Tailwind CSS / Material UI – для оформлення інтерфейсу;
- Бібліотеки візуалізації (наприклад, Leaflet або Google Maps API) – для відображення маршрутів на карті.

Допоміжні програмні засоби:

- TypeORM або Prisma – для взаємодії з базою даних;
- Zod / Joi – для валідації вхідних даних;
- Swagger – для автоматичної документації API;
- JWT / OAuth – для авторизації користувачів.

Усі вищенаведені засоби є широко використовуваними у промисловій розробці, мають активну підтримку та забезпечують високу якість, безпеку й масштабованість програмного продукту.

3.3 Експериментальний аналіз ефективності алгоритмів

У межах дослідження розглядається задача побудови маршруту виконавця мікрозавдань, який має обмежений часовий ресурс C_{max} (наприклад, 2 години) та прагне максимізувати сумарну винагороду за виконані завдання.

Кожне завдання характеризується:

- Координатами (lat, lon),
- Винагородою r_i,
- Орієнтовним часом виконання c_i;
- Відстанню між завданнями d_{ij}.

Виконавець може обрати підмножину завдань $S \subseteq T$, де $T=\{ t_1,t_2,...,t_n\}$ - множина усіх доступних завдань.

Мета оптимізації полягає у максимізації функції:

$$max \sum_{i \in S}^{\square} r_i ,$$

за умови, що сумарний час виконання не перевищує допустимого:

$$\sum_{i \in S}^{\square} \left(c_i + d_{prev,i}\right) \leq C_{max} ,$$

де $d_{prev,i}$ - час на переміщення між послідовними завданнями.

Алгоритми, відібрані для порівняння:

- Жадібний алгоритм (Greedy Approach). На кожному кроці вибирає найближчу задачу з максимальною віддачею (винагорода / час);
- Genetic Algorithm (GA). Кожен маршрут - «хромосома», використовуються операції кросовера і мутації для пошуку найкращого порядку задач;
- Ant Colony Optimization (ACO). Мурахи прокладають маршрути з урахуванням феромонів, що відображають комбінацію відстані і винагороди.

Параметри експерименту:

Кількість задач: 20, 50, 100.

Максимальний час: 1 год, 2 год, 3 год.

Критерії порівняння:

- Сумарна винагорода;
- Кількість виконаних задач;
- Загальний час виконання;
- Відстань маршруту;
- Час розрахунку алгоритму.

3.4 Greedy Approach

Для початку, створимо 20 задач з центром в точці S:

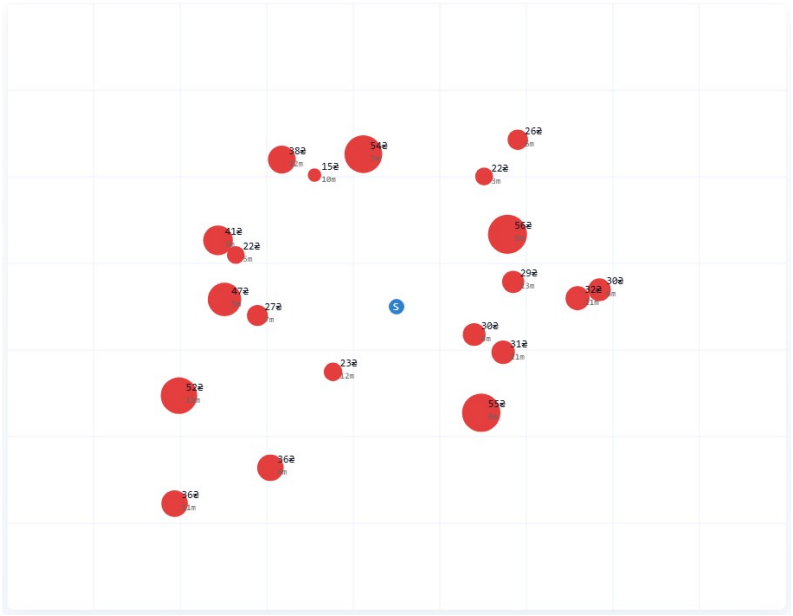

Рисунок 3.1 - 20 задач

Запускаємо жадібний алгоритм при 20 задачах і максимальним часом виконання 1 година:

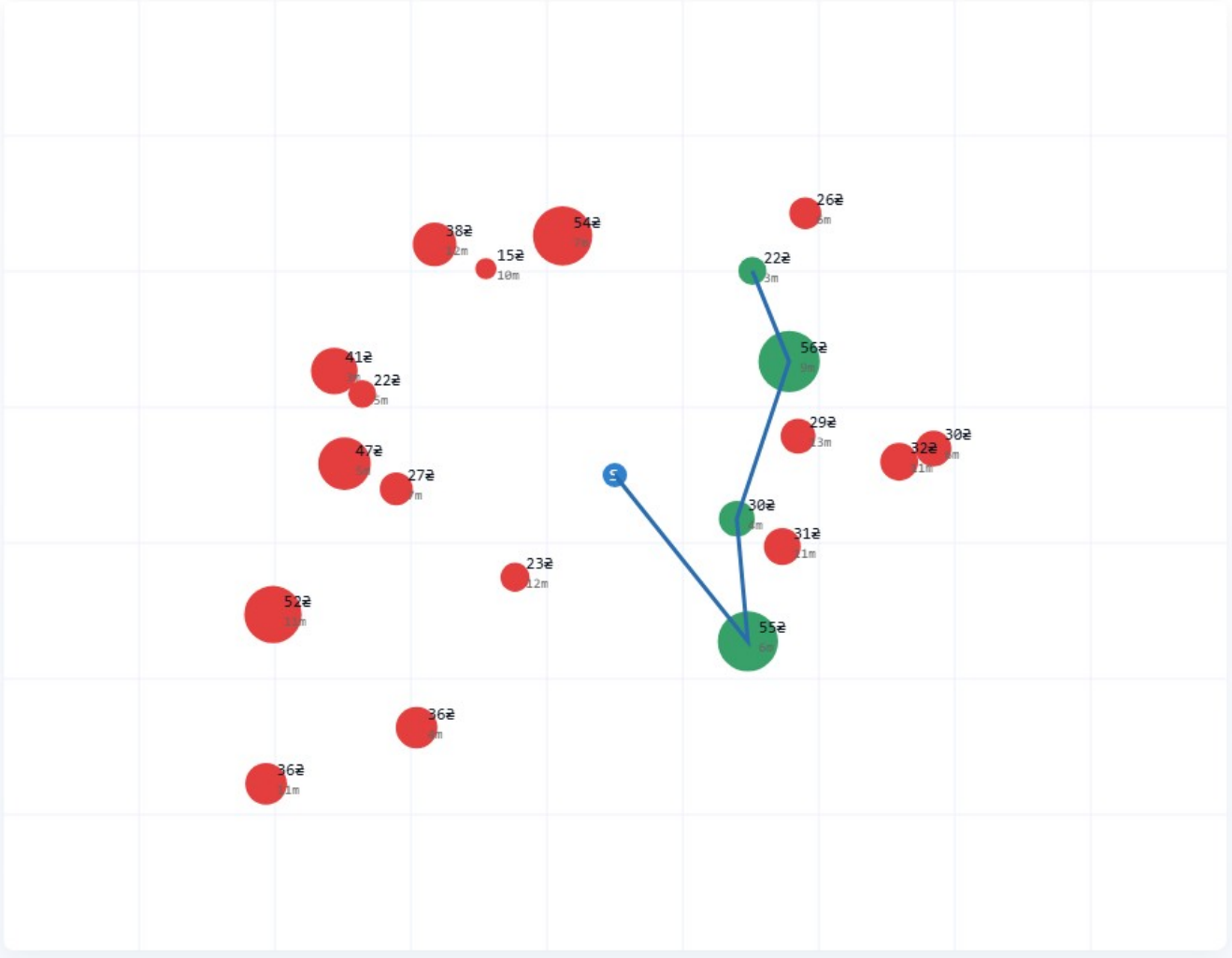

Рисунок 3.2 - 20 задач, результат жадібного алгоритму

Результати:

- Винагорода: 163 грн
- Витрачений час: 117 / 120 хв
- Загальна відстань: 8.83 км
- Час роботи алгоритму: 0.10 мс
- Виконано задач: 4

Тепер згенеруємо 50 задач:

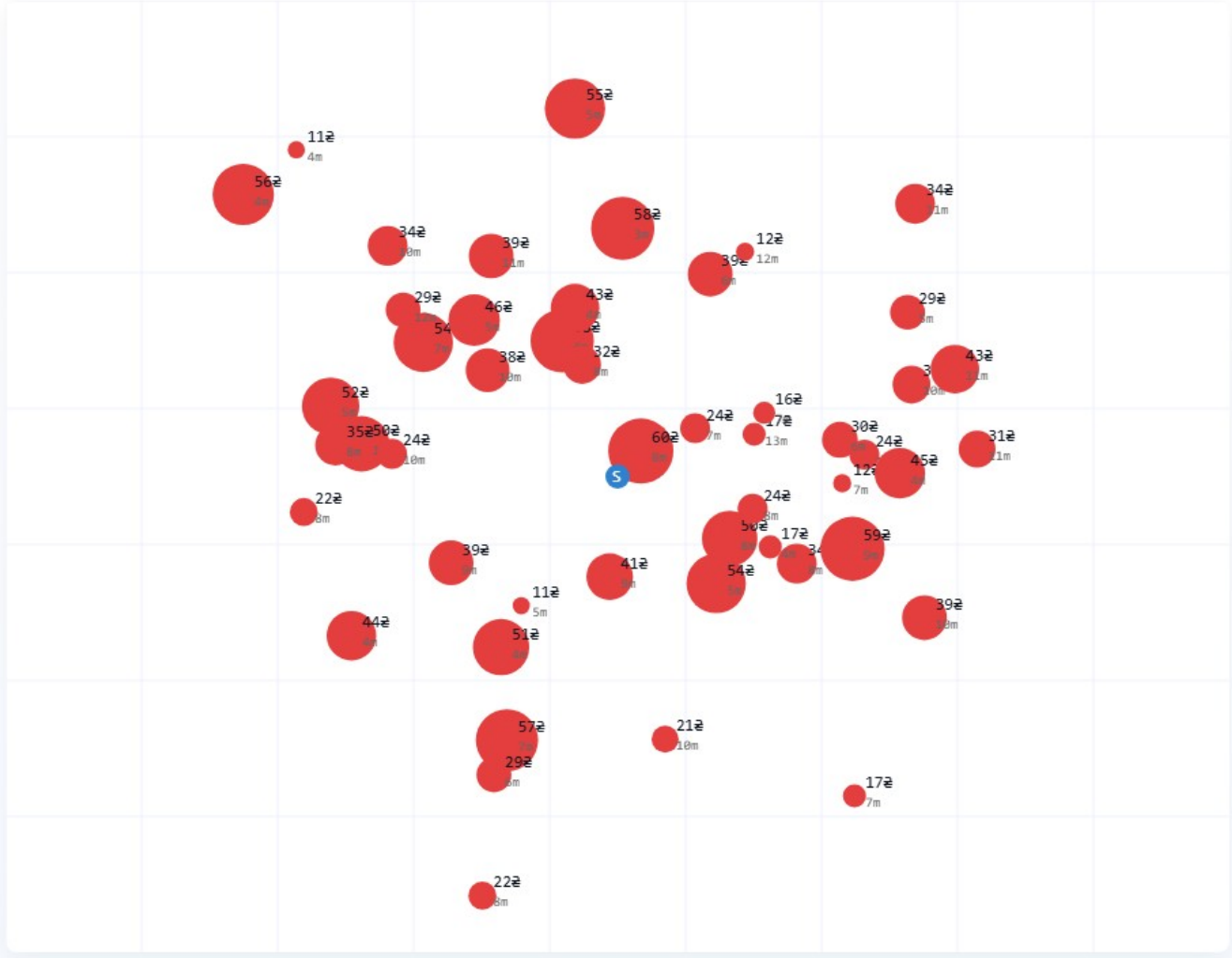

Рисунок 3.3 - 50 задач

Запускаємо жадібний алгоритм при 50 задачах і максимальним часом виконання 2 години:

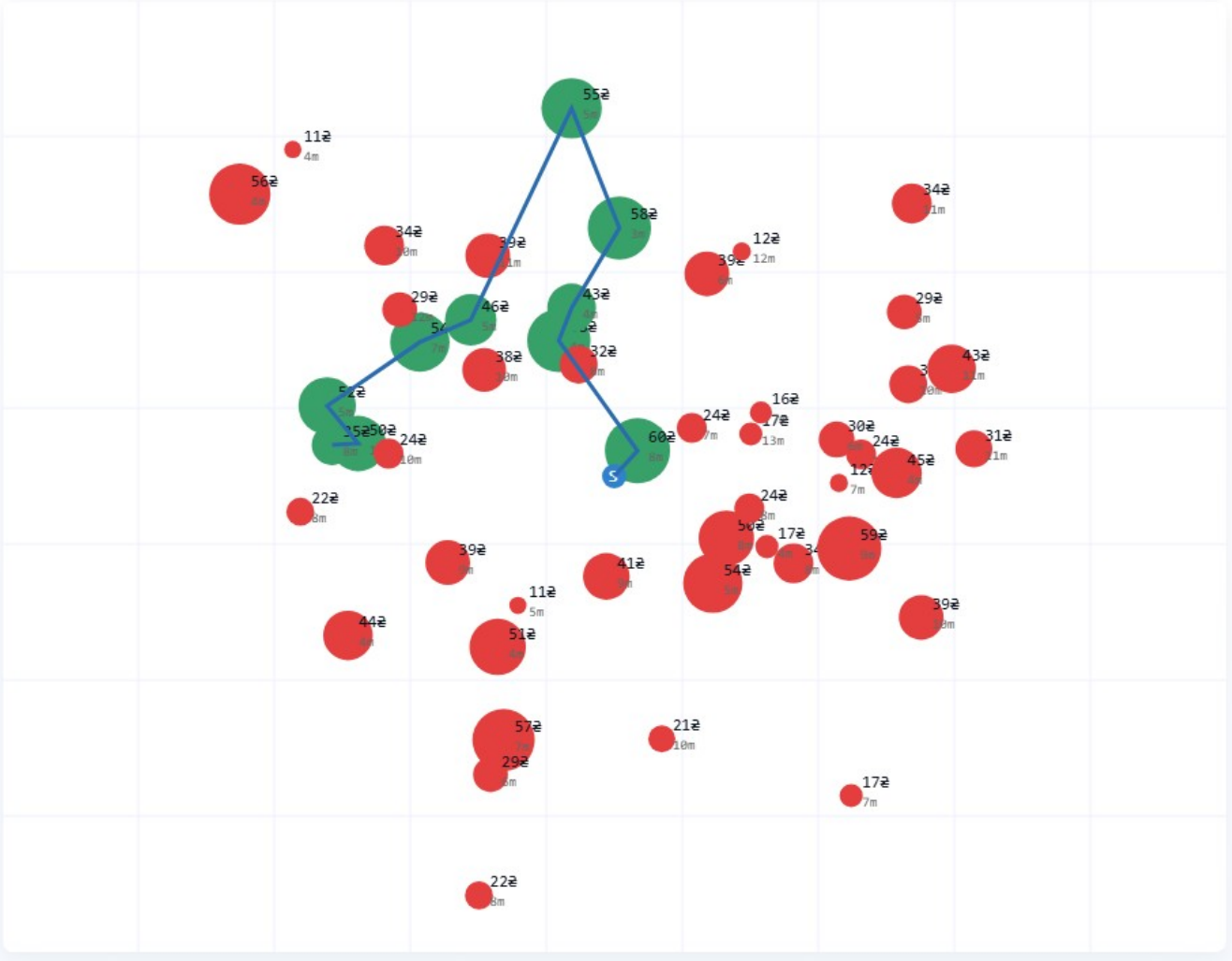

Рисунок 3.4. - 50 задач, результат жадібного алгоритму

Результати:

- Винагорода: 511 грн
- Витрачений час: 112 / 120 хв
- Загальна відстань: 13.30 км
- Час роботи алгоритму: 0.10 мс
- Виконано задач: 10

І під кінець 100 задач:

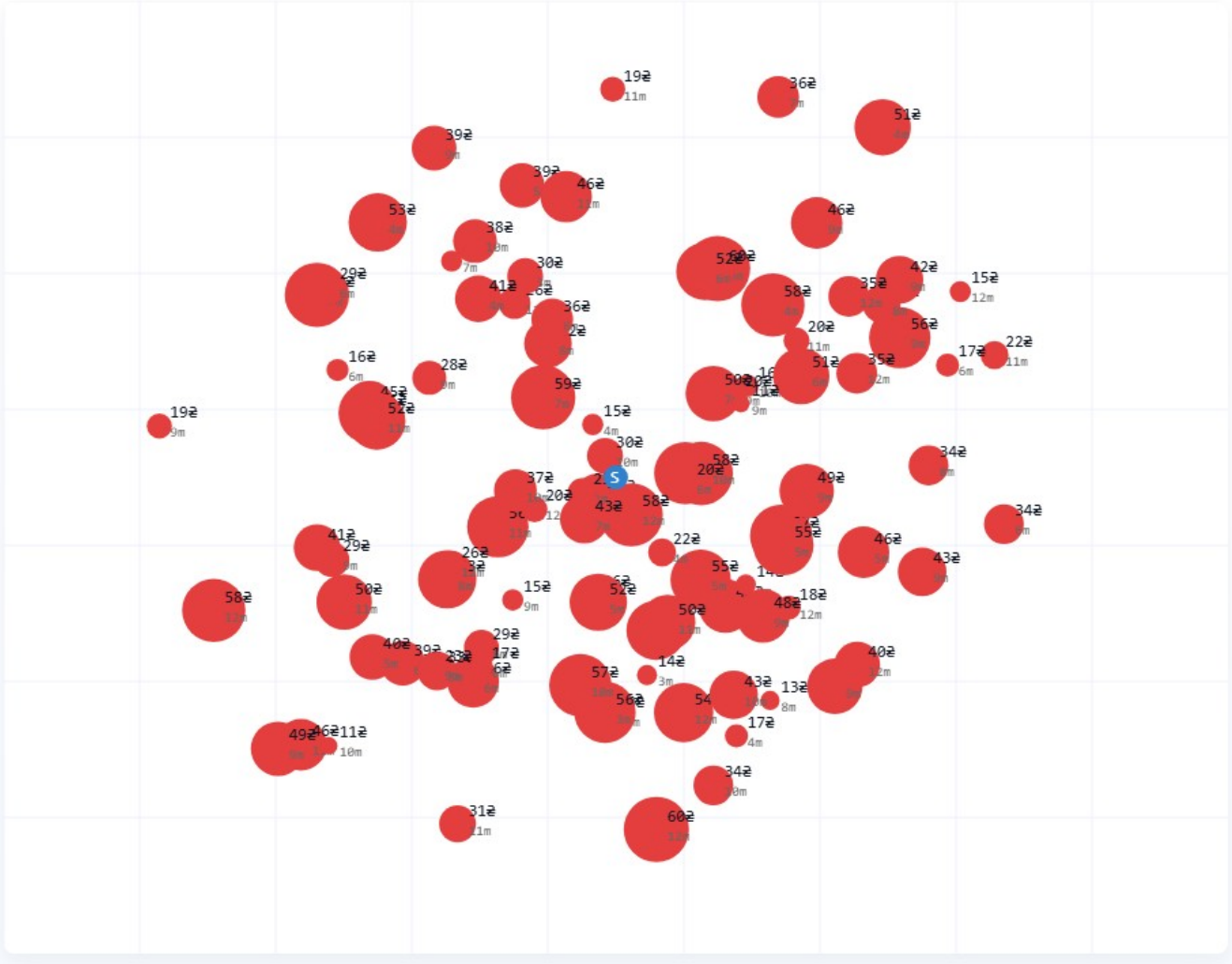

Рисунок 3.5 - 100 задач

Запускаємо жадібний алгоритм:

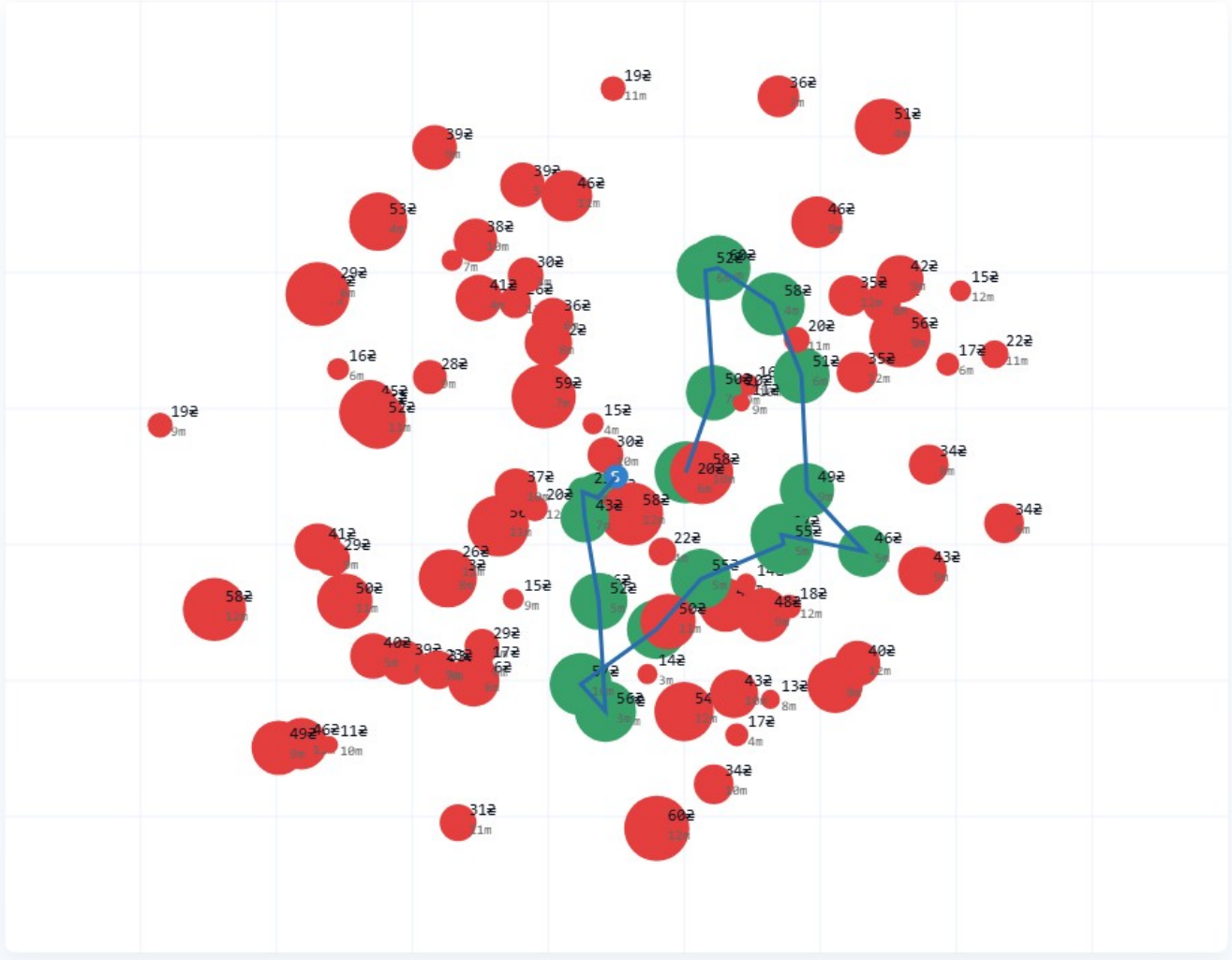

Рисунок 3.6 - 100 задач, результат жадібного алгоритму

Результати:

- Винагорода: 919 грн
- Витрачений час: 178 / 180 хв
- Загальна відстань: 17.75 км
- Час роботи алгоритму: 0.30 мс
- Виконано задач: 18

3.5 Genetic Algorythm

Тепер протестуємо генетичний алгоритм. Почнемо з 20 задач, згенерованих вище:

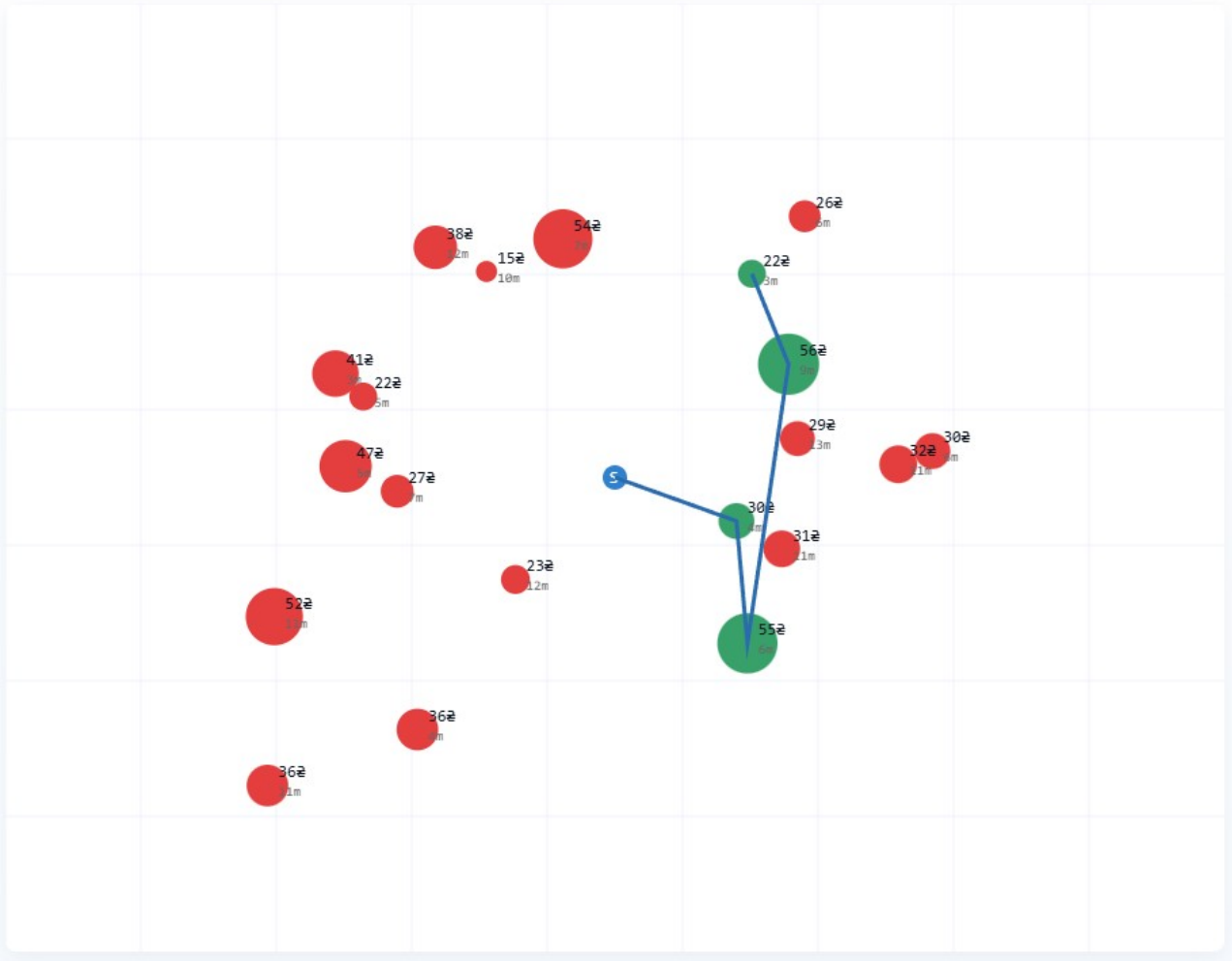

Рисунок 3.7 - 20 задач, результат генетичного алгоритму

Результати:

- Винагорода: 163.0 грн
- Витрачений час: 59.3 / 60 хв
- Загальна відстань: 9.32 км
- Час роботи алгоритму: 3.80 мс
- Виконано задач: 4

Результат такий же. Протестуємо на 50 задачах:

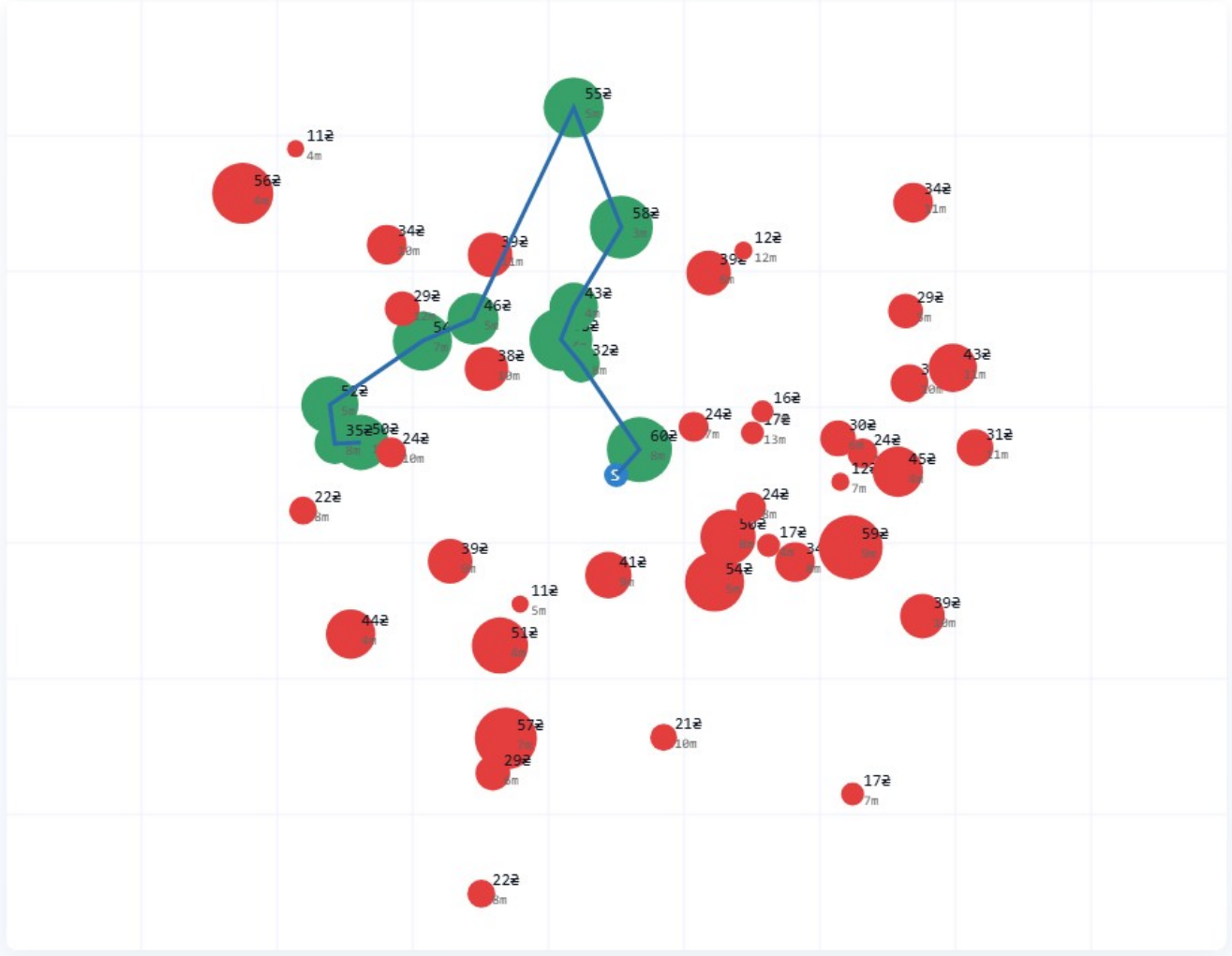

Рисунок 3.8 - 50 задач, результат генетичного алгоритму

Результати:

- Винагорода: 543.0 грн
- Витрачений час: 119.7 / 120 хв
- Загальна відстань: 13.16 км
- Час роботи алгоритму: 9.20 мс
- Виконано задач: 11

Одразу кидається в очі перевага по всім пунктам над жадібним алгоритмом. Наступними будуть 100 задач:

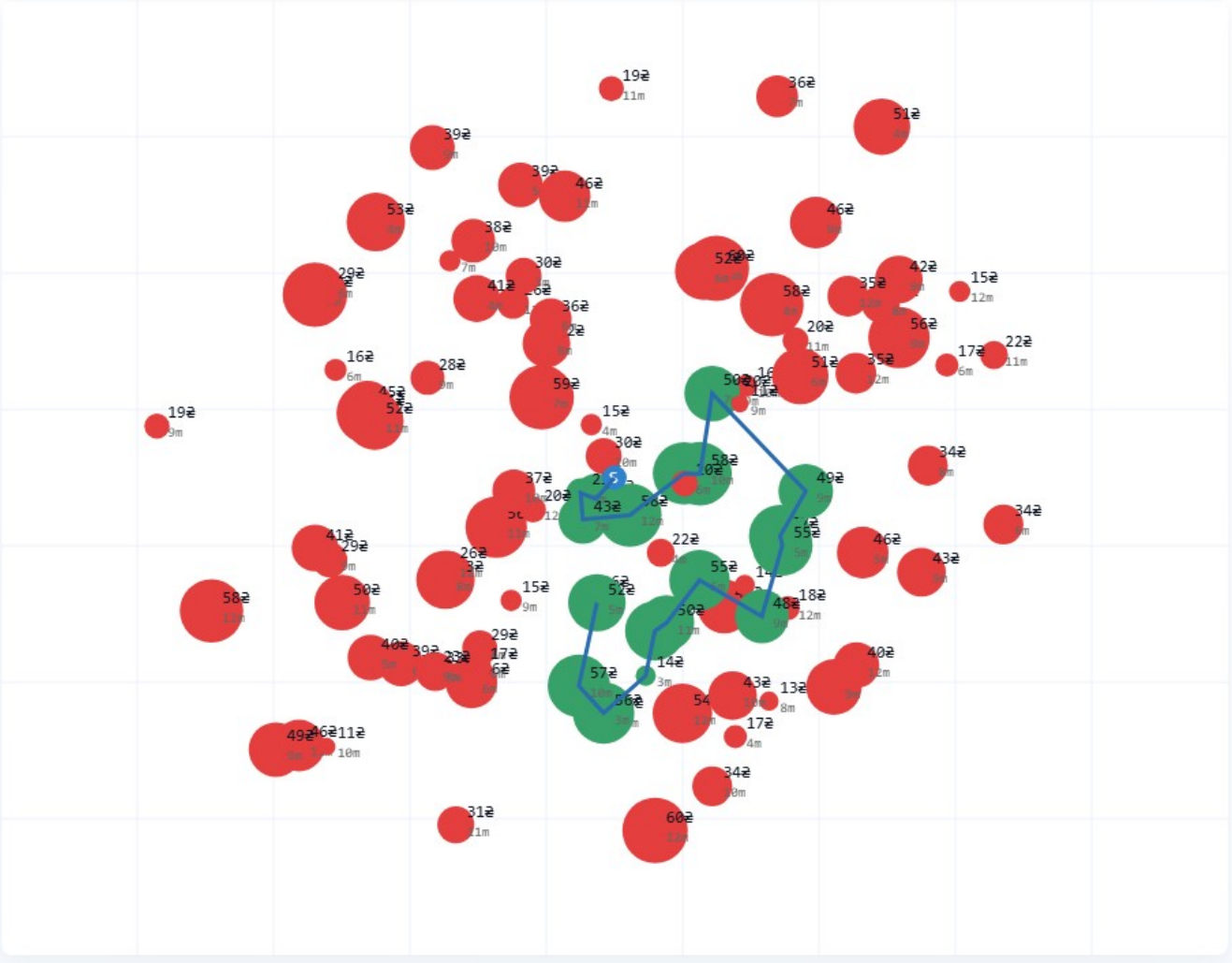

Рисунок 3.9 - 100 задач, результат генетичного алгоритму

Результати:

- Винагорода: 880.0 грн
- Витрачений час: 178.9 / 180 хв
- Загальна відстань: 13.47 км
- Час роботи алгоритму: 22.00 мс
- Виконано задач: 18

Жадібний алгоритм переміг у цьому випадку.

3.6 Ant Colony Optimization

Наступним алгоритмом буде Ant Colony Optimization (ACO). Почнемо, як завжди, з 20 задач:

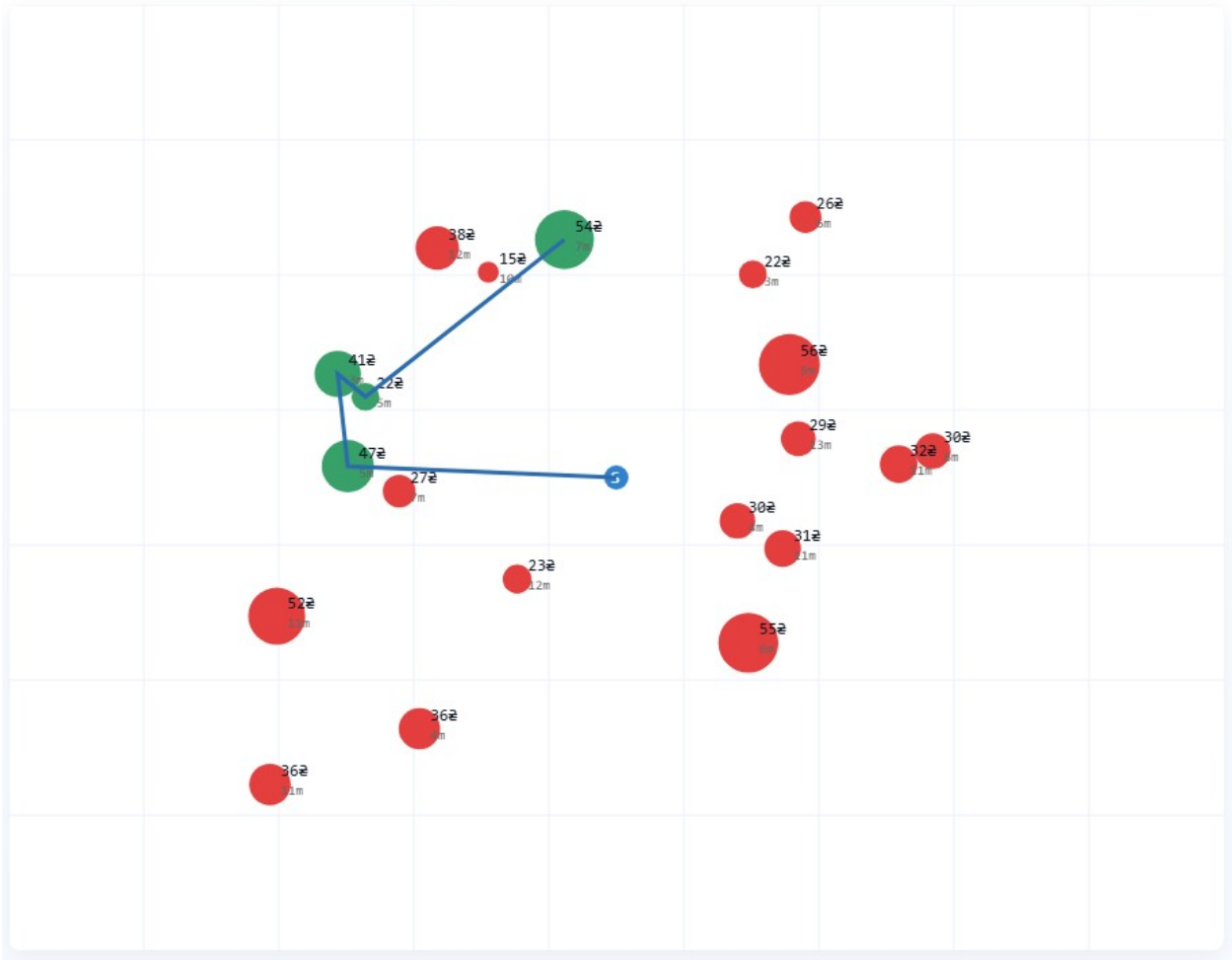

Рисунок 3.10 - 20 задач, результат алгоритму ACO

Результати:

- Винагорода: 164.0 грн
- Витрачений час: 59.8 / 60 хв
- Загальна відстань: 9.95 км

- Час роботи алгоритму: 13.50 мс
- Виконано задач: 4

Результат такий же, як і в попередньому алгоритмі, перейдемо до 50 задач:

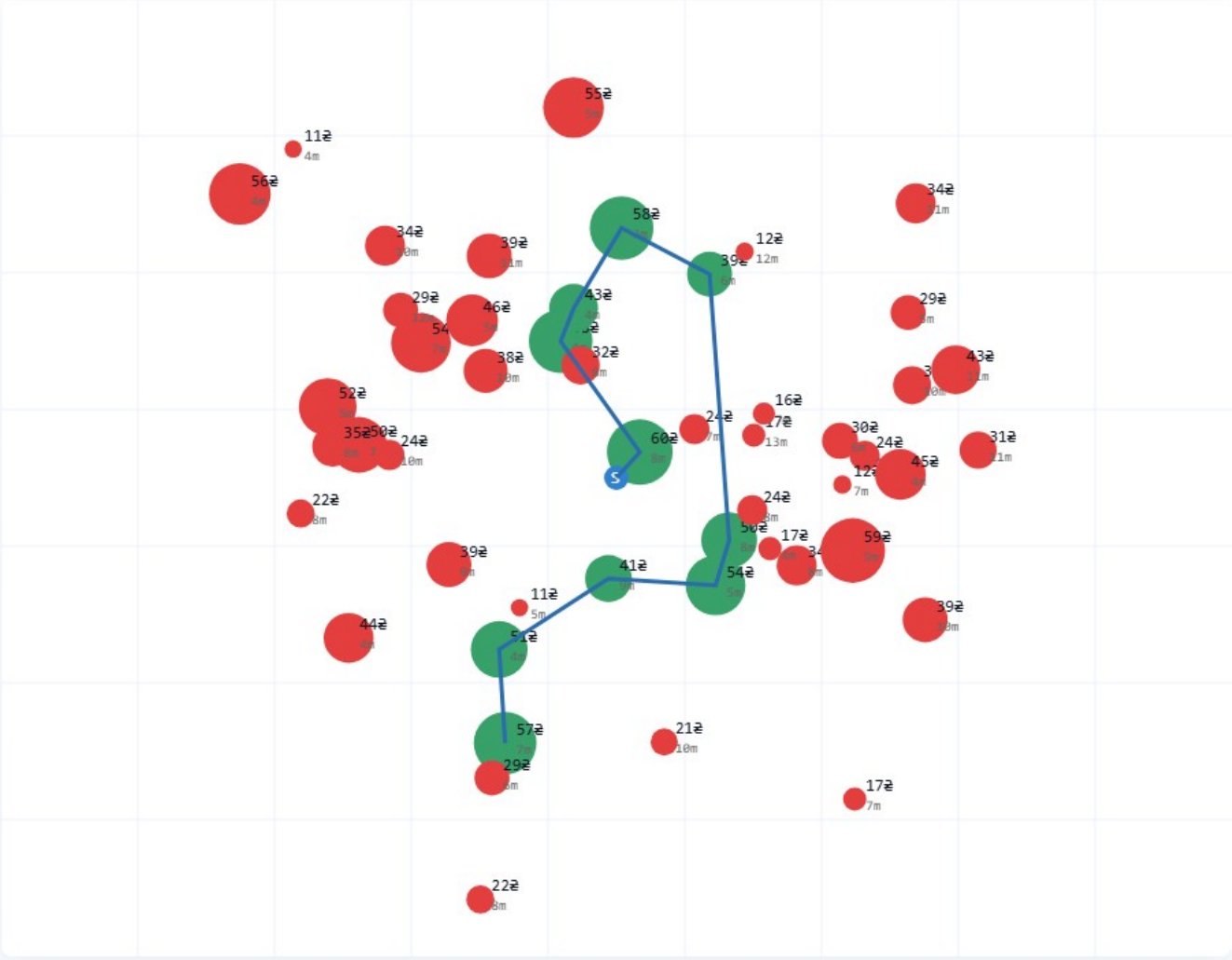

Рисунок 3.11 - 50 задач, результат алгоритму ACO

Результат:

- Винагорода: 511.0 грн
- Витрачений час: 119.1 / 120 хв
- Загальна відстань: 15.27 км

- Час роботи алгоритму: 68.00 мс
- Виконано задач: 10

Як можна побачити, результат є гіршим, ніж обидва кандидати до цього. Також час роботи зростає дуже швидко. Наступними будуть 100 задач:

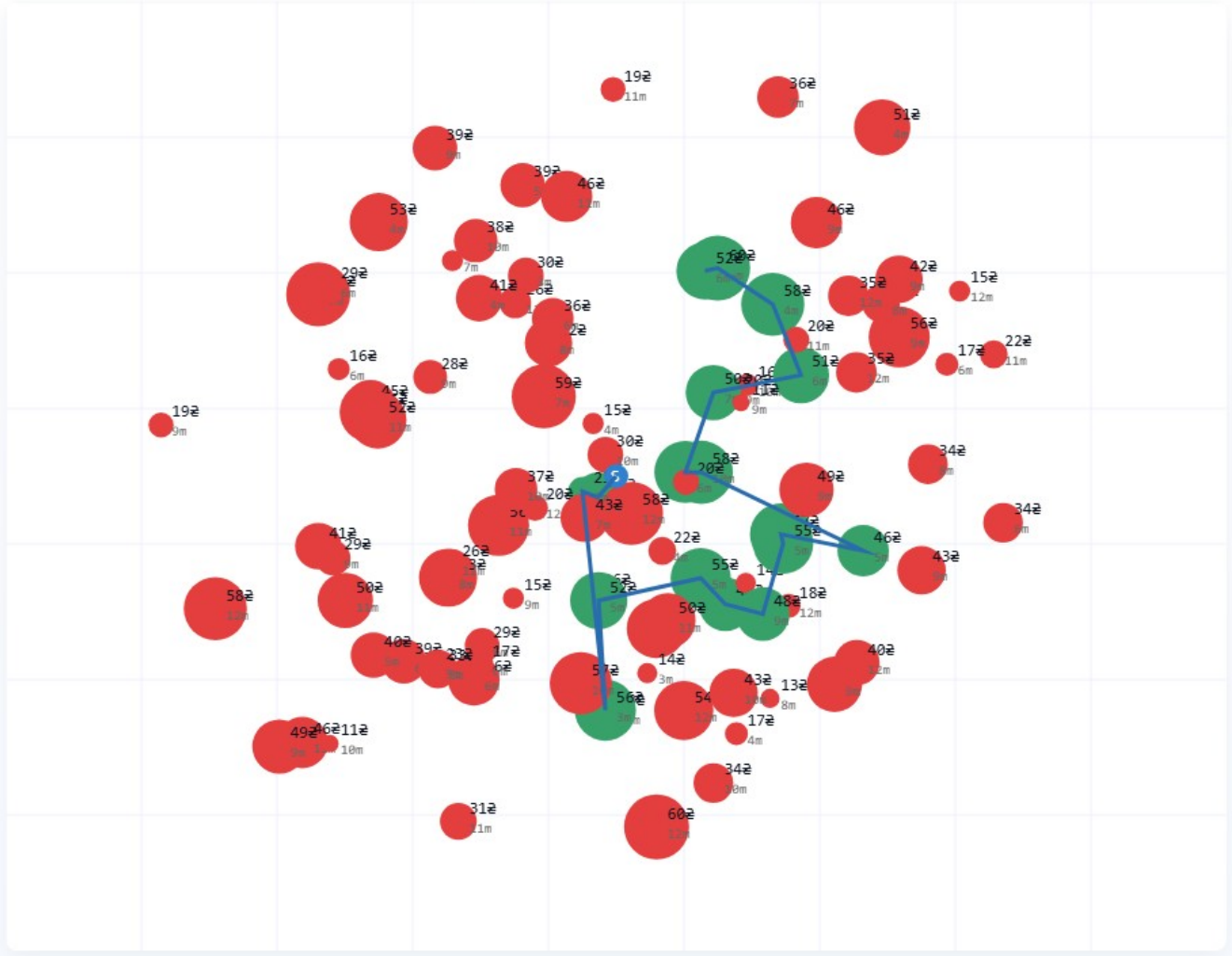

Рисунок 3.12 - 100 задач, результат алгоритму ACO

Результат:

- Винагорода: 870.0 грн
- Витрачений час: 178.4 / 180 хв

- Загальна відстань: 18.36 км
- Час роботи алгоритму: 244.70 мс
- Виконано задач: 17

Нажаль, винагорода програє жадібному алгоритму, та час роботи вже на грані допустимого.

Висновки до розділу

Створимо таблицю результатів екперименту:

Для 20 задач:

Таблиця 3.1

	Загальна відстань	Витрачений час	Виконано задач	Час роботи алгоритму	Винагорода
Жадібний алгоритм	8.83 км	58/60	4	0.10 мс	163
Генетичний алгоритм	9.27 км	57.1 / 60	4	3.30 мс	164
Алгоритм ACO	9.95 км	59.8 / 60	4	13.50 мс	164

Для 50 задач:

Таблиця 3.2

	Загальна відстань	Витрачений час	Виконано задач	Час роботи алгоритму	Винагорода
Жадібний алгоритм	13.30 км	112 / 120	10	0.10 мс	511
Генетичний алгоритм	13.16 км	119.7 / 120	11	9.20 мс	543

Алгоритм ACO	15.27 км	119.1 / 120	10	68.00 мс	511

Для 100 задач:

Таблиця 3.3

	Загальна відстань	Витрачений час	Виконано задач	Час роботи алгоритму	Винагорода
Жадібний алгоритм	17.75 км	178 / 180	18	0.30 мс	919
Генетичний алгоритм	13.47 км	178.9 / 180	18	22.00 мс	880
Алгоритм ACO	18.36 км	178.4 / 180	17	244.70 мс	870

Підсумувавши, можна зробити висновок, що жадібний алгоритм показує непогані результати і витрачає набагато менше часу на знаходження маршруту. Але генетичний алгоритм показував більш стабільні маршрути, які підходять в більшій кількості випадків, при цьому його час знаходження є допустимим.

Найгіршим алгоритмом в даному випадку виявився ACO - він програє по кожній метриці.

Тому найкращим варіантом буде модифікація на основі генетичного алгоритму.

РОЗДІЛ 4. ПРАКТИЧНА РЕАЛІЗАЦІЯ

4.1 Структура БД

У проекті було використано СУБД PostgreSQL. Його переваги:

- Безкоштовний і з відкритим кодом;
- Кросплатформеність;
- Швидкість;
- Маштабованість;
- Безпека.

Основні таблиці:

users:

- id - uuid not null primary key;
- username - charvar not null;
- password - charvar not null.

tasks:

- id - uuid not null primary key;
- title - charvar not null;
- description - text;
- reward - numeric not null;
- latitude - numeric not null;
- longitude - numeric not null;
- estimated_duration - integer;
- status - enum not null;
- created_by - uuid foreign key users;
- assigned_to - uuid foreign key users.

skills:

- id - uuid not null primary key;
- title - charvar not null.

user_skills:

- user_id - uuid foreign key users;
- skill_id - uuid foreign key skills.

task_skills:

- task_id - uuid foreign key tasks;
- skill_id - uuid foreign key skills.

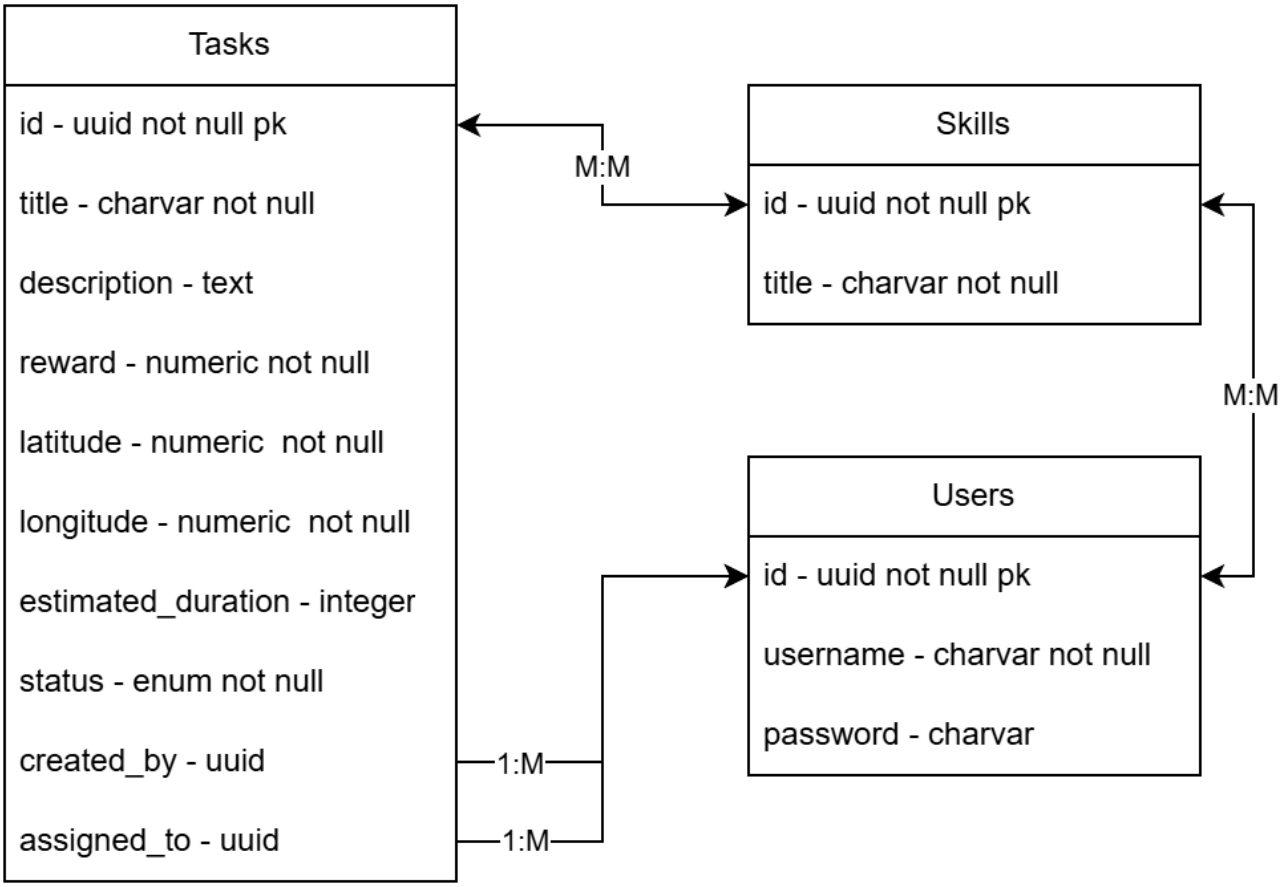

Рисунок 4.1 – Структура БД

4.2 Фронтенд

Фронтенд реалізовано за допомогою фреймворку Next.Js. Його переваги:

- Оптимізована робота з великими обсягами даних (маршрути, точки, метрики);

- Вбудована маршрутизація та зручна структура проєкту;
- Легка інтеграція з візуалізаційними бібліотеками (Canvas, D3.js, Chart.js);
- Підтримка сучасних стандартів React (hooks, server components);
- Висока масштабованість.

4.2.1 Компоненти

4.2.1.1 Мапа (Map / MapContent)

Візуалізує географічні дані, розташування задач та оптимізовані маршрути.
Бібліотека: react-leaflet (обгортка над Leaflet).
Функціонал:

- Маркери задач: Відображають доступні задачі на мапі. Колір маркера змінюється залежно від статусу (обрана/звичайна).
- Маркер користувача: Показує стартову точку (поточне місцезнаходження).
- Маршрут (Polyline): Візуалізація шляху, побудованого генетичним алгоритмом.
- Popup: Спливаючі вікна з деталями задачі при кліку на маркер.
- Auto-fit: Автоматичне масштабування мапи, щоб охопити всі вибрані задачі або побудований маршрут.

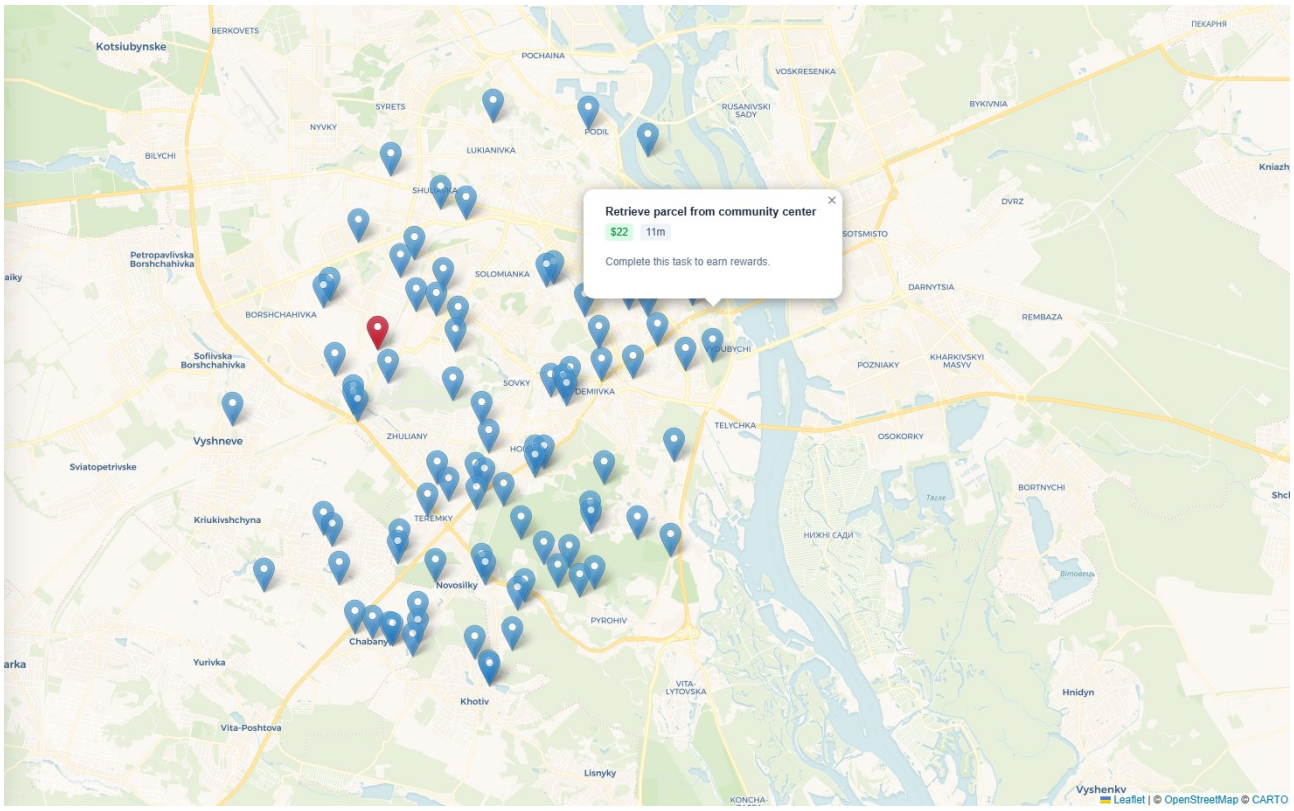

Рисунок 4.3 – Компонент карта

4.2.1.2 Сайдбар (Sidebar)

Панель управління, що забезпечує взаємодію користувача з системою.

Функціонал:

- Список задач: Прокручуваний список карток задач, знайдених поблизу.
- Фільтрація/Пошук: Кнопка для запиту задач у радіусі навколо користувача.
- Панель оптимізації: Кнопки для вибору задач, очищення вибору та запуску розрахунку оптимального маршруту ("Generate Optimal Path").
- Адаптивність: На мобільних пристроях може згортатися/розгортатися.

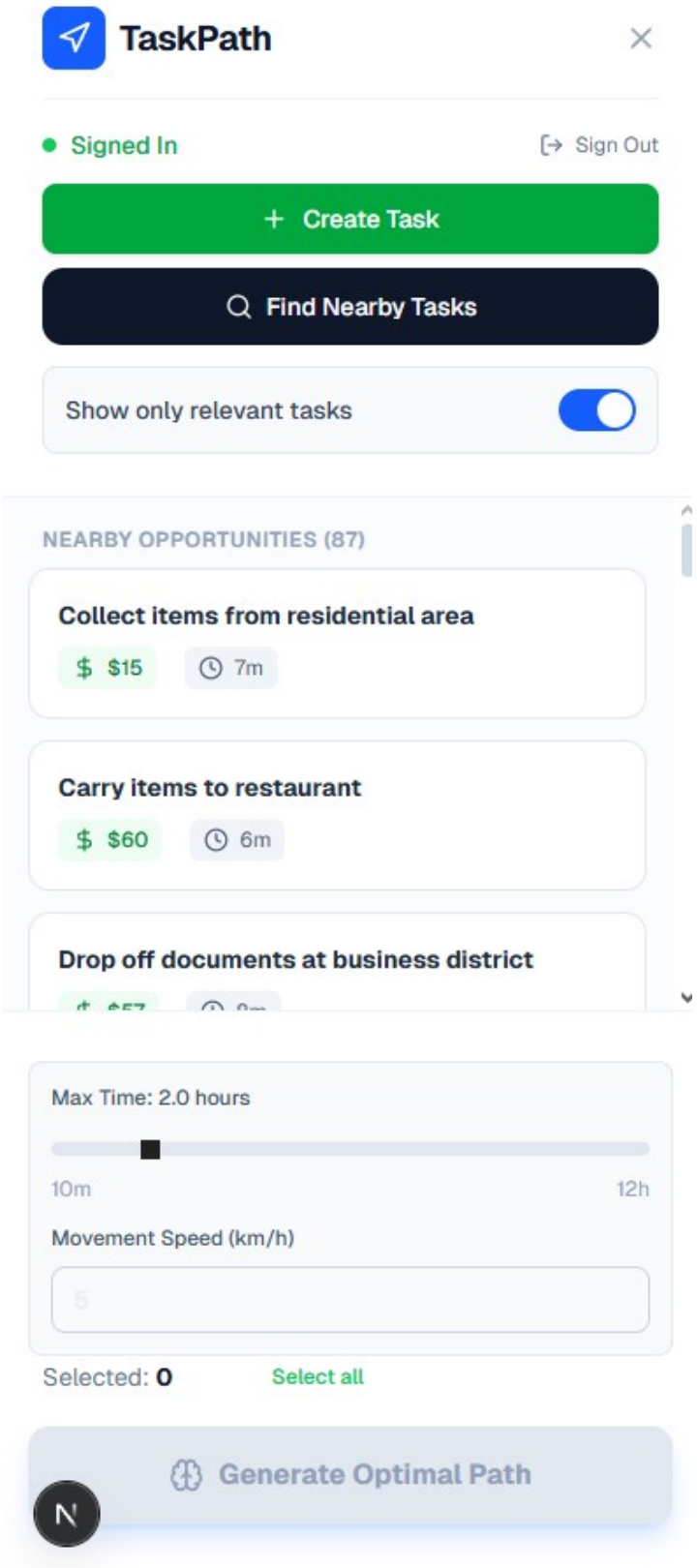

Рисунок 4.4 – Компонент сайдбар

4.2.1.3 Задача (Task / TaskCard)

Картка, що представляє окрему одиницю роботи в списку.

Дані: Назва, винагорода (валюта), орієнтовний час виконання, відстань.

Взаємодія:

Клік по картці додає або видаляє задачу зі списку для оптимізації маршруту.

Візуальна індикація стану "обрано" (підсвічування, іконка галочки).

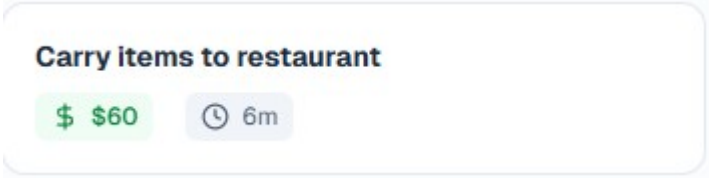

Рисунок 4.5 – Компонент Задача

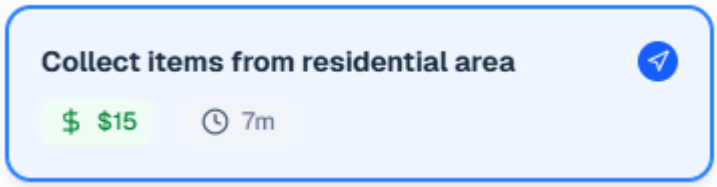

Рисунок 4.6 – Компонент Задача (обрана)

4.2.2 Сторінки (Pages)

Маршрутизація застосунку (Next.js App Router).

4.2.2.1 Головна (Main Page)

Основна робоча область користувача (/).

Склад: Інтегрує компоненти Sidebar та MapContainer.

Логіка: При завантаженні ініціалізує карту, завантажує початкові дані про задачі навколо користувача.

SSR/CSR: Використовує клієнтський рендеринг ("use client") для коректної роботи мапи.

4.2.2.2 Реєстрація (Registration Page)

Сторінка створення нового облікового запису (/auth/register).

Поля: Ім'я користувача, пошта, пароль, вибір початкових навичок (skills).

Дія: Відправляє POST-запит на бекенд для створення сутності User.

4.2.2.3 Авторизація (Authorization Page)

Сторінка входу в систему (/auth/login).

Поля: Логін/Пошта, пароль.

Дія: Отримує JWT токен, зберігає його та завантажує профіль користувача в Redux store.

4.2.2.4 Сторінка задачі (Task Details Page)

Окрема сторінка для детального перегляду задачі (/tasks/[id]).

Використання: Для перегляду розширеного опису, вимог до навичок, фотографій місця виконання або коментарів замовника.

Дії: Кнопка "Взяти в роботу" або "Поскаржитись".

4.2.3 Утиліти та Стан (Utilities & Store)

Логіка управління даними та взаємодії з API.

4.2.3.1 Навички (Skills Utility)

Управління компетенціями користувача та вимогами задач.

Призначення: Фільтрація задач, які користувач не може виконати через брак навичок.

Структура: Список ID навичок (наприклад, ['painting', 'delivery']).

4.2.3.2 Задачі (Tasks Utility / Slice)

Redux-слайс tasksSlice для управління списком доступних задач.

State:

- items: Масив об'єктів задач.
- status: Стан завантаження (idle, loading, succeeded, failed).

Actions: fetchNearbyTasks (Async Thunk) — запит до API для отримання задач за координатами.

4.2.3.3 Шлях (Path Utility / Slice)

Redux-слайс pathSlice для управління логікою маршрутизації.

State:

- selectedTaskIds: Масив ID задач, які користувач хоче виконати.
- optimizedPath: Впорядкований масив задач (результат роботи генетичного алгоритму).

Actions:

- toggleTaskSelection: Додавання/видалення задачі з вибірки.
- fetchOptimalPath: Відправка запиту на бекенд для запуску генетичного алгоритму.

4.2.3.4 Користувач (User Utility)

Управління даними поточної сесії.

State: ID користувача, поточні геолокаційні координати, баланс, список навичок.

Логіка: Забезпечує передачу userId та координат при запитах до API.

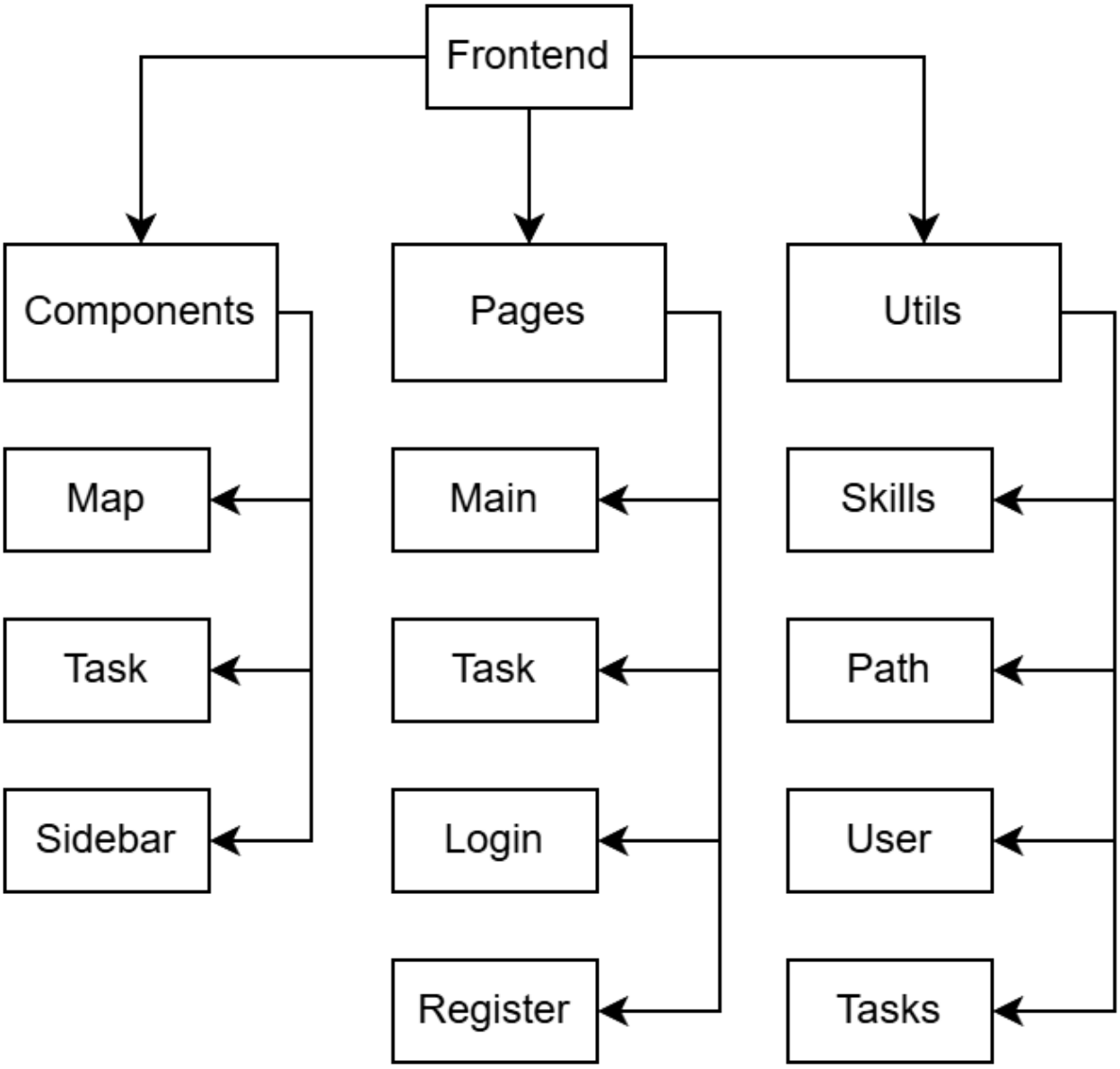

Рисунок 4.7 – Структура фронтенду

4.3 Бекенд

Бекенд спроєктовано за модульним принципом NestJS. Застосунок розділено на логічні блоки (модулі), кожен з яких відповідає за певну доменну область.

Основні технології:

Runtime: Node.js

Framework: NestJS (TypeScript)

Database: PostgreSQL

ORM: TypeORM

Validation: class-validator, class-transformer

4.3.1 Модулі

4.3.1.1 AppModule

Кореневий модуль застосунку.

Відповідальність: Конфігурація підключення до бази даних, імпорт інших модулів, налаштування глобальних змінних середовища.

4.3.1.2 TasksModule

Відповідає за управління задачами (створення, пошук, фільтрація).

Контролери: TasksController

Сервіси: TasksService

Сутності: Task, Skill

4.3.1.3 PathfindingModule

Модуль, що містить логіку штучного інтелекту для побудови маршрутів.

Контролери: PathfindingController

Сервіси: PathfindingService

Особливість: Ізольована логіка для важких обчислень (генетичний алгоритм).

4.3.2 База Даних (Entities)

Схема бази даних спроєктована для підтримки геопросторових запитів та зв'язків між користувачами і задачами.

4.3.2.1 User (Користувач)

Представляє учасника системи (виконавця або замовника).

id (UUID): Унікальний ідентифікатор.

username (String): Ім'я користувача.

skills (ManyToMany): Зв'язок з таблицею Skill. Визначає, які типи робіт може виконувати користувач.

4.3.2.2 Task (Задача)

Основна одиниця роботи на мапі.

- id (UUID): Унікальний ідентифікатор.
- title (String): Короткий заголовок.
- description (Text): Детальний опис.
- reward (Decimal): Грошова винагорода.
- latitude / longitude (Decimal): Географічні координати.

- estimated_duration_minutes (Int): Орієнтовний час виконання.
- status (Enum): OPEN, ASSIGNED, COMPLETED.
- requiredSkills (ManyToMany): Навички, необхідні для виконання цієї задачі.

4.3.2.3 Skill (Навичка)

Довідник компетенцій (наприклад, "Малярні роботи", "Доставка", "Електрика").

id (String): Код навички (наприклад, painting).

description (String): Опис для UI.

4.3.3 Сервіси та Бізнес-логіка

4.3.3.1 TasksService

Стандартний CRUD-сервіс.

Пошук поблизу (findNearby): Використовує формулу Haversine (або PostGIS у продакшені) для фільтрації задач у заданому радіусі від координат користувача.

Створення: Валідує вхідні дані та зберігає нову задачу.

4.3.3.2 PathfindingService (Генетичний Алгоритм)

Це "серце" системи оптимізації. Сервіс знаходить найефективніший порядок виконання списку задач, враховуючи відстань та час.

Етапи роботи алгоритму:

- Ініціалізація (Initial Population):
- Створюється набір випадкових маршрутів (хромосом). Кожен маршрут - це повний список обраних задач у випадковому порядку.

- Оцінка Пристосованості (Fitness Function):
- Для кожного маршруту розраховується "вартість" (Cost).
- Cost = (Час в дорозі) + (Час виконання задач).
- Чим менша вартість, тим кращий маршрут.
- Селекція (Selection): Використовується турнірний відбір (Tournament Selection). З популяції випадковим чином обираються кілька маршрутів, і найкращий з них стає "батьком" для наступного покоління.
- Схрещування (Crossover): Два батьківські маршрути обмінюються частинами шляху для створення "нащадка". Використовується впорядковане схрещування (Ordered Crossover), щоб уникнути дублювання задач у маршруті.
- Мутація (Mutation): З невеликою ймовірністю (наприклад, 1-5%) у маршруті міняються місцями дві задачі. Це запобігає застряганню алгоритму в локальних мінімумах.
- Еволюція: Процес повторюється протягом певної кількості поколінь (наприклад, 100). Найкращий маршрут останнього покоління повертається користувачу.

4.3.4 API Інтерфейс (REST)

4.3.4.1 GET /tasks/nearby

Пошук доступних задач.

Query Params:

- lat: Широта користувача.
- lng: Довгота користувача.
- radius: Радіус пошуку в км.

Response: Масив об'єктів Task.

4.3.4.2 POST /tasks

Створення нової задачі.

Body:

- Назва;
- Опис;
- Координати;
- Нагорода;
- Необхідні навички.

4.3.4.3 POST /pathfinding/optimize

Запит на оптимізацію маршруту.

Body:

- taskIds: Масив ID задач, які користувач хоче відвідати.
- startLatitude, startLongitude: Точка старту.
- userId: ID користувача (для перевірки навичок).

Response: Впорядкований масив Task (оптимальний шлях).

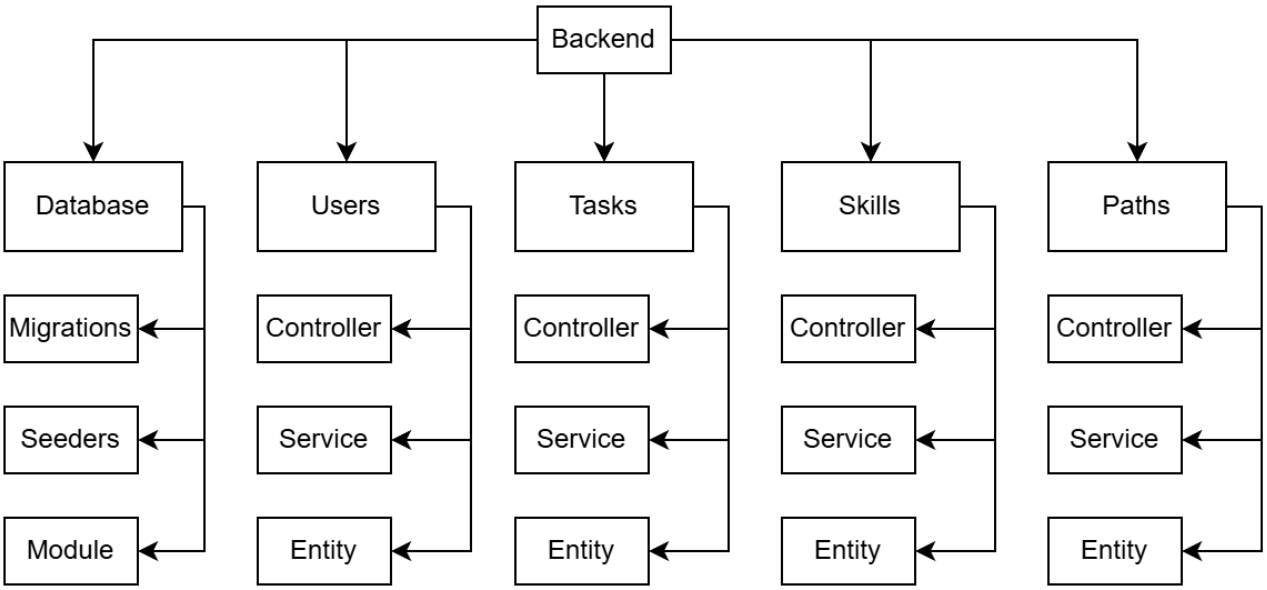

Рисунок 4.8 – Структура бекенду

Структура Tasks модулю:

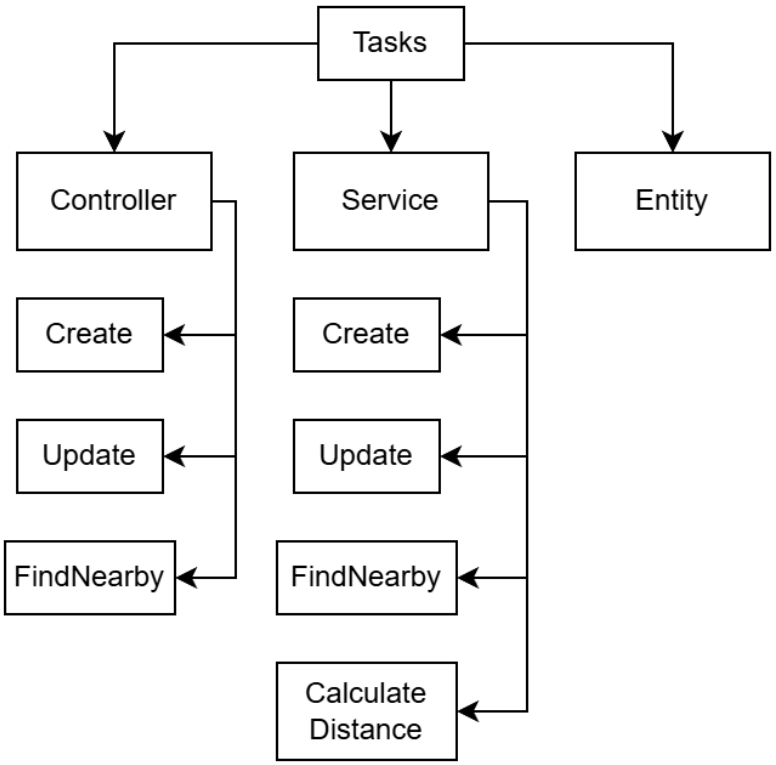

Рисунок 4.9 – Модуль Tasks

Модуль Auth:

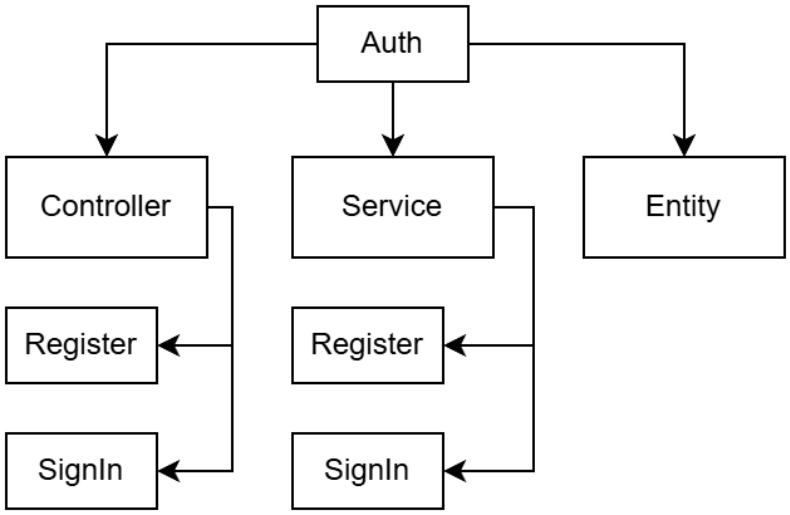

Рисунок 4.10 – Модуль Auth

Модуль Paths:

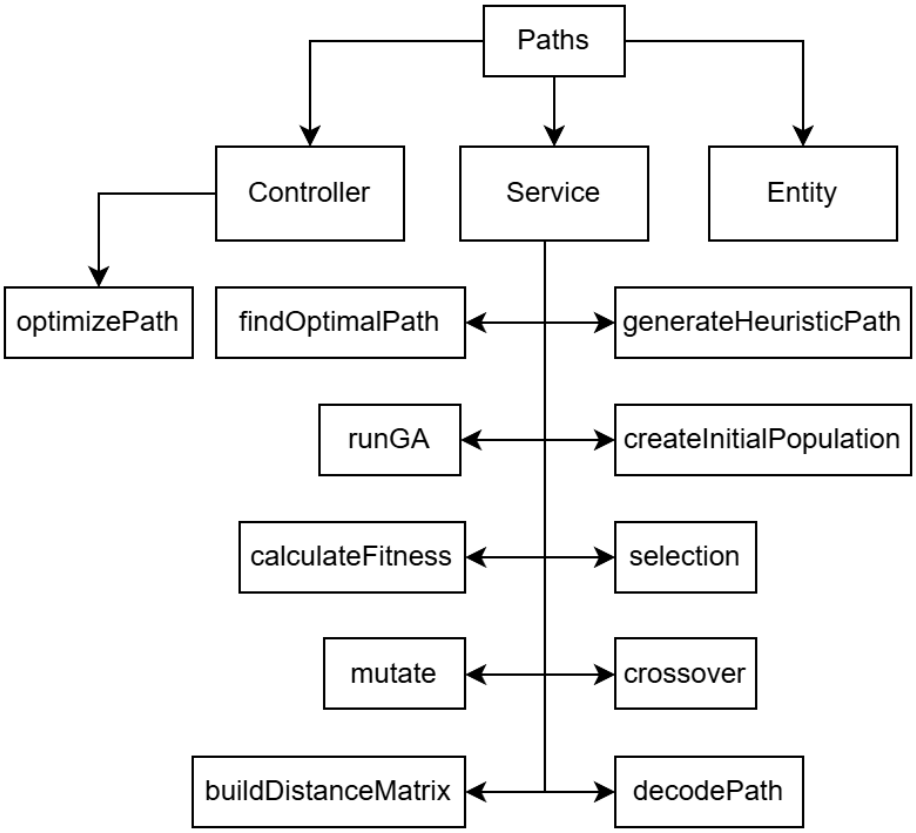

Рисунок 4.11 – Модуль Paths

4.4 Опис алгоритмів

Алгоритм оптимізації маршруту виглядає таким чином:

Крок 1. Отримати список задач;

Крок 2. Створюємо матрицю дистанцій;

Крок 3. Створюємо початкову популяцію (підалгоритм CreateInitialGeneration);

Крок 4. Рахуємо fitness для поточної популяції (підалгоритм CalculateFitness);

Крок 5. Зберігаємо 2 найкращих маршрути;

Крок 6. Решту перегенеровуємо – використовуючи підалгоритми mutate та crossover;

Крок 7. Зберігаємо нову популяцію та повертаємось до кроку 4 поки не вичерпаємо кількість генерацій;

Крок 8. Рахуємо fitness;

Крок 9. Повертаємо найкращий маршрут.

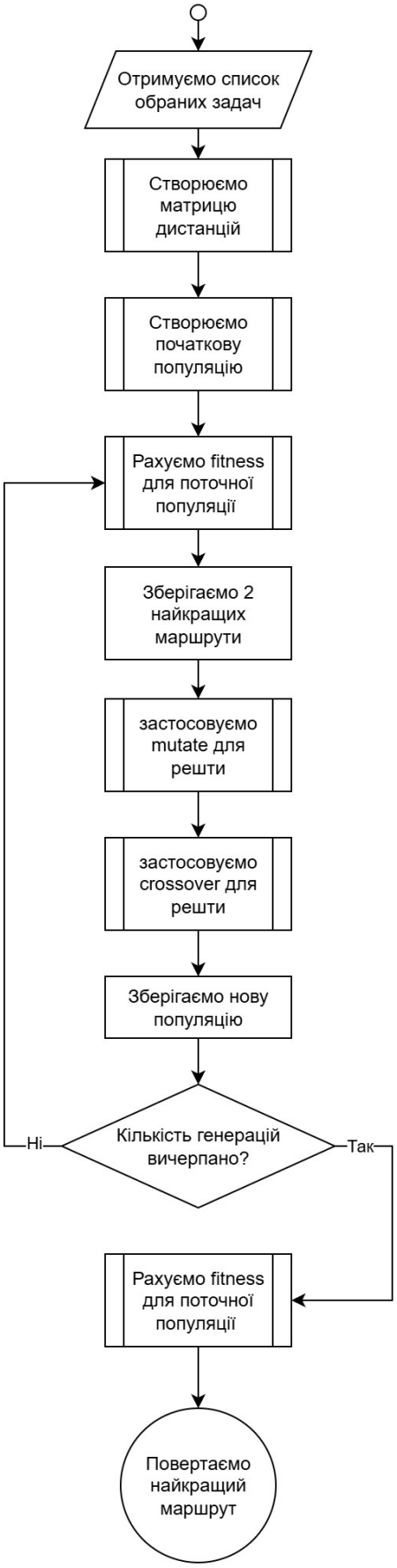

Рисунок 4.12 – Алгоритм пошуку маршруту

Підалгоритм CalculateFitness:

Крок 1. Отримуємо дистанцію між останньою і поточною задачею з матриці дистанцій;

Крок 2. Рахуємо час пересування до поточної задачі + час виконання задачі

Крок 3. Якщо час не вичерпано – додаємо винагороду до суми, перехід до кроку 1.

Крок 4. Повертаємо сумарну винагороду

Рисунок 4.13 – Алгоритм CalculateFitness

Підалгоритм CreateInitialPopulation:

Крок 1. Створюємо найкращий шлях за допомогою Жадібного алгоритму;

Крок 2. Створюємо шлях з найближчими задачами;

Крок 3. Створюємо шлях з найдорожчими задачами;

Крок 4. Генеруємо решту шляхів з випадковою вибіркою;

Крок 5. Повертаємо шляхи.

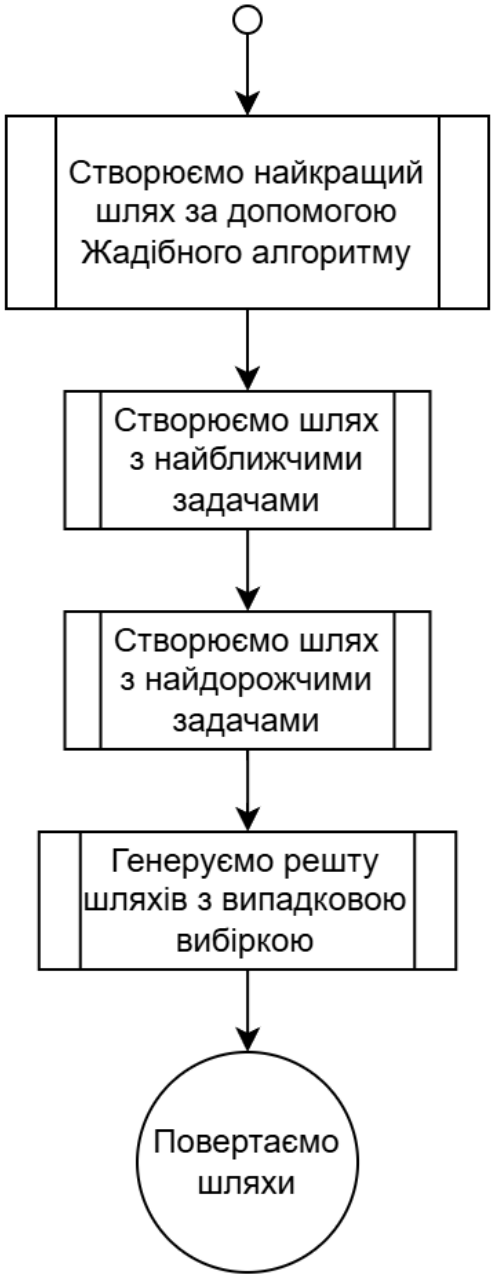

Рисунок 4.14 – Алгоритм CreateInitialPopulation

4.5 Екранні форми

Головна сторінка містить:

- Кнопку входу/виходу з акаунту;
- Кнопку пошуку найближчих задач;
- Список доступних задач;
- Форму вибору максимального часу та швидкості переміщення;
- Кнопку генерації шляху;
- Мапу, що містить:
    - Позиції задач;
    - Позицію користувача;
    - Оптимізований шлях (якщо згенеровано).

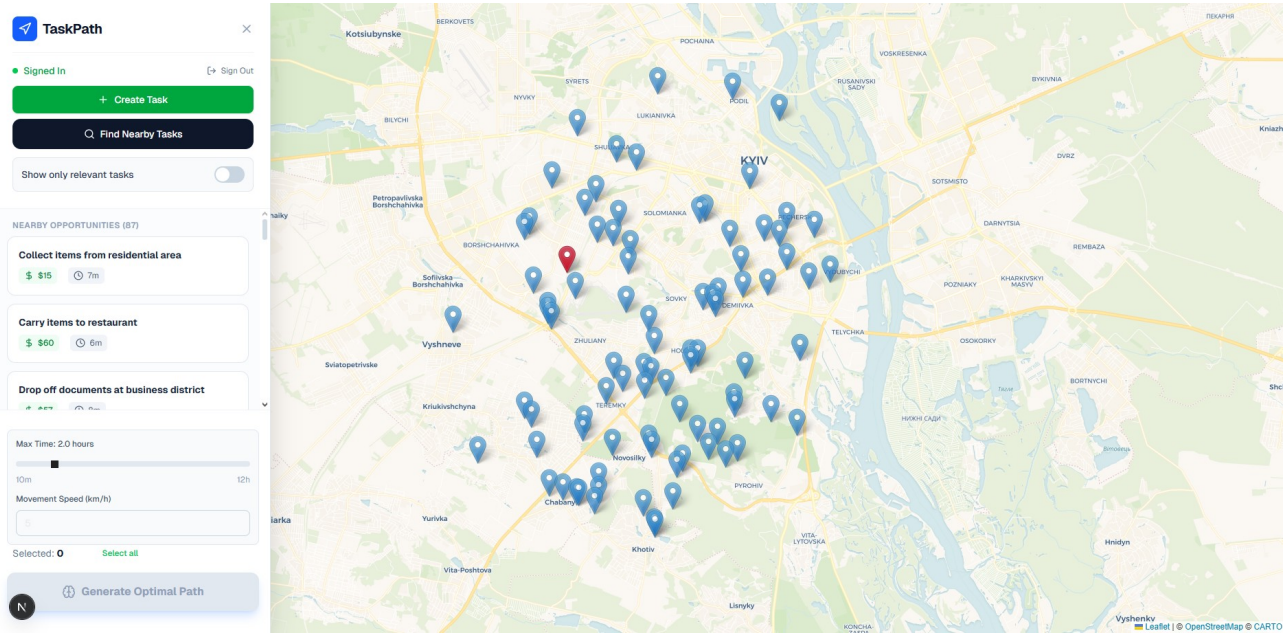

Рисунок 4.15 – Головна сторінка

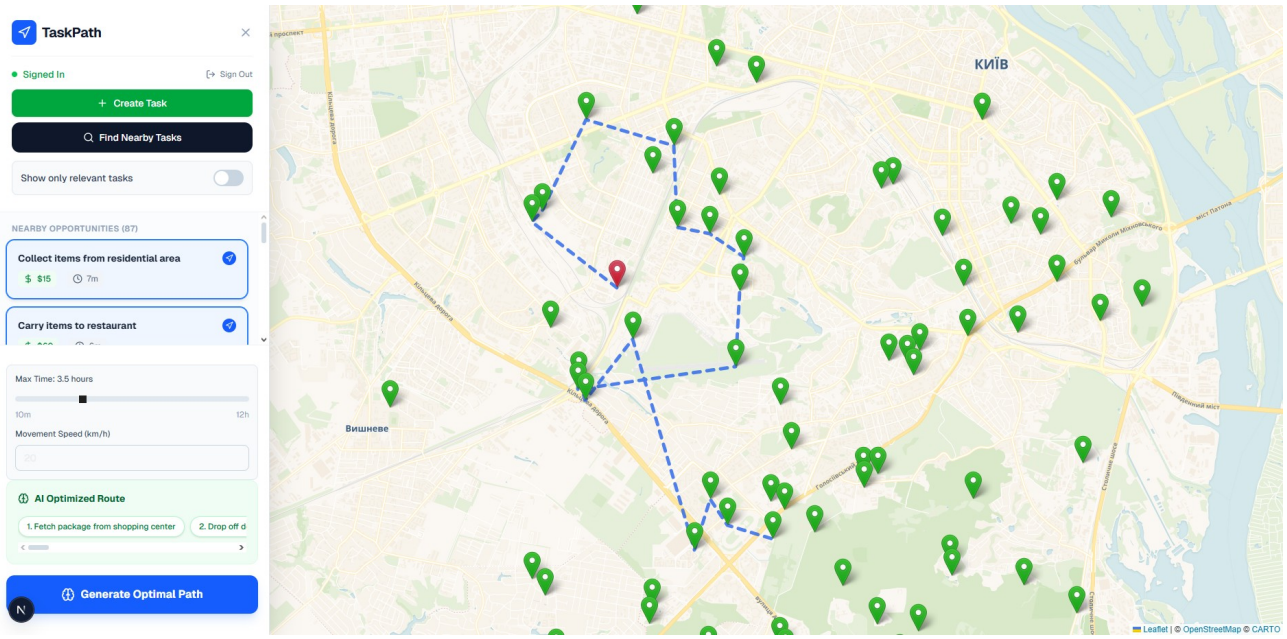

Рисунок 4.16 – Головна сторінка, з оптимальним шляхом

Сторінка входу, яка складається з полей username та password і посиланням на сторінку реєстрації.

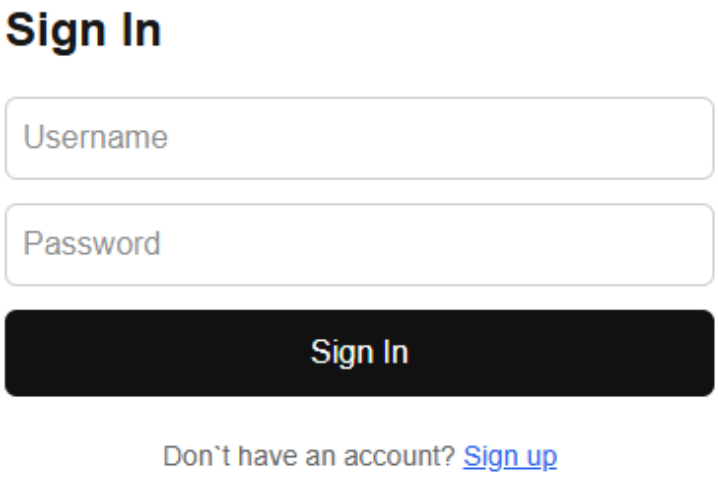

Рисунок 4.17 – Форма входу/реєстрації

Сторінка створення/редагування задачі має такі поля:

- Назва задачі
- Опис
- Локація – з можливістю вибору через мапу
- Винагорода
- Приблизний час виконання
- Обов’язкові навички

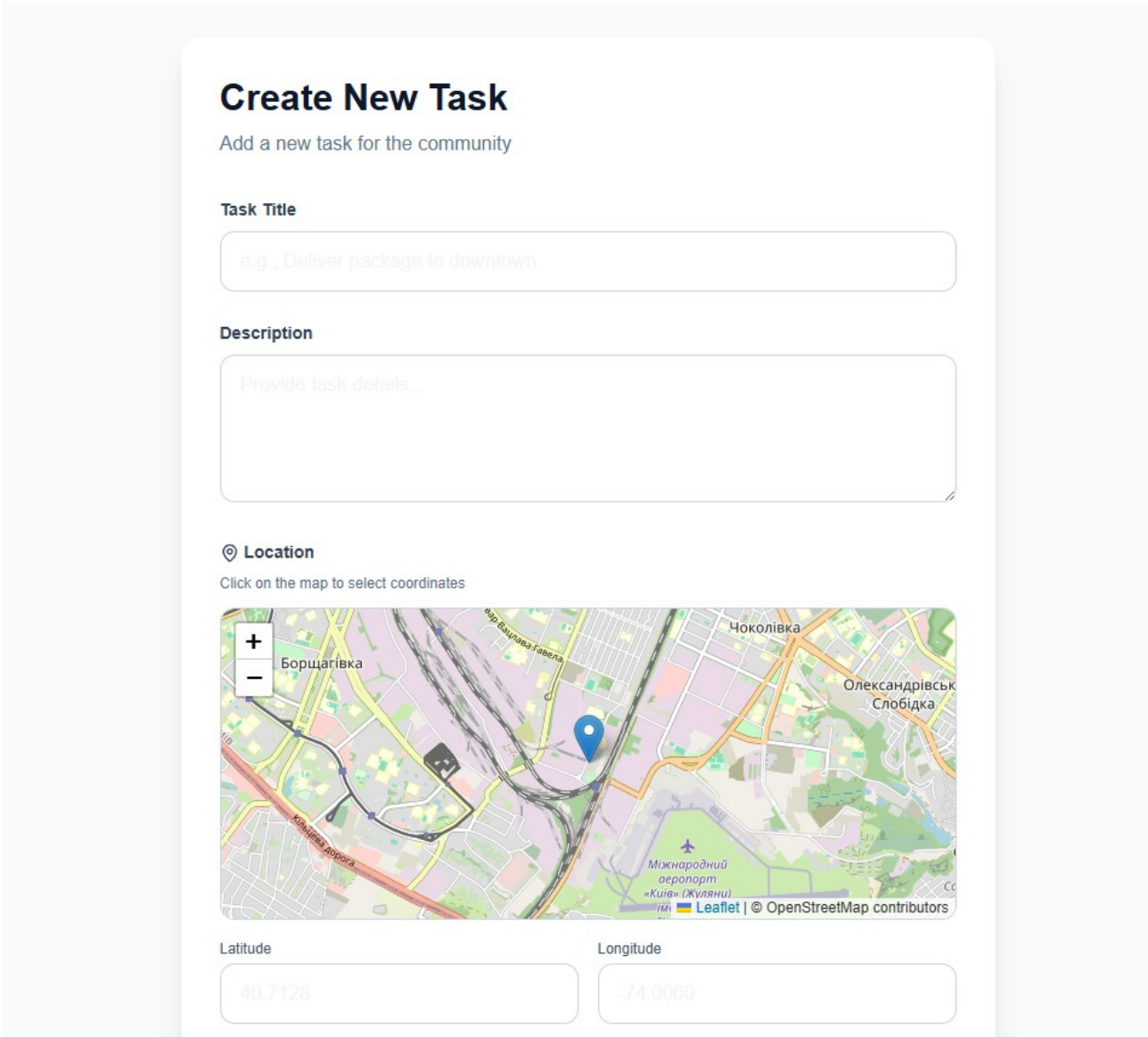

Рисунок 4.18 – Сторінка задачі

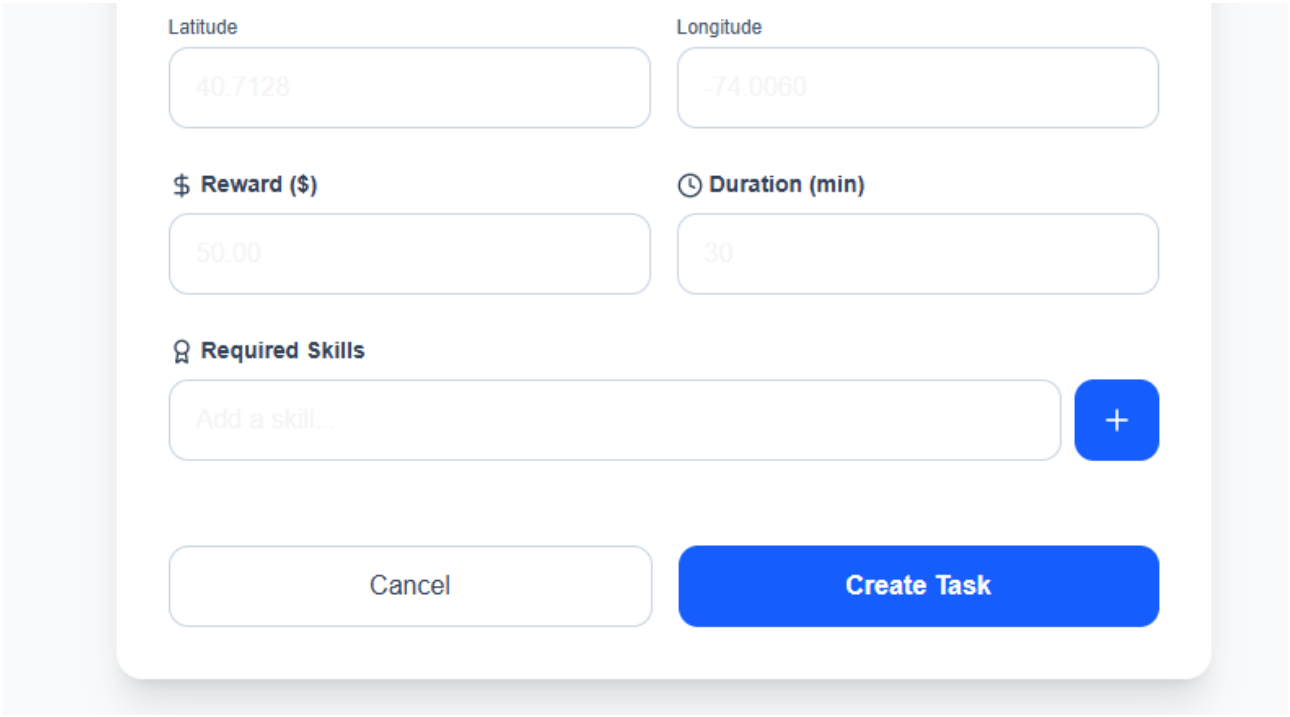

Рисунок 4.13 – Сторінка задачі ч. 2

Висновки до розділу

У цьому розділі було обґрунтовано вибір технологій, описано основні алгоритми та наведено екранні форми додатку.

ВИСНОВКИ

У результаті виконання проекту було розроблено алгоритм створення ефективних маршрутів у web-сервісах для виконаня завдань а також веб-застосунок, який його використовує.

Під час розробки було:

- Проведено аналіз предметної області та огляд існуючих аналогів;
- Досліджено та обрано найкращі алгоритми для виконання поточної задачі;
- Спроектовано архітектуру системи та структуру бази даних;
- Реалізовано алгоритм та веб-додаток з його використанням;

Створений алгоритм може бути використаний у вже існуючих системах або стати основою для нових додатків.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Статистика соціальних мереж в світі - Inclient [Електронний ресурс] // Режим доступу:

https://inclient.ru/global-social-networks-stats

2. Оптимізація маршрутів для бізнес-ефективності - Shifton [Електронний ресурс] // Режим доступу:

https://shifton.com/uk/blog/shcho-take-opytymizatsiya-marshrutu-vsebizhnyy-putivnyk

3. Larsen, A. The dynamic vehicle routing problem - Technical University of Denmark, 2000.

4. Mardešić N. Review of Stochastic Dynamic Vehicle Routing in the Urban Logistics Landscape. Mathematics, 2023.

5. Konovalenko A., Hvattum L. M. Optimizing a Dynamic Vehicle Routing Problem with Deep Reinforcement Learning. Machines, 2024.

6. Necula R., Breaban M., Raschip M. Tackling Dynamic Vehicle Routing Problem with Time Windows by means of Ant Colony System. ArXiv, 2017.

7. Abdirad M., Krishnan K., Gupta D. A Two-Stage Metaheuristic Algorithm for the Dynamic Vehicle Routing Problem in Industry 4.0. ArXiv, 2020.

8. Andersen T. Reoptimisation strategies for dynamic vehicle routing with proximity-dependent nodes. TOP Journal, 2024.

9. Horváth M., Tamási T. A general modeling and simulation framework for dynamic vehicle routing. ArXiv, 2024.

10. Deep Policy Dynamic Programming for Vehicle Routing Problems - OpenReview [Електронний ресурс] // Режим доступу:

https://openreview.net/forum?id=OAMrSPRRxJx

11. Baty L., Jungel K., Klein S. P., Parmentier A., Schiffer M. Combinatorial Optimization enriched Machine Learning to solve the Dynamic Vehicle Routing Problem with Time Windows. ArXiv, 2023.

ДОДАТОК А. КОД ПРОГРАМИ

AuthController:

```
@Controller('auth')
export class AuthController {
  constructor(private readonly authService: AuthService) {}

  @Post('register')
  register(@Body() dto: RegisterDto) {
    return this.authService.register(dto.username, dto.password);
  }

  @Post('sign-in')
  signIn(@Body() dto: SignInDto) {
    return this.authService.signIn(dto.username, dto.password);
  }
}
```

AuthService:

```
@Injectable()
export class AuthService {
  constructor(
    private usersService: UsersService,
    private readonly jwtService: JwtService,
  ) {}

  async register(
    username: string,
    password: string,
  ): Promise<{ access_token: string }> {
    const user = await this.usersService.createUser(username, password);
    const payload = { sub: user.id, username: user.username };
```

```
    const access_token = await this.jwtService.signAsync(payload);
    return { access_token };
  }

  async signIn(
    username: string,
    pass: string,
  ): Promise<{ access_token: string }> {
    const user = await this.usersService.findByUsername(username);
    if (!user) {
      throw new UnauthorizedException();
    }
    const valid = await bcrypt.compare(pass, user.password);
    if (!valid) {
      throw new UnauthorizedException();
    }
    const payload = { sub: user.id, username: user.username };
    const access_token = await this.jwtService.signAsync(payload);
    return { access_token };
  }
}
```

PathfindingController:

```
@Controller('pathfinding')
export class PathfindingController {
  constructor(private readonly pathfindingService: PathfindingService) {}

  @Post('optimize')
  @UseGuards(JwtAuthGuard)
  optimizePath(@Body() optimizePathDto: OptimizePathDto, @Request() req) {
    const userId = req.user.userId;
    return this.pathfindingService.findOptimalPath(
      { ...optimizePathDto },
```

```
      userId,
    );
  }
}
```

PathfindingService:

```
@Injectable()
export class PathfindingService {
  constructor(
    @InjectRepository(Task)
    private tasksRepository: Repository<Task>,
    @InjectRepository(User)
    private usersRepository: Repository<User>,
    private tasksService: TasksService,
  ) {}

  /**
   * Main entry point for the Genetic Algorithm
   * Finds the optimal path for a user to complete a list of tasks.
   */
  async findOptimalPath(dto: OptimizePathDto, userId: string): Promise<Task[]> {
    const user = await this.usersRepository.findOne({ where: { id: userId } });
    if (!user) throw new HttpException('User not found', HttpStatus.NOT_FOUND);

    const tasks = await this.tasksRepository.find({
      where: { id: In(dto.taskIds), status: TaskStatus.OPEN },
    });
    if (tasks.length === 0) return [];

    const startPoint: LocationPoint = {
      latitude: dto.startLatitude,
      longitude: dto.startLongitude,
```

```
    };

    const distanceMatrix = this.buildDistanceMatrix(startPoint, tasks);

    // 3. Calculate Total Possible Reward (The "Perfect Score")
    // We use this to calculate how much reward we "missed" later
    const maxPossibleReward = tasks.reduce(
      (sum, t) => sum + (t.reward || 1),
      0,
    );

    // 4. Run GA
    const populationSize = 100;
    const generations = 250;

    // Smart Initialization (Optional but recommended)
    // We seed the population with random paths AND one "Greedy" path (highest value/time
ratio)
    const population = this.createInitialPopulation(
      tasks,
      startPoint,
      populationSize,
      distanceMatrix,
    );

    const bestPath = this.runGA(
      population,
      distanceMatrix,
      dto.maxTimeMinutes,
      dto.movementSpeed,
      generations,
      0.05, // Mutation
      0.8, // Crossover
      maxPossibleReward,
```

```
  );

  return bestPath;
}

/**
 * Runs the main GA loop
 * @param population - Initial population
 * @param startPoint - The user's starting location
 * @param maxTimeMinutes - Maximum time available for completing tasks
 * @param movementSpeed - Movement speed in km/h
 * @param generations - How many generations to run
 * @param mutationRate - Chance of mutation
 * @param crossoverRate - Chance of crossover
 */
private runGA(
  population: Population,
  distanceMatrix: Map<string, number>,
  maxTimeMinutes: number,
  movementSpeed: number,
  generations: number,
  mutationRate: number,
  crossoverRate: number,
  maxPossibleReward: number,
): Chromosome {
  let currentPopulation = population;

  for (let i = 0; i < generations; i++) {
    // Calculate Fitness
    const fitnessScores = currentPopulation.map((chromo) => ({
      path: chromo,
      fitness: this.calculateFitness(
        chromo,
        distanceMatrix,
```

```
      maxTimeMinutes,
      movementSpeed,
      maxPossibleReward,
    ),
  }));

  // Sort: Lower fitness is better
  fitnessScores.sort((a, b) => a.fitness - b.fitness);

  const newPopulation: Population = [];

  // Elitism: Keep the top 2 absolute best paths found so far
  newPopulation.push(fitnessScores[0].path);
  newPopulation.push(fitnessScores[1].path);

  // Generate rest
  while (newPopulation.length < currentPopulation.length) {
    const parent1 = this.selection(fitnessScores);
    const parent2 = this.selection(fitnessScores);

    let child = parent1;
    if (Math.random() < crossoverRate)
      child = this.crossover(parent1, parent2);
    if (Math.random() < mutationRate) child = this.mutate(child);

    newPopulation.push(child);
  }
  currentPopulation = newPopulation;
}

// Final result processing
const finalScores = currentPopulation.map((chromo) => ({
  path: chromo,
  fitness: this.calculateFitness(
```

```
      chromo,
      distanceMatrix,
      maxTimeMinutes,
      movementSpeed,
      maxPossibleReward,
    ),
  }));
  finalScores.sort((a, b) => a.fitness - b.fitness);

  return this.decodePath(
    finalScores[0].path,
    distanceMatrix,
    maxTimeMinutes,
    movementSpeed,
  );
}

/**
 * Filters a path to only include tasks that fit within the time limit
 */
private filterPathByTimeLimit(
  path: Chromosome,
  startPoint: LocationPoint,
  maxTimeMinutes: number,
  movementSpeed: number,
): Chromosome {
  const filteredPath: Chromosome = [];
  let totalTime = 0;
  let currentPoint = startPoint;

  for (const task of path) {
    const travelDistance = this.tasksService.haversineDistance(
      currentPoint.latitude,
      currentPoint.longitude,
```

```
      task.latitude,
      task.longitude,
    );

    const travelTimeMinutes = (travelDistance / movementSpeed) * 60;
    const timeWithThisTask =
      totalTime + travelTimeMinutes + task.estimatedDurationMinutes;

    if (timeWithThisTask > maxTimeMinutes) {
      break;
    }

    filteredPath.push(task);
    totalTime = timeWithThisTask;
    currentPoint = task;
  }

  return filteredPath;
}

/**
 * Creates an initial population of random paths
 */
private createInitialPopulation(
  allTasks: Task[],
  startPoint: LocationPoint, // We need start point now
  populationSize: number,
  distanceMatrix: Map<string, number>,
): Population {
  const population: Population = [];
  // const geneIds = allTasks.map((t) => t.id);

  // 1. STRATEGY A: "High Density" (Reward / Distance)
  // Prioritizes tasks that are close AND valuable.
```

```
const densityPath = this.generateHeuristicPath(
  allTasks,
  startPoint,
  distanceMatrix,
  (task, currentPos, dist) => {
    const reward = task.reward || 1;
    // Avoid division by zero. If dist is 0, treat as 0.1
    const safeDist = dist < 0.1 ? 0.1 : dist;
    // The higher the score, the more likely we pick it
    return reward / safeDist;
  },
);
population.push(densityPath);

// 2. STRATEGY B: "Nearest Neighbor"
// Just goes to the closest task next. Great for finding clusters of medium tasks.
const nearestPath = this.generateHeuristicPath(
  allTasks,
  startPoint,
  distanceMatrix,
  (task, currentPos, dist) => {
    // Inverse of distance (closer = higher score)
    return 1 / (dist + 0.01);
  },
);
population.push(nearestPath);

// 3. STRATEGY C: "Pure Greed" (Reward only)
const greedyPath = [...allTasks].sort(
  (a, b) => (b.reward || 0) - (a.reward || 0),
);
population.push(greedyPath);

// 4. Fill the rest with Random (for genetic diversity)
```

```
  while (population.length < populationSize) {
    const randomPath = [...allTasks].sort(() => Math.random() - 0.5);
    population.push(randomPath);
  }

  return population;
}

private generateHeuristicPath(
  tasks: Task[],
  startPoint: LocationPoint,
  distanceMatrix: Map<string, number>,
  scoreFn: (task: Task, currentId: string, dist: number) => number,
): Task[] {
  const remainingTasks = new Set(tasks);
  const path: Task[] = [];
  let currentId = 'START'; // Corresponds to key in distanceMatrix

  while (remainingTasks.size > 0) {
    let bestTask: Task | null = null;
    let bestScore = -Infinity;

    for (const task of remainingTasks) {
      const dist = distanceMatrix.get(`${currentId}-${task.id}`) || 10000;
      const score = scoreFn(task, currentId, dist);

      if (score > bestScore) {
        bestScore = score;
        bestTask = task;
      }
    }

    if (bestTask) {
      path.push(bestTask);
```

```
      remainingTasks.delete(bestTask);
      currentId = bestTask.id;
    } else {
      break;
    }
  }
  return path;
}

/**
 * Fitness function: Calculates the total cost of a path.
 * A lower score is better.
 * Cost = Total Travel Time + Total Task Duration
 * If path exceeds maxTimeMinutes, adds heavy penalty
 */
private calculateFitness(
  path: Chromosome,
  distanceMatrix: Map<string, number>,
  maxTimeMinutes: number,
  movementSpeed: number,
  maxPossibleReward: number,
): number {
  let currentTime = 0;
  let collectedReward = 0;
  let lastId = 'START';

  for (const task of path) {
    const distKm = distanceMatrix.get(`${lastId}-${task.id}`) || 0;
    const travelTime = (distKm / movementSpeed) * 60;
    const nextTime = currentTime + travelTime + task.estimatedDurationMinutes;

    // If we can fit this task in the day, take the reward
    if (nextTime <= maxTimeMinutes) {
      currentTime = nextTime;
```

```
      collectedReward += task.reward || 1; // <--- Accumulate Value
      lastId = task.id;
    } else {
      // Stop counting once we run out of time
      break;
    }
  }

  // Weighting Logic:
  // We want High Reward first, Low Time second.
  // Penalty = (Money we left on the table) * HugeFactor
  const uncollectedReward = maxPossibleReward - collectedReward;

  // The Factor must be larger than the max possible time (minutes)
  // so that saving 1 min never outweighs gaining 1 reward point.
  const rewardPriorityFactor = 10000;

  return uncollectedReward * rewardPriorityFactor + currentTime;
}

/**
 * Selects a parent from the population based on fitness.
 * (Using simple Tournament Selection)
 */
private selection(
  fitnessScores: { path: Chromosome; fitness: number }[],
): Chromosome {
  // Get 2 random individuals and return the one with better fitness
  const k = 2;
  let best = fitnessScores[Math.floor(Math.random() * fitnessScores.length)];
  for (let i = 1; i < k; i++) {
    const next =
      fitnessScores[Math.floor(Math.random() * fitnessScores.length)];
    if (next.fitness < best.fitness) {
```

```
      best = next;
    }
  }
  return best.path;
}

/**
 * Creates a new path (child) from two parents.
 * (Using simple Ordered Crossover)
 */
private crossover(parent1: Chromosome, parent2: Chromosome): Chromosome {
  const len = parent1.length;
  const start = Math.floor(Math.random() * len);
  const end = Math.floor(Math.random() * (len - start) + start);

  const child: Chromosome = new Array(len).fill(null);
  // Create a Set for O(1) lookups of what is already in the child
  const childSet = new Set<string>();

  // Copy segment
  for (let i = start; i <= end; i++) {
    child[i] = parent1[i];
    childSet.add(parent1[i].id);
  }

  // Fill remaining
  let p2Index = 0;
  for (let i = 0; i < len; i++) {
    if (child[i] === null) {
      while (childSet.has(parent2[p2Index].id)) {
        p2Index++;
      }
      child[i] = parent2[p2Index];
      childSet.add(parent2[p2Index].id);
```

```
    }
  }
  return child;
}

private mutate(path: Chromosome): Chromosome {
  const newPath = [...path];
  const i = Math.floor(Math.random() * newPath.length);
  let j = Math.floor(Math.random() * newPath.length);
  while (i === j) j = Math.floor(Math.random() * newPath.length);

  [newPath[i], newPath[j]] = [newPath[j], newPath[i]];
  return newPath;
}

private buildDistanceMatrix(
  start: LocationPoint,
  tasks: Task[],
): Map<string, number> {
  const matrix = new Map<string, number>();

  // Helper to store key
  const setDist = (id1: string, id2: string, dist: number) => {
    matrix.set(`${id1}-${id2}`, dist);
  };

  // 1. Distance from Start to every Task
  tasks.forEach((task) => {
    const dist = this.tasksService.haversineDistance(
      start.latitude,
      start.longitude,
      task.latitude,
      task.longitude,
    );
```

```
    setDist('START', task.id, dist);
  });

  // 2. Distance between every Task pair
  for (let i = 0; i < tasks.length; i++) {
    for (let j = 0; j < tasks.length; j++) {
      if (i === j) continue;
      const dist = this.tasksService.haversineDistance(
        tasks[i].latitude,
        tasks[i].longitude,
        tasks[j].latitude,
        tasks[j].longitude,
      );
      setDist(tasks[i].id, tasks[j].id, dist);
    }
  }
  return matrix;
}

private decodePath(
  path: Chromosome,
  distanceMatrix: Map<string, number>,
  maxTimeMinutes: number,
  movementSpeed: number,
): Task[] {
  const result: Task[] = [];
  let currentTime = 0;
  let lastId = 'START';

  for (const task of path) {
    const distKm = distanceMatrix.get(`${lastId}-${task.id}`) || 0;
    const travelTime = (distKm / movementSpeed) * 60;
    const nextTime = currentTime + travelTime + task.estimatedDurationMinutes;
```

```
      if (nextTime <= maxTimeMinutes) {
        result.push(task);
        currentTime = nextTime;
        lastId = task.id;
      } else {
        break;
      }
    }
    return result;
  }
}
```

TasksController:

```
@Controller('tasks')
export class TasksController {
  constructor(private readonly tasksService: TasksService) {}

  @Post()
  @UseGuards(JwtAuthGuard)
  create(@Body() createTaskDto: CreateTaskDto, @Request() req) {
    const userId = req.user.userId;
    return this.tasksService.create(createTaskDto, userId);
  }

  @Get('nearby')
  @UseInterceptors(ClassSerializerInterceptor)
  @UseGuards(OptionalJwtAuthGuard)
  findNearby(@Query() query: TaskQueryDto, @Request() req) {
    const userId = req.user?.userId;
    return this.tasksService.findNearby(query, userId);
  }
}
```

TasksService:

```
@Injectable()
export class TasksService {
 constructor(
  @InjectRepository(Task)
  private tasksRepository: Repository<Task>,
  @InjectRepository(User)
  private usersRepository: Repository<User>,
  @InjectRepository(Skill)
  private skillsRepository: Repository<Skill>,
 ) {}

 public async create(
  createTaskDto: CreateTaskDto,
  userId: string,
 ): Promise<Task> {
  const user = await this.usersRepository.findOne({ where: { id: userId } });
  if (!user) throw new HttpException('User not found', 404);

  const skills = await this.skillsRepository.findBy({
   title: In(createTaskDto.requiredSkills),
  });
  const newSkillTitles = createTaskDto.requiredSkills.filter(
   (title) => !skills.find((skill) => skill.title === title),
  );

  for (const title of newSkillTitles) {
   const newSkill = this.skillsRepository.create({ title });
   await this.skillsRepository.save(newSkill);
   skills.push(newSkill);
  }

  const task = this.tasksRepository.create({
```

```
    ...createTaskDto,
    status: TaskStatus.OPEN,
    createdBy: user,
    requiredSkills: skills,
  });

  return this.tasksRepository.save(task);
}

public async findAll(): Promise<Task[]> {
  return this.tasksRepository.find();
}

public async findNearby(
  query: TaskQueryDto,
  userId?: string,
): Promise<Task[]> {
  let allTasks = await this.tasksRepository.find({
    relations: ['requiredSkills'],
  });
  if (!allTasks || allTasks.length === 0) {
    return [];
  }
  if (!query.ignoreUserSkills && userId) {
    const user = await this.usersRepository.findOne({
      where: { id: userId },
      relations: ['skills'],
    });
    if (user) {
      const userSkillIds = user.skills?.map((s) => s.id) || [];
      // Filter tasks by user's skills
      const eligibleTasks = allTasks.filter((task) =>
        task.requiredSkills.every((skill) => userSkillIds.includes(skill.id)),
      );
```

```
      allTasks = eligibleTasks;
    }
  }

  // In a real app, you'd do this with a PostGIS query for efficiency
  return allTasks.filter((task) => {
    const distance = this.haversineDistance(
      query.lat,
      query.lng,
      task.latitude,
      task.longitude,
    );
    return distance <= query.radius && task.status === TaskStatus.OPEN;
  });
}

/**
 * Calculates distance between two coordinates using Haversine formula
 * @returns Distance in kilometers
 */
public haversineDistance(
  lat1: number,
  lon1: number,
  lat2: number,
  lon2: number,
): number {
  const R = 6371; // Earth's radius in km
  const dLat = (lat2 - lat1) * (Math.PI / 180);
  const dLon = (lon2 - lon1) * (Math.PI / 180);
  const a =
    Math.sin(dLat / 2) * Math.sin(dLat / 2) +
    Math.cos(lat1 * (Math.PI / 180)) *
      Math.cos(lat2 * (Math.PI / 180)) *
      Math.sin(dLon / 2) *
```

```
      Math.sin(dLon / 2);
    const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
    return R * c;
  }
}
```

UsersService:

```
@Injectable()
export class UsersService {
  constructor(
    @InjectRepository(User)
    private usersRepository: Repository<User>,
  ) {}

  async findOne(id: string): Promise<User | undefined> {
    return this.usersRepository.findOneOrFail({ where: { id } });
  }

  async findByUsername(username: string): Promise<User | null> {
    return this.usersRepository.findOne({ where: { username } });
  }

  async createUser(username: string, password: string): Promise<User> {
    const existing = await this.findByUsername(username);
    if (existing) {
      throw new ConflictException('Username already taken');
    }
    const hashed = await bcrypt.hash(password, 10);
    const user = this.usersRepository.create({ username, password: hashed });
    return this.usersRepository.save(user);
  }
}
```

Frontend:

MapComponent:

```
export default function Map() {
        const MOCK_USER_START_POS = { lat: 50.41420606283159, lng: 30.430412292480472 }; // Kyiv

    const ICONS = {
        default: createCustomIcon('https://raw.githubusercontent.com/pointhi/leaflet-color-markers/master/img/marker-icon-2x-blue.png'),
        selected: createCustomIcon('https://raw.githubusercontent.com/pointhi/leaflet-color-markers/master/img/marker-icon-2x-green.png'),
        start: createCustomIcon('https://raw.githubusercontent.com/pointhi/leaflet-color-markers/master/img/marker-icon-2x-red.png'),
    };

    const map = useMap();
    const tasks = useAppSelector(selectTasks);

    const { selectedTaskIds, optimizedPath } = useAppSelector((state) => state.path);

    const pathCoordinates = useMemo(() => {
      const coords = optimizedPath?.map(t => ({ lat: t.latitude, lng: t.longitude })) || [];
        if (coords.length > 0) coords.unshift({ lat: MOCK_USER_START_POS.lat, lng: MOCK_USER_START_POS.lng });
      return coords;
    }, [MOCK_USER_START_POS.lat, MOCK_USER_START_POS.lng, optimizedPath]);

    // Fix Icons on mount
    useEffect(() => { fixLeafletIcons(); }, []);

    // Auto-fit bounds
    useEffect(() => {
```

```
            if (pathCoordinates.length > 0) {
              const bounds = L.latLngBounds(pathCoordinates);
              map.fitBounds(bounds, { padding: [100, 100], maxZoom: 15 });
            } else if (tasks?.length > 0) {
              const bounds = L.latLngBounds(tasks.map(t => [t.latitude, t.longitude]));
              // Add start pos to bounds so we don't lose the user
                                                    bounds.extend([MOCK_USER_START_POS.lat,
MOCK_USER_START_POS.lng]);
              map.fitBounds(bounds, { padding: [80, 80], maxZoom: 14 });
            }
          }, [tasks, pathCoordinates, map]);

          return (
            <>
              <TileLayer
                url="https://{s}.basemaps.cartocdn.com/rastertiles/voyager/{z}/{x}/{y}{r}.png"
                                                                attribution='&copy;      <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>              &copy;              <a
href="https://carto.com/attributions">CARTO</a>'
              />

              {/* Start Position */}
                                          <Marker    position={[MOCK_USER_START_POS.lat,
MOCK_USER_START_POS.lng]} icon={ICONS.start}>
                <Popup className="font-sans">
                  <div className="text-center">
                    <div className="font-bold text-slate-800">Start Location</div>
                    <div className="text-xs text-slate-500">You are here</div>
                  </div>
                </Popup>
              </Marker>

              {/* Task Markers */}
              {tasks?.map((task) => {
```

```
                const isSelected = selectedTaskIds.includes(task.id);
                const isOptimized = optimizedPath?.find(t => t.id === task.id);

                return (
                  <Marker
                    key={task.id}
                    position={[task.latitude, task.longitude]}
                    icon={isSelected ? ICONS.selected : ICONS.default}
                    opacity={isSelected || isOptimized ? 1 : 0.7}
                  >
                    <Popup offset={[0, -20]}>
                      <div className="p-1 min-w-[150px]">
                        <h3 className="font-bold text-slate-800 mb-1">{task.title}</h3>
                        <div className="flex gap-2 mb-2">
                          <span className="text-xs bg-green-100 text-green-700 px-1.5 py-0.5
rounded font-medium">${task.reward}</span>
                            <span className="text-xs bg-slate-100 text-slate-600 px-1.5 py-0.5
rounded font-medium">{task.estimatedDurationMinutes}m</span>
                        </div>
                        <p className="text-xs text-slate-500 leading-snug mb-2">
                          Complete this task to earn rewards.
                        </p>
                      </div>
                    </Popup>
                  </Marker>
                );
              })}

              {/* Route Line */}
              {pathCoordinates.length > 0 && (
                <Polyline
                  positions={pathCoordinates}
                  pathOptions={{
                    color: '#2563eb', // Blue-600
```

```
                    weight: 5,
                    opacity: 0.8,
                    dashArray: '10, 10',
                    lineCap: 'round'
                  }}
                />
              )}
            </>
          );
        }
```

MapSelector:

```
        interface MapSelectorProps {
          latitude: number
          longitude: number
          onLocationSelect: (lat: number, lng: number) => void
        }

        export default function MapSelector({ latitude, longitude, onLocationSelect }:
MapSelectorProps) {
          const mapRef = useRef<L.Map | null>(null)
          const markerRef = useRef<L.Marker | null>(null)
          const containerRef = useRef<HTMLDivElement>(null)
          const [isClient, setIsClient] = useState(false)
          const prevCoordsRef = useRef({ latitude, longitude })

          useEffect(() => {
            setIsClient(true)
          }, [])

          useEffect(() => {
            if (!isClient || !containerRef.current) return
```

```
        // Initialize map only once
        if (!mapRef.current) {
          mapRef.current = L.map(containerRef.current).setView([latitude, longitude], 13)

          L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
            attribution: '© OpenStreetMap contributors',
          }).addTo(mapRef.current)

          // Add click handler
          mapRef.current.on('click', (e: L.LeafletMouseEvent) => {
            const { lat, lng } = e.latlng
            onLocationSelect(lat, lng)
          })

          // Fix icon issue
          delete (L.Icon.Default.prototype as any)._getIconUrl
          L.Icon.Default.mergeOptions({
            iconRetinaUrl: 'https://cdnjs.cloudflare.com/ajax/libs/leaflet/1.9.4/images/marker-icon-
2x.png',
            iconUrl: 'https://cdnjs.cloudflare.com/ajax/libs/leaflet/1.9.4/images/marker-icon.png',
                shadowUrl:    'https://cdnjs.cloudflare.com/ajax/libs/leaflet/1.9.4/images/marker-
shadow.png',
          })

          // Create initial marker
          if (latitude && longitude) {
            markerRef.current = L.marker([latitude, longitude]).addTo(mapRef.current)
          }
        }

        return () => {
          if (mapRef.current) {
            mapRef.current.remove()
            mapRef.current = null
```

```
      markerRef.current = null
    }
   }
  }, [isClient])

  // Only update marker when coordinates actually change meaningfully
  useEffect(() => {
   if (!mapRef.current || !latitude || !longitude) return

   const latChanged = Math.abs(prevCoordsRef.current.latitude - latitude) > 0.00001
   const lngChanged = Math.abs(prevCoordsRef.current.longitude - longitude) > 0.00001

   if (latChanged || lngChanged) {
    prevCoordsRef.current = { latitude, longitude }

    if (markerRef.current) {
     markerRef.current.setLatLng([latitude, longitude])
    } else {
     markerRef.current = L.marker([latitude, longitude]).addTo(mapRef.current)
    }
    mapRef.current.setView([latitude, longitude], mapRef.current.getZoom())
   }
  }, [latitude, longitude])

  if (!isClient) {
   return <div className="w-full h-64 bg-slate-100 rounded-xl animate-pulse" />
  }

  return <div ref={containerRef} className="w-full h-64" />
}
```

TaskCard:

```
type TaskCardProps = {
  task: Task
  isSelected: boolean;
  onSelect: (taskId: string) => void;
}

export default function TaskCard({ task, isSelected, onSelect }: TaskCardProps) {

  return (
    <div className={`
      relative p-4 rounded-xl border transition-all duration-200 cursor-pointer group
      ${isSelected
        ? 'bg-blue-50 border-blue-500 shadow-md ring-1 ring-blue-500'
        : 'bg-white border-slate-200 hover:border-blue-300 hover:shadow-sm'
      }
    `}
      onClick={() => onSelect(task.id)}
    >
      <div className="flex justify-between items-start mb-2">
        <h3 className="font-bold text-slate-800 text-sm">{task.title}</h3>
        {isSelected && (
          <span className="bg-blue-600 text-white p-1 rounded-full">
            <Navigation className="w-3 h-3" />
          </span>
        )}
      </div>

      <div className="flex items-center gap-4 text-xs text-slate-600">
          <div className="flex items-center gap-1.5 font-medium text-green-700 bg-green-50
px-2 py-1 rounded-md">
          <DollarSign className="w-3.5 h-3.5" />
          <span>${task.reward}</span>
        </div>
        <div className="flex items-center gap-1.5 bg-slate-100 px-2 py-1 rounded-md">
```

```
            <Clock className="w-3.5 h-3.5" />
            <span>{task.estimatedDurationMinutes}m</span>
          </div>
        </div>
      </div>
    );
  }
```

Sidebar:

```
export default function Sidebar() {
        const MOCK_USER_START_POS = { lat: 50.41420606283159, lng: 30.430412292480472 }; // Kyiv
    const dispatch = useAppDispatch();
    const router = useRouter();
    const tasks = useAppSelector(selectTasks);
    const tasksIsLoading = useAppSelector(selectIsLoading);

     const { selectedTaskIds, optimizedPath, status: pathStatus } = useAppSelector((state) => state.path);
    const [isOpen, setIsOpen] = useState(true);
    const [authenticated, setAuthenticated] = useState(false);
    const [ignoreUserSkills, setIgnoreUserSkills] = useState(false);
    const [maxTimeMinutes, setMaxTimeMinutes] = useState(120); // Default 2 hours
    const [movementSpeed, setMovementSpeed] = useState(5); // Default 5 km/h

    useEffect(() => {
      setAuthenticated(isAuthenticated());
    }, []);

    const handleSignOut = () => {
      clearToken();
      setAuthenticated(false);
```

```
        router.push('/login');
      };

      // Initial Fetch
      useEffect(() => {
        dispatch(clearPath());
                dispatch(fetchNearbyTasks({ ...MOCK_USER_START_POS, radius: 10, ignoreUserSkills }));
      }, [dispatch, ignoreUserSkills]);

      const handleOptimize = () => {
        if (selectedTaskIds.length < 1) return;
        dispatch(fetchOptimalPath({
          taskIds: selectedTaskIds,
          startLatitude: MOCK_USER_START_POS.lat,
          startLongitude: MOCK_USER_START_POS.lng,
          maxTimeMinutes,
          movementSpeed
        }));
      };

      if (!isOpen) {
        return (
          <button
            onClick={() => setIsOpen(true)}
              className="absolute top-6 left-6 z-1000 bg-white text-slate-700 p-3 rounded-xl shadow-xl hover:bg-slate-50 transition-all border border-slate-200"
            >
            <Layers className="w-6 h-6" />
          </button>
        );
      }

      return (
```

```
<div className="absolute top-0 left-0 h-full w-full sm:w-[400px] bg-white z-1000 shadow-2xl flex flex-col border-r border-slate-200 animate-in slide-in-from-left duration-300">
{/* Header */}
<div className="p-6 border-b border-slate-100 bg-white">
<div className="flex justify-between items-center mb-4">
<div className="flex items-center gap-2">
<div className="bg-blue-600 p-2 rounded-lg">
<Navigation className="w-5 h-5 text-white" />
</div>
<h1 className="text-xl font-bold text-slate-900 tracking-tight">TaskPath</h1>
</div>
<button onClick={() => setIsOpen(false)} className="text-slate-400 hover:text-slate-600">
<X className="w-5 h-5" />
</button>
</div>
{/* Auth Status */}
<div className="mt-4 pt-4 border-t border-slate-100 mb-2">
{authenticated ? (
<div className="space-y-3">
<div className="flex items-center justify-between">
<div className="flex items-center gap-2 text-sm text-green-600">
<div className="w-2 h-2 bg-green-500 rounded-full animate-pulse"></div>
<span className="font-medium">Signed In</span>
</div>
<button
onClick={handleSignOut}
className="flex items-center gap-1.5 text-xs text-slate-500 hover:text-red-600 transition-colors"
>
<LogOut className="w-3.5 h-3.5" />
Sign Out
```

```
                    </button>
                  </div>
                  <Link
                    href="/create-task"
                      className="w-full flex items-center justify-center gap-2 text-sm font-medium text-white bg-green-600 hover:bg-green-700 py-2.5 px-4 rounded-lg transition-all shadow-sm"
                  >
                    <Plus className="w-4 h-4" />
                    Create Task
                  </Link>
                </div>
              ) : (
                <div className="flex gap-2">
                  <Link
                    href="/login"
                     className="flex-1 flex items-center justify-center gap-1.5 text-xs font-medium text-slate-600 hover:text-blue-600 py-2 px-3 rounded-lg hover:bg-blue-50 transition-all border border-slate-200"
                  >
                    <LogIn className="w-3.5 h-3.5" />
                    Sign In
                  </Link>
                  <Link
                    href="/register"
                     className="flex-1 flex items-center justify-center gap-1.5 text-xs font-medium text-white bg-blue-600 hover:bg-blue-700 py-2 px-3 rounded-lg transition-all"
                  >
                    <UserPlus className="w-3.5 h-3.5" />
                    Sign Up
                  </Link>
                </div>
              )}
            </div>
```

```
                <button
                  onClick={() => {
                    dispatch(clearPath());
                      dispatch(fetchNearbyTasks({ ...MOCK_USER_START_POS, radius: 10, ignoreUserSkills }));
                  }}
                    className="w-full flex justify-center items-center gap-2 bg-slate-900 text-white text-sm font-medium py-3 px-4 rounded-xl hover:bg-slate-800 transition-all"
                >
                    {tasksIsLoading ? <Loader2 className="w-4 h-4 animate-spin" /> : <Search className="w-4 h-4" />}
                  Find Nearby Tasks
                </button>
                {/* Ignore User Skills Toggle */}
                {authenticated && (
                    <div className="mt-3 flex items-center justify-between p-3 bg-slate-50 rounded-lg border border-slate-200">
                      <label htmlFor="ignoreSkills" className="text-sm text-slate-700 font-medium cursor-pointer">
                        Show only relevant tasks
                      </label>
                      <button
                        id="ignoreSkills"
                        onClick={() => setIgnoreUserSkills(!ignoreUserSkills)}
                            className={`relative w-11 h-6 rounded-full transition-colors ${!ignoreUserSkills ? 'bg-blue-600' : 'bg-slate-300'
                          }`}
                      >
                        <span
                            className={`absolute top-0.5 left-0.5 w-5 h-5 bg-white rounded-full transition-transform ${!ignoreUserSkills ? 'translate-x-5' : 'translate-x-0'
                          }`}
                        />
                      </button>
```

```
        </div>
      )}




    </div>

    {/* Task List */}
    <div className="flex-1 overflow-y-auto p-4 space-y-3 bg-slate-50/50">
      <div className="text-xs font-semibold text-slate-400 uppercase tracking-wider
px-2 mb-2">
        Nearby Opportunities ({tasks.length})
      </div>

      {tasks.map(task => (
        <TaskCard
          key={task.id}
          task={task}
          isSelected={selectedTaskIds.includes(task.id)}
          onSelect={(taskId) => dispatch(toggleTaskSelection(taskId))}
        />
      ))}

      {tasks.length === 0 && !tasksIsLoading && (
        <div className="text-center py-10 text-slate-400">
          <MapPin className="w-10 h-10 mx-auto mb-2 opacity-50" />
          <p>No tasks found in this area</p>
        </div>
      )}
    </div>

    {/* Footer / Optimization Panel */}
    <div className="p-4 bg-white border-t border-slate-100 pb-8">
```

```
                {/* Time Range and Movement Speed */}
                 <div className="mt-3 space-y-3 p-3 bg-slate-50 rounded-lg border border-slate-200">
                   <div>
                        <label htmlFor="timeRange" className="text-xs text-slate-600 font-medium block mb-2">
                          Max Time: {maxTimeMinutes < 60 ? `${maxTimeMinutes} mins` : `${(maxTimeMinutes / 60).toFixed(1)} hours`}
                        </label>
                        <input
                          id="timeRange"
                          type="range"
                          min="10"
                          max="720"
                          step="10"
                          value={maxTimeMinutes}
                          onChange={(e) => setMaxTimeMinutes(Number(e.target.value))}
                           className="w-full h-2 bg-slate-200 rounded-lg appearance-none cursor-pointer accent-blue-600"
                        />
                        <div className="flex justify-between text-xs text-slate-400 mt-1">
                          <span>10m</span>
                          <span>12h</span>
                        </div>
                      </div>

                      <div>
                          <label htmlFor="movementSpeed" className="text-xs text-slate-600 font-medium block mb-2">
                            Movement Speed (km/h)
                          </label>
                          <input
                            id="movementSpeed"
                            type="number"
```

```
                        min="1"
                        max="50"
                        step="0.5"
                        value={movementSpeed}
                        onChange={(e) => setMovementSpeed(Number(e.target.value))}
                         className="w-full px-3 py-2 text-sm border border-slate-300 rounded-lg focus:outline-none focus:ring-2 focus:ring-blue-500 focus:border-transparent"
                        placeholder="15"
                      />
                    </div>
                  </div>

                  {optimizedPath.length > 0 ? (
                    <div className="mb-4 bg-green-50 rounded-xl p-4 border border-green-100">
                        <h4 className="text-sm font-bold text-green-800 mb-2 flex items-center gap-2">
                        <Brain className="w-4 h-4" /> AI Optimized Route
                      </h4>
                      <div className="flex gap-2 overflow-x-auto py-2">
                        {optimizedPath.map((t, i) => (
                            <div key={t.id} className="shrink-0 text-xs bg-white text-green-700 px-3 py-1.5 rounded-full border border-green-200 font-medium shadow-sm">
                              {i + 1}. {t.title}
                            </div>
                        ))}
                      </div>
                    </div>
                  ) : (
                    <div className="flex justify-between items-center mb-4 px-2">
                        <span className="text-sm text-slate-500">Selected: <b className="text-slate-900">{selectedTaskIds.length}</b></span>
                      <button
                        onClick={() => dispatch(selectAllTasks(tasks.map(task => task.id)))}
                        className="text-xs text-green-500 hover:text-green-700 font-medium"
```

```
        >
          Select all
        </button>
        {selectedTaskIds.length > 0 ? (
          <button
            onClick={() => dispatch(clearPath())}
            className="text-xs text-red-500 hover:text-red-700 font-medium"
          >
            Clear Selection
          </button>
        ) : <div className="opacity-0 cursor-default">Clear Selection</div>}
      </div>
    )}

    <button
      onClick={handleOptimize}
      disabled={selectedTaskIds.length === 0 || pathStatus === 'loading'}
      className={`w-full flex justify-center items-center gap-2 font-bold
            py-4 px-6 rounded-xl shadow-lg shadow-blue-500/20 transition-all
        ${selectedTaskIds.length > 0
          ? 'bg-blue-600 hover:bg-blue-700 text-white translate-y-0'
          : 'bg-slate-200 text-slate-400 cursor-not-allowed'
        }
            `}
    >
      {pathStatus === 'loading' ? (
        <Loader2 className="w-5 h-5 animate-spin" />
      ) : (
        <>
          <Brain className="w-5 h-5" />
          Generate Optimal Path
        </>
      )}
    </button>
```

```
            </div>
          </div>
        );
      }
```

CreateTaskPage:

```
      const MapSelector = dynamic(() => import('../components/Map/MapSelector'), {
        ssr: false,
        loading: () => <div className="w-full h-64 bg-slate-100 rounded-xl animate-pulse flex items-center justify-center text-slate-400">Loading map...</div>,
      })

      export default function CreateTaskPage() {
        const router = useRouter()
        const [loading, setLoading] = useState(false)
        const [error, setError] = useState<string | null>(null)

        const [formData, setFormData] = useState({
          title: '',
          description: '',
          reward: '',
          estimatedDurationMinutes: '',
          requiredSkills: [] as string[],
        })

        const [latitude, setLatitude] = useState<number | null>(null)
        const [longitude, setLongitude] = useState<number | null>(null)

        const [skillInput, setSkillInput] = useState('')

        const handleSubmit = async (e: React.FormEvent) => {
          e.preventDefault()
```

```
  setError(null)
  setLoading(true)

  try {
    await api.createTask({
      title: formData.title,
      description: formData.description,
      reward: parseFloat(formData.reward),
      latitude: latitude !== null ? parseFloat(latitude.toPrecision(6)) : 0,
      longitude: longitude !== null ? parseFloat(longitude.toPrecision(6)) : 0,
      estimatedDurationMinutes: parseInt(formData.estimatedDurationMinutes, 10),
      requiredSkills: formData.requiredSkills,
    })
    router.push('/')
  } catch (err: any) {
    setError(err?.response?.data?.message || 'Failed to create task')
  } finally {
    setLoading(false)
  }
}

const addSkill = () => {
  if (skillInput.trim() && !formData.requiredSkills.includes(skillInput.trim())) {
    setFormData({
      ...formData,
      requiredSkills: [...formData.requiredSkills, skillInput.trim()],
    })
    setSkillInput('')
  }
}

const removeSkill = (skill: string) => {
  setFormData({
    ...formData,
```

```
    requiredSkills: formData.requiredSkills.filter(s => s !== skill),
   })
  }

  const handleMapClick = (lat: number, lng: number) => {
   setLatitude(lat)
   setLongitude(lng)
  }

  return (
   <div className="min-h-screen bg-slate-50 py-8 px-4">
    <div className="max-w-2xl mx-auto">
     <div className="bg-white rounded-2xl shadow-xl p-8">
      <h1 className="text-3xl font-bold text-slate-900 mb-2">Create New Task</h1>
      <p className="text-slate-500 mb-8">Add a new task for the community</p>

      <form onSubmit={handleSubmit} className="space-y-6">
       {/* Title */}
       <div>
        <label className="block text-sm font-semibold text-slate-700 mb-2">
         Task Title
        </label>
        <input
         type="text"
         value={formData.title}
         onChange={(e) => setFormData({ ...formData, title: e.target.value })}
         required
             className="w-full px-4 py-3 border border-slate-300 rounded-xl focus:ring-2
focus:ring-blue-500 focus:border-transparent"
         placeholder="e.g., Deliver package to downtown"
        />
       </div>

       {/* Description */}
```

```
                <div>
                  <label className="block text-sm font-semibold text-slate-700 mb-2">
                    Description
                  </label>
                  <textarea
                    value={formData.description}
                    onChange={(e) => setFormData({ ...formData, description: e.target.value })}
                    rows={4}
                        className="w-full px-4 py-3 border border-slate-300 rounded-xl focus:ring-2
focus:ring-blue-500 focus:border-transparent"
                    placeholder="Provide task details..."
                  />
                </div>

                {/* Location */}
                <div>
                  <label className="block text-sm font-semibold text-slate-700 mb-2">
                    <MapPin className="w-4 h-4 inline mr-1" />
                    Location
                  </label>
                        <p className="text-xs text-slate-500 mb-3">Click on the map to select
coordinates</p>

                  <div className="mb-4 rounded-xl overflow-hidden border border-slate-300">
                    <MapSelector
                      latitude={latitude !== null ? latitude : 50.41420606283159}
                      longitude={longitude !== null ? longitude : 30.430412292480472}
                      onLocationSelect={handleMapClick}
                    />
                  </div>

                  <div className="grid grid-cols-2 gap-4">
                    <div>
                      <label className="block text-xs font-medium text-slate-600 mb-1">
```

```
                Latitude
              </label>
              <input
                type="number"
                step="any"
                value={latitude !== null ? latitude : ''}
                    onChange={(e) => setLatitude(e.target.value ? parseFloat(e.target.value) :
null)}
                required
                  className="w-full px-4 py-3 border border-slate-300 rounded-xl focus:ring-2
focus:ring-blue-500 focus:border-transparent"
                placeholder="40.7128"
              />
            </div>
            <div>
              <label className="block text-xs font-medium text-slate-600 mb-1">
                Longitude
              </label>
              <input
                type="number"
                step="any"
                value={longitude !== null ? longitude : ''}
                    onChange={(e) => setLongitude(e.target.value ? parseFloat(e.target.value) :
null)}
                required
                  className="w-full px-4 py-3 border border-slate-300 rounded-xl focus:ring-2
focus:ring-blue-500 focus:border-transparent"
                placeholder="-74.0060"
              />
            </div>
          </div>
        </div>

        {/* Reward and Duration */}
```

```
                <div className="grid grid-cols-2 gap-4">
                  <div>
                    <label className="block text-sm font-semibold text-slate-700 mb-2">
                      <DollarSign className="w-4 h-4 inline mr-1" />
                      Reward ($)
                    </label>
                    <input
                      type="number"
                      step="0.01"
                      value={formData.reward}
                      onChange={(e) => setFormData({ ...formData, reward: e.target.value })}
                      required
                      min="0"
                        className="w-full px-4 py-3 border border-slate-300 rounded-xl focus:ring-2
focus:ring-blue-500 focus:border-transparent"
                      placeholder="50.00"
                    />
                  </div>
                  <div>
                    <label className="block text-sm font-semibold text-slate-700 mb-2">
                      <Clock className="w-4 h-4 inline mr-1" />
                      Duration (min)
                    </label>
                    <input
                      type="number"
                      value={formData.estimatedDurationMinutes}
                            onChange={(e) => setFormData({ ...formData, estimatedDurationMinutes:
e.target.value })}
                      required
                      min="1"
                        className="w-full px-4 py-3 border border-slate-300 rounded-xl focus:ring-2
focus:ring-blue-500 focus:border-transparent"
                      placeholder="30"
                    />
```

```
          </div>
        </div>

        {/* Skills */}
        <div>
          <label className="block text-sm font-semibold text-slate-700 mb-2">
            <Award className="w-4 h-4 inline mr-1" />
            Required Skills
          </label>
          <div className="flex gap-2 mb-3">
            <input
              type="text"
              value={skillInput}
              onChange={(e) => setSkillInput(e.target.value)}
              onKeyPress={(e) => e.key === 'Enter' && (e.preventDefault(), addSkill())}
                 className="flex-1 px-4 py-3 border border-slate-300 rounded-xl focus:ring-2
focus:ring-blue-500 focus:border-transparent"
              placeholder="Add a skill..."
            />
            <button
              type="button"
              onClick={addSkill}
                  className="px-4 py-3 bg-blue-600 text-white rounded-xl hover:bg-blue-700
transition-colors"
            >
              <Plus className="w-5 h-5" />
            </button>
          </div>
          <div className="flex flex-wrap gap-2">
            {formData.requiredSkills.map((skill) => (
              <span
                key={skill}
                  className="inline-flex items-center gap-1 px-3 py-1.5 bg-blue-100 text-blue-
700 rounded-full text-sm font-medium"
```

```
              >
                {skill}
                <button
                  type="button"
                  onClick={() => removeSkill(skill)}
                  className="hover:text-blue-900"
                >
                  <X className="w-3.5 h-3.5" />
                </button>
              </span>
            ))}
          </div>
        </div>

        {error && (
            <div className="p-4 bg-red-50 border border-red-200 rounded-xl text-red-700
text-sm">
            {error}
          </div>
        )}

        {/* Buttons */}
        <div className="flex gap-4 pt-4">
          <button
            type="button"
            onClick={() => router.push('/')}
              className="flex-1 px-6 py-3 border border-slate-300 text-slate-700 rounded-xl
hover:bg-slate-50 transition-colors font-medium"
          >
            Cancel
          </button>
          <button
            type="submit"
            disabled={loading}
```

```
                    className="flex-1 px-6 py-3 bg-blue-600 text-white rounded-xl hover:bg-blue-
700 transition-colors font-bold disabled:opacity-50 disabled:cursor-not-allowed"
                  >
                    {loading ? 'Creating...' : 'Create Task'}
                  </button>
                </div>
              </form>
            </div>
          </div>
        </div>
      )
    }
```

LoginPage:

```
    export default function LoginPage() {
      const router = useRouter()
      const [username, setUsername] = useState('')
      const [password, setPassword] = useState('')
      const [loading, setLoading] = useState(false)
      const [error, setError] = useState<string | null>(null)

      const onSubmit = async (e: React.FormEvent) => {
        e.preventDefault()
        setError(null)
        setLoading(true)
        try {
          await signIn(username, password)
          router.push('/')
          router.refresh()
          // eslint-disable-next-line @typescript-eslint/no-explicit-any
        } catch (err: any) {
          setError(err?.message || 'Login failed')
```

```
    } finally {
      setLoading(false)
    }
  }

  return (
    <div style={{ maxWidth: 360, margin: '64px auto' }}>
      <h1 style={{ fontSize: 24, fontWeight: 600, marginBottom: 16 }}>Sign In</h1>
      <form onSubmit={onSubmit} style={{ display: 'grid', gap: 12 }}>
        <input
          placeholder='Username'
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          autoComplete='username'
          required
          style={{ padding: 8, border: '1px solid #ccc', borderRadius: 6 }}
        />
        <input
          type='password'
          placeholder='Password'
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          autoComplete='current-password'
          required
          style={{ padding: 8, border: '1px solid #ccc', borderRadius: 6 }}
        />
        {error && <div style={{ color: 'crimson' }}>{error}</div>}
          <button type='submit' disabled={loading} style={{ padding: 10, borderRadius: 6,
background: '#111', color: '#fff' }}>
          {loading ? 'Signing in…' : 'Sign In'}
        </button>
        <div style={{ textAlign: 'center', marginTop: 8, fontSize: 14, color: '#666' }}>
          Don`t have an account?{' '}
          <a href='/register' style={{ color: '#2563eb', textDecoration: 'underline' }}>
```

```
          Sign up
        </a>
      </div>
    </form>
  </div>
 )
}
```

RegisterPage

```
export default function RegisterPage() {
 const router = useRouter()
 const [username, setUsername] = useState('')
 const [password, setPassword] = useState('')
 const [loading, setLoading] = useState(false)
 const [error, setError] = useState<string | null>(null)

 const onSubmit = async (e: React.FormEvent) => {
  e.preventDefault()
  setError(null)
  setLoading(true)
  try {
   await register(username, password)
   router.push('/')
   router.refresh()
   // eslint-disable-next-line @typescript-eslint/no-explicit-any
  } catch (err: any) {
   setError(err?.message || 'Registration failed')
  } finally {
   setLoading(false)
  }
 }
```

```
  return (
    <div style={{ maxWidth: 360, margin: '64px auto' }}>
      <h1 style={{ fontSize: 24, fontWeight: 600, marginBottom: 16 }}>Register</h1>
      <form onSubmit={onSubmit} style={{ display: 'grid', gap: 12 }}>
        <input
          placeholder='Username'
          value={username}
          onChange={(e) => setUsername(e.target.value)}
          autoComplete='username'
          required
          style={{ padding: 8, border: '1px solid #ccc', borderRadius: 6 }}
        />
        <input
          type='password'
          placeholder='Password'
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          autoComplete='new-password'
          required
          style={{ padding: 8, border: '1px solid #ccc', borderRadius: 6 }}
        />
        {error && <div style={{ color: 'crimson' }}>{error}</div>}
          <button type='submit' disabled={loading} style={{ padding: 10, borderRadius: 6,
background: '#111', color: '#fff' }}>
          {loading ? 'Registering…' : 'Create Account'}
        </button>
        <div style={{ textAlign: 'center', marginTop: 8, fontSize: 14, color: '#666' }}>
          Already have an account?{' '}
          <a href='/login' style={{ color: '#2563eb', textDecoration: 'underline' }}>
            Sign in
          </a>
        </div>
      </form>
    </div>
```

```
  )
 }
```

Auth:

```
 import axios from "axios";

 export type AuthResponse = { access_token: string };

 const API_BASE_URL =
        process.env.NEXT_PUBLIC_API_URL || process.env.API_BASE_URL ||
"http://localhost:4000";

 const authAxios = axios.create({
  baseURL: API_BASE_URL,
 });

 export function getToken(): string | null {
  if (typeof window === "undefined") return null;
  return localStorage.getItem("token");
 }

 export function setToken(token: string) {
  if (typeof window === "undefined") return;
  localStorage.setItem("token", token);
 }

 export function clearToken() {
  if (typeof window === "undefined") return;
  localStorage.removeItem("token");
 }

 export function isAuthenticated(): boolean {
  return !!getToken();
```

```
}

export async function signIn(username: string, password: string): Promise<AuthResponse>
{
  try {
    const response = await authAxios.post<AuthResponse>("/auth/sign-in", {
      username,
      password,
    });
    setToken(response.data.access_token);
    return response.data;
  } catch (error) {
    throw new Error("Invalid credentials");
  }
}

export async function register(username: string, password: string): Promise<AuthResponse>
{
  try {
    const response = await authAxios.post<AuthResponse>("/auth/register", {
      username,
      password,
    });
    setToken(response.data.access_token);
    return response.data;
  // eslint-disable-next-line @typescript-eslint/no-explicit-any
  } catch (error: any) {
    const message = error.response?.data?.message || "Registration failed";
    throw new Error(message);
  }
}
```

Api

```
const API_BASE_URL =
        process.env.NEXT_PUBLIC_API_URL || process.env.API_BASE_URL ||
"http://localhost:4000";

const axiosInstance = axios.create({
  baseURL: API_BASE_URL,
});

// Add token to requests automatically
axiosInstance.interceptors.request.use((config) => {
  const token = getToken();
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
});

// Clear token on 401 responses
axiosInstance.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      clearToken();
      // Optionally redirect to login
      if (typeof window !== 'undefined') {
        window.location.href = '/login';
      }
    }
    return Promise.reject(error);
  }
);
```

```
const api = {
  fetchNearbyTasks: async ({ lat, lng, radius, ignoreUserSkills }: GetNearbyTasksDto) => {
    try {
      const response = await axiosInstance.get(
        `/tasks/nearby?lat=${lat}&lng=${lng}&radius=${radius}&ignoreUserSkills=${ignoreUserSkills}`
      );
      return response.data;
    } catch (e) {
      console.warn("API Error (Using Mock Data):", e);
      return [];
    }
  },

  fetchOptimalPath: async ({
    taskIds,
    startLatitude,
    startLongitude,
    maxTimeMinutes,
    movementSpeed,
  }: GetOptimalPathDto) => {
    try {
      const response = await axiosInstance.post(`/pathfinding/optimize`, {
        taskIds,
        startLatitude,
        startLongitude,
        maxTimeMinutes,
        movementSpeed,
      });
      return response.data;
    } catch (e) {
      console.warn("API Error (Returning unoptimized):", e);
      return [];
    }
```

```
  },

  createTask: async (taskData: CreateTaskDto) => {
    try {
      const response = await axiosInstance.post(`/tasks`, taskData);
      return response.data;
    } catch (e) {
      console.error("API Error (Task creation failed):", e);
      throw e;
    }
  },
};
export default api;
```

PathSlice:

```
export interface PathSliceState {
  selectedTaskIds: string[];
  optimizedPath: Task[];
  status: 'idle' | 'loading' | 'failed';
}

const initialState: PathSliceState = {
  selectedTaskIds: [],
  optimizedPath: [],
  status: 'idle',
};

export const pathSlice = createAppSlice({
  name: "path",
  initialState,
  reducers: (create) => ({
    toggleTaskSelection: create.reducer((state, action: PayloadAction<string>) => {
```

```
    const taskId = action.payload;
    const index = state.selectedTaskIds.indexOf(taskId);
    if (index >= 0) state.selectedTaskIds.splice(index, 1);
    else state.selectedTaskIds.push(taskId);
   }),
   selectAllTasks: create.reducer((state, action: PayloadAction<string[]>) => {
    state.selectedTaskIds = action.payload;
   }),
   clearPath: create.reducer((state) => {
    state.selectedTaskIds = [];
    state.optimizedPath = [];
    state.status = 'idle';
   }),
   fetchOptimalPath: create.asyncThunk(
    async (params: GetOptimalPathDto) => {
     const response = await api.fetchOptimalPath(params);
     return response;
    },
    {
     pending: (state) => {
      state.status = 'loading';
     },
     fulfilled: (state, action) => {
      state.status = 'idle';
      state.optimizedPath = action.payload;
     },
     rejected: (state) => {
      state.status = 'failed';
     },
    },
   ),
  }),
 selectors: {
  selectOptimizedPath: (path) => path.optimizedPath,
```

```
    selectStatus: (path) => path.status,
  },
});

export const { fetchOptimalPath, toggleTaskSelection, clearPath, selectAllTasks } =
  pathSlice.actions;

export const { selectOptimizedPath, selectStatus } = pathSlice.selectors;
```

TasksSlice:

```
export interface TasksSliceState {
  tasks: Task[];
  isLoading: boolean;
  error: string | null;
}

const initialState: TasksSliceState = {
  tasks: [],
  isLoading: false,
  error: null,
};

export const tasksSlice = createAppSlice({
  name: "tasks",
  initialState,
  reducers: (create) => ({
    fetchNearbyTasks: create.asyncThunk(
      async (params: GetNearbyTasksDto) => {
        const response = await api.fetchNearbyTasks(params);
        return response;
      },
      {
```

```
      pending: (state) => {
        state.isLoading = true;
      },
      fulfilled: (state, action) => {
        state.isLoading = false;
        state.tasks = action.payload ?? [];
      },
      rejected: (state) => {
        state.error = "failed";
        state.isLoading = false;
      },
    },
  ),
 }),
 selectors: {
  selectTasks: (tasks) => tasks.tasks,
  selectIsLoading: (tasks) => tasks.isLoading,
  selectError: (tasks) => tasks.error,
 },
});

export const { fetchNearbyTasks } =
 tasksSlice.actions;

export const { selectTasks, selectIsLoading, selectError } = tasksSlice.selectors;
```