

ПРИВАТНЕ АКЦІОНЕРНЕ ТОВАРИСТВО «ВИШИЙ НАВЧАЛЬНИЙ ЗАКЛАД
«МІЖРЕГІОНАЛЬНА АКАДЕМІЯ УПРАВЛІННЯ ПЕРСОНАЛОМ»

Інститут комп'ютерно-інформаційних технологій

Кафедра комп'ютерних інформаційних систем і технологій

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

Картографічна система пошуку оптимальних маршрутів

Виконав студент групи Т16-9-20-М1ППі(1.6д)

Горшкова Я.В.

Керівник: к.т.н., доцент

Чолишкіна О.Г.

Рецензент: к.т.н., доцент

Шибасва Н.О.

Допущено до захисту

Завідувач кафедри

д.т.н., професор Кавун С.В.

(підпис)

Київ, 2022

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи: 118 с., 31 рис., 4 додатки, 25 джерел.

НАВІГАЦІЯ, МУРАШИНІ КОЛОНІЇ, ЛОГІСТИКА, ТРАНСПОРТНІ СИСТЕМИ, KOTLIN, FIREBASE.

Об'єктом дослідження є алгоритми визначення ефективних маршрутів з використанням орієнтованих графів та побудови прогнозу часу пересування та ефективності обраного транспортного засобу.

Метою кваліфікаційної роботи є розробка системи з пошуку оптимального маршруту для картографічного мобільного додатку на навігації. Розроблене рішення має використовувати орієнтовану графову структуру та декілька функціональних методів з оптимізації побудови транспортних маршрутів, згідно обраного транспортного рішення.

Основні інструменти розробки безпосередньо використані для реалізації: Kotlin, Firestore Firebase.

Розроблена інформаційна система дозволяє будувати оптимальні маршрути між заданими точками та планувати оптимальний час досягнення результату.

NAVIGATION, ANT COLONIES, LOGISTICS, TRANSPORT SYSTEMS, KOTLIN, FIREBASE.

The object of research is algorithms for determining effective routes using oriented graphs and building a forecast of travel time and efficiency of the selected vehicle.

The purpose of the qualification work is to develop a system for finding the optimal route for a cartographic mobile application for navigation. The developed solution should use a oriented graph structure and several functional methods to optimize the construction of transport routes, according to the chosen transport solution.

The main development tools are directly used for implementation: kotlin, firestore firebase.

The developed information system allows you to build optimal routes between specified points and plan the optimal time to achieve the result.

ЗМІСТ

ВСТУП.....	5
1 ОБҐРУНТУВАННЯ ЕФЕКТИВНОСТІ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ З ОПТИМІЗАЦІЇ АЛГОРИТМУ ПОШУКУ МАРШРУТУ.....	8
1.1 Обґрунтування актуальності застосування сучасних рішень з пошуку маршрутів.....	8
1.2 Аналіз методів вирішення транспортних задач.....	12
1.3 Впровадження алгоритмічних компонентів пошуку оптимальних маршрутів.....	15
1.4 Опис технічних і програмних засобів розробки.....	19
1.5 Постанова мети та завдання	25
1.6 Висновок до першого розділу	26
2 РОЗРОБКА ЛОГІКИ РОБОТИ АЛГОРИТМУ ПОШУКУ ОПТИМАЛЬНОГО МАРШРУТУ	28
2.1 Застосування алгоритмів мурашиних колоній для розробки інформаційних систем	28
2.2 Впровадження алгоритмів повного перебору для пошуку оптимальних маршрутів.....	35
2.3 Впровадження гібридного алгоритму мурашиної колонії до логістичної галузі.....	39
2.4 Висновок до другого розділу.....	43
3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ З ПОШУКУ ОПТИМАЛЬНОГО МАРШРУТУ ДЛЯ ЛОГІСТИЧНИХ ОРГАНІЗАЦІЙ	44
3.1 Проектування розробки програмної системи	44
3.2 Опис інтерфейсу розробленої інформаційної системи	57
3.3 Практичне використання розробленої системи.....	69
3.4 Висновок до третього розділу	74
ВИСНОВКИ.....	75

	4
ПЕРЕЛІК ПОСИЛАНЬ	77
ДОДАТОК А	79
ДОДАТОК Б	103
ДОДАТОК В	110
ДОДАТОК Г	114

ВСТУП

З точки зору логістики, транспортні перевезення вантажів являють собою складний процес, який включає в себе планування, організацію і виконання доставки продукції, підготовку партій відправлень до перевезення, організацію і проведення вантажно-розвантажувальних робіт, розфасовку, пакування, складування товару, страхування перевезень, іноді – митні послуги.

Важливими складовими організації транспортних перевезень є раціональний вибір транспортних засобів, найбільш повне використання вантажопідйомності транспортних засобів за допомогою правильної завантаження, дотримання технологій при веденні вантажно-розвантажувальних робіт, розрахунок необхідних запасів товару для забезпечення безперебійної навантаження. У разі, коли підприємство-виробник користується послугами перевізника, до елементів транспортного процесу додаються оформлення необхідних перевізних документів, укладення договору на перевезення, розрахунок за послуги перевізника. Практична реалізація всіх перелічених складових транспортного процесу вимагає відповідних грошових і тимчасових витрат.

Ефективне застосування логістичних принципів передбачає широке використання інформаційних технологій, розроблення та впровадження систем імітаційного моделювання. Оптимальна організація окремих етапів не є умовою оптимальності всього процесу. Однак уміння визначати оптимально можливі результати на будь-якому етапі дозволяє забезпечити дослідників і виробників важливим критерієм для оцінки впливу цього етапу на ефективність усього процесу. Тому проблема оптимізації кожного етапу транспортного процесу завжди буде залишатися актуальною, але не завжди – визначальною.

Актуальність теми. Застосування різних спеціалізованих алгоритмів оптимізаційного пошуку маршрутів при вирішенні транспортних задач є актуальною та сучасною темою з впровадження сучасних інформаційних технологій в середовище малого та середнього бізнесу.

Зв'язок роботи з програмами наукових досліджень кафедри ПІ. Розробка картографічних, навігаційних та оптимізаційних систем тісно пов'язана з вивченням автоматизованих систем, мобільними застосунками та іншими програмними рішеннями. Розроблена інформаційна система має спеціалізоване призначення та може використовуватися як програмний засіб прямого застосування.

Мета і задачі дослідження. Метою кваліфікаційної роботи є розробка системи з пошуку оптимального маршруту для картографічного мобільного додатку на навігації. Розроблене рішення має використовувати орієнтовану графову структуру та декілька функціональних методів з оптимізації побудови транспортних маршрутів, згідно обраного транспортного рішення. Інформаційна система має будувати маршрути згідно доданих координатних точок користувача та оброблювати більше ніж 2 такі точки на мапі. Також слід запровадити автоматизовані засоби геопозиціонування з використанням апаратної частини користувацького пристрою. Інформаційна система має формувати оптимальний прогноз часу на пересування користувача згідно сформованого маршруту та надавати альтернативні маршрути за умови можливості їх формування.

Завданням кваліфікаційної роботи є розробка сучасного мобільного програмного рішення з пошуку оптимального маршруту між заданими точками з оптимізаційною логікою на базі алгоритму мурашиної колонії.

Вимогами до кваліфікаційної роботи є:

- виконати огляд сучасних рішень при вирішенні транспортних задач;
- виконати оцінку ефективності застосування транспортних задач в картографічних системах;
- виконати оцінку ефективності алгоритмів побудови оптимальних маршрутів;
- провести порівняльний аналіз засобів розробки картографічних систем;
- спроектувати архітектуру програмної системи;
- виконати опис алгоритмічної складової розробленої інформаційної системи;

- виконати опис функціональних можливостей розробленого програмного рішення;
- провести аналіз ефективності розробленого рішення;
виконати розробку програмного засобу.

Об’єкт дослідження. Об’єктом дослідження є алгоритми визначення ефективних маршрутів з використанням орієнтованих графів та побудови прогнозу часу пересування та ефективності обраного транспортного засобу.

Методи дослідження. При розробці кваліфікаційної роботи, були використані методи мурашиної колонії для пошуку оптимальних маршрутів, а також система оптимізації маршруту з використанням системи аналізу функціонального графу.

Практичне значення одержаних результатів. Розроблена інформаційна система може використовуватися як самостійний програмний модуль, або інтегруватися до інших програмних засобів як окремий функціональний компонент.

Публікації. Розроблена системи висвітлені в двох науково-практичних конференціях, за результатами яких сформовані збірники тез.

1 ОБҐРУНТУВАННЯ ЕФЕКТИВНОСТІ РОЗРОБКИ ІНФОРМАЦІЙНОЇ СИСТЕМИ З ОПТИМІЗАЦІЇ АЛГОРИТМУ ПОШУКУ МАРШРУТУ

1.1 Обґрунтування актуальності застосування сучасних рішень з пошуку маршрутів

Розвиток міст визначається взаємозв'язком зростання міських територій, чисельності населення, планування та розміщення різних функціональних зон. Рівень розвитку міст залежить від організації та технічних можливостей транспортних систем. Створення ефективної транспортної системи міста – це складна комплексна проблема, що включає низку завдань, різних за значимістю, складністю та трудомісткістю, серед яких визначення маршрутів руху міського пасажирського транспорту (МПТ), обґрунтування типу, виду та кількості рухомого складу по кожному маршруту, розподіл маршрутів по перевізникам, розробка розкладу та оптимізація режимів руху на маршруті та ін [2].

Формування маршрутної мережі є важливим етапом розробки ефективної транспортної системи міста. Від того, наскільки раціонально розроблено маршрутну мережу, наскільки вдало і гармонійно вона інтегрована в транспортну мережу міста, залежить задоволення населення перевезеннями та ефективність роботи транспортних компаній. Під маршрутною мережею мається на увазі сукупність всіх маршрутів руху пасажирського транспорту біля міста, району тощо. Маршрут руху, у свою чергу, є шляхом руху транспортного засобу між початковим і кінцевим зупинковими пунктами відповідно до розкладу.

Усі існуючі підходи до проектування раціональних маршрутних мереж ДПТ можна поділити на три групи:

- автоматизоване проектування маршрутів пасажирського транспорту на основі формалізованих математичних моделей;
- часткова автоматизація процесу побудови маршрутів пасажирського транспорту та експертна оцінка результатів спеціалістом;

– прийняття рішень на основі досвіду та неформалізованого аналізу експертів.

Застосування жорстко формалізованих математичних моделей дає оптимальне рішення з погляду суворо закладеного в програму алгоритму, проте за такого підходу неможливо врахувати традиції та звички пасажирів, що склалися в місті, екологічну обстановку та інші вимоги, що не піддаються формальному опису. Тому найбільш ефективним вважається другий підхід, у якому експерт проводить аналіз отриманих результатів та приймає остаточне рішення [3].

Основою всього проектування є визначення величини пасажиропотоків за напрямками транспортних кореспонденцій, своєю чергою визначені з урахуванням транспортних розрахункових районів. Чим правильніша територія міста розділена на транспортні райони, тим правильніше (точніше) величини пасажиропотоків. І тим самим найбільшою мірою маршрутна мережа громадського пасажирського транспорту відповідає потребам населення міста.

В даний час відомо досить багато методів вирішення задачі маршрутизації транспорту. Завдання маршрутизації транспорту є узагальненням відомої задачі комівояжера, у якому необхідно побудувати відразу кілька замкнутих маршрутів, які проходять через деяку загальну вершину (депо). Ці завдання належать до класу завдань комбінаторної оптимізації та є NP-важкими. Методів знаходження їх точних рішень та перевірки наближених до оптимальності за кінцевий час немає. Існує точний алгоритм для вирішення задачі маршрутизації транспорту на основі методу гілок та кордонів, але в силу швидкого зростання часу обчислень його неможливо застосовувати для завдань із більш ніж 25-30 вершин [4-5].

Перевагами даного алгоритму можна назвати високу ефективність у порівнянні з іншими методами глобальної оптимізації (наприклад, нейронні мережі, генетичні алгоритми), адаптованість та масштабованість, а також гарантовану збіжність, що дозволяє отримати оптимальне рішення незалежно від розмірності графа.

Мурашиний алгоритм відноситься до категорії алгоритмів роєвого інтелекту і моделює поведінку мурашиної колонії. Мурахи – це соціальні комахи, здатні

утворювати колективи (колонії). Саме колективна система дозволяє ефективно вирішувати завдання динамічного характеру, які могли б бути виконані окремими елементами системи без наявності відповідного зовнішнього управління та координації. Основу поведінки мурашиної колонії становить здатність самоорганізації, що дозволяє швидко адаптуватися до умов навколишнього середовища, що змінюються, і забезпечує досягнення загальних цілей колонії на основі низькорівневої взаємодії, рис.1.1.

Взаємодія відбувається за допомогою феромонів, якими окремі особини позначають пройдений ними шлях. Чим більше феромонів, тим частіше використовується стежка, що вказує на оптимальність маршруту з погляду його довжини.

Логічно припустити, що спочатку мурахи оминатимуть перешкоду як зліва, так і праворуч з рівною ймовірністю. Однак ті представники колонії, які випадково оберуть найкоротший шлях, подолають відстань від початкової точки до мети і назад за короткий проміжок часу, а значить, за кілька пересувань цей шлях збагатиться більше феромонами. А оскільки орієнтиром при русі для мурах служить саме феромон, шлях з більшою концентрацією буде обраний рештою членів колонії.

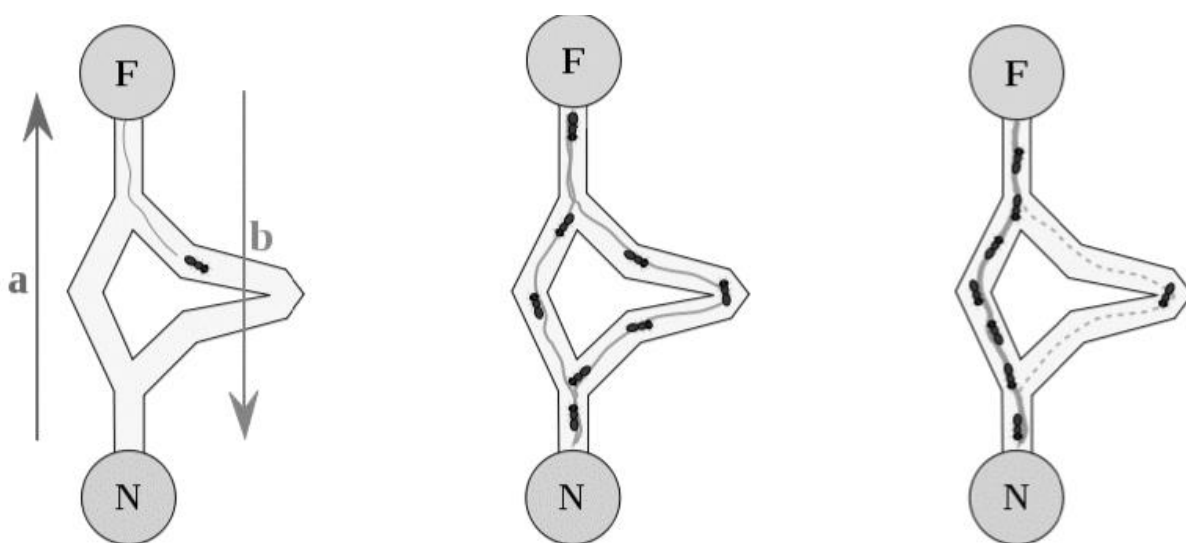


Рисунок 1.1 – Візуалізація схеми роботи мурашиного алгоритму

Мурашиний алгоритм можна описати наступною послідовністю дій:

1. Створення мурах. Спосіб розміщення мурах є визначальним і залежить від умов задачі: всі мурахи можуть бути поміщені в одну точку або в різні. Також у момент створення мурах необхідно задати початковий рівень феромону, що характеризується деяким невеликим позитивним числом. Це необхідно для забезпечення ненульової ймовірності переходу до наступної точки на початковому кроці.

2. Пошук рішення. Маршрут є сукупність вершин графа. Імовірність переходу з вершини i у вершину j визначається за такою формулою:

$$p_{ij} = \begin{cases} \frac{\tau_{ij}^\alpha * \eta_{ij}^\beta}{\sum_{h \notin tabu_k} \tau_{ih}^\alpha * \eta_{ih}^\beta}, & \text{якщо } j \notin tabu_k \\ 0 & \text{інакше} \end{cases} \quad (1.1)$$

де τ_{ij} – кількість феромонів на ребрі (i,j) , «нюх» мурахи;

η_{ij} – привабливість ребра (i, j) , відстань між вершинами i та j , «зір» мурахи;

α, β – регульовані параметри, що визначають важливість складових (вага ребра та рівень феромонів) при виборі шляху;

$tabu_k$ – список вже відвіданих вершин, пам'ять мурахи.

Чим кращий шлях, тим більше феромона належатиме ребрам цього шляху. У загальному випадку, ребра, які використовуються великою кількістю мурах і які є частиною найкоротшого шляху, отримують більше феромонів, і тому найчастіше вибираються мурахами в наступних ітераціях [6-9].

Цей ітераційний процес триватиме до виконання певної умови завершення:

- виконано задану кількість ітерацій;
- вся задана кількість мурах завершила пошук;
- досягнуто необхідної якості рішення;
- минув квант процесорного часу.

Найважливішою складовою транспортної інфраструктури, що багато в чому визначає динаміку розвитку сучасного міста, є функціональна система загального

транспорт. У процесі розвитку міста його транспортна система потребує активного перегляду на оновлення та змін. Це може бути пов'язано з великою кількістю функціональних факторів, які суттєво впливають на структуру розподілу населення та його маршрутних особистостей.

Перепроєктування маршрутної мережі МПТ (або розробка нової раціональної маршрутної мережі) – трудомісткий процес, що включає кілька етапів робіт. Найбільш ефективним підходом розв'язання даної задачі є її автоматизація, але за участю експерта, який проводить аналіз отриманих результатів та приймає остаточне рішення. Функціональна автоматизація потребує активного наукового вивчення, та активної формалізації через розробку нових активних рішень та засобів покращення сучасної транспортної структури.

1.2 Аналіз методів вирішення транспортних задач

Серед всіх етапів планування транспортних перевезень етапи моделювання процесу перевезень і «рішення транспортної задачі» залишаються центральними. Пояснюється це тим, що моделювання процесу перевезення з подальшим визначенням оптимального розподілу наявних обсягів вантажу у постачальників між одержувачами дозволяє не тільки в повній мірі забезпечити їх вимоги, але і зробити це з мінімальними витратами [10].

Іншими словами, такий підхід дозволяє «на кінчику пера» отримати прибуток, яка втрачається при евристичному або інтуїтивному плануванні перевезень. На рис. 1.2 наведена схема, що відображає класифікацію найбільш поширених різновидів транспортних завдань. Згідно схеми, всі транспортні задачі за змістом і особливостями математичної моделі можна розділити на дев'ять класів:

- перевезення вантажів;
- розподіл транспортних засобів;
- вибір засобів доставки вантажу;

- розстановка вантажного флоту;
- розвозка вантажів;
- закриття підприємства;
- двохетапна перевезення вантажів;
- розіграш кубка;
- задача комівояжера.

Найбільш відомі транспортні задачі пов'язані з перевезенням вантажів. Невипадково, до даного класу задач належить класична транспортна задача (0.3) – (0.7). Даний клас не тільки найбільш відомий, але і найбільш представницький. Він включає чотири типи завдань:

- з безперервною закритій математичною моделлю;
- з безперервної відкритої математичною моделлю;
- з дискретною закритій математичною моделлю;
- з дискретною відкритої математичною моделлю.

Другий клас пов'язаний з вибором засобів доставки вантажу і включає два типи завдань:

- з простою математичною моделлю;
- з фіксованими доплатами.

Наступний клас транспортних завдань пов'язаний з двоетапною перевезенням вантажу. Він включає три типи завдань:

- з простою математичною моделлю;
- з багатовимірної математичної моделлю;
- з багатовимірної математичної моделлю за запитами.

Кожен з решти класів завдань представлений одним типом математичних моделей.



Рисунок 1.2 – Класифікація транспортних задач за математичними моделями

1.3 Впровадження алгоритмічних компонентів пошуку оптимальних маршрутів

Тема електронної картографії та навігації дуже актуальна та затребувана на сьогоднішній день, цьому сприяв стрімкий розвиток:

- систем глобального позиціонування;
- телекомунікаційних технологій передачі на будь-які відстані;
- інформаційних банків даних картографічної інформації.

Серед систем електронної картографії є навігаційно-картографічні додатки - повнофункціональні програмні системи з різноструктуровим масивом інформації, що зберігається в базах даних. Технічні засоби, на яких можна скористатися різноманітні:

- персональні комп'ютери та ноутбуки;
- автомобільні навігатори;
- мобільні телефони, смартфони та планшети;
- смарт-годинник, смарт-окуляри;
- навігаційні стаціонарні термінали.

Існуючі картографічні системи використовують велику кількість динамічних даних:

- назви адміністративних та географічних об'єктів, назви доріг, адреси будинків, розмітка дорожнього руху, встановлений допустимий швидкісний режим;

- офіси організацій та компаній, соціальні установи, пам'ятки;
- фотографії, описи, коментарі користувачів, оцінки експертів;
- тимчасові події (наприклад, ярмарки);
- зупинки громадського транспорту, лінії його руху, розклад;
- поточний стан руху на дорогах (пробки, затори, ДТП);
- текуче положення окремих одиниць громадського транспорту.

Навігаційно-картографічні системи вирішують такі завдання:

- надання актуальних статичних та динамічних даних;
- локалізація положення користувача;
- робота з маршрутами.

Робота з маршрутами має такі підзадачі:

- пошук найбільш актуального маршруту згідно вхідних умов;
- побудова деталізованих складових маршруту;
- розпізнавання завантаження доріг та транспортні обставини;
- відстеження переміщення заданим маршрутом.

Алгоритм базується на вхідних даних певного змісту, складається з послідовних кроків їх перетворення на підсумковий текстовий опис. Дані, одержувані на вхід від навігаційно-картографічної системи, є трійкою $In = \{L, M, D\}$:

- L – полігон маршруту. Є масив двійок $\{Latitude, Longitude\}$ вершин ламаної лінії, що описує траєкторію маршруту в реальному світі;
- M – текстові позначки з назвами вулиць. Є масивом трійок $\{Latitude, Longitude, Name\}$, в якому зберігається інформація про вулиці, що проїжджають користувачем;
- D – навігаційні директиви; Є масивом трійок $\{Latitude, Longitude, Directive\}$, в якому зберігається інформація про послідовні дії, які необхідно зробити для досягнення фінішної точки виду «прямо», «направо», «розворот» та інші.

Крім вхідних даних від навігаційно-картографічної системи використовуються додаткові джерела даних, що зберігаються локально:

- NL – визначені улюблені маршрути користувача. Являє собою масив з сформованої інформації $\{Latitude, Longitude\}$, є результуючими користувачевими маршрутами чи його частинами, тобто тими фрагментами шляху, якими він найбільш часто користується та формує їх персональну назву, наприклад, «далі, як у роботу»;
- NP - знайомі користувачеві POI. Являє собою масив функціональних даних $\{Latitude, Longitude, Name, Popularity\}$, які формують інформацію про

найбільш актуальні місця перебування користувача системи, в поле Popularity заноситься загальна кількість візитів.

Всі дані, що зберігаються на пристрої користувача, мають власний унікальний ідентифікатор, що є персональним набором користувацької інформації. Завдяки використанню повністю унікального ідентифікатора з'являється можливість використовувати адресацію до інформації щодо нього без зазначення типу даних. Тут і надалі ідентифікатор позначається як ID.

У ході роботи алгоритму можуть бути потрібні додаткові дані, що запитуються від навігаційно-картографічної системи:

- RP – набір POI у певному радіусі від заданої точки, запитуються за необхідністю доповнити набір вхідної та аналітичної інформації;
- являє собою масив даних {Latitude, Longitude, Name, Popularity}, в якому зберігається інформація про громадські місця (магазини, організації, парки, зупинки, пам'ятки тощо), у полі Popularity зберігається узагальнена оцінка місця, заснована на користувацьких коментарях.

В алгоритмі використовується поняття каркаса - набір лексем, об'єднаних у послідовність на основі вхідних даних, після чого замінюються на слово або словосполучення російською відповідно до визначених для кожної лексеми правил.

Таким чином, виходить список усіх POI, які знаходяться поряд з маршрутом, при цьому в лістингу не вказано, але в алгоритмі враховується:

- NPr-мінімальна відстань, яка може бути від полігону до POI, розраховується аналогічно описаному вище методом визначення належності точки полігону маршруту;

Далі відбувається фільтрація всіх знайдених POI та виділення з них найбільш знайомих користувачеві або, якщо таких немає, найбільш відомих з точки зору суспільної думки, вираженої в оцінці POI, що зберігається в навігаційно-картографічній системі (здійснюється додатковий запит на отримання RP). Для цього:

- для кожного логічного інтервалу (між навігаційними директивами) полігону вибирається одна POI з найбільшим рейтингом популярності користувачеві або, якщо його немає в жодного знайденого POI, з найбільшим громадським рейтингом;

- дані POI згодом братимуть участь у формулюваннях виду «проїжджайте повз ...».

- для кожного закінчення інтервалу (поруч із географічним місцем, де необхідно виконати навігаційну директиву, тобто на поворотах, розворотах тощо) аналогічним чином вибирається по одній POI.

- дані POI згодом братимуть участь у формулюваннях виду «поверніть поруч...».

Угрупування інтервалів за однотипними навігаційними директивами відбувається таким чином:

- всі директиви «прямо» групуються в одну, відстань, що долається, підсумовується (відстань обчислюється виходячи з полігону, що описує конкретний інтервал шляху);

- всі директиви поворотів, відстань між якими менша за заданий налаштуванням порога (передбачається, що це дуже близькі один до одного повороти), групуються в одну із зазначенням їх числа.

Подібні угруповання дозволяють скоротити підсумковий текстовий опис, тим самим спростивши його та зробивши легшим для сприйняття та запам'ятовування.

Формування каркасу результату з лексем є одним із найважливіших кроків алгоритму. На цьому етапі будується послідовність лексем, яка в результаті буде перетворена на текст. При побудові каркаса використовуються такі форми, що у вигляді однієї чи кількох лексем:

- повні та спрощені назви POI, що використовуються у прямій мові;
- словесний опис зовнішнього вигляду POI;
- відмінювання назв POI ({POI_NAME});
- знайомі користувачеві інтервали шляху ({KNOWN_NAME});

- директиви до переміщення (`{ACTION}`, `{DIRECTION}`);
- спрощені заокруглені відстані (`{GROUP_DISTANCE}`);
- групи однотипних дій (`{GROUP_DIRECTION}`);
- прийменники, спілки, розділові знаки (`{TILL}`, «,»);
- "сміттєві" слова, що надають тексту "людяності" (`{JUNK}`).

Для побудови каркаса використовується формальна граматика - спосіб опису формальної мови, тобто виділення деякої підмножини з багатьох слів деякого кінцевого алфавіту. Розрізняють граматики, що породжують і розпізнають (або аналітичні) — перші задають правила, за допомогою яких можна побудувати будь-яке слово мови, а другі дозволяють за цим словом визначити, чи входить воно в мову чи ні. Для завдання побудови каркаса використовується граматика, що породжується, яка визначається наступними характеристиками:

E – набір (алфавіт) термінальних символів;

N – набір (алфавіт) нетермінальних символів;

P - набір правил виду: "ліва частина" → "права частина", де: "ліва частина" - непуста послідовність терміналів і нетерміналів, що містить хоча б один нетермінал; "права частина" - будь-яка послідовність терміналів та нетерміналів;

S – стартовий (або початковий символ) граматики із набору нетерміналів.

1.4 Опис технічних і програмних засобів розробки

Мова програмування Java є об'єктно-орієнтованою мовою програмування, який був створений Джеймсом Гослінгом (James Gosling) і іншими інженерами в компанії Sun Microsystems. Він був розроблена в 1991 році, як частина проекту "Green Project", і офіційно оголошений 23 травня 1995 року, в SunWorld, а випущений в листопаді. Java була спочатку розроблена як заміна для C++ (хоча набір функцій більше схожа на Objective C) і відомий як Дуб (в честь дерева за межами офісу Гослінга). Детальніше про історію Java можна знайти в статті про

платформу Java, яка включає в себе мову, Java Virtual Machine, і Java API. Java є власністю компанії Sun Microsystems; Java є торговою маркою компанії Sun Microsystems.

Існували чотири основних мети при створенні мови Java:

- Об'єктно-орієнтована мова.
- Незалежний від цільової платформи (більш-менш).
- Повинен містити об'єкти і бібліотеки для роботи в мережі.
- Він призначений для виконання коду з віддалених джерел надійно.

Об'єктна орієнтація

Перша характеристика, об'єктно-орієнтований підхід ("ГО"), відноситься до методу програмування для чайників і дизайну мову. Основна ідея ГО полягає в розробці програмного забезпечення все це "речі" (тобто об'єкти), якими можна маніпулювати, а не дії, які потрібно виконувати. Це засновано на тому, що перше (речі) змінюються рідше і радикальніше, ніж дії, що робить такі об'єкти (насправді речі, що містять дані) більш стабільною основою для розробки програмного забезпечення. Метою є, робити великі проекти програмного забезпечення легко керованими, таким чином, можна домогтися підвищення якості і скорочення числа невдалих проектів програмування для початківців.

Незалежність від платформи. Друга характеристика, незалежність від платформи, означає, що програми, написані на мові Java повинні працювати аналогічно на різному устаткуванні. Програміст повинен бути в змозі написати програму один раз і запустити її в будь-якому місці. Це досягається шляхом компіляції Java-коду "наполовину" в байт-код - спрощені машинні команди, яким відповідають стандартний набір реальних команд процесору. Цей код потім необхідно запустити на віртуальній машині, тобто програмою, напісаной на машинному коді для взаємодії з апаратними засобами, яка переводить байт-код Java в придатний для використання код на конкретному обладнанні. Крім того, надаються стандартизовані бібліотеки для забезпечення доступу до особливостей архітектури конкретної машини (наприклад, графіки і мереж) єдиним способом.

Мова Java також включає підтримку для багатопоточних програм - життєво важлива необхідність для багатьох мережових додатків і основа програмування.

Перша реалізація мови використовували інтерпретує віртуальну машину для досягнення мобільності, і багато реалізації використовують такий підходо досі. Однак, ці реалізації створюють програми, які виконуються повільніше, ніж повністю скомпільовані програми, створені типовим компілятором C ++ і трохи пізніше з'явилися компіляторами мови Java, тому мова страждає від придбаної репутації "производитіель повільних програм". У більш пізніх реалізаціях Java VM створює програми, які працюють набагато швидше, використовуючи кілька методів.

Перший метод - це просто компіляція безпосередньо в машинний код, як традиційні компілятор, пропускаючи етап перетворення програми в байт-код цілком. Цим досягається велика продуктивність, але за рахунок втрати мобільності і переносимості програм. Інший метод, "В поцессе виконання" або "JIT", компілює байт-код Java в машинний код під час виконання програми. Більш складні віртуальні машини навіть використовували динамічні перекомпіляції, в якій JVM може аналізувати поведінку роботи програми і вибірково перекомпілювати і оптимізувати критичні частини програми. Обидві ці технології дозволяють програмі скористатися швидкістю машинного код без втрати мобільності.

Портативність є технічно важкою метою для досягнення і успіх Java вдостіженні цієї мети є предметом суперечок. Хоча насправді можна писати програми для платформи Java, які ведуть себе однаково на багатьох платформах, але все ж більшу кількість доступних платформ не виконує програму як очікувалося, а видає невелику кількість помилок або невідповідностей, що призвело до появи пародії на відоме гасло компанії Sun "Написав один раз, працює скрізь" в інший "Написав один раз, налагоджував всюди".

Незалежний від платформи Java, проте, став дуже успішним для серверів додатків, таких як веб-сервіси, сервлети, або Enterprise Java Beans.

Kotlin представляє статично типізований мова програмування від компанії JetBrains. Kotlin можна використовувати для створення мобільних і веб-додатків.

Kotlin працює поверх віртуальної машини Java (JVM) і при компіляції компілюється в байткод. Тобто, як і в випадку з Java, ми можемо запускати додаток на Kotlin всюди, де встановлена JVM. Хоча також можна компілювати код в JavaScript і запускати в браузері. І, крім того, можна компілювати код Kotlin в нативні бінарні файли, які будуть працювати без будь-якої віртуальної машини. Таким чином, коло платформ, для яких можна створювати додатки на Kotlin, надзвичайно широкий - Windows, Linux, Mac OS, iOS, Android [13].

Перша версія мови вийшла 15 лютого 2016 року. Хоча сама розробка мови велася з 2010 року. Поточною версією мови на даний момент є версія 1.2, яка вийшла 28 листопада 2017 року.

Також варто відзначити, що Kotlin розвивається як opensource, вихідний код проекту можна подивитися в репозиторії на github.

Kotlin зазнав впливу багатьох мов: Java, Scala, Groovy, C #, JavaScript, Swift та дозволяє писати програми як в об'єктно-орієнтованому, так і в функціональному стилі. Він має ясний і зрозумілий синтаксис і досить легкий для навчання.

Найпопулярнішим напрямком, де застосовується Kotlin, є перш за все розробка під ОС Android. Причому настільки популярним, що компанія Google на конференції Google I / O 2017 проголосила Kotlin одним з офіційних мов для розробки під Android (поряд з Java і C ++), а інструменти по роботі з даними мовою були за замовчуванням включені в функціонал середовища розробки Android Studio починаючи з версії 3.0.

Cloud Firestore - це гнучка, масштабується хмарна база даних від Firebase і Google Cloud Platform для інтернету, мобільних платформ, і серверних додатків. Як і Firebase Realtime Database, вона синхронізує ваші дані між клієнтськими додатками за допомогою слухачів реального часу а також пропонує підтримку оффлайн режиму для мобільних платформ і веба. В даний момент Firestore знаходиться в бета релізі.

Модель даних Cloud Firestore підтримує гнучкі, ієрархічні структури даних. Зберігає дані в документах, які в свою чергу зберігаються в колекціях. Документи можуть вкладення об'єкти і підколекції.

Пакет розробника Firebase об'єднує інтуїтивно зрозумілі API, позбавляючи вас від необхідності управляти окремими пакетами. Вибирайте тільки те, що вам потрібно, і користуйтеся перевагами інтегрованого рішення. Платформа, яка використовує інфраструктуру Google, надає необхідні можливості для кожного етапу розробки і зростання.

Firestore надає використовує потужну інфраструктуру Google Cloud Platform: автоматичну мультирегіональну реплікацію даних, надійні гарантії цілісності, атомарні batch операції, підтримку транзакцій.

Cloud Firestore - це збережена в хмарі, NoSQL база даних, з якої iOS, Android і веб додатки працюють безпосередньо з допомогою нативних SDK. Крім того доступні SDK для Node.js, Java, Python, Go SDK, а також REST api.

Основні переваги в роботі СКБД Firebase. Швидкість роботи. У пакеті розробника Firebase зібрані інтуїтивно зрозумілі API, які спрощують і прискорюють розробку якісних додатків. Також у вашому розпорядженні всі необхідні інструменти для розширення користувацької бази і підвищення доходів – вам залишається тільки вибрати відповідні для ваших цілей.

Готова інфраструктура. Вам не доведеться створювати складну інфраструктуру або працювати з декількома панелями управління. Замість цього ви зможете зосередитися на потребах користувачів.

Статистика. В основі Firebase лежить безкоштовний аналітичний інструмент, розроблений спеціально для мобільних пристроїв. Google Analytics для Firebase дозволяє отримувати дані про дії ваших користувачів і відразу ж вживати заходів за допомогою додаткових функцій.

Кроссплатформенність. Firebase працює на будь-яких платформах завдяки пакетам розробника для Android, iOS, JavaScript і C++. Ви також можете звертатися до Firebase, використовуючи серверні бібліотеки або REST API.

Масштабованість. Якщо ваш додаток стане популярним і навантаження на нього зросте, вам не доведеться змінювати код сервера або залучати додаткові ресурси – Firebase зробить це за вас. Крім того, більшість функцій Firebase безкоштовні і залишаються такими незалежно від масштабу ваших проектів.

Платних функцій чотири. В них передбачений безкоштовний пробний період і два тарифних плани.

Безкоштовна підтримка по електронній пошті. Крім того, команда Firebase і фахівці з розробки Google дадуть відповідь на ваші запитання на ресурсах Stack Overflow і GitHub.

Завдання Cloud Firestore - це надавати якісний досвід користувачам, поставляючи їм додатки краще і швидше. Пакети програм Android, iOS і Javascript в Cloud Firestore синхронізують дані додатків в реальному часі, завдяки чому програмістам простіше створювати додатки.

Android, iOS, і веб-версія Cloud Firestore підтримують роботу баз даних в офлайн-режимі, щоб користувачі могли заходити в мобільні додатки, писати і читати в них навіть в авіарежимі. Дані можна просінхронізувати пізніше в хмарі, коли знову з'явиться інтернет.

Cloud Firestore використовує колекції і документи, щоб структурувати і запитувати дані, які програмісти потім можуть використовувати для виразних запитів.

Програмісти також можуть займатися внесерверної розробкою, що дозволить мобільним додаткам встановлювати з'єднання безпосередньо з базою даних.

Cloud Functions можна легко налаштувати так, щоб писати кастомний код в момент записування даних. Пакети програм автоматично інтегруються з Firebase Authentication, щоб прискорити процес запуску.

Firestore - повністю керований продукт і Мультирегіональна база даних з тиражуванням. Firestore допомагає програмістам легше створювати додатки незалежно від їх масштабу.

У Cloud Firestore, ви можете використовувати запити для отримання одного документа, визначеного документа, або всіх документів в колекції, які відповідають параметрам вашого запиту.

Запити можуть містити ланцюжка з декількох фільтрів або фільтри комбіновані з сортуванням. Також всі колекції індексуються за замовчуванням, з

цього продуктивність запиту прямо пропорційна результату, а не розміром колекції.

Як і Realtime Database, Cloud Firestore використовує синхронізацію даних для поновлення даних на кожному підключеному пристрої.

Cloud Firestore кеширує дані, які ваше додатки часто використовує, з цього додаток може писати, читати, слухати поновлення і робити запити навіть коли девайс знаходиться в офлайн. Коли девайс повернеться в онлайн Firestore всі локальні зміни в хмару.

1.5 Постанова мети та завдання

Метою кваліфікаційної роботи є розробка системи з пошуку оптимального маршруту для картографічного мобільного додатку на навігації. Розроблене рішення має використовувати орієнтовану графову структуру та декілька функціональних методів з оптимізації побудови транспортних маршрутів, згідно обраного транспортного рішення. Інформаційна система має будувати маршрути згідно доданих координатних точок користувача та оброблювати більше ніж 2 такі точки на мапі. Також слід запровадити автоматизовані засоби геопозиціонування з використанням апаратної частини користувацького пристрою. Інформаційна система має формувати оптимальний прогноз часу на пересування користувача згідно сформованого маршруту та надавати альтернативні маршрути за умови можливості їх формування.

Розробку слід виконати засобами мови програмування Kotlin та з використання системи керування даними Firestore Firebase.

Завданням кваліфікаційної роботи є розробка сучасного мобільного програмного рішення з пошуку оптимального маршруту між заданими точками з оптимізаційною логікою на базі алгоритму мурашиної колонії.

Вимогами до кваліфікаційної роботи є:

- виконати огляд сучасних рішень при вирішенні транспортних задач;
- виконати оцінку ефективності застосування транспортних задач в картографічних системах;
- виконати оцінку ефективності алгоритмів побудови оптимальних маршрутів;
- провести порівняльний аналіз засобів розробки картографічних систем;
- спроектувати архітектуру програмної системи;
- виконати опис алгоритмічної складової розробленої інформаційної системи;
- виконати опис функціональних можливостей розробленого програмного рішення;
- провести аналіз ефективності розробленого рішення;
- виконати розробку програмного засобу.

1.6 Висновок до першого розділу

Проаналізувавши предметну область та методи вирішення задач пошуку оптимального маршруту, було визначено напрям розробки та особливості застосування ефективних рішень. Серед усіх оптимізаційних алгоритмів пошуку слід застосовувати алгоритм мурашиної колонії, який дозволить зменшити час на пошук оптимального рішення, а також мінімізує апаратні витрати на його роботу. Для перевірки ефективності його роботи слід запровадити додаткове рішення з розрахунку оптимального маршруту на графі, та забезпечити кореляцію результатів в залежності від складності маршруту та додаткових параметрів, які можуть застосовуватися при його побудові.

Для реалізації такого програмного рішення слід використати концепцію мобільних додатків, яка дозволить розробити швидке, надійне та портативне програмне рішення, яке може використовуватися в якості самостійного

програмного продукту, або в якості окремого функціонального модуля, який може запроваджуватися до інших програмних систем.

2 РОЗРОБКА ЛОГІКИ РОБОТИ АЛГОРИТМУ ПОШУКУ ОПТИМАЛЬНОГО МАРШРУТУ

Системи пошуку оптимальних маршрутів є актуальними та сучасними програмними системами, які активно впроваджуються в різні функціональні структури та програмні засоби, що застосовуються в різних напрямках ведення бізнесу. Отже використання сучасних програмних алгоритмів у поєднанні з потужними математичними рішеннями дозволить розробити сучасну та актуальну інформаційну систему в галузі пошуку оптимальних маршрутів.

2.1 Застосування алгоритмів мурашиних колоній для розробки інформаційних систем

Як основна математична модель транспортної логістики, як правило, приймається класичне транспортне завдання, в якому вирішується питання про доставку продукту з пунктів його складування до пунктів споживання. При цьому перевезення здійснюються реальною транспортною мережею, яка легко інтерпретується на графах $\Gamma(V, S)$. Кожному вузлу мережі відповідає вершина $v_i \in V$, а ланці – дуга $s = (i_s, j_s)$. У аналізованій основній задачі мережа складається з m вузлів і n ланок. Для кожного вузла i задані речові числа b_{ik} , $i = 1, m$, $k = 1, q$. Якщо ж $b_{ik} < 0$, то в i -му вузлі розташований споживач (клієнт) та його замовлення на товар k -го виду складає b_{ik} . Якщо $b_{ik} = 0$, i -ий вузол проміжний. Для кожної ланки задано дійсне число d_s , яке може мати різний зміст, наприклад: довжина ланки, час проходження ланки ТС, вартість провезення ланкою одиниці товару, пропускна здатність ланки. Потрібно скласти план розвезення товару, який мінімізує значення цільової функції $\sum d_s$.

Зазвичай вибирається одне з цільових функцій й нею вирішується завдання. Якщо потрібно врахувати кілька критеріїв оптимальності, то завдання вирішується по кожному з них і вибирається одне рішення, яке підходить до кожного критерію. При цьому моделі можуть бути різними. У разі критеріїв протяжності або вартості базовою є транспортна модель, а у разі тимчасового критерію або пропускної спроможності використовується модель мережевого планування.

Для вирішення оптимізаційної задачі та спроектувати з подальшою розробкою функціонального алгоритму пошуку оптимального шляху з використанням зовнішніх функціональних факторів – слід обрати найбільш ефективні алгоритмічні методи, які використовуються при вирішенні аналогічних задач, рис. 2.1

Для вирішення сформованої задачі, слід використати гібридну комбінацію алгоритмів пошуку оптимального, які включають в себе поліпшення кількох маршрутів та алгоритму мурашиних колоній.

У роботі розглядається критерій довжини чи вартості сумарного шляху маршрутів. Ця модель належить до класу багатопродуктових мережевих транспортних моделей. Її окремим випадком є однопродуктова модель, яка полягає в наступному.

Завдання Т. Потрібно мінімізувати сумарну довжину маршруту

$$L = \sum_{s=1}^n d_s \quad (2.1)$$

на безлічі $\{D\}$ кістяків і планів розвезення клієнтів

$$x = (x_1, x_2, \dots, x_s, \dots, x_n),$$

де x_s - обсяг вантажу, що перевозиться за ланкою $s \in D$, що задовольняють умовам

$$\sum_{s \in N_i^+} x_s - \sum_{s \in N_i^-} x_s = b_i, \quad i = \overline{1, m} \quad (2.2)$$

де N_i^+ - безліч ланок s , що входять у вузол i , N_i^- - безліч ланок s , що виходять з i .

При цьому m

$$\sum_{i=1}^m b_i = 0. \quad (2.3)$$

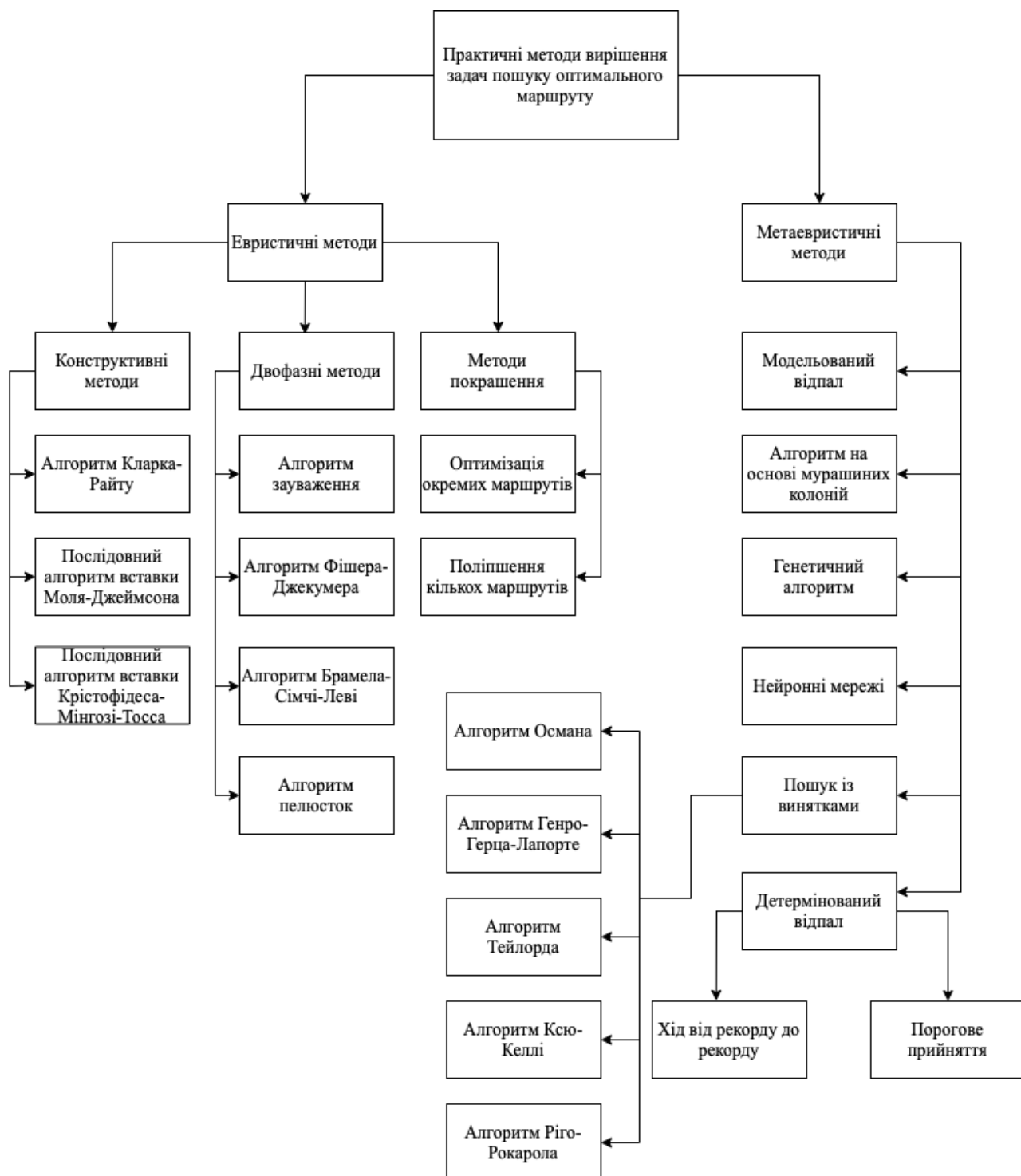


Рисунок 2.1 – Схема актуальних методів вирішення пошукових задач

Пошук шуканих маршрутів на безлічі кістяків доцільний з таких причин: обсяг перероблюваної інформації на дереві істотно менше, ніж на мережі; на дереві існує єдина дорога, що зв'язує дві вершини, тому при пошуку маршрутів відсутня перебір; об'єднання маршрутів з однією вихідною вершиною є деревом, тому безліч кістяків містить і цікаве для нас підмножина оптимальних (найкоротших) маршрутів; використання допустимого кістяка гарантує реалізацію перевезення вантажів по всіх клієнтах.

Перебір кістяків пропонується реалізовувати еволюційною метаевристикою, зберігаючи рекордне значення загальної протяжності маршрутів. Таким чином, на кожному кроці процесу випадковим чином будується кістякове дерево і на наступному етапі алгоритму відбувається його розкладання на маршрути та розрахунок їх протяжності.

Для визначення логічної складової роботи програмного засобу та застосування алгоритму побудови оптимальних маршрутів та пошуку альтернативних типів транспорту, необхідно спроектувати функціональну блок-схему роботи метода побудови графу та пошуку оптимального вектору. На рис. 2.2 зображено функціональну блок-схему роботи алгоритму.

Алгоритм розкладання кістяного дерева на маршрути.

Позначимо: S_f - Файл фрагментів маршрутів; S_m - файл безлічі маршрутів; S_v - Файл безлічі вершин.

Вхід: основне дерево D .

Крок 1. Початок: Вибирається одна вихідна вершина $i_0 \notin S_v$ і поміщається у S_v . Дуга $s_0 = (i_0, j_0)$, $j_0 \neq i_0$ міститься у S_f . j_0 – додається у S_v . Якщо таких дуг кілька, то вони розміщуються у S_f , а інцидентні їм вершини у S_v . Крок 2. Вибір чергової дуги. Нехай фрагмент чергового маршруту містить дуг. Остання дуга $s_k = (i_{sk}, j_{sk})$. Знаходимо у дереві D дугу s_{k+1} , з початком $i_{sk+1} = j_{sk}$ і кінцем $j_{sk+1} \notin S_v$. Якщо таких дуг кілька, кожен додаємо до останнього непорожнього фрагмента, отримуючи кілька нових фрагментів. Інакше, якщо такої дуги немає, то маршрут побудований і він переноситься з файлу S_f в S_m . Якщо $S_f \neq \emptyset$, то крок 2 виконується для

останнього фрагмента з S_f , інакше якщо $S_v \neq m$, тона крок 1, інакше, якщо $S_v = m$, то крок 4.

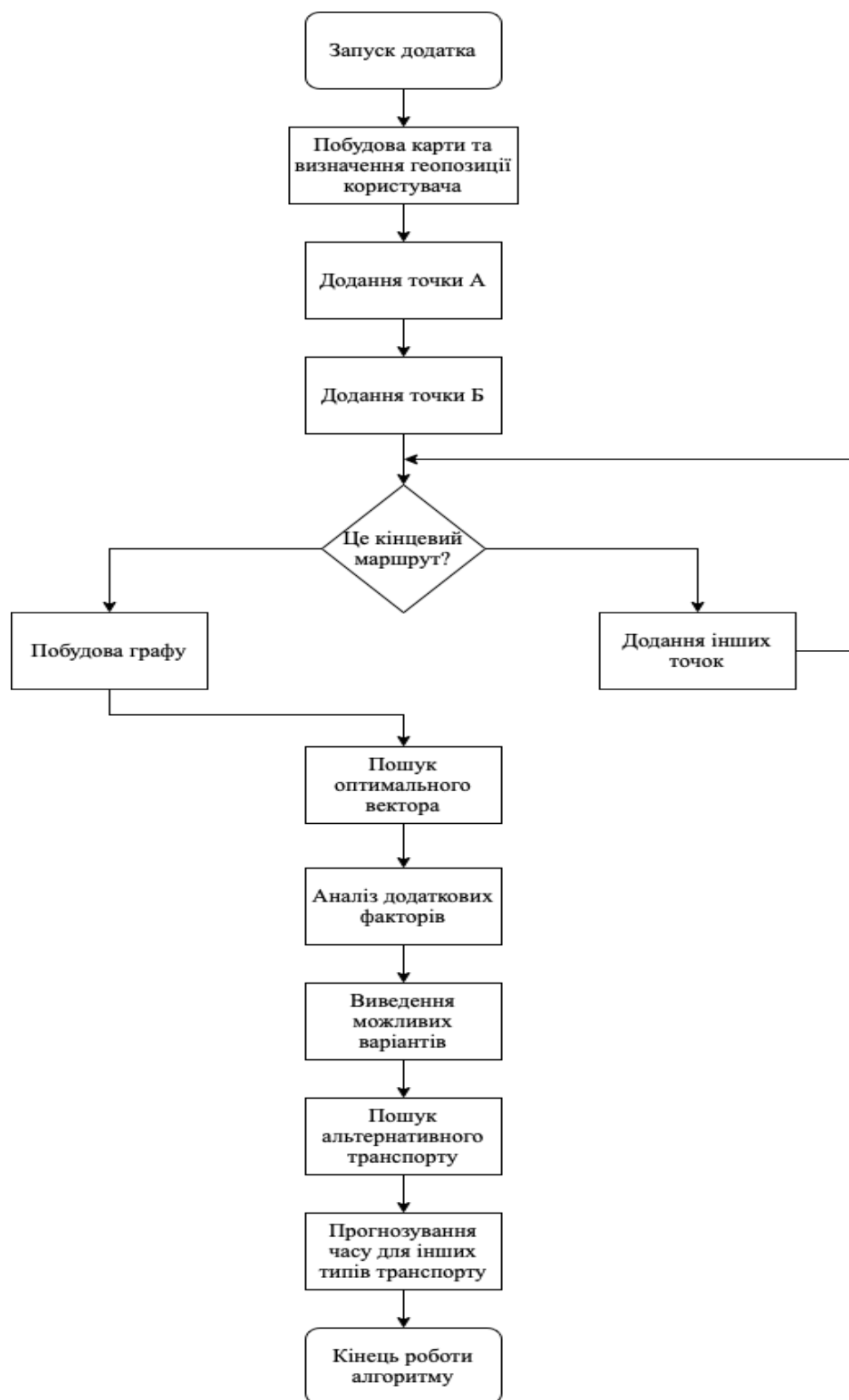


Рисунок 2.2 – Функціональна блок-схема роботи алгоритму пошуку оптимального маршруту

Крок 3. Поєднання маршрутів. Якщо в S_m існують два такі маршрути, що один з них є фрагментом другого, цей маршрут виключається з S_m і обсяги x_s вантажу, що перевозиться, коригуються. Таким чином, кількість маршрутів зменшується. В іншому випадку перехід до кроку 4. Крок 4. Кінець. Файл S_m містить перелік усіх маршрутів. Підрахунок сумарної довжини маршрутів.

Вихід: список маршрутів S_m ; $L(S_m)$ – сумарна довжина маршрутів

При розвезенні різних товарів доцільно, наскільки можна, компонувати консолідовані замовлення кожному за клієнта і розміщувати в одному транспортному засобі та одному пакеті. З цією метою пропонується наступний алгоритм.

Алгоритм поєднання консолідованих маршрутів. Алгоритм складається із виконання $(p+1)$ етапів. На кожному етапі виконуються ідентичні

процедури пошуку маршруту перевезення для продукту $k = 1, p$; на $(p+1)$ -му етапі деякі з маршрутів поєднуються. Для виконання поєднання вибирається найдовший за кількістю ланок маршрут. Він оголошується «провідним». Серед інших маршрутів знаходяться ті, які представляють пов'язані частини «провідного» маршруту, вони поєднуються з ним. Для фіксування обсягів розвезень по кожній ланці s використовується векторний запис

$$x^s = (x_1^s, x_2^s, \dots, x_p^s).$$

Перестановки різних товарів у процесах «розкладання» кістяка і «суміщення» маршрутів призводять до різних результатів. Тому слід виконати процедури розкладання дерева та суміщення маршрутів на безлічі перестановок товарів. І тому можна застосувати одноточечну еволюційну метаевристику двох рівнях загального алгоритму.

Еволюційний метод безлічі кістяків для вирішення задачі незамкнутої маршрутизації консолідованих вантажів.

На вході маємо транспортну мережу. Одна ітерація методу кістякового дерева складається з виконання наступних кроків.

Крок 1. Побудова остовного дерева D . Виконується алгоритм Пріма. Крок 2. Розв'язання системи рівнянь $\sum_{s \in D} x_s + \beta = 0$, і коригування дерева з метою

отримання $x_s \geq 0$, $s = 1, n$, $\beta = (b_1, b_2, \dots, b_i, \dots, b_m)$, b – вага i -ої вершини (наявність/запас клієнта); x_s – кількість товару, що перевозиться за ланкою s .

Крок 3. Розкладання кістяного дерева на маршрути.

Крок 4. Поєднання (часткове об'єднання) маршрутів. Маршрути поєднуються, якщо один із них є підмаршрутом іншого, обсяги товару складаються. Місткість ТЗ, що залишилася, коригується.

4.1. Пошук пари маршрутів зі списків 1 та 2, які підходять для суміщення. Нехай це M_1 та M_2 і один з них є підмножиною іншого.

4.2. Маршрути M_1 і M_2 поєднуються та обсяги товарів з урахуванням місткості ТЗ складаються.

4.3. Підготовка інформації для завантаження наступного у списку $(k+2)$ товару. Якщо $(k+1)=p$, то перехід крок 5.

Виконується задана кількість p ітерацій алгоритму синтезу двох маршрутів. На $(k+1)$ -ой ітерації відбувається поєднання відповідних маршрутів і складання обсягів k видів продуктів з $(k+1)$ -им продуктом, якщо це дозволяє зробити решту місткості ТЗ, інакше цей продукт залишається до виконання наступної ітерації.

Крок 5. Розрахунок сумарної протяжності маршрутів та порівняння з рекордною.

Крок 6. Кінець ітерації. На виході маємо сукупність маршрутів рекордної протяжності, для ланок у яких вказано обсяги завантаження.

Крок 7. Коригування вихідної інформації. Якщо кількість нижніх ітерацій вичерпано, то перехід на Крок 1, інакше для дерева D перехід на крок 2 для β вектора, відповідного наступного продукту.

На виході формується сукупність маршрутів для консолідованих перевезень.

Завдання побудови замкнутих (циклічних) маршрутів – це NP-повне завдання комівояжера, яке полягає в наступному. Дано повний неорієнтований граф $G = (V, S)$, кожному з ребер $s \in S$ якого зіставляється невід'ємна цілочисленна вартість c_s . Необхідно знайти шлях мінімальної вартості (довжини), що проходить точно по одному разу через кожну вершину графа.

2.2 Впровадження алгоритмів повного перебору для пошуку оптимальних маршрутів

Алгоритм повного перебору (АПП) реалізує відбір у просторі N рішень за допомогою перебору всіх альтернатив. Підсумком діяльності цього методу вважається чітке рішення.

Мінус АПП – це його тимчасова складність, площа пошуку збільшується експоненційно (швидкість зростання пропорційно значенню самої величини), тому, якщо N ніяк не є істотно-малим - застосовують евристичні та пошукові алгоритми.

Метод гілок та кордонів – це розвиток АПП. Завдання методу полягає у додаванні контролю критерію для обмежуючої функції, що виходить із знання проблеми, згідно з яким у конкретному рівні можна припинити створення цієї гілки дерева перестановок. Він зберігає всі без винятку позитивні якості АПП, проте не досить придатний для питань, у місці, де N не вважається суттєво невеликим.

Переваги цього алгоритму - його можна розпаралелити і цей алгоритм має точне рішення.

Спосіб включення далекого (СВД) полягає в наступному: міста, кордонно-далекі один від одного, в жодному разі не стануть суміжними в ланцюзі. Дані 2 міста та стануть базовими з метою подальшого рішення. Далі знову знаходиться вершина, гранично далека від вершин, раніше ув'язнених у ланцюг. Знаходиться найменша сума довжин ребер серед виявленої вершиною і двома суміжних вершин у ланцюзі, що ставить місце у ланцюзі виявленої вершині.

BV-метод базується на розгляді існуючого еталонного маршруту та його оптимізації. Рішення умовно складається із двох стадій:

- отримання початкового еталонного рішення. Рішення являє собою найкраще рішення з усіх рішень, отриманих на основі «жадібного» способу;
- оптимізація початкового рішення – полягає у модернізації отриманого початкового еталонного маршруту з підтримкою BV-модифікаторів.

Цей алгоритм має квадратичну складність, який надає наближене рішення і може бути розпаралеленим на 2-му етапі.

Генетичний алгоритм (ГА) та мурашиний алгоритм (Ant Colony System – ACS) вважаються фаворитами пошукових алгоритмів.

Найбільш найкращим із числа пошукових алгоритмів вважається ГА. Щоправда, він також має мінуси, пов'язані з передчасною збіжністю. ГА надає наближене рішення задачі, він також може бути запроваджений. На рис. 2.3 зображено блок-схему роботи генетичного алгоритму при вирішенні транспортної задачі.

Мурашиний алгоритм є розвитком алгоритму «Система мурах». Його основні відмінності:

- 1) у функції підбору нового міста явно існує баланс між використанням тих знань, які накопичилися та дослідженням нових рішень;
- 2) при глобальному оновленні феромону його додаток відбувається тільки до дуг, що належать глобальному короткому шляху;
- 3) доки мурахи знаходять рішення, відбувається локальне оновлення феромона.

Ідея мурашиного алгоритму – моделювання реальної мурашиної колонії. Колонія є концепцією з дуже елементарними правилами автономної дії мурах. Незважаючи на простоту поведінки будь-якої мурахи, дії всієї колонії виявилось розумним. Таким чином, основою поведінки мурашиної колонії призначається низькорівневий зв'язок, внаслідок якої колонія являє собою розумну систему, блок схема мурашиного алгоритму показана на рис. 2.4.

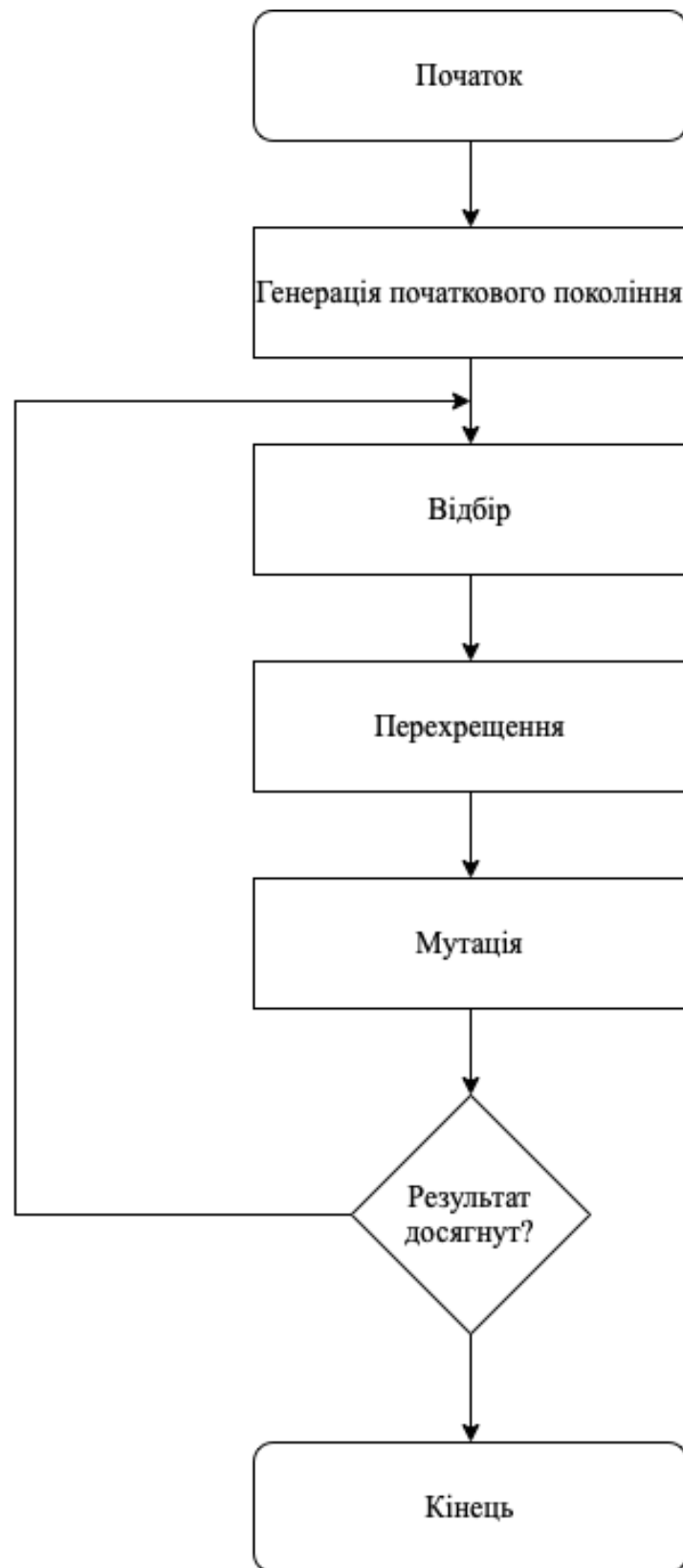


Рисунок 2.3 – Блок-схема роботи генетичного алгоритму для вирішення транспортної задачі

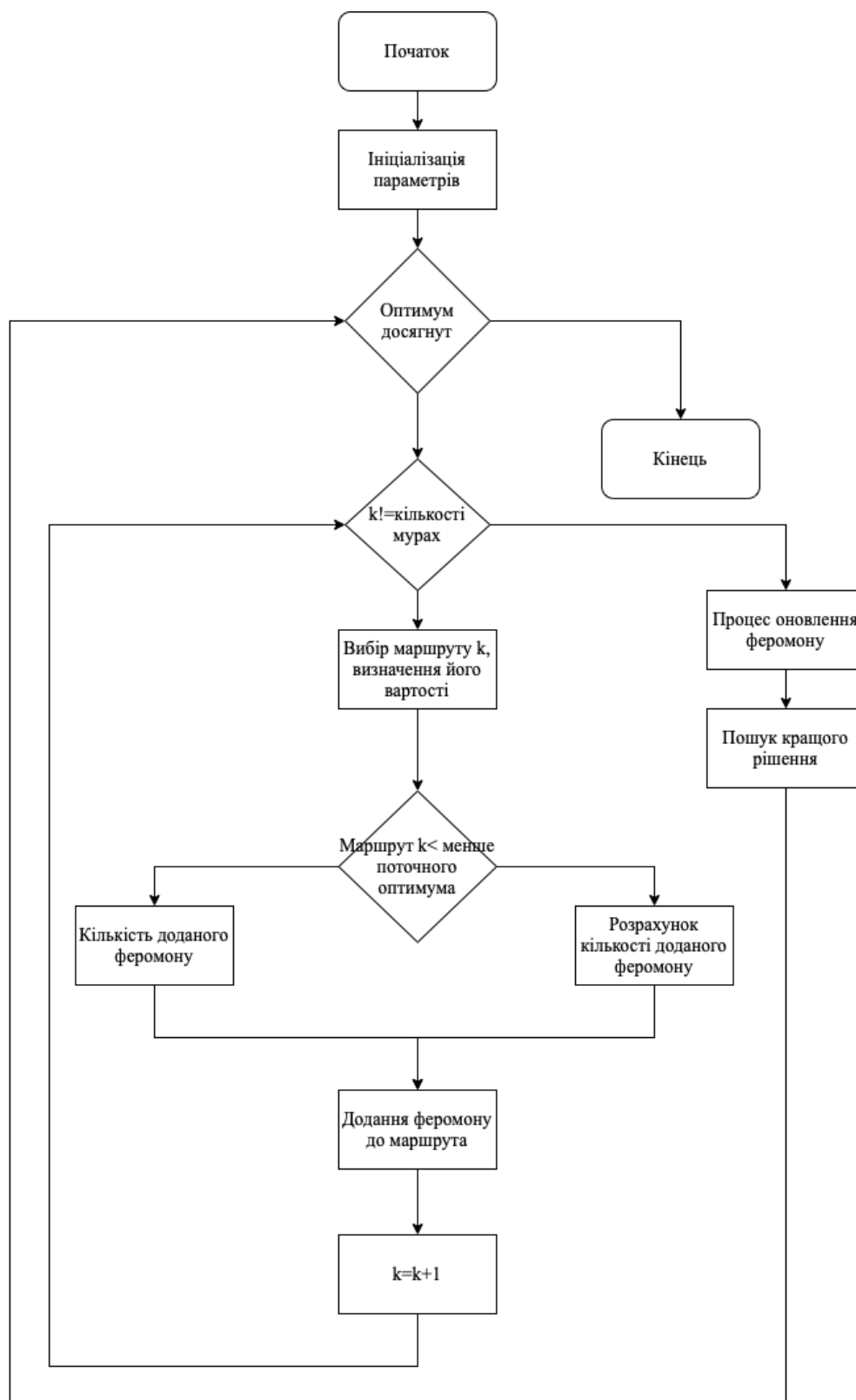


Рисунок 2.4 – Схема роботи мурашиного алгоритма

Феромон – спеціально хімічна речовина, яка визначає взаємодію між мурахами, які відкладають цю речовину на пройдений шлях. При виборі шляху, мураха не бере тільки факт короткого шляху, а й враховує досвід інших мурах, цю

інформацію, мураха отримує з феромонів на кожному шляху. Виходить, що феромони визначають бажання мурахи зробити вибір між тим чи іншим маршрутом. Хоча при цьому підході не можна уникнути попадання в локальний оптимум. Ця проблема вирішується за допомогою випарів феромонів.

Пам'ять мурахи (список табу) - це пройдені мурахою міст, в які зайти ще раз заборонено. При використанні списку табу мураха абсолютно точно не потрапить до одного міста двічі. Цей список зростає при проходженні шляху та скидається у старті будь-якої ітерації алгоритму. Тоді «Список міст, які ще необхідно відвідати мурашкою k , що знаходиться в місті i , є доповненням до пам'яті мурахи».

Видимість – це зворотна відстані $= 1/D_{ij}$, де D_{ij} – відстань між містами i та j . «Видимість є локальною інформацією, що характеризує бажання мурашки відвідати місто j із міста i – що вона вище, тим ближче місто i тим більше бажання його відвідати».

Безпосередній вибір наступного міста здійснюється за принципом «колеса рулетки»: для кожної мурахи генерується маршрут руху з останнього місцезнаходження випадковим чином, з урахуванням ймовірностей переходу. Потім кожного з отриманих маршрутів розраховується цільова функція загальної протяжності маршруту.

2.3 Впровадження гібридного алгоритму мурашиної колонії до логістичної галузі

Для запровадження сформованої логіки роботи алгоритму та вирішенні транспортних та логістичних задач, спроектовано блок-схему загального принципу роботи мурашиного алгоритма в поєднанні з інформаційними системами, рис. 2.5.

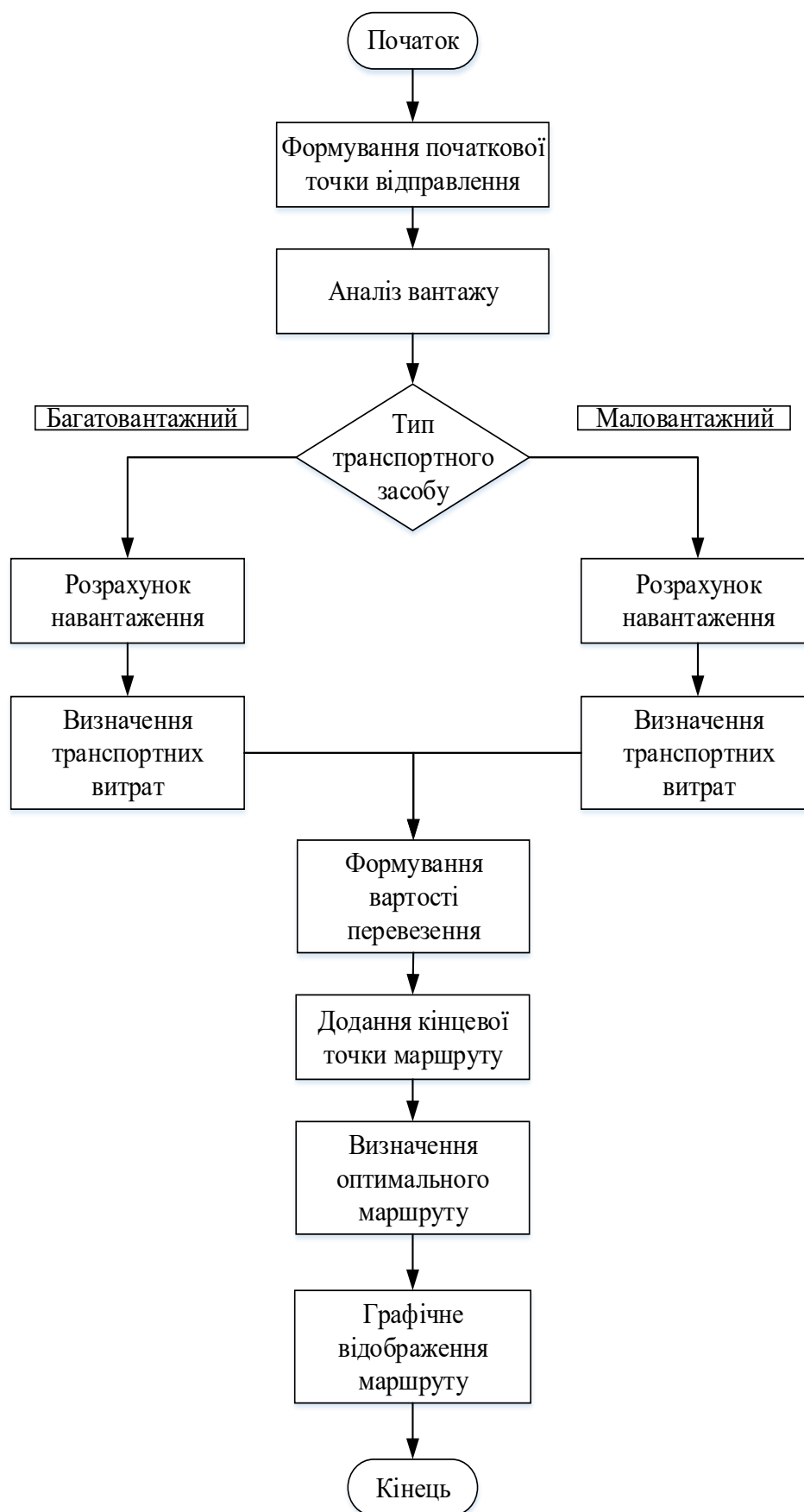


Рисунок 2.5 – Алгоритм роботи комплексу логістики

Алгоритмічна блок-схема включає у себе наступні блоки:

- Формування початкової точки відправлення – дозволяє сформувати точку відправлення вантажу. Є початковою точкою розрахунку в системі.
- Аналіз вантажу – ініціалізує процес розрахунку вартості перевезення. Враховує розміри вантажу, а також його вагу. Є складовою частиною при розрахунку вартості перевезення.
- Тип транспортного засобу – дозволяє сформувати коштовний коефіцієнт. Включає в себе вартість перевезення вантажу, відстань, маршрутний лист, а також умови перевезення.
- Розрахунок навантаження – формує сумісність вантажу з обраним транспортним засобом.
- Визначення транспортних витрат – коефіцієнти які формуються під час перевезення вантажу, а також додаткові розрахунки на перевезення вантажу.
- Формування вартості перевезення – фінальне формування вартості перевезення вантажу, без додання маршрутних витрат.
- Додавання кінцевої точки маршруту – дозволяє додати остаточну точку перевезення вантажу, та розрахувати фінальну вартість перевезення.
- Визначення оптимального маршруту – використання різних алгоритмічних та математичних методів розрахунку оптимальних маршрутів.
- Графічне відображення маршруту – дозволяє вивести маршрут руху транспортного засобу на інтерактивну мапу.

В результаті розробки алгоритму роботи функціональної логіки, з'являється можливість сформувати логіку розташування мережевих, програмних та архітектурних компонентів. Для цього застосовуються спеціалізовані функціональні діаграми.

На етапі проектування розробки програмної системи, необхідно визначити всі функціональні можливості програмної системи та визначити перелік об'єктів функціонування, як апаратних так і програмних. Для цього є необхідність в розробці діаграми розміщення, рис. 2.6.

Діаграма розміщення (deployment diagram) відображає фізичні взаємозв'язки між програмними і апаратними компонентами системи. Вона є хорошим засобом для того, щоб показати маршрути переміщення об'єктів і компонентів в розподіленій системі.

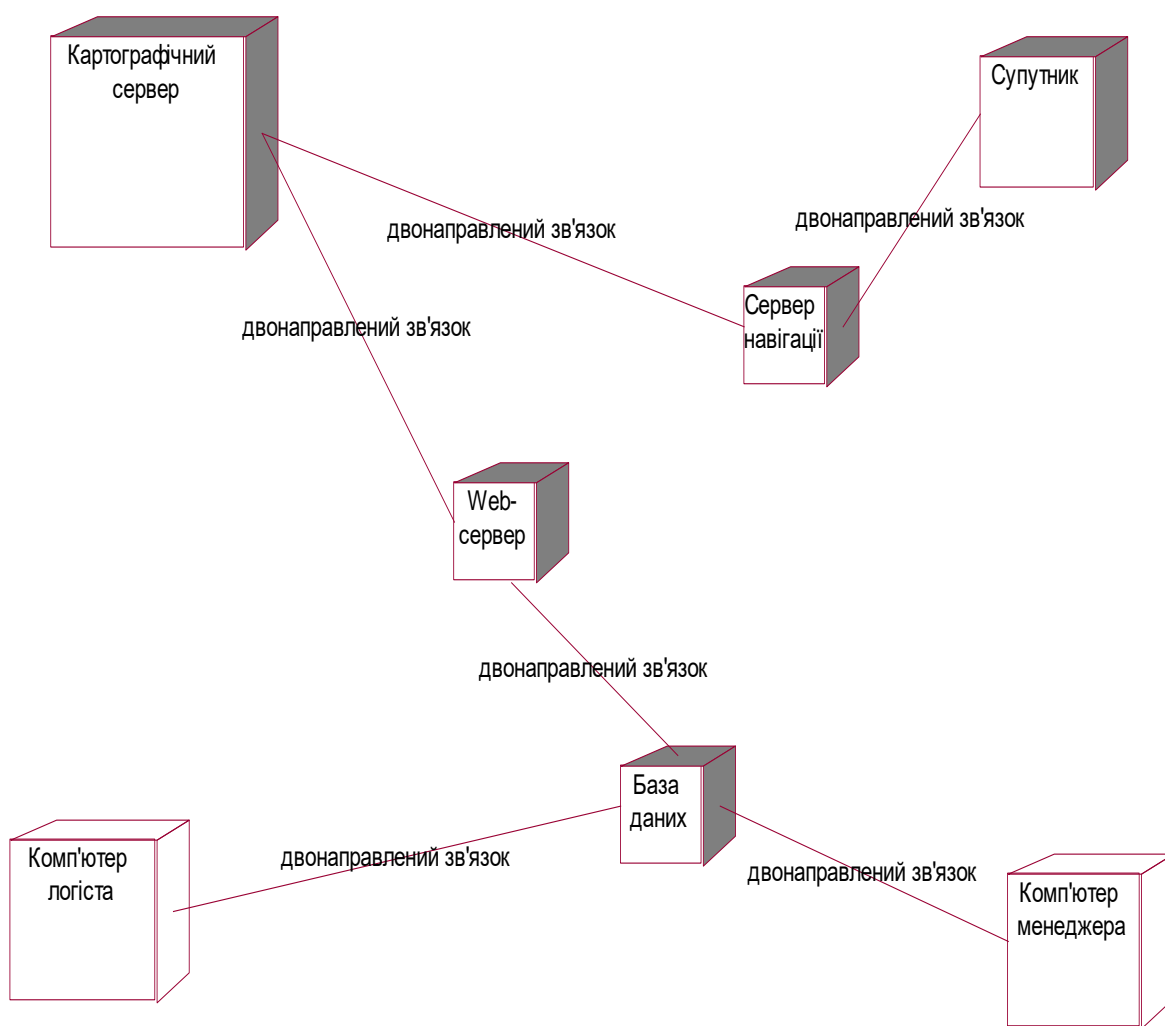


Рисунок 2.6 – Діаграма розміщення компонентів технічної системи

Кожен вузол на діаграмі розміщення являє собою певний тип обчислювального пристрою, в більшості випадків – частина апаратури. Ця апаратура може бути простим пристроєм або датчиком, а може бути і мейнфреймами.

2.4 Висновок до другого розділу

Розроблений алгоритм поведінки мурашиних колоній має актуальне значення для використання в інформаційних системах пошуку оптимальних маршрутів, а також при вирішенні транспортних та логістичних задач. Розроблена логічна концепція дозволяє оптимізувати маршрутний граф та визначити найбільш актуальні напрямки руху, згідно сформованих початкових та кінцевих точок, які задають користувачі системи. Мінімізація часу на обробку вхідних умов дозволяє користувачам системи прорахувати усі варіанти оптимальних маршрутів на мапі, та побудувати найкращих з пропозицій.

Розроблена логіка алгоритма використовує орієнтовану графову структуру та декілька функціональних методів з оптимізації побудови транспортних маршрутів, згідно обраного транспортного рішення. Інформаційна система дозволяє будувати маршрути згідно доданих координатних точок користувача та оброблювати більше ніж 2 такі точки на мапі.

3 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ З ПОШУКУ ОПТИМАЛЬНОГО МАРШРУТУ ДЛЯ ЛОГІСТИЧНИХ ОРГАНІЗАЦІЙ

У третьому розділі кваліфікаційної роботи, описано процес проектування інформаційної системи та створення спеціалізованої архітектурної документації, необхідної для розробки функціональної інформаційної системи. Також виконано опис структурних компонентів системи, та розробленої інформаційної системи, її практичного використання.

3.1 Проектування розробки програмної системи

На етапі проектування системи, першою дією є розробка діаграми варіантів використання яка зображає відношення між акторами і прецедентами. На рис. 3.1 зображена діаграма варіантів використання функціональних можливостей системи.

Діаграми варіантів використання дозволяють створювати список операцій, які виконує система. Як правило цей вид діаграм називають діаграмами функцій, так як набір таких діаграм створює список вимог до системи і визначається з множиною функцій, що виконує система. Кожна діаграма - це опис послідовності дій та взаємодій між акторами та іншими компонентами системи. Діаграма варіантів використання, має можливість визначити архітектурну логіку та взаємодію елементів програмної системи, можливість зміни їх станів, та їх функціональне призначення.

Діаграма варіантів використання дозволяє розробникам системи визначити головні складові компоненти та їх роль, а також взаємодію між іншими функціональними складовими об'єктами. Діаграма варіантів дозволяє відшукати слабкі сторони системи, а також мінімізувати ризик великих помилок на етапах кодування та тестування інформаційної системи.

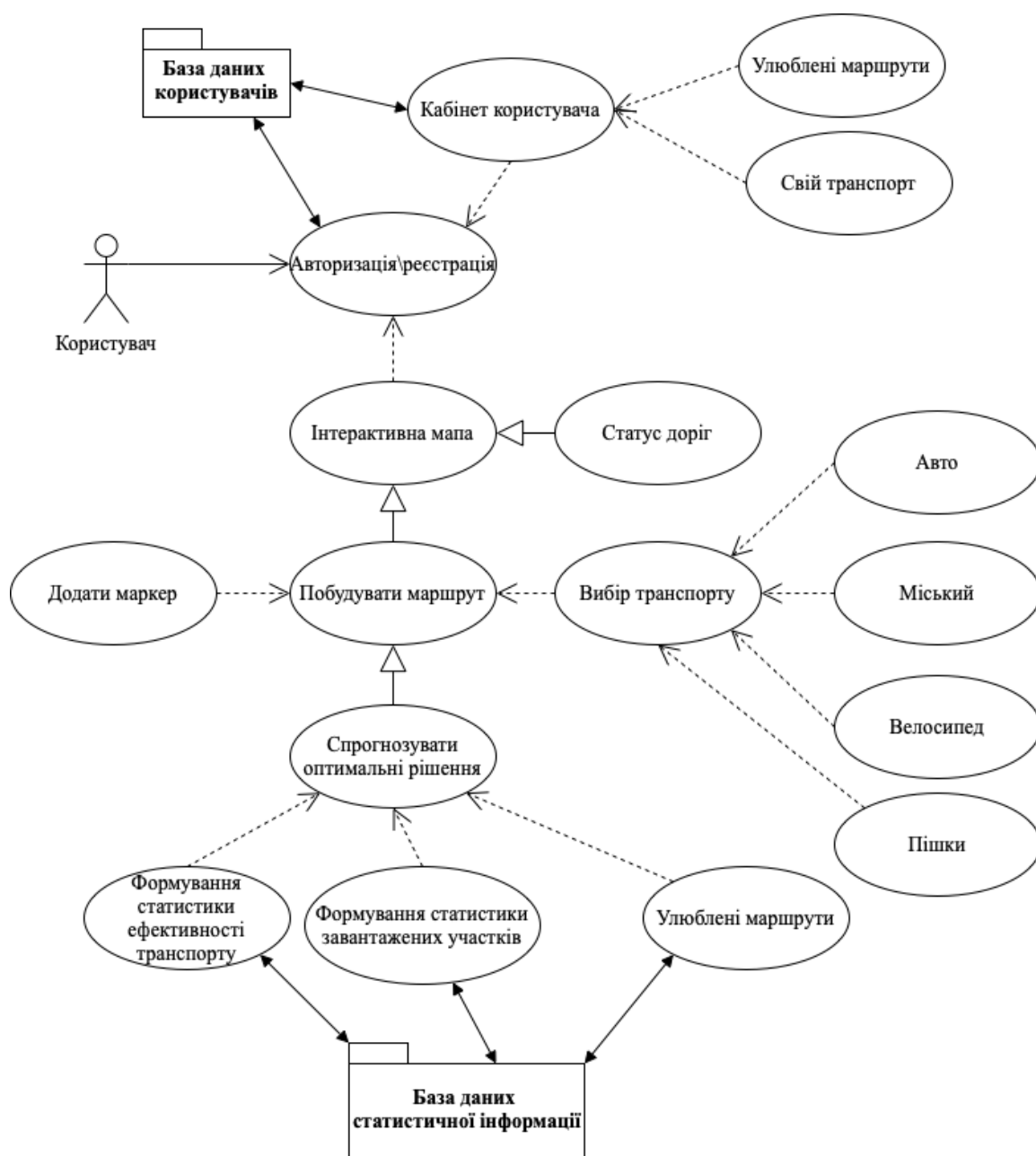


Рисунок 3.1 – Діаграма варіантів використання інформаційної системи

Діаграма варіантів використання включає в себе наступні об'єкти:

- Авторизація/реєстрація. Кожен користувач інформаційної системи має необхідність пройти авторизацію та реєстрацію в системі. Це є обов'язковою умовою використання інформаційної системи.
- Інтерактивна мапа. Спеціалізований компонент, який інтегрується до інформаційної системи та дозволяє додавати зовнішні шари та інформацію, визначати геопозиціонування користувача, а також формувати маршрути для переміщення.

- Побудувати маршрут. Системна функція визначення відстані між доданими маркерами.
- Додати маркер. Дозволяє додавати геопозиційні маркери на інтерактивну мапу. Використовуються для побудови маршрутів.
- Спрогнозувати оптимальні рішення. Спеціалізоване рішення, яке будує усі можливі алгоритми маршруту та визначають мінімальний час за який можливо дістатися від точки А до точки Б.
- Формування статистики ефективності транспорту. Накопичувана інформація згідно усіх побудов маршруту користувачів, яка використовується в якості засоба постійного покращення результатів роботи алгоритму.
- Формування статистики завантажених участків. Алгоритм який накопичує статистичну інформацію про завантажені участки доріг впродовж різного часу побудови маршрутів.
- Улюблені маршрути. Маршрути які додає користувач до свого профілю з можливістю швидкого виклику для застосування.
- Статус доріг. Інформація про поточне завантаження доріг, яке функціонує завдяки виклику системної бібліотеки з аналізу трафіку через супутникові засоби фіксації та навантаженні на GSM станції впродовж доріг.
- Кабінет користувача. Власний репозиторій кожного створеного користувача системи.
- Свій транспорт. Додання типу транспортного засоба, на якому користувач системи пересувається більшу кількість часу.
- Вибір транспорту. Можливість обрання різних типів транспортних засобів, які можуть використовуватися для побудови оптимальних маршрутів.

Сформовані варіанти використання є актуальними для застосування в проектній частині та можуть використовуватися як головні архітектурні складові програмного засобу. Наступним етапом є проектування логічної частини поведінки користувача, для цього будується алгоритмічна блок-схема.

Алгоритмічна блок-схема роботи програмної системи, згідно визначених варіантів використання, зображено на рис. 3.2. Вона дозволяє визначити набір

алгоритмічних дій, які повинен виконати кожен користувач для отримання реального результату.



Рисунок 2.2 – Алгоритмічна блок-схема роботи інформаційної системи

Кожна блок-схема дозволяє відповісти на велику кількість питань, які формуються під час загальної розробки програмного рішення. Отже використання такого засобу розробки вимог до систем є одним з найкращих.

Блок-схема до програмного засобу з побудування маршруту включає в себе роботу з доданням транспортного засобу, а також визначення його оптимальності для виконання кінцевої задачі. Кожний транспортний засіб має своє обмеження та можливості руху.

Після визначення транспортного засобу, формується маршрут перевезення який включає в себе точку А та точку Б на мапі. Для підвищення якості роботи використовується інтерактивна мапа від компанії «Google», яка дозволяє отримувати більш насичену систему маршрутів, розраховувати відстань руху, а також візуально виводити показники маршрутів на екран користувача. Таке рішення є доступним за рахунок використання спеціалізованих програмних модулів взаємодії, які знаходяться в свободному доступі.

Під звітом пропонується виведення декількох актуальних маршрутів з визначенням часу на кожних з них.

В якості схеми взаємодії різних структурних елементів програмної системи, використовується діаграма послідовності дії. Вона визначає процес та завдання, які виконують програмні компоненти, та дозволяють визначити конкретну необхідність в кожному з них. Діаграма послідовності дії зображена на рис. 3.3.

Враховуючи можливість масштабування програмного засобу, в якості головного засобу з оцінки ефективності маршрутів для перевезення, необхідно виконати розрахунок навантаження на кожний компонент програмного засобу. Для цього використовується кооперативна діаграма, дозволяюча розрахувати та спроектувати навантаження, зображено на рис. 3.4.

На кооперативній діаграмі екземпляри об'єктів показано у вигляді піктограм. Лінії між ними позначають повідомлення, обмін якими здійснюється в рамках даного варіанту використання.

Кожен вид діаграм взаємодії має свої переваги, вибір зазвичай здійснюється виходячи з переваг розробника. На діаграмах послідовності робиться акцент саме на послідовності повідомлень, при цьому легше спостерігати порядок, в якому відбуваються різні події. У разі кооперативних діаграм можна використовувати просторове розташування об'єктів для того, щоб показати їх статична взаємодія.

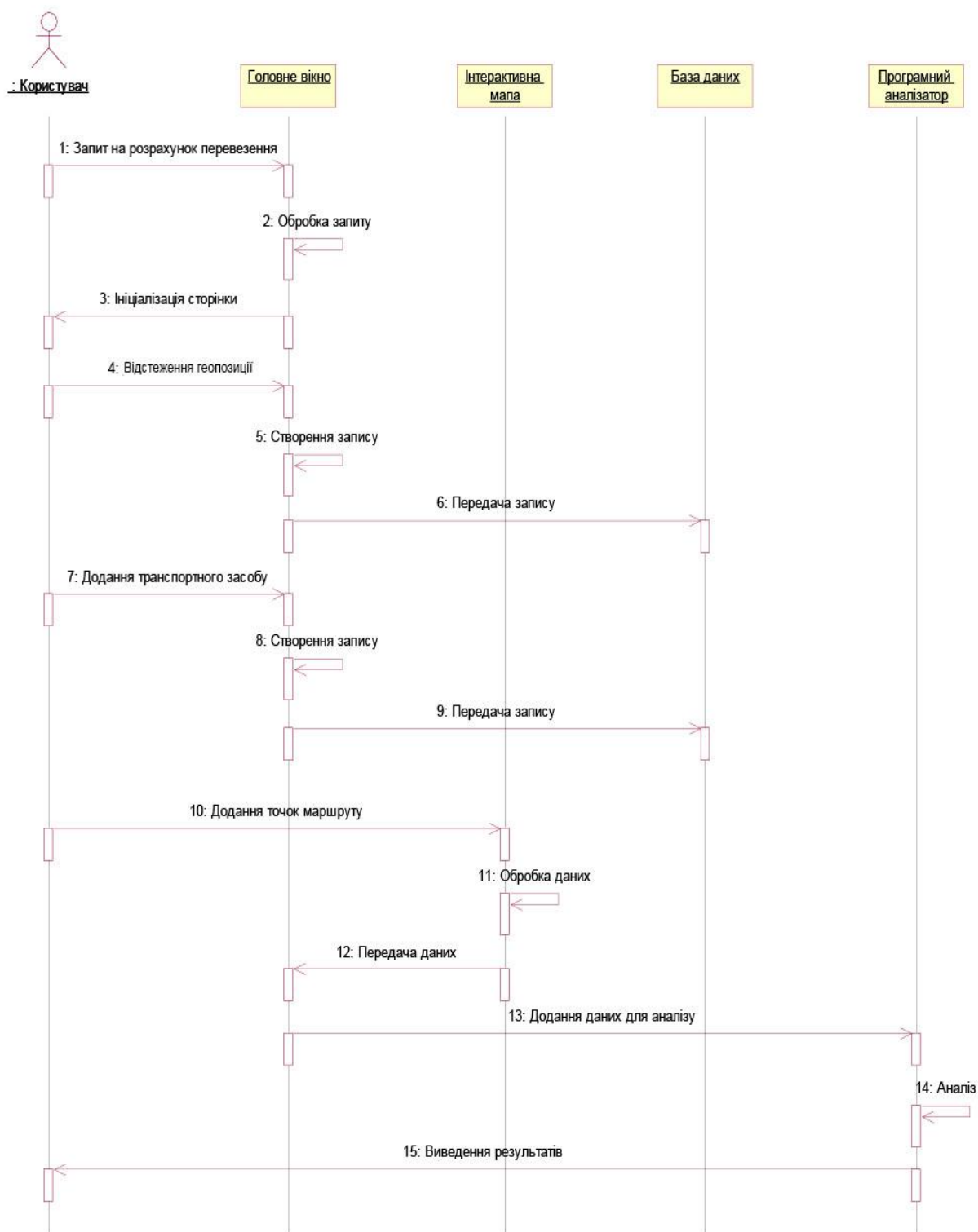


Рисунок 3.3 – Діаграма послідовності дії роботи інформаційної системи

Однією з головних властивостей будь якої діаграми взаємодії є її простота. Глянувши на діаграму, можна легко побачити всі повідомлення.

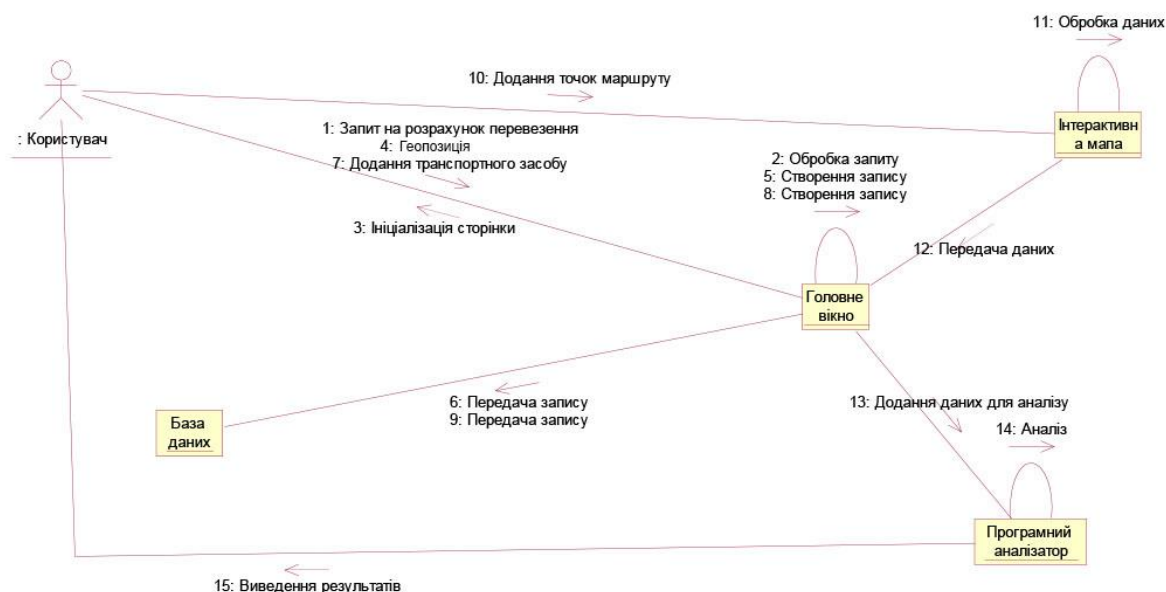


Рисунок 3.4 – Кооперативна діаграма визначення навантаження на об’єкти системи

Однак при спробі зобразити щось більш складне, ніж єдиний послідовний процес без безлічі умовних переходів або циклів, даний підхід може не спрацювати.

Для відображення умовного поведінки на діаграмах взаємодії існує два підходи. Один з них полягає у використанні окремих діаграм для кожного сценарію. Другий полягає в тому, що повідомлення супроводжуються умовами, що показують поведінку об’єктів.

В залежності від сформованих розрахунків навантаження, формується перелік об’єктів які мають бути змінені, або до них використано рефакторинг. Отже розрахунок навантаження є досить важливою складовою частиною при якій визначають найбільш активні компоненти програмного засобу.

Кооперативна діаграма входить до переліку потужних засобів розробки програмного забезпечення, та дозволяє автоматизувати наступні процеси тестування та якості коду, виконують функцію оцінки ефективності коду та інше.

Після визначення поведінки програмних компонентів, необхідно визначити усі можливі стани кожного компоненту програмної системи діагностування стану та оцінки надійності технічних систем. Для цього розроблено діаграму станів, рис. 3.5.



Рисунок 3.5 – Діаграма станів програмної системи діагностування стану та оцінки надійності технічних систем

Кожна діаграма станів в UML описує всі можливі стани одного примірника певного класу і можливі послідовності його переходів з одного стану в інший, тобто моделює всі зміни станів об'єкта як його реакцію на зовнішні впливи.

Діаграми станів найчастіше використовуються для опису поведінки окремих об'єктів, але також можуть бути застосовані для специфікації функціональності інших компонентів моделей, таких як варіанти використання, актори, підсистеми, операції та методи.

Діаграма станів є графом спеціального виду, який представляє деякий автомат. Вершинами графа є можливі стани автомата, зображувані відповідними графічними символами, а дуги позначають його переходи з стану в стан.

В мета моделі UML автомат є пакетом, в якому визначено безліч понять, необхідних для подання поведінки модельованої сутності у вигляді дискретного простору з кінцевим числом станів і переходів.

Тривалість перебування системи в будь-якому з можливих станів суттєво перевищує час, який витрачається на перехід з одного стану в інший. Передбачається, що в межі час переходу може бути одно нулю (якщо не обумовлено інше), тобто зміна станів об'єкта може відбуватися миттєво. Поведінка автомата моделюється як послідовне переміщення по графу від вершини до вершини з урахуванням орієнтації пов'язаних дуг.

Діаграма класів (Static Structure diagram) — діаграма, що демонструє класи системи, їх атрибути, методи і взаємозв'язки між ними. Є класичною нотацією UML.

Класи є основою розробки та проектування програмного забезпечення. Діаграма класів допомагає визначити суттєву складову програмного забезпечення, зображено на рис. 3.6.

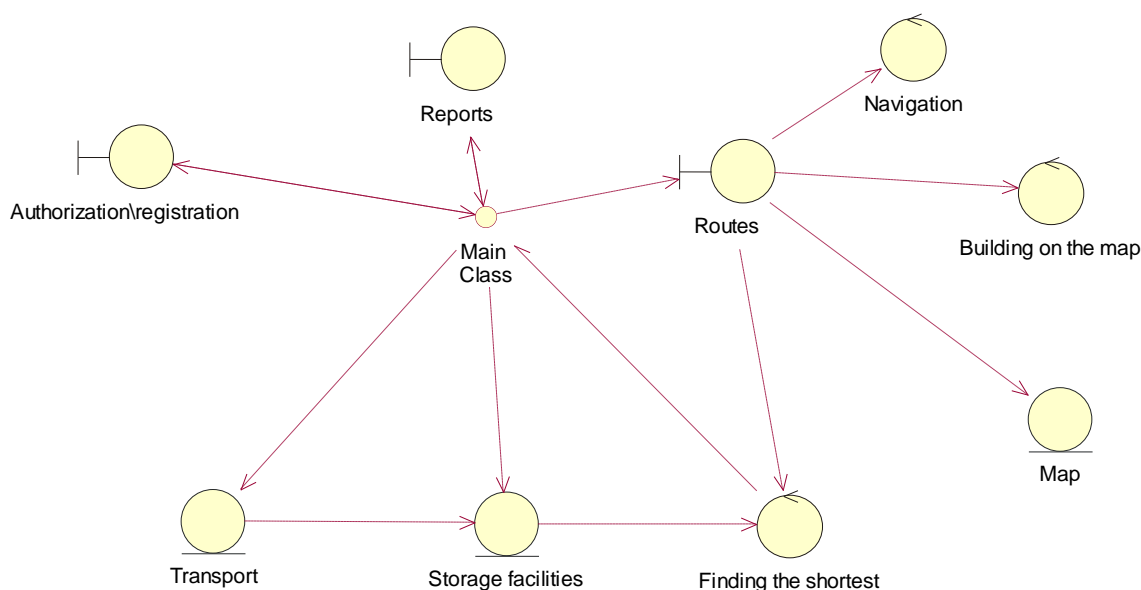


Рисунок 3.6 – Діаграма класів програмного засобу

Клас (class) у мові UML служить для позначення безлічі об'єктів, що володіють однаковою структурою, поведінкою, стосунками з об'єктами інших класів. Графічно клас зображується у вигляді прямокутника, який додатково може бути розділений горизонтальними лініями на розділи або секції. В цих розділах можуть вказуватися ім'я класу, атрибути (змінні) та операції (методи).

Крім внутрішнього пристрою або структури класів на відповідній діаграмі зазначаються відносини між класами. При цьому сукупність типів таких відносин фіксована в мові UML і зумовлена семантикою цих типів відносин. Базовими відносинами в UML є:

- залежності (dependency relationship);
- асоціації (association relationship);
- узагальнення (generalization relationship).

Кожне з цих відносин має власне графічне представлення на діаграмі, яка відображає взаємозв'язки між об'єктами відповідних класів.

На діаграмі визначено взаємодію кожного класу програмного засобу, та його взаємодію між собою. Це дозволяє врахувати архітектурні рішення, які були прийняті під час розробки програмного засобу. Головним класом, є клас програмний, який дозволяє поєднати усі класи між собою. Він використовується для роботи інших класів між собою.

Головним класом системи є клас main який використовується в якості поєднання програмного засобу з базою даних, та дозволяє виконувати трансляцію інформації, між іншими класами, які пов'язані з роботою бази даних.

В якості класу конектора використовується системний клас, який виконує роль протоколу передачі інформації між таблицями бази даних, інформації користувача, та іншими компонентами системи.

Інші класи, які були побудовані в системі, виконують задачу роботоздатності програмного засобу, зберігають у собі інформаційні, навігаційні, контролюючі та інші засоби, які є необхідністю роботи системи.

Також деякі класи, використовують додаткову класову структуру, наприклад побудування реєстрації та авторизації, для яких необхідно розробити систему

криптографічного захисту елементів системі від несанкціонованого доступу до приватної інформації користувачів.

Для побудови бази даних розробляємої інформаційної системи, необхідно провести формалізацію задачі, що є необхідним етапом розробки завдання і полягає в побудові структури таблиць для зберігання інформації, схеми їх взаємозв'язків і опису алгоритмів обробки.

На підставі вхідних документів створюються таблиці бази даних, опис яких, наведено в таблицях 3.1-3.3.

Таблиця 3.1 – Таблиця reports у базі даних

Стовпець	Тип даних	Що виконує
Id	Int	Системний ідентифікатор
ChosenTransport	varchar	Текстове поле
TransportId	Int	Системний ідентифікатор
goodWeight	varchar	Текстове поле
StartCity	varchar	Текстове поле
finishCity	varchar	Текстове поле

Таблиця звітування дозволяє групувати інформацію, отриману внаслідок роботи системи розрахунку коротшого шляху.

Звіт є важливою складовою математичного розрахунку ефективності, який дозволяє переглянути вартість перевезення вантажу, на обраному транспортному засобі з використанням додаткових умов, які можуть бути додані перед початком перевезення.

Ключовими полями є – поле обрання транспорту, його спеціалізованого ідентифікатора, допустимого навантаження, початкового та кінцевого пунктів слідування.

Таблиця 3.2 – Таблиця transport в базі даних

Стовпець	Тип даних	Що виконує
Id	Int	Системний ідентифікатор
type	varchar	Текстове поле

Продовження таблиці

Стовпець	Тип даних	Що виконує
PermWeight	Int	Системний ідентифікатор
Capacity	Int	Системний ідентифікатор
fuel	Int	Системний ідентифікатор
speed	Int	Системний ідентифікатор
priceTransit	Int	Системний ідентифікатор

Таблиця транспорту необхідна для створення гнучкої системи додання транспортних засобів до програмного засобу та використовувати їх для розрахунку оптимальних маршрутів. Таблиця складається з типу транспорту, розмірів, максимальної швидкості, кількості витрачаємого палива, швидкості руху, вартості перевезення. Усі ці складові дозволяють розрахувати можливість перевезення за кожною з вулиць на маршруті.

Таблиця користувачів необхідна для створення персональних облікових записів користувачів та використання їх для роботи з інформаційною системою. Таблиця користувачів складається з персонального ідентифікатора користувача, персонального імені користувача, пароля, а також імені користувача.

Таблиця 3.3 – Таблиця users в базі даних

Стовпець	Тип даних	Що виконує
Id	Int	Системний ідентифікатор
login	varchar	Текстове поле
password	varchar	Текстове поле
name	varchar	Текстове поле

Схема бази даних — її структура, описана на формальній мові, що підтримує СКБД. В реляційних базах даних схема визначає таблиці, поля в кожній таблиці (звичайно з зазначенням їх назви, типу, обов'язковості), і обмеження цілісності (первинний, потенційні й зовнішні ключі й інші обмеження). Схеми в загальному випадку зберігаються в словник даних. Хоча схема визначена на мові бази даних у вигляді тексту, термін часто використовується для позначення графічного представлення структури бази даних. Основними об'єктами графічного представлення схеми є таблиці та зв'язки, обумовлені зовнішніми ключами. В якості графічного уявлення взаємозв'язку таблиць у базі даних, використовується побудовання діаграми бази даних, зображено на рис. 3.7.

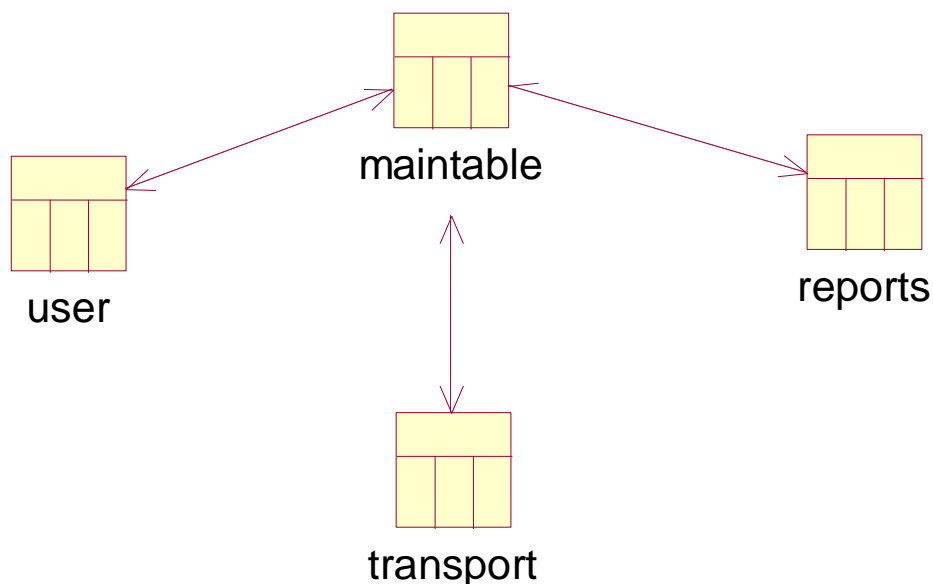


Рисунок 3.7 – Діаграма бази даних

Висновок. В результаті проектування інформаційної системи, було отримано набори вхідної інформації для розробки програмного додатку з прогнозування оптимальних маршрутів руху при поїздках з точки А до точки Б та можливістю додання додаткових точок. Проектна документація дозволяє більш детально сформулювати головні вимоги до програмного засобу, а також визначити головні

складові системи. В процесі проектування було розроблено діаграму варіантів використання, діаграму послідовності дій, діаграму компонентів, діаграму станів системи, діаграму класів, а також діаграму бази даних.

3.2 Опис інтерфейсу розробленої інформаційної системи

Розробка сучасних мобільних додатків потребує запровадження сучасних інтерактивних та функціональних рішень, які направлені на роботу з великими обсягами даними, та адаптацією до різних пристроїв з різною роздатністю екранів. За рахунок таких факторів розробка сучасних мобільних додатків потребує використання спеціалізованих програмних засобів, які адаптують контент для виведення користувачу, та покращують взаємодію.

Першим кроком при роботі з інформаційною системою є створення облікового запису для нового користувача чи авторизація вже існуючого. З метою забезпечення захищеності персональних даних користувачів в базі даних, використовується алгоритм шифрування даних, який забезпечує унікальний набір персональних даних з неможливістю зовнішнього перегляду.

Для цього застосовується алгоритм SHA-1, який інтегрується в якості окремої бібліотеки для взаємодії між формою авторизації та базою даних. Таким самим чином виконується постійне шифрування персональних даних та передача їх в зашифрованому вигляді до бази даних.

Також застосовано алгоритм salt який дозволяє створювати однакові набори даних з розподілом користувачів на персональні ідентифікатори. На рис. 3.8 зображено вікно авторизації в мобільному додатку.

Для розробки інформаційної системи використано набір сучасних візуальних компонентів, які дозволяють забезпечити єдину візуальну складову, а також запровадити єдиний набір графічної стилізації для різних робочих просторів користувача.

Авторизация

Логин

Введите Ваш логин

Пароль

Введите Ваш пароль

Войти

[Нет аккаунта? Зарегистрируйтесь](#)

Рисунок 3.8 – Вікно авторизації в мобільному додатку

Під час реєстрації нового користувача в системі, рис. 3.9, необхідно додати інформацію за основними полями, які дозволяють персоналізувати акаунт, а також обрати актуальну інформацію для побудови маршрутів згідно доданого типу транспортних засобів. Для цього новому користувачу необхідно додати наступну інформацію:

- Повне ім'я.
- Логін.
- Пароль.
- Підтвердження пароля.
- Тип транспортного засобу який використовує користувач в більшості часу.

Після додання усієї необхідної інформації, створюється новий обліковий запис та персональний профіль.

Регистрация

Полное имя *

Диана

Логин *

d

Пароль *

1234567890

Повторите пароль *

1234567890

Транспорт *

Велосипед

Зарегистрироваться



Рисунок 3.9 – Вікно реєстрації нового користувача

Після реєстрації користувача, формується його персональний кабінет, рис. 3.10, який зберігає різну інформацію стосовно його діяльності в фоновому режимі. Така інформація дозволяє покращувати роботу інформаційної системи та платформи Android.

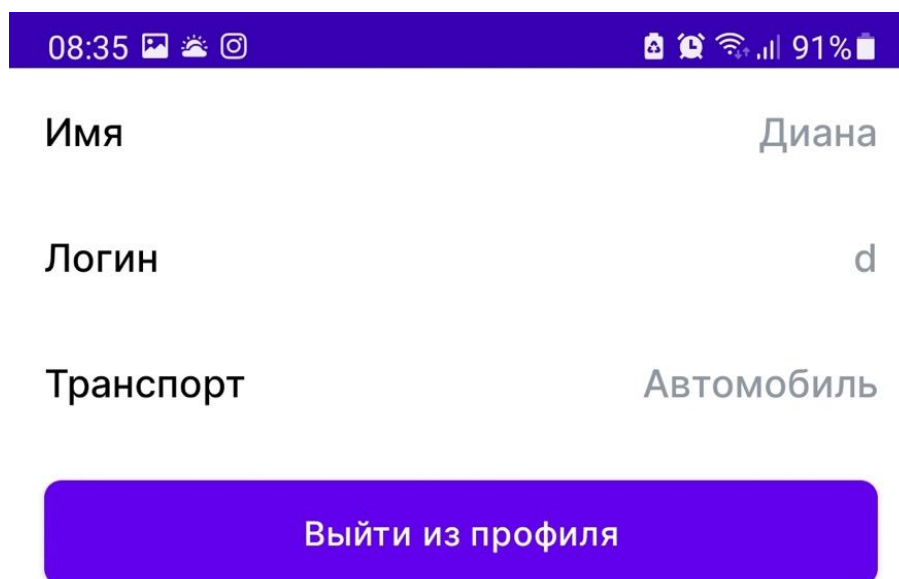


Рисунок 3.10 – Вікно з персональним кабінетом користувача

Після авторизації користувач переходить до головного вікна системи, де завантажується мапа світу. Користувач має дозволити системі автоматично використовувати геопозиціонування засобами мобільного телефону. Для цього необхідно в самому додатку чи в налаштуваннях додати відповідний запит.

Автоматично налаштувати геопозиціонування для нових пристроїв неможливо, так як це потребує спеціалізованих наборів доступу до керування пристроєм. Розроблене рішення використовує декілька засобів геопозиціонування. За замовчуванням використовується визначення об'єкта за телефонними станціями зв'язку. Таке рішення дає точність до 5-ти метрів. Додатково використовується супутникове позиціонування, яке підтримується на апаратному рівні мобільного пристрою. Супутникове геопозиціонування дозволяє збільшити точність отримання координат до 1-го метра. Також, з метою додаткового алгоритму перевірки позиції об'єкта – застосовано засіб отримання інформації від провайдера мережі інтернет. Точність такого позиціонування до 2-ух метрів в місті з розвитими засобами геопозиції. На рис. 3.11 зображено вікно з мапою перед початком пошуку геопозиції.



Рисунок 3.11 – Вікно з виведенням мапи

Після визначення геопозиції користувача, стає доступним головний інтерфейс розробленого програмного рішення. Користувач може побудувати маршрут між декількома точками, а також переглянути навантаження трафіка в реальному часі, рис. 3.12.

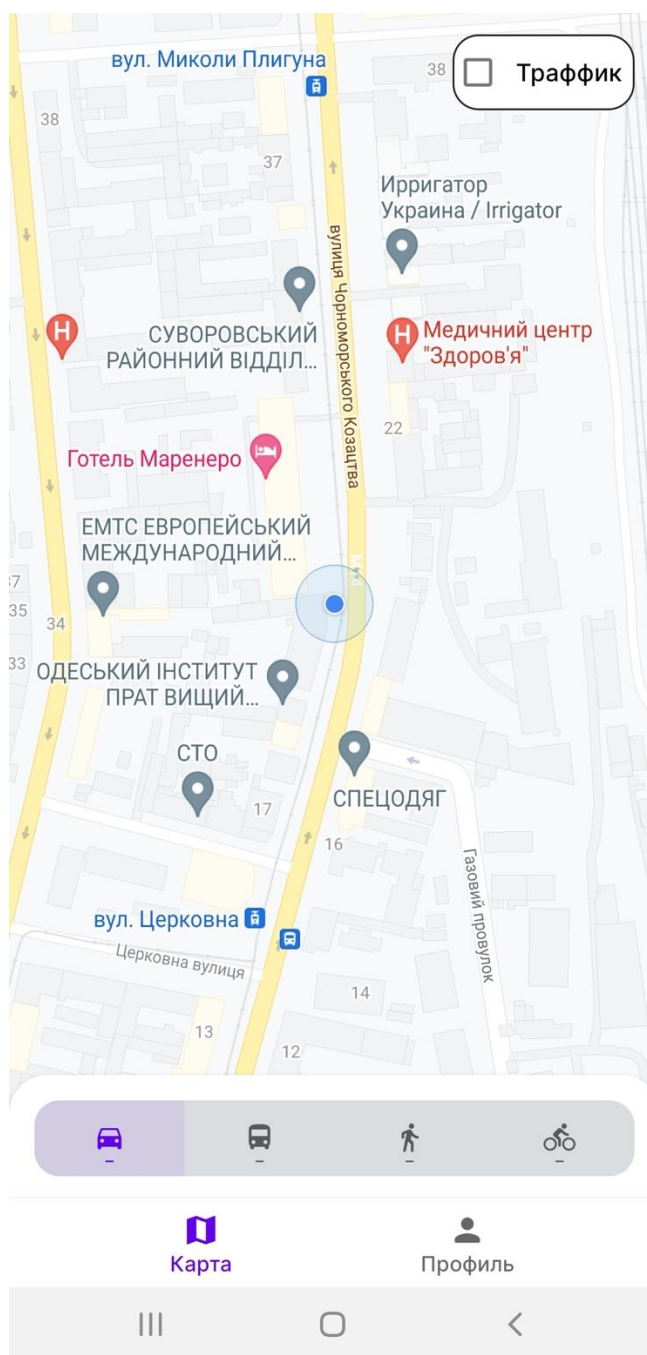


Рисунок 3.12 – Вікно головної частини програмного засобу

Алгоритм роботи трафіка дозволяє в реальному часі отримувати поточне навантаження на конкретні складові дороги, в залежності від обраних напрямків та маршрутів.

Такий алгоритм дозволяє оптимізувати роботу картографічної служби, а також зменшувати неточність при пошукових запитах в побудові маршрутів між точками. Для роботи алгоритма використовується спеціалізований набір системних бібліотек, рис. 3.13.

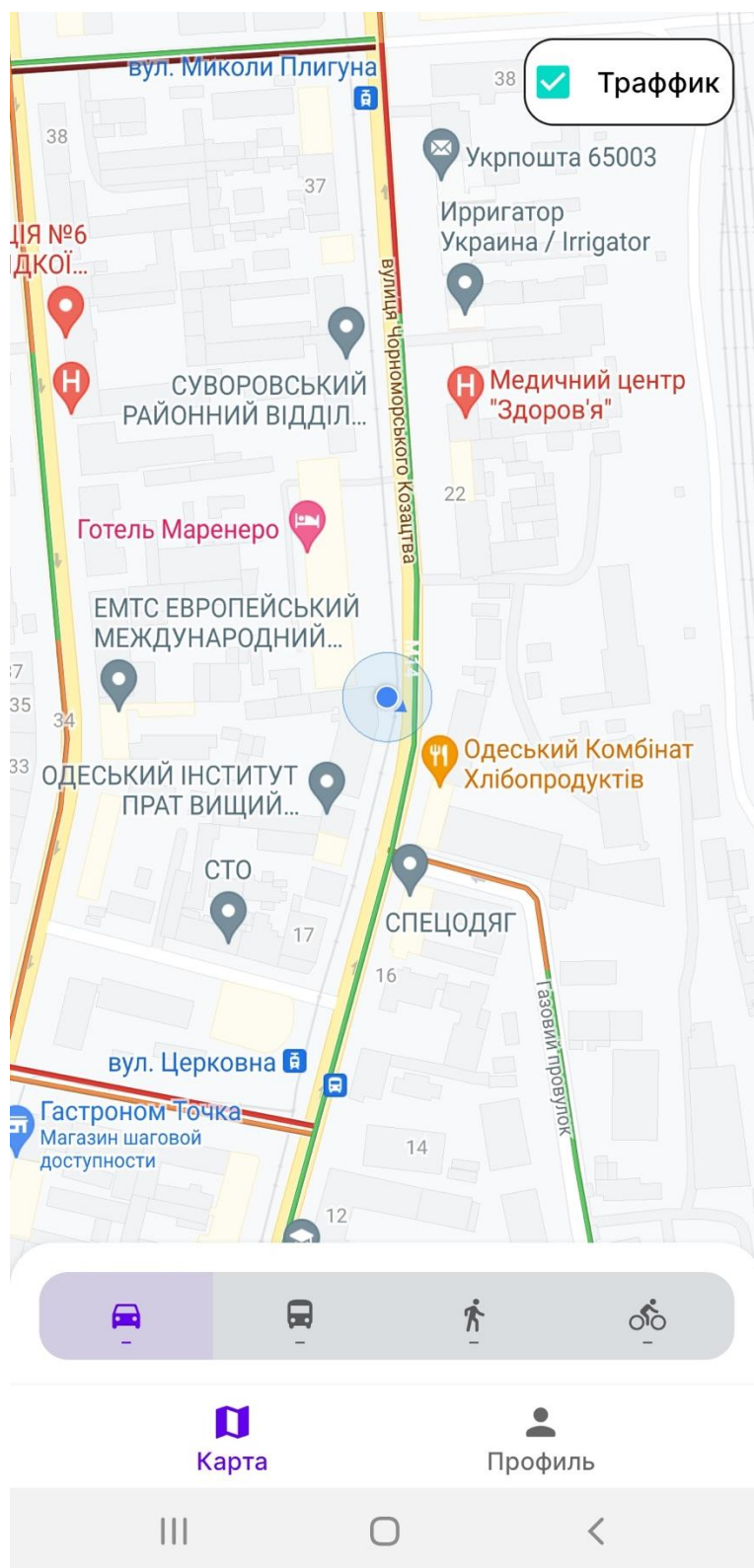


Рисунок 3.13 – Вікно додатку з відображенням трафіку

Користувач додатку має можливість додати до 10 точок для побудови маршрутів. Мобільний додаток буде обробляти усі додані точки та будувати

оптимально-допустимі маршрути які можливо використати для пересування тим транспортом – який обрано в системі. Нова точка додається простим тапом по екрану, після чого створюється перевіряєчне повідомлення стосовно того – чи бажає користувач додати нову точку для формування маршруту. Це забезпечує захист від помилок дій користувача та додатково перевіряє необхідну точку додання на мапу, рис. 3.14.

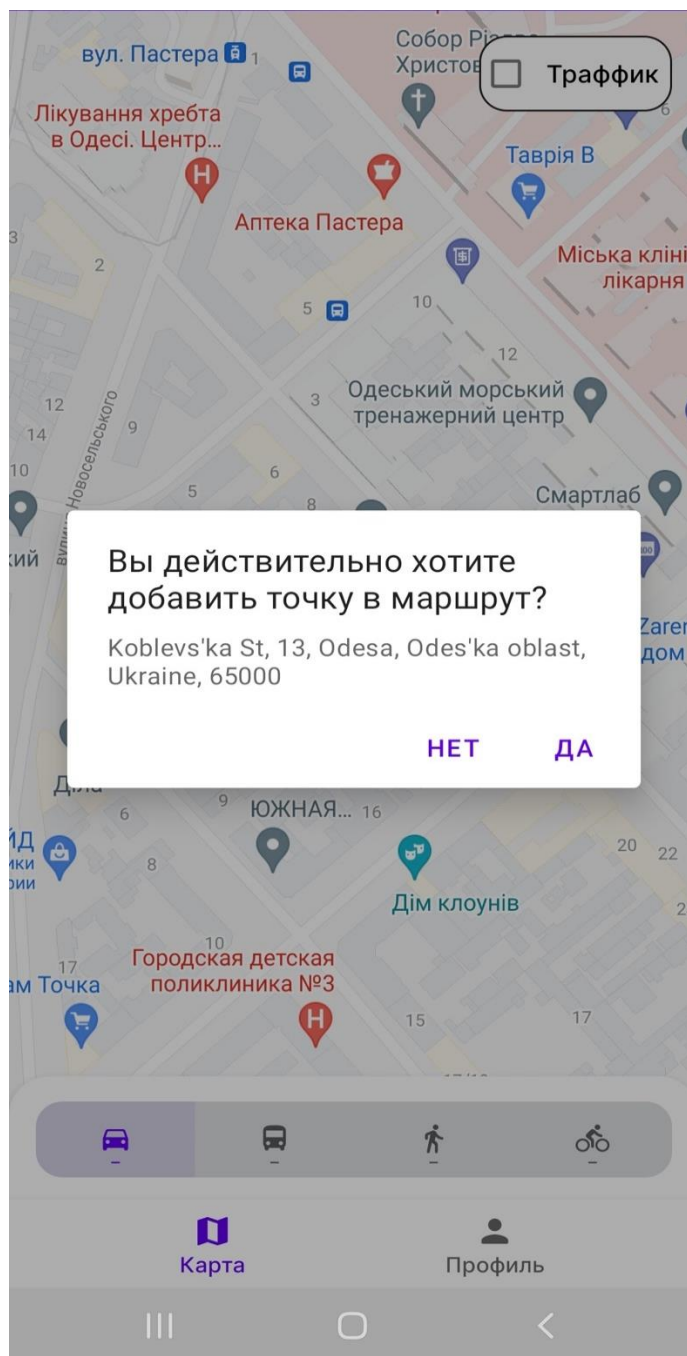


Рисунок 3.14 – Вікно мобільного додатку з доданням точки на мапу

Додана точка на мапі відображається червоним маркером. Такий в системі можливо додати тільки один, який використовується для визначення необхідної точки на мапі. Кожна точка є окремим елементом системи вхідних даних, для побудови та розрахунку ефективного маршруту, рис. 3.15.

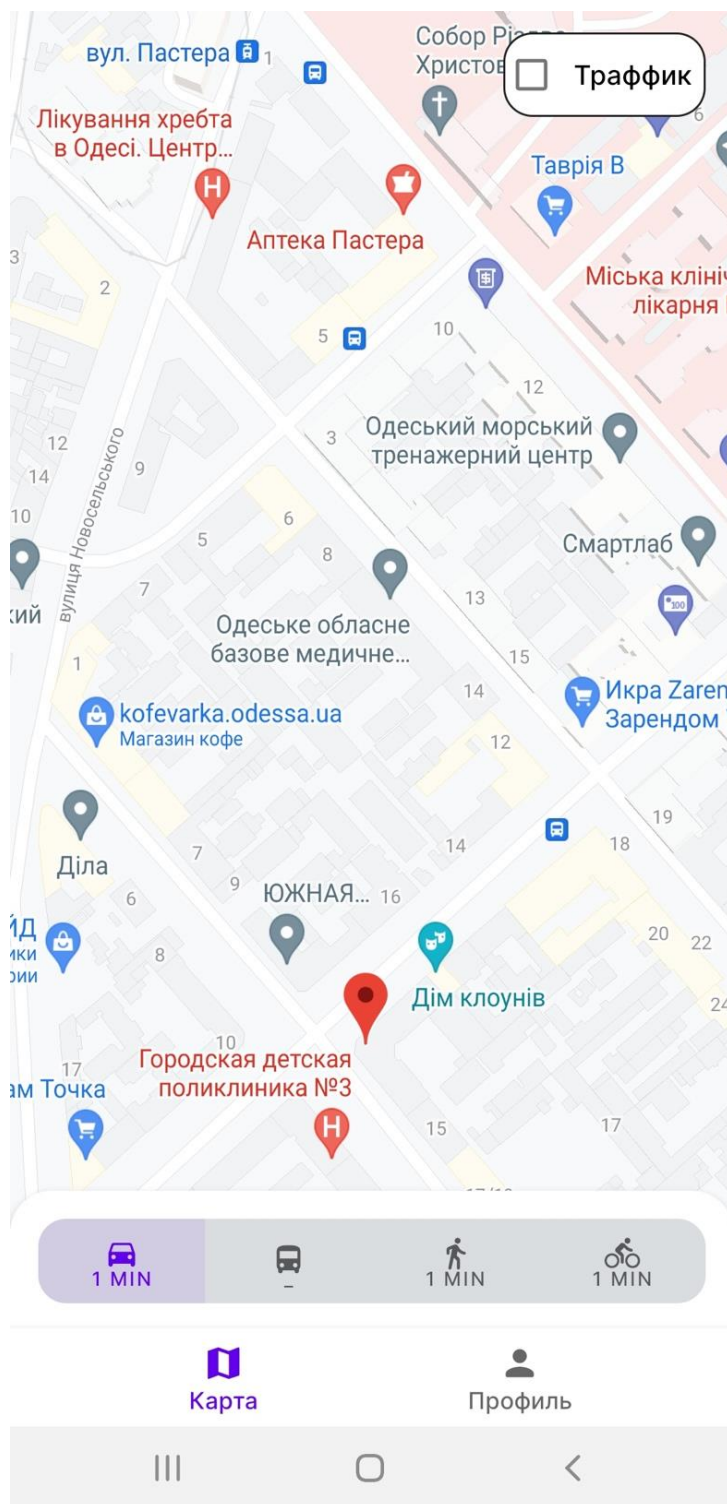


Рисунок 3.15 – Вікно доданої точки на мапу

Інформаційна система побудована таким чином, що формується прогноз на оптимальне переміщення з точки А в точку Б, засобами аналізу різних транспортних маршрутів. Якщо до обраної точки неможливо дістатися одним з транспортних засобів, то система не визначає час до цього місця. Одночасно розроблений додаток може обробити до 10 точок на мапі. На рис. 3.16 зображено вікно з маршрутом та розрахунком іншими типами транспортних засобів.



Рисунок 3.16 – Вікно додатку з маршрутами та часом

При створенні маршрутів, формується логіка пересування згідно можливих напрямків та особистостей місцевості. Алгоритм ґрунтується на роботі згідно таких зовнішніх факторів:

- Час поїздки.
- Завантаження кожної окремої ділянки.
- Врахування транспортних знаків.
- Відстань векторів на графі.
- Особистість обраного транспортного засобу.

Згідно кожного окремого та загального переліку входних факторів – формується оптимальний чи оптимальні маршрути руху, рис. 3.17.

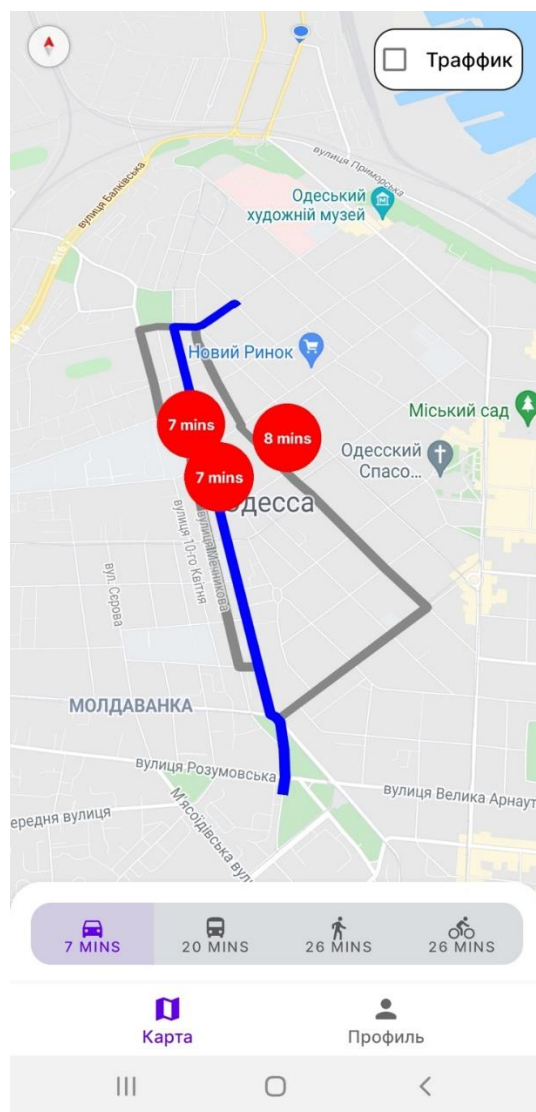


Рисунок 3.17 – Вікно додатку з сформованими маршрутами

При сформованому маршруті можливо відображати навантаження трафіку, яке має можливість корегувати оптимальні рішення та формувати різний час на поїздку, рис. 3.18.

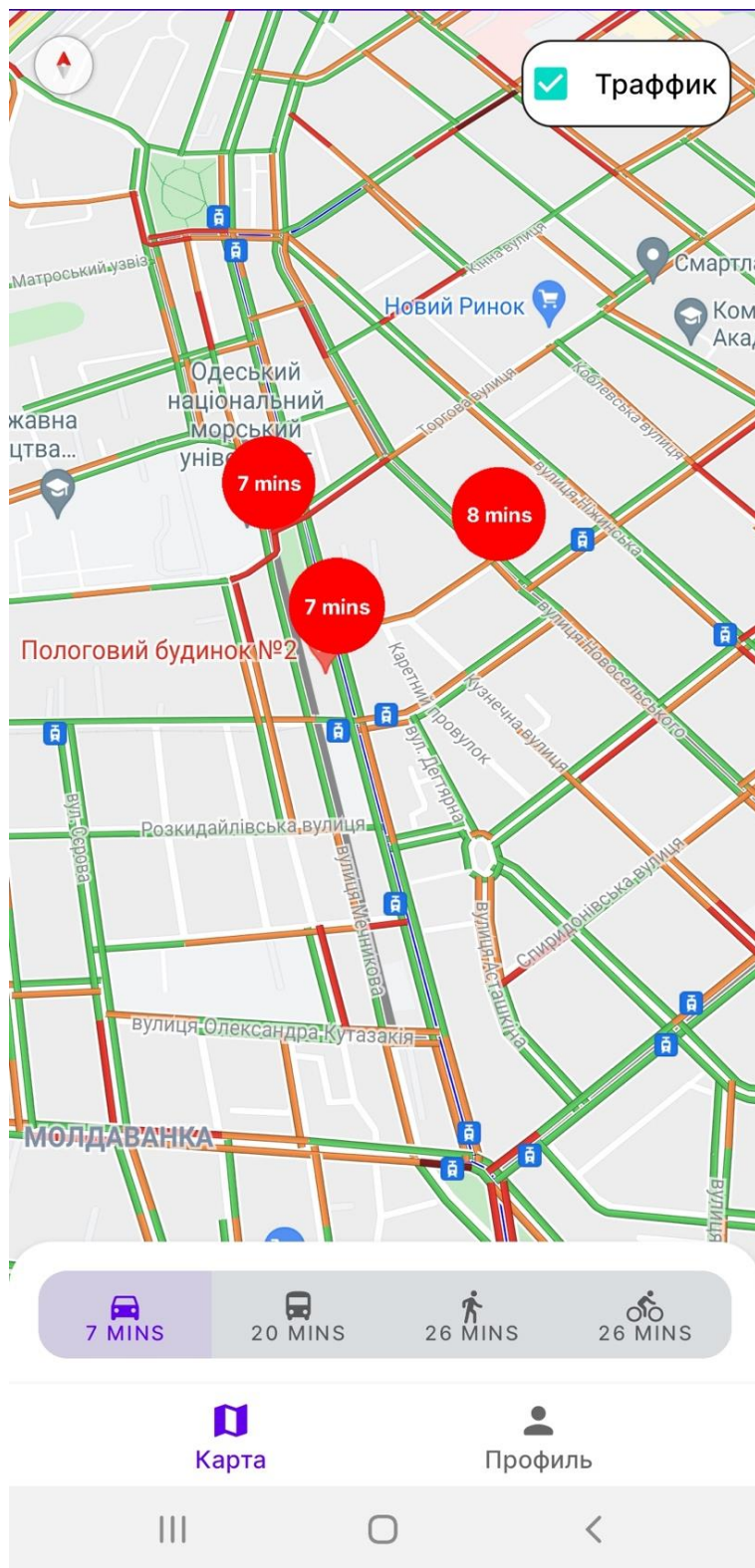


Рисунок 3.18 – Вікно виведення маршрутів з трафіком

Розроблене програмне рішення може використовуватися в якості платформи для побудови засобів динамічної навігації та визначенні оптимальних маршрутів. Мобільний додаток може бути інтегрованим до сучасних картографічних засобів, через виведення активних шарів та їх оптимізаційної структури до іншого програмного засобу.

3.3 Практичне використання розробленої системи

З метою перевірки працездатності розробленої інформаційної системи, слід виконати порівняння функціональних можливостей з існуючими програмними рішеннями та їх особистими алгоритмами побудови маршрутів та визначення орієнтованого часу на переїзд між заданими точками.

Для цього виконаємо порівняння:

- Розробленого рішення.
- Мобільного додатку Google Maps.
- Мобільного додатку 2GIS.
- Web версії Google Maps.

Такий набір дозволить точно визначити якість процедури побудови маршруту та його роботи з мапою.

При побудові маршруту встановимо дві однакові точки, які будуть використовуватися в усіх системах. Розроблена інформаційна система визначає оптимальний та ефективний маршрут згідно усіх зовнішніх факторів, які присутні на мапі в поточний період часу.

Також виконується аналіз трафіку, та система прогнозує час руху на інших типах транспорту, та додає відповідний маркер на головний екран користувача. На рис. 3.19 зображено розроблену інформаційну систему з побудованим маршрутом.

На різних пристроях, візуалізація та масштаб застосунку може відрізнятися, однак на функціональну складову компонентів та на працездатність це не впливає.

Мобільний додаток розроблено згідно усіх діючих стандартів та функціональних особливостей для платформи Android.

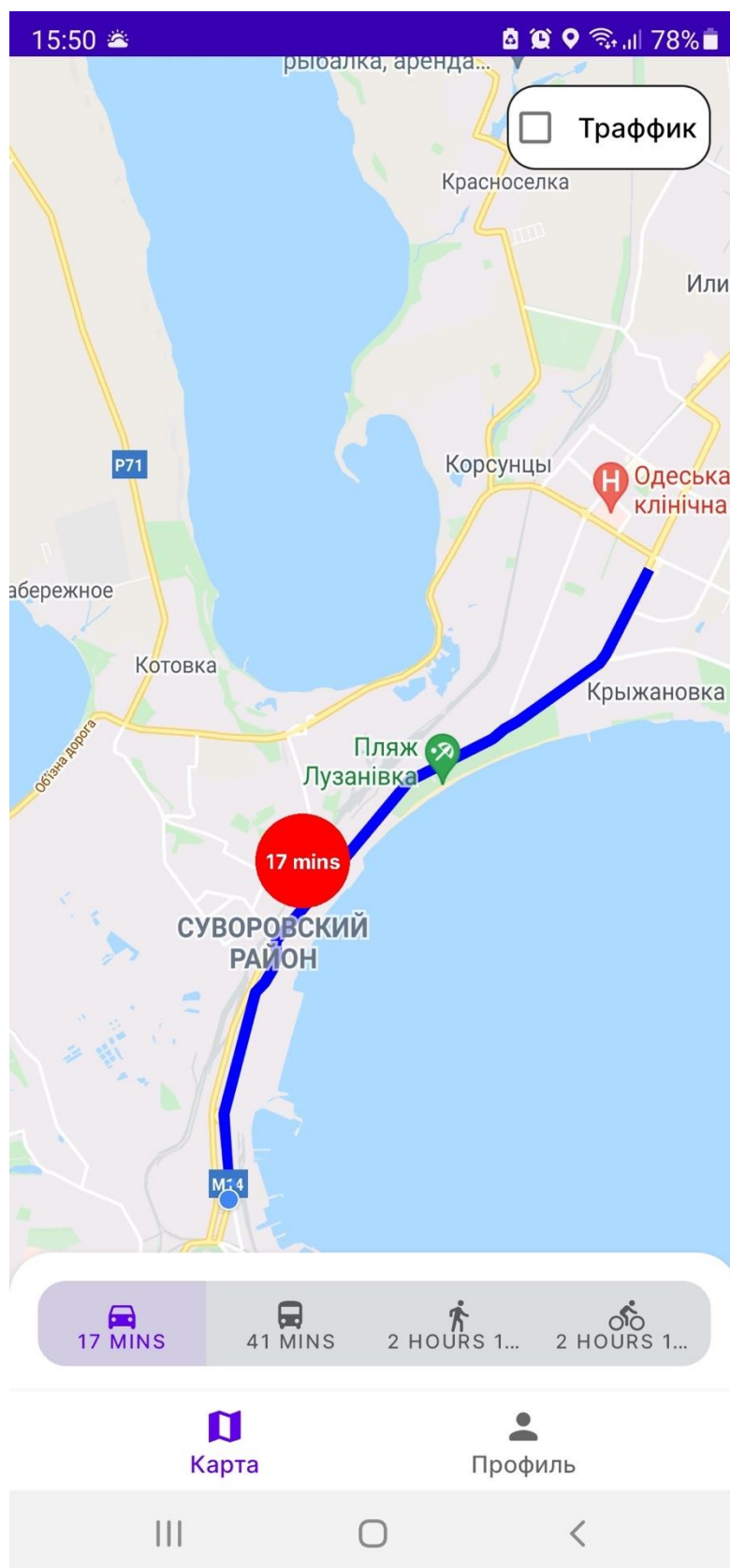


Рисунок 3.19 – Вікно з виведенням побудованого маршруту

Наступним етапом є створення аналогічного маршруту в програмному засобі Google Maps. Для цього алгоритм Google аналізує зовнішні фактори та транспорту ситуацію та формує свій варіант маршруту. Різниця часу при побудові маршрутів є 1 хвилина. Результат відрізняється на 2 хвилини в більшу сторону в інформаційному засобі Google для власного авто та на 3 хвилини для міського транспорту. На рис. 3.20 зображено вікно побудованого маршруту в засобі Google Maps.

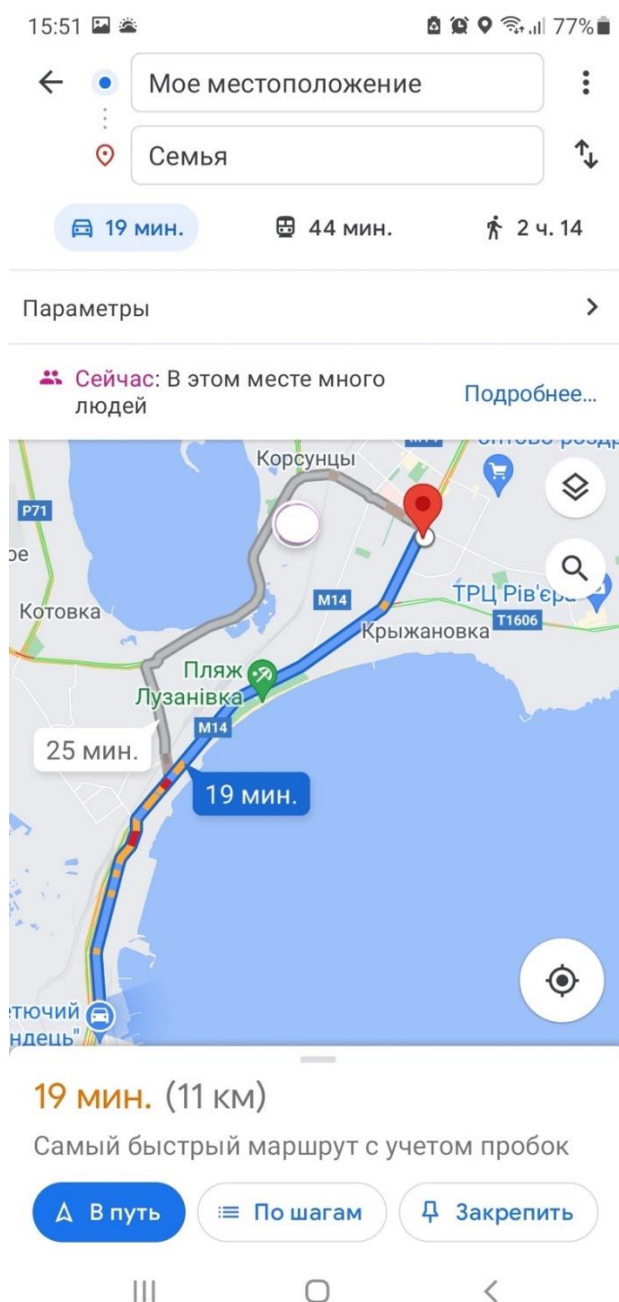


Рисунок 3.20 – Вікно побудованого маршруту засобом Google Maps

Іншим перевірочним засобом було обрано сервіс 2GIS який є як в мобільній версії, так й через web. Було обрано експериментальні точки маршруту та отримані аналогічні результати з розробленим програмним засобом. Також додаток 2GIS побудував додатковий маршрут, більшої довжини який не є ефективним в обраних умовах трафіка, рис. 3.21.

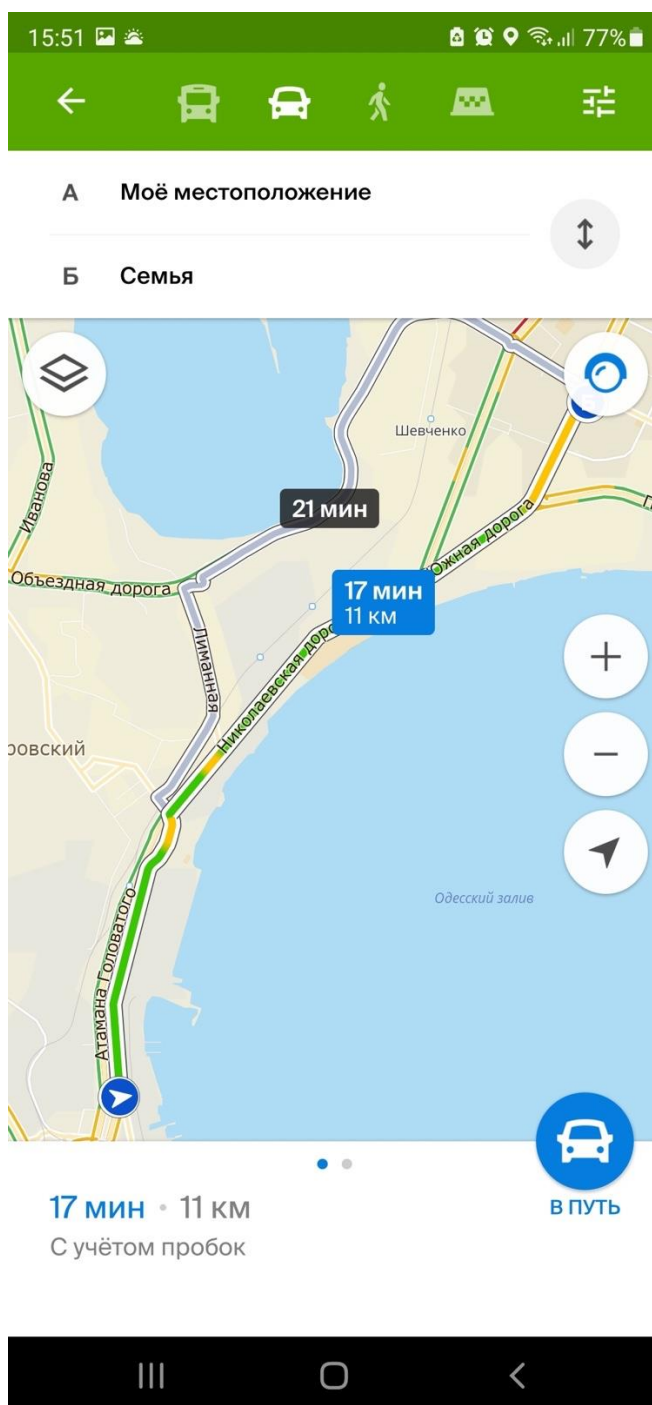


Рисунок 3.21 – Вікно побудованого маршруту в 2GIS

Останнім засобом перевірки ефективності роботи розробленого програмного рішення є побудова маршруту в web-версії застосунку Google Maps. Це дозволить визначити якість роботи алгоритму побудови маршруту без використання супутникових засобів геопозиції, а також перевірити якість побудови прогнозу часу на подолання маршруту. На рис. 3.22 зображено сторінку web-застосунку Google Maps з побудованим маршрутом.

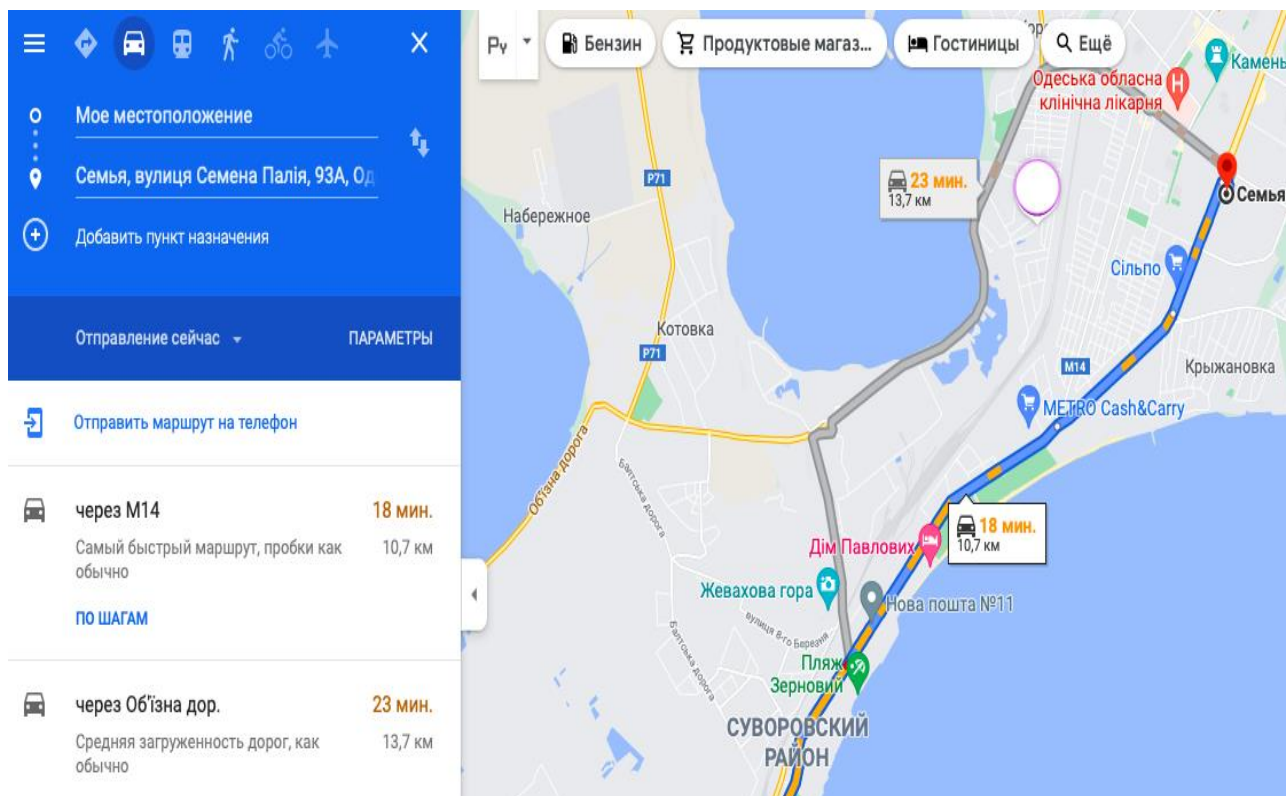


Рисунок 3.22 – Сторінка застосунку Google Maps з побудованим маршрутом

Результатом проведеного експерименту є перевірка ефективності роботи розробленого алгоритму побудови маршрутів та прогнозування часу на подолання такого маршруту. Провівши результативний експеримент з існуючими програмними засобами, було визначено, що алгоритм який використовується в розробленому програмному засобі є ефективним програмним рішенням, та може використовуватися як альтернативний метод побудови маршрутів в картографічних застосунках.

3.4 Висновок до третього розділу

У третьому розділі було спроектовано інформаційну систему з використанням сучасних стандартів мови UML та специфікацій з проектування та формування архітектурної документації. В програмний проект увійшли такі діаграми:

- діаграма варіантів використання;
- діаграма послідовності дії користувачів системи;
- діаграма станів поведінки системи;
- діаграма класів;
- діаграма розгортання інформаційної системи.

Кожна з діаграм дозволяє визначити головні складові проекту, а також сформувані задачі для розробки інформаційної системи.

Після визначення архітектурних вимог до інформаційної системи, було сформовані завдання для розробки та виконано процес написання програмного коду. Розроблена інформаційна система може використовуватися в якості самостійного програмного продукту чи взаємодіяти з іншим програмним забезпеченням. Готовий продукт є мобільним додатком, який може використовуватися на будь-яких пристроях під керуванням операційної системи Android.

ВИСНОВКИ

Результатом кваліфікаційної роботи є розробка програмного засобу з побудови оптимальних маршрутів згідно сформованих початкових умов користувача. Розроблена система використовує велику кількість вхідної інформації, яку аналізує та оброблює в реальному часі та зостосовує в якості додаткових умов при прогнозуванні подій.

Для розробки інформаційної системи було використано середовище Android Studio. Це є потужним засобом розробки сучасного програмного забезпечення, яке може використовуватися в різних напрямках ведення проектів. Вибір середовища розробки зумовлено й вибором мови розробки та набору спеціалізованих технологічних рішень, які використовуються в розробці інформаційної системи.

Для реалізації інформаційної системи використовується мова програмування Kotlin та фреймворк для розробки мобільних застосунків Flutter. Flutter використовується в якості засобу розробки інтерфейсної частини застосунку та оптимізації робочого простору на екрані користувача.

При виборі систем керування базами даних було обрано нереляційну СКБД Firestore Firebase. Основною причиною є відсутність чіткої структури даних, які потрібно зберігати на рівні бази даних. Структура буде залежати від задач, які користувач буде створювати в системі.

В кваліфікаційній роботі були виконані наступні задачі:

- виконано огляд сучасних рішень при вирішенні транспортних задач;
- виконано оцінку ефективності застосування транспортних задач в картографічних ситемах;
- виконано оцінку ефективності алгоритмів побудови оптимальних маршрутів;
- проведено порівняльний аналіз засобів розробки картографічних систем;
- спроектовано архітектуру програмної системи;

- виконано опис алгоритмічної складової розробленої інформаційної системи;
- виконано опис функціональних можливостей розробленого програмного рішення;
- проведено аналіз ефективності розробленого рішення;
- виконано розробку програмного засобу.

Розроблена інформаційна система може використовуватися в якості самостійного програмного продукту чи взаємодіяти з іншим програмним забезпеченням.

ПЕРЕЛІК ПОСИЛАНЬ

1. Айзерман Н. А. Выбор вариантов: основы теории [Текст] / Н.Л. Айзерман, Ф.Т. Алескерев. Наука,-М., 1990.-240 с.
2. Антошвили М.Е. Оптимизация городских автобусных перевозок / М.Е. Антошвили, С.Ю. Либерман, И.В. Спирин. - М.: Транспорт, 1985. - 102 с.
3. Баламирзоев А.Г. Оценка адекватности работы имитационной модели движения транспортного потока / А.Г. Баламирзоев, М.А. Султанахмедов // Журнал актуальной научной информации «Естественные и технические науки», 2007, № 3,с.235-238.
4. Беллман Р. Прикладные задачи динамического программирования / Р. Беллман, С. Дрейфус. - М.: Транспорт, 1965. - 458 с.
5. Большаков А.М. Повышение качества обслуживания пассажиров и эффективности работы автобусов / А.М. Большаков, Е.А. Кравченко, Л.С. Черникова. - М.: Транспорт, 1981.- 206 с.
6. Вентцель Е. С. Теория вероятностей [Текст]: учебник для студентов вузов / Е. С. Вентцель,-М.: Академия, 2005. -576 с
7. Гафт М. Г. Принятие решений при многих критериях [Текст] / М. Г.Гафт.-М.; Знание, 1979.-64 с.
8. Герами В,Д. Методология формирования системы городского пассажирского общественного транспорта. - М.: МАДИ, 2001.-313 с.
9. Гмурман В.Е. Теория вероятностей и математическая статистика. - М., Высш. шк.. 1998. - 479 с.
10. Григоров М.А. Проблемы моделирования и управления движением транспортных потоков в крупных городах / М.А. Григоров, А.Ф. Дашенко, А.В. Усов. Одесса,: Астропринт, 2004.-272 с.
11. Еремин В.М. Оценка практической применимости имитационной модели транспортного потока. -В кн.: Проектирование автомобильных дорог и безопасность движения. - М.: МАДИ, 1980, с.52-60.

12. Котиков Ю.Г. Основы теории транспортных систем. - СПб.: СПбГАСУ, 2000. - 216 с.
13. Лопатин А.П. Моделирование перевозочного процесса на городском пассажирском транспорте. - М.: Транспорт, 1985. - 144 с.
14. Нечепуренко М.И. Алгоритмы и программы решения задач на графах и сетях. - М.: Наука, 1990. - 245 с.
15. Лившица В.Н. Оптимизация планирования и управления транспортными системами. - М.: Транспорт, 1987. - 208 с.
16. Правдин Н.В. Прогнозирование пассажирских потоков (методика, расчеты, примеры) / Н.В. Правдин, В.Л. Негрей. - М.: Транспорт, 1980. - 222 с.
17. Берд, Барри Java для чайников / Барри Берд. - М.: Диалектика / Вильямс, 2013. - 521 с.
18. Монахов, В. Язык программирования Java и среда NetBeans (+ CD-ROM) / В. Монахов. - М.: БХВ-Петербург, 2012. - 720 с.
19. Шилдт, Герберт Java 8. Руководство для начинающих / Герберт Шилдт. - М.: Вильямс, 2015. - 720 с.
20. SQLite, MySQL и PostgreSQL: сравниваем популярные реляционные СУБД [Электронный ресурс]. – Режим доступа: <https://postgrespro.ru/docs/postgresql/9.6/> (дата доступа: 15.11.2019).
21. Яргер Р.Д. MySQL и msSQL. Базы данных для небольших предприятий и Интернета / Р.Д. Яргер. – М.: Символ-Плюс, 2010. – 929 с.
22. Уорсли Дж., Дрейк Дж. PostgreSQL. Для профессионалов / Дж. Уорсли, Дж. Дрейк — СПб.: Питер, 2003. — 496 с.
23. Стоунз Р. PostgreSQL. Основы / Р. Стоунз, Н. Мэттью — СПб.: Символ-Плюс, 2002. — 640 с.
24. Rocha G.D. Learning SQLite for iOS / G.D. Rocha // Packt Publishing, 2016. – 579p.
25. Owens M. The Definitive Guide to SQLite / M. Owens, G. Allen // Apress, 2010. – 739p.

ДОДАТОК А

Лістинг програмного коду

```

package com.example.wayprediction.ui.screen.map
import android.content.Context
import android.graphics.Color
import com.example.wayprediction.Constants.USERS_COLLECTION_NAME
import com.example.wayprediction.R
import com.example.wayprediction.entity.FavouriteTrip
import com.example.wayprediction.entity.geocoding.Results
import com.example.wayprediction.network.service.MapService
import com.example.wayprediction.ui.presenter.BasePresenter
import com.example.wayprediction.util.SharedPreferencesManager
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.model.LatLng
import com.google.android.gms.maps.model.Marker
import com.google.android.gms.maps.model.MarkerOptions
import com.google.maps.android.PolyUtil
import moxy.InjectViewState
import com.google.android.gms.maps.model.PolylineOptions
import com.google.firebase.firestore.FirebaseFirestore
import com.google.android.gms.maps.model.BitmapDescriptorFactory
import android.graphics.Bitmap
import android.graphics.Bitmap.Config.ARGB_8888
import android.graphics.Canvas
import android.graphics.Paint
import androidx.core.content.res.ResourcesCompat
import android.graphics.Rect
import
com.example.wayprediction.entity.directions.DirectionsResponseBody

@InjectViewState
class MapPresenter(
    private val mapService: MapService,
    private val sharedPreferencesManager: SharedPreferencesManager,
    private val firebaseFirestore: FirebaseFirestore
) : BasePresenter<MapView>() {

    private var googleMap: GoogleMap? = null
    private var mode: String = "driving"

    private val markers = mutableListOf<Marker>()

    private val mapAdapter = MapAdapter(::notifyPointRemoved)

    private var vehicleTrip: DirectionsResponseBody? = null
    private var busTrip: DirectionsResponseBody? = null

```

```

private var walkingTrip: DirectionsResponseBody? = null

override fun onFirstViewAttach() {
    super.onFirstViewAttach()
    viewState.setAdapter(mapAdapter)
}

fun notifyMapReady(googleMap: GoogleMap) {
    this.googleMap = googleMap
}

fun notifyBuildRouteButtonClicked(context: Context) {
    when (mode) {
        "driving" -> vehicleTrip
        "transit" -> busTrip
        else -> walkingTrip
    }.also {
        googleMap?.clear()
        viewState.hideBottomSheet()
        it?.routes?.sortedByDescending {
            it.legs.last().duration.value
        }.forEachIndexed { index, route ->
            val color = if (index == it.routes.lastIndex)
                Color.BLUE else Color.GRAY
            val options =
                PolylineOptions().width(20f).color(color)
            val decodedRoute =
                PolyUtil.decode(route.overview_polyline.points)
            decodedRoute.forEachIndexed { index, latLng ->
                if (index == decodedRoute.size / 2) {

                    val bmp = Bitmap.createBitmap(140, 140,
                        ARGB_8888)

                    val canvas = Canvas(bmp)
                    val paint = Paint()
                    paint.textSize = 30f
                    paint.color = Color.WHITE
                    val customTypeface =
                        ResourcesCompat.getFont(context,
                            R.font.inter_bold)

                    paint.typeface = customTypeface

                    canvas.drawCircle(
                        70f,
                        70f,
                        70f,
                        Paint().apply { this.color =
                            Color.RED })

                    val bounds = Rect()
                    paint.getTextBounds(
                        route.legs.last().duration.text,
                        0,

```



```

route.legs.last().duration.text.length,
                                bounds
                                )
                                val x = ((bmp.width - bounds.width()) /
2).toFloat()
                                val y = ((bmp.height + bounds.height())
/ 2).toFloat()

canvas.drawText(route.legs.last().duration.text, x, y, paint)

                                googleMap?.addMarker(
                                    MarkerOptions()
                                        .position(latLng)

                                .icon(BitmapDescriptorFactory.fromBitmap(bmp))
                                    )
                                    }
                                    options.add(latLng)
                                    }
                                googleMap?.addPolyline(options)
                                }
                                }

fun notifyMapPointSelected(mapPoint: LatLng) {
    launchIOCoroutine(
        { mapsService.getPlaceInfo(latLng =
mapPoint.toMapsString()) },
        {
viewState.showConfirmPointSelectionDialog(it?.results?.first()!!) },
        { it.printStackTrace() }
    )
}

fun notifyPositiveButtonDialogClicked(result: Results) {
    mapAdapter.addItem(result)
    markers.add(
        googleMap?.addMarker(
            MarkerOptions().position(
                LatLng(
                    result.geometry.location.lat,
                    result.geometry.location.lng
                )
            ).title(result.formatted_address)
        )!!
    )

    if
(sharedPreferencesManager.currentUser.favouriteTrips.contains(Favour
iteTrip(mapAdapter.points as MutableList<Results>)))
viewState.setIsInFavourites(

```

```

        R.drawable.round_star_24
    ) else
viewState.setIsInFavourites(R.drawable.round_star_border_24)

        getTrip("driving")
        getTrip("transit")
        getTrip("walking")
    }

    fun notifyDirectionModeSelected(modeButtonId: Int) {
        mode = when (modeButtonId) {
            R.id.directionsCarMaterialButton -> "driving"
            R.id.directionsBusMaterialButton -> "transit"
            else -> "walking"
        }
    }

    fun notifyAddToFavouritesClicked() {
        if (mapAdapter.points.isNotEmpty()) {
            if
(sharedPreferencesManager.currentUser.favouriteTrips.contains(
                FavouriteTrip(
                    mapAdapter.points as MutableList<Results>
                )
            )
        ) {
            sharedPreferencesManager.currentUser =
sharedPreferencesManager.currentUser.apply {
                favouriteTrips.removeAll {

sharedPreferencesManager.currentUser.favouriteTrips.contains(
                    FavouriteTrip(mapAdapter.points as
MutableList<Results>)
                )
            }
        }

viewState.setIsInFavourites(R.drawable.round_star_border_24)
        } else {
            sharedPreferencesManager.currentUser =
sharedPreferencesManager.currentUser.apply {
                favouriteTrips.add(
                    FavouriteTrip(mapAdapter.points as
MutableList<Results>)
                )
            }
        }

viewState.setIsInFavourites(R.drawable.round_star_24)
    }

    firebaseFirestore.collection(USERS_COLLECTION_NAME)
        .document(sharedPreferencesManager.currentUser.id)
        .set(sharedPreferencesManager.currentUser)

```

```

    }
}

private fun notifyPointRemoved(result: Results) {
    if
    (sharedPreferencesManager.currentUser.favouriteTrips.contains(FavouriteTrip(
        mapAdapter.points as MutableList<Results>)))
    viewState.setIsInFavourites(
        R.drawable.round_star_24
    ) else
    viewState.setIsInFavourites(R.drawable.round_star_border_24)
    markers.first { it.title == result.formatted_address
    }.remove()
}

private fun getTrip(mode: String) {
    launchIOCoroutine(
        {
            mapsService.getRouteInfo(
                origin =
                "place_id:${mapAdapter.points.first().place_id}",
                destination =
                "place_id:${mapAdapter.points.last().place_id}",
                waypoints = if (mapAdapter.points.size > 2) {
                    mapAdapter.points
                        .subList(1, mapAdapter.points.lastIndex)
                        .joinToString(
                            separator = "|place_id:",
                            prefix = "place_id:"
                        ) { it.place_id }
                } else {
                    ""
                },
                mode = mode
            )
        },
        {
            when (mode) {
                "driving" -> {
                    vehicleTrip = it

                    viewState.setVehicleTime(it?.routes?.minByOrNull {
                        it.legs.last().duration.value }!!.legs.last().duration.text)
                }
                "transit" -> {
                    busTrip = it
                    viewState.setBusTime(it?.routes?.minByOrNull
                    { it.legs.last().duration.value }!!.legs.last().duration.text)
                }
                else -> {
                    walkingTrip = it
                }
            }
        }
    )
}

```

```

viewState.setWalkingTime(it?.routes?.minByOrNull {
it.legs.last().duration.value }!!.legs.last().duration.text)
        }
    },
    { it.printStackTrace() }
)
}

private fun LatLng.toMapsString() = "$latitude, $longitude"
}
package com.example.wayprediction.ui.screen.profile

import android.content.Intent
import android.os.Bundle
import android.view.View
import androidx.recyclerview.widget.LinearLayoutManager
import com.example.wayprediction.R
import com.example.wayprediction.ui.fragment.BaseFragment
import
com.example.wayprediction.ui.screen.authentication.AuthenticationAct
ivity
import
kotlinx.android.synthetic.main.fragment_profile.favouriteRoutesRecyc
lerView
import
kotlinx.android.synthetic.main.fragment_profile.logOutMaterialButton
import
kotlinx.android.synthetic.main.fragment_profile.userEmailTitledTextV
iew
import
kotlinx.android.synthetic.main.fragment_profile.userNameTitledTextVi
ew
import
kotlinx.android.synthetic.main.fragment_profile.userRoleTitledTextVi
ew
import moxy.presenter.InjectPresenter
import moxy.presenter.ProvidePresenter
import org.koin.android.ext.android.get

class ProfileFragment : BaseFragment(), ProfileView {

    @InjectPresenter
    lateinit var profilePresenter: ProfilePresenter

    @ProvidePresenter
    fun provideProfilePresenter() = get<ProfilePresenter>()

    override fun getLayoutResId() = R.layout.fragment_profile

```

```

        override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
            super.onViewCreated(view, savedInstanceState)

            logOutMaterialButton.setOnClickListener {
                profilePresenter.notifyLogOutButtonClicked()
            }

            favouriteRoutesRecyclerView.layoutManager =
                LinearLayoutManager(context)
        }

        override fun setUsername(userName: String) {
            userNameTitledTextView.value = userName
        }

        override fun setUserEmail(userEmail: String) {
            userEmailTitledTextView.value = userEmail
        }

        override fun setUserRole(userRole: String) {
            userRoleTitledTextView.value = userRole
        }

        override fun setAdapter(mapAdapter: FavouritesAdapter) {
            favouriteRoutesRecyclerView.adapter = mapAdapter
        }

        override fun showAuthenticationActivity() {
            startActivity(Intent(requireActivity(),
                AuthenticationActivity::class.java))
            requireActivity().finish()
        }
    }
}

package com.example.wayprediction.ui.screen.profile

import com.example.wayprediction.Constants
import com.example.wayprediction.entity.FavouriteTrip
import com.example.wayprediction.entity.User
import com.example.wayprediction.util.SharedPreferencesManager
import com.google.firebase.firestore.FirebaseFirestore
import moxy.InjectViewState
import moxy.MvpPresenter

@InjectViewState
class ProfilePresenter(
    private val sharedPreferencesManager: SharedPreferencesManager,
    private val firebaseFirestore: FirebaseFirestore
) : MvpPresenter<ProfileView>() {

    override fun onFirstViewAttach() {
        super.onFirstViewAttach()
    }

```

```

        with(sharedPreferencesManager.currentUser) {
            viewState.setUserName(fullName)
            viewState.setUserEmail(login)
            viewState.setUserRole(
                listOf(
                    "Автомобиль",
                    "Автобус",
                    "Ноги",
                    "Велосипед"
                ) [personnelTransport]
            )
        }

viewState.setAdapter(FavouritesAdapter(::notifyTripRemoved).apply {

sharedPreferencesManager.currentUser.favouriteTrips.forEach {
    addItem(it)
}
})
}

private fun notifyTripRemoved(result: FavouriteTrip) {
    sharedPreferencesManager.currentUser =
sharedPreferencesManager.currentUser.apply {
    favouriteTrips.removeAll { it.points.toTypedArray() contentEquals
result.points.toTypedArray() } }

firebaseFirestore.collection(Constants.USERS_COLLECTION_NAME)
    .document(sharedPreferencesManager.currentUser.id)
    .set(sharedPreferencesManager.currentUser)
}

fun notifyLogoutButtonClicked() {
    sharedPreferencesManager.isUserAuthorized = false
    sharedPreferencesManager.currentUser = User()
    viewState.showAuthenticationActivity()
}

}

package com.example.wayprediction.ui.screen.map

import android.Manifest.permission
import android.annotation.SuppressLint
import android.app.Activity
import android.content.Intent
import android.content.IntentSender
import android.os.Bundle
import android.view.View
import androidx.core.app.ActivityCompat
import com.example.wayprediction.R
import com.example.wayprediction.entity.geocoding.Results

```

```

import com.example.wayprediction.ui.fragment.BaseFragment
import
com.example.wayprediction.util.extensions.checkLocationPermissions
import com.example.wayprediction.util.extensions.isLocationAvailable
import com.google.android.gms.common.api.ResolvableApiException
import com.google.android.gms.location.LocationRequest
import com.google.android.gms.location.LocationServices
import com.google.android.gms.location.LocationSettingsRequest
import com.google.android.gms.maps.CameraUpdateFactory
import com.google.android.gms.maps.GoogleMap
import com.google.android.gms.maps.OnMapReadyCallback
import com.google.android.gms.maps.SupportMapFragment
import com.google.android.gms.maps.model.LatLng
import com.google.android.material.bottomsheet.BottomSheetBehavior
import com.google.android.material.dialog.MaterialAlertDialogBuilder
import
kotlinx.android.synthetic.main.fragment_map.addTripToFavouritesImage
Button
import
kotlinx.android.synthetic.main.fragment_map.addedPointsRecyclerView
import
kotlinx.android.synthetic.main.fragment_map.buildTripMaterialButton
import
kotlinx.android.synthetic.main.fragment_map.directionsBikeMaterialBu
tton
import
kotlinx.android.synthetic.main.fragment_map.directionsBusMaterialBut
ton
import
kotlinx.android.synthetic.main.fragment_map.directionsCarMaterialBut
ton
import
kotlinx.android.synthetic.main.fragment_map.directionsWalkMaterialBu
tton
import
kotlinx.android.synthetic.main.fragment_map.drivingModeMaterialButto
nToggleGroup
import
kotlinx.android.synthetic.main.fragment_map.showTrafficMaterialCheck
Box
import
kotlinx.android.synthetic.main.fragment_map.tripPanelConstraintLayou
t
import moxy.presenter.InjectPresenter
import moxy.presenter.ProvidePresenter
import org.koin.android.ext.android.get

class MapFragment : BaseFragment(), MapView, OnMapReadyCallback {

    @InjectPresenter
    lateinit var mapPresenter: MapPresenter

    @ProvidePresenter

```

```

fun provideMapPresenter() = get<MapPresenter>()

private var bottomSheetPlaceBehavior: BottomSheetBehavior<*>? =
null

override fun getLayoutResId() = R.layout.fragment_map

override fun onCreateView(view: View, savedInstanceState:
Bundle?) {
    super.onCreateView(view, savedInstanceState)

    val mapFragment =
childFragmentManager.findFragmentById(R.id.mapFragmentContainerView)
as SupportMapFragment?
    mapFragment?.getMapAsync(this)

drivingModeMaterialButtonToggleGroup.addOnButtonCheckedListener { _,
checkedId, _ ->
    mapPresenter.notifyDirectionModeSelected(checkedId)
}

    bottomSheetPlaceBehavior =
BottomSheetBehavior.from(tripPanelConstraintLayout).apply {
        state = BottomSheetBehavior.STATE_HIDDEN
    }

    buildTripMaterialButton.setOnClickListener {

mapPresenter.notifyBuildRouteButtonClicked(requireContext())
    }
    addTripToFavouritesImageButton.setOnClickListener {
        mapPresenter.notifyAddToFavouritesClicked()
    }
}

@SuppressLint("MissingPermission")
override fun onMapReady(googleMap: GoogleMap) {
    if (checkLocationPermissions()) {

LocationServices.getFusedLocationProviderClient(requireActivity()).l
astLocation.addOnCompleteListener {
    if (!isLocationAvailable()) {
        requestLocationSettings()
    } else if (it.result != null) {
        googleMap.animateCamera(
            CameraUpdateFactory.newLatLngZoom(
                LatLng(
                    it.result.latitude,
                    it.result.longitude
                ),
                17f
            ), null
        )
    }
}

```



```

        )

        googleMap.isMyLocationEnabled = true
        googleMap.uiSettings.isMyLocationButtonEnabled =
false
        googleMap.setOnMapClickListener { latLng ->
            mapPresenter.notifyMapPointSelected(
                latLng
            )
        }
        mapPresenter.notifyMapReady(googleMap)

        showTrafficMaterialCheckBox.setOnCheckedChangeListener { _, b ->
            googleMap.isTrafficEnabled = b
        }
    }
}
else {
    requestLocationSettings()
}
}

private fun requestLocationSettings() {
    ActivityCompat.requestPermissions(
        requireActivity(),
        arrayOf(permission.ACCESS_FINE_LOCATION),
        1
    )
}

override fun onActivityResult(requestCode: Int, resultCode: Int,
data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode==1 && resultCode != Activity.RESULT_OK){
        val mapFragment =
childFragmentManager.findFragmentById(R.id.mapFragmentContainerView)
as SupportMapFragment?
        mapFragment?.getMapAsync(this)
    }
}

override fun showConfirmPointSelectionDialog(result: Results) {
    MaterialAlertDialogBuilder(requireContext())
        .setTitle("Вы действительно хотите добавить точку в
маршрут?")
        .setMessage(result.formatted_address)
        .setPositiveButton("Да") { _, _ ->
mapPresenter.notifyPositiveButtonDialogClicked(result) }
        .setNegativeButton("Нет", null)
        .show()
}

```



```

        .addOnCompleteListener {
            viewState.dismissDialog()
        }
    }

    fun notifyTaskDateAutoCompleteTextViewTouched(event: MotionEvent)
    {
        if (event.action == MotionEvent.ACTION_UP)
        viewState.showTaskDatePickerDialog()
    }

    fun notifyTaskDatePickerPositiveButtonClicked(selectedTime: Long)
    {
        selectedTaskLocalDate = selectedTime.toLocalDate()
    }
}

package com.example.bedbooking.ui.screen.add_tourist

import android.graphics.drawable.Drawable
import android.net.Uri
import android.view.MotionEvent
import android.view.MotionEvent.ACTION_UP
import com.example.bedbooking.Constants
import com.example.bedbooking.Constants.TOURISTS_COLLECTION_NAME
import com.example.bedbooking.entity.Booking
import com.example.bedbooking.entity.Tourist
import
com.example.bedbooking.ui.screen.add_tourist.phone_numbers.PhoneNumb
ersAdapter
import
com.example.bedbooking.util.DateTimeFormatters.Companion.SIMPLE_DATE
_FORMATTER
import com.example.bedbooking.util.extension.toByteArray
import
com.example.bedbooking.util.extension.toEpochMilliAtDefaultZone
import com.example.bedbooking.util.extension.toLocalDate
import com.google.firebase.firestore.FirebaseFirestore
import com.google.firebase.firestore.auth.User
import com.google.firebase.storage.FirebaseStorage
import moxy.InjectViewState
import moxy.MvpPresenter
import org.threeten.bp.LocalDate

@InjectViewState
class AddTouristPresenter(
    private val firebaseFirestore: FirebaseFirestore,
    private val firebaseStorage: FirebaseStorage,
    private val userId: String?
) : MvpPresenter<AddTouristView>() {

    private val phoneNumbersAdapter = PhoneNumbersAdapter()

```

```

private var selectedLocalDate: LocalDate? = null
    set(value) {
        viewState.setSelectedDate(value?.format(SIMPLE_DATE_FORMATTER)!!)
        field = value
    }

override fun onFirstViewAttach() {
    super.onFirstViewAttach()

    viewState.setAdapter(phoneNumbersAdapter)
    selectedLocalDate = LocalDate.now()

    if (userId != null) {
        firebaseFirestore.collection(TOURISTS_COLLECTION_NAME)
            .document(userId)
            .get()
            .addOnCompleteListener {
                val user =
it.result?.toObject(Tourist::class.java)!!
                viewState.setupUser(user)
                notifyDatePickerPositiveButtonClicked(user.birthday)

                phoneNumbersAdapter.phoneNumbers.addAll(user.phoneNumbers)

                phoneNumbersAdapter.notifyDataSetChanged()
            }
    }
}

fun notifyUserBirthdayAutoCompleteTextViewTouched(event: MotionEvent) {
    if (event.action == ACTION_UP)
viewState.showDatePickerDialog()
}

fun notifyDatePickerPositiveButtonClicked(selectedTime: Long) {
    selectedLocalDate = selectedTime.toLocalDate()
}

fun notifyAddPhoneEndIconClicked(phoneNumber: String) {
    phoneNumbersAdapter.addPhoneNumber(phoneNumber)
}

fun notifyAddTouristButtonClicked(tourist: Tourist,
userImageDrawable: Drawable) {
    pushImageInStorage(tourist, userImageDrawable)
}

private fun pushImageInStorage(tourist: Tourist,
userImageDrawable: Drawable) {
    val id = userId ?:
firebaseFirestore.collection(TOURISTS_COLLECTION_NAME).document().id

```

```

        val storageRef = firebaseStorage.reference.child("Users
images").child("$id.jpeg")

        storageRef.putBytes(userImageDrawable.toByteArray())
            .addOnCompleteListener {
                storageRef.downloadUrl.addOnSuccessListener {
uri: Uri ->

                    tourist.id = id
                    tourist.photoUrl = uri.toString()
                    tourist.birthday =
selectedLocalDate?.toEpochMilliAtDefaultZone()!!
                    tourist.phoneNumbers =
phoneNumbersAdapter.phoneNumbers

                    firebaseFirestore.collection(TOURISTS_COLLECT
ION_NAME)
                        .document(id)
                        .set(tourist)
                        .addOnCompleteListener {
                            viewState.dismissDialog()
                        }
                }
            }
    }

    fun notifyDeleteTouristMaterialButtonClicked() {
        firebaseFirestore.collection(TOURISTS_COLLECTION_NAME)
            .document(userId!!)
            .delete()
            .addOnCompleteListener {
                viewState.dismissDialog()
            }
    }
}

package com.example.bedbooking.ui.screen.book_room

import android.view.MotionEvent
import android.view.MotionEvent.ACTION_UP
import com.example.bedbooking.Constants.BOOKINGS_COLLECTION_NAME
import com.example.bedbooking.Constants.FOOD_COLLECTION_NAME
import com.example.bedbooking.Constants.ROOMS_COLLECTION_NAME
import com.example.bedbooking.Constants.TOURISTS_COLLECTION_NAME
import com.example.bedbooking.entity.Booking
import com.example.bedbooking.entity.Food
import com.example.bedbooking.entity.Room
import com.example.bedbooking.entity.Tourist
import com.example.bedbooking.util.DateTimeFormatters
import
com.example.bedbooking.util.extension.toEpochMilliAtDefaultZone
import com.example.bedbooking.util.extension.toLocalDate

```

```

import com.google.firebase.firestore.FirebaseFirestore
import moxy.InjectViewState
import moxy.MvpPresenter
import org.threeten.bp.LocalDate
import org.threeten.bp.Period

@InjectViewState
class BookRoomPresenter(
    private val firebaseFirestore: FirebaseFirestore,
    private val bookingId: String?
) : MvpPresenter<BookRoomView>() {

    var selectedRoomId: String = ""

    var selectedTouristId: String = ""

    var selectedFoodId: String = ""

    private val tourists = mutableListOf<Tourist>()

    private val rooms = mutableListOf<Room>()

    private val food = mutableListOf<Food>()

    private var selectedCheckInLocalDate: LocalDate? =
LocalDate.now()
        set(value) {
            viewState.setSelectedCheckInDate(value?.format(DateTimeFo
rmatters.SIMPLE_DATE_FORMATTER)!!)
            field = value
            notifyDateSelected()
        }

    private var selectedDepartureLocalDate: LocalDate? =
LocalDate.now()
        set(value) {
            viewState.setSelectedDepartureDate(value?.format(DateTime
Formatters.SIMPLE_DATE_FORMATTER)!!)
            field = value
            notifyDateSelected()
        }

    override fun onFirstViewAttach() {
        super.onFirstViewAttach()

        firebaseFirestore.collection(ROOMS_COLLECTION_NAME).addSnapsh
otListener { value, error ->
            rooms.clear()
            rooms.addAll(value!!.toObjects(Room::class.java))

            selectedRoomId = rooms.first().id
            viewState.setRoomsAdapter(rooms.map { it.name })
        }
    }

```

```

        firebaseFirestore.collection(TOURISTS_COLLECTION_NAME).addSnapshotListener { value, error ->
            tourists.clear()
            tourists.addAll(value!!.toObjects(Tourist::class.java))

            selectedTouristId = tourists.first().id
            viewState.setTouristsAdapter(tourists.map { it.fullName })
        })
    }

    firebaseFirestore.collection(FOOD_COLLECTION_NAME).addSnapshotListener { value, error ->
        food.clear()
        food.addAll(value!!.toObjects(Food::class.java))

        selectedFoodId = food.first().id
        viewState.setFoodAdapter(food.map { it.name })

        selectedCheckInLocalDate = LocalDate.now()
        selectedDepartureLocalDate = LocalDate.now()
    }

    if (bookingId != null) {
        firebaseFirestore.collection(BOOKINGS_COLLECTION_NAME)
            .document(bookingId)
            .get()
            .addOnCompleteListener {
                val booking =
it.result?.toObject(Booking::class.java)!!
                viewState.setupBooking(booking)
                notifyCheckInDatePickerPositiveButtonClicked(
booking.startDate)
                notifyDepartureDatePickerPositiveButtonClicked(
booking.endDate)
                viewState.setTouristName(booking.touristName)
                viewState.setRoomName(booking.roomName)
                viewState.setFoodName(booking.foodName)
                selectedTouristId = booking.touristId
                selectedRoomId = booking.roomId
                selectedFoodId = booking.foodId
            }
    }
}

private fun notifyDateSelected() {
    val daysCount = Period.between(selectedCheckInLocalDate,
selectedDepartureLocalDate).days + 1
    viewState.setDaysCount(daysCount.toString())

    viewState.setTotalPrice((daysCount * rooms.first { it.id ==
selectedRoomId }.priceForDay.toInt() + daysCount * food.first {
it.id == selectedFoodId }.price.toInt()).toString())
}

```

```

    }

    fun notifyCheckInDatePickerPositiveButtonClicked(selectedTime:
Long) {
        selectedCheckInLocalDate = selectedTime.toLocalDate()
    }

    fun notifyDepartureDatePickerPositiveButtonClicked(selectedTime:
Long) {
        selectedDepartureLocalDate = selectedTime.toLocalDate()
    }

    fun notifyShowUserInfoButtonClicked() {
        viewState.showUserInfoDialog(selectedTouristId)
    }

    fun notifyBookRoomMaterialButtonClicked(
        daysCount: Int,
        prepayment: Float,
        prepaymentCurrency: String,
        discount: Float,
        balance: Float
    ) {
        val id = bookingId ?:
firebaseFirestore.collection(BOOKINGS_COLLECTION_NAME).document().id

        firebaseFirestore.collection(BOOKINGS_COLLECTION_NAME)
            .document(id)
            .set(
                Booking(
                    id,
                    selectedCheckInLocalDate!!.toEpochMil
liAtDefaultZone(),
                    selectedDepartureLocalDate!!.toEpochM
illiAtDefaultZone(),
                    selectedTouristId,
                    tourists.first { it.id ==
selectedTouristId }.fullName,
                    selectedRoomId,
                    rooms.first { it.id == selectedRoomId
}.name,
                    daysCount,
                    prepayment,
                    selectedFoodId,
                    food.first { it.id == selectedFoodId
}.name,
                    true,
                    discount,
                    balance,
                    prepayment + balance
                )
            )
        .addOnCompleteListener {

```



```

        viewState.dismissDialog()
    }
}

fun notifyDeleteBookingMaterialButtonClicked(bookingId: String) {
    firebaseFirestore.collection(BOOKINGS_COLLECTION_NAME)
        .document(bookingId)
        .delete()
        .addOnCompleteListener {
            viewState.dismissDialog()
        }
}

fun notifyRoomSelected(position: Int) {
    selectedRoomId = rooms[position].id
}

fun notifyFoodSelected(position: Int) {
    selectedFoodId = food[position].id

    val daysCount = Period.between(selectedCheckInLocalDate,
selectedDepartureLocalDate).days + 1

    viewState.setTotalPrice((daysCount * rooms.first { it.id ==
selectedRoomId }.priceForDay.toInt() + daysCount * food.first {
it.id == selectedFoodId }.price.toInt()).toString())
}

fun notifyUserSelected(position: Int) {
    selectedTouristId = tourists[position].id
}

fun notifyCheckInDateAutoCompleteTextViewTouched(event:
MotionEvent) {
    if (event.action == ACTION_UP)
viewState.showCheckInDatePickerDialog()
}

fun notifyDepartureDateAutoCompleteTextViewTouched(event:
MotionEvent) {
    if (event.action == ACTION_UP)
viewState.showDepartureDatePickerDialog()
}
}
package com.example.bedbooking.ui.screen.day_info

import com.example.bedbooking.Constants.BOOKINGS_COLLECTION_NAME
import com.example.bedbooking.entity.Booking
import
com.example.bedbooking.util.extension.toEpochMilliAtDefaultZone
import com.google.firebase.firestore.FirebaseFirestore
import moxy.InjectViewState

```

```

import moxy.MvpPresenter
import org.threeten.bp.LocalDate

@InjectViewState
class DayInfoPresenter(
    private val firebaseFirestore: FirebaseFirestore,
    private val date: LocalDate
) : MvpPresenter<DayInfoView>() {

    private val bookingsAdapter = BookingsAdapter()

    override fun onFirstViewAttach() {
        super.onFirstViewAttach()

        bookingsAdapter.bookingCallback = bookingCallback
        viewState.setAdapter(bookingsAdapter)

        val bookings = mutableListOf<Booking>()

        firebaseFirestore.collection(BOOKINGS_COLLECTION_NAME)
            .whereEqualTo("startDate",
date.toEpochMilliAtDefaultZone())
            .get()
            .addOnCompleteListener {
                bookings.addAll(it.result?.toObjects(Booking::cla
ss.java)!!)

                firebaseFirestore.collection(BOOKINGS_COLLECTION_
NAME)
                    .whereEqualTo("endDate",
date.toEpochMilliAtDefaultZone())
                    .get()
                    .addOnCompleteListener {
                        bookings.addAll(it.result?.toObjects(
Booking::class.java)!!)
                        bookingsAdapter.setData(bookings)
                    }
            }
    }

    private val bookingCallback = object : BookingCallback {
        override fun onBookingClick(bookingId: String) {
            viewState.openBookingInfo(bookingId)
        }
    }
}

package com.example.bedbooking.ui.screen.rooms_list

import com.example.bedbooking.Constants.BOOKINGS_COLLECTION_NAME
import com.example.bedbooking.Constants.ROOMS_COLLECTION_NAME
import com.example.bedbooking.R
import com.example.bedbooking.entity.Room

```

```

import
com.example.bedbooking.util.extension.toEpochMilliAtDefaultZone
import com.google.firebase.firestore.FirebaseFirestore
import moxy.InjectViewState
import moxy.MvpPresenter
import org.threeten.bp.LocalDate
import org.threeten.bp.LocalDateTime

@InjectViewState
class RoomsListPresenter(
    private val firebaseFirestore: FirebaseFirestore,
    private val isFree: Boolean
) : MvpPresenter<RoomsListView>() {

    private val roomsAdapter = RoomsAdapter()

    override fun onFirstViewAttach() {
        super.onFirstViewAttach()

        roomsAdapter.roomListItemCallback = roomListItemCallback
        viewState.setAdapter(roomsAdapter)
        getAvailableRooms()
    }

    private fun getAvailableRooms() {
        val currentDateTimeInMillis =
LocalDate.now().toEpochMilliAtDefaultZone()

        firebaseFirestore.collection(BOOKINGS_COLLECTION_NAME)
            .get()
            .addOnCompleteListener {
                if (it.result!!.isEmpty) {
                    if (isFree) {
                        viewState.hideEmptyListMessage()

                        firebaseFirestore.collection(ROOMS_COLLEC
TION_NAME)
                            .addSnapshotListener { value,
error ->
                                roomsAdapter.setData(value?.t
oObjects(Room::class.java)!!)
                            }

                    } else {
                        viewState.showEmptyListMessage(R.string.a
ll_rooms_free)
                    }
                } else {
                    firebaseFirestore.collection(BOOKINGS_COLLECT
ION_NAME)
                        .whereLessThanOrEqualTo("startDate",
currentDateTimeInMillis)
                            .get()

```

```

        .addOnCompleteListener {
            if (it.isSuccessful) {
                val bookings =
it.result!!.documents
                    .filter { document ->
document.getLong("endDate")!! >= currentDateTimeInMillis }
                    .map { document ->
document.getString("roomId") }

                if (bookings.isNotEmpty()) {
                    if (isFree) {
                        firebaseFirestore.col
lection(ROOMS_COLLECTION_NAME).whereNotIn("id", bookings)
                            .addSnapshotL
istener { value, error ->
                                roomsAdap
ter.setData(value?.toObjects(Room::class.java)!!)
                            }
                    } else {
                        viewState.hideEmptyLi
stMessage()
                        firebaseFirestore.col
lection(ROOMS_COLLECTION_NAME).whereIn("id", bookings)
                            .addSnapshotL
istener { value, error ->
                                roomsAdap
ter.setData(value?.toObjects(Room::class.java)!!)
                            }
                    }
                } else {
                    if (isFree) {
                        viewState.showEmptyLi
stMessage(R.string.no_free_rooms)
                    } else {
                        viewState.showEmptyLi
stMessage(R.string.all_rooms_free)
                    }
                }
            }
        }
    }
}

private val roomListItemCallback = object : RoomListItemCallback
{
    override fun onRoomClick(roomId: String) {
        viewState.showRoomInfo(roomId)
    }
}

}
package com.example.bedbooking.ui.screen.statistics

```

```

import com.example.bedbooking.Constants.BOOKINGS_COLLECTION_NAME
import com.example.bedbooking.entity.Booking
import
com.example.bedbooking.util.extension.toEpochMilliAtDefaultZone
import com.example.bedbooking.util.extension.toLocalDate
import com.google.firebase.firestore.FirebaseFirestore
import moxy.InjectViewState
import moxy.MvpPresenter
import org.threeten.bp.LocalDate
import org.threeten.bp.Year

@InjectViewState
class StatisticsPresenter(
    private val firebaseFirestore: FirebaseFirestore
) : MvpPresenter<StatisticsView>() {

    override fun onFirstViewAttach() {
        super.onFirstViewAttach()

        firebaseFirestore.collection(BOOKINGS_COLLECTION_NAME)
            .whereLessThanOrEqualTo("endDate",
LocalDate.now().toEpochMilliAtDefaultZone())
            .get()
            .addOnCompleteListener {
                val bookings = mutableListOf<Float>()

                bookings.addAll(
                    it.result?.toObjects(Booking::class.java)
                        ?.filter {
Year.from(it.startDate.toLocalDate()).value >= Year.now().value }
                        ?.map { it.summary }!!
                )

                viewState.setCurrentYearIncomes(bookings.sum().to
String())
            }
    }
}

package com.example.bedbooking.ui.screen.tasks

import com.example.bedbooking.Constants.TASKS_COLLECTION_NAME
import com.example.bedbooking.entity.Task
import com.google.firebase.firestore.FirebaseFirestore
import moxy.InjectViewState
import moxy.MvpPresenter

@InjectViewState
class TasksPresenter(
    private val firebaseFirestore: FirebaseFirestore
) : MvpPresenter<TasksView>() {

```

```
private val tasksAdapter = TasksAdapter()

override fun onCreateViewAttach() {
    super.onCreateViewAttach()

    viewState.setAdapter(tasksAdapter)

    firebaseFirestore.collection(TASKS_COLLECTION_NAME)
        .orderBy("date")
        .addSnapshotListener { value, error ->
            tasksAdapter.setData(value?.toObjects(Task::class
.java)!!)
        }
    }
```

ДОДАТОК Б

Слайди презентації

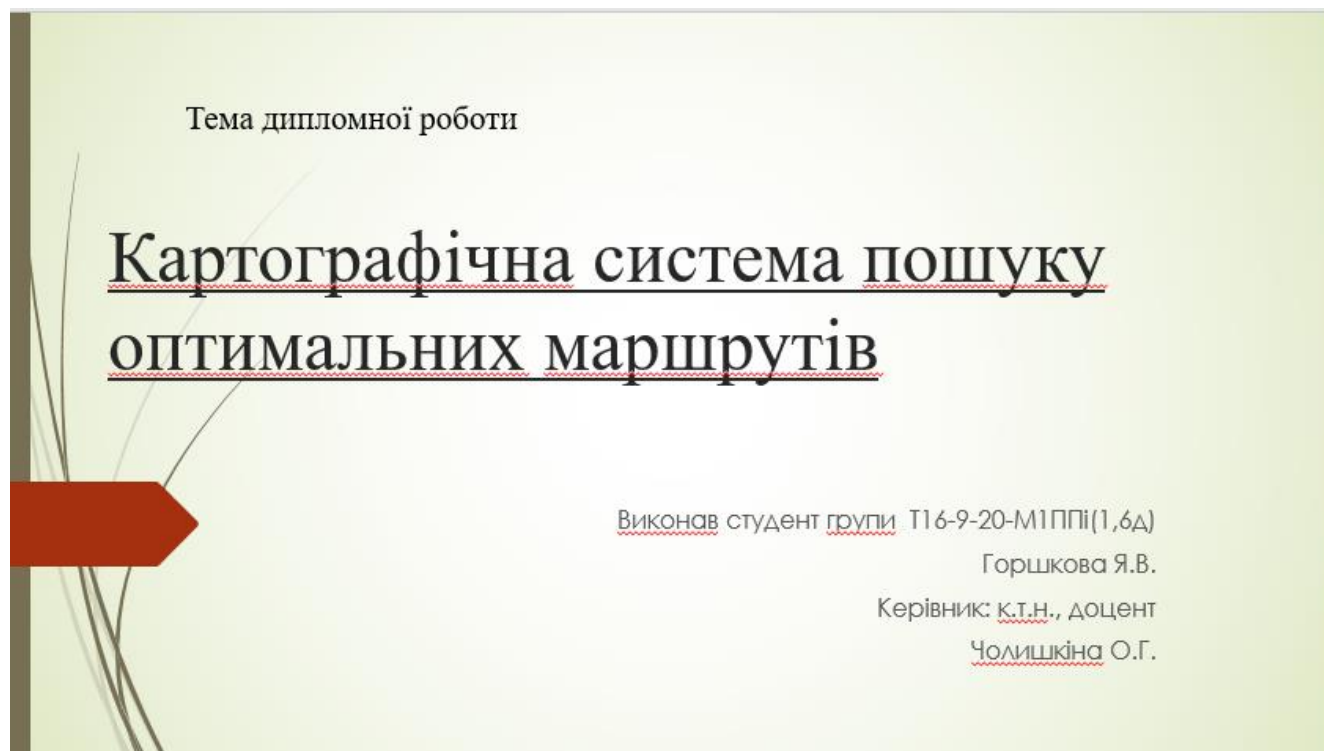


Рисунок Б1 – Титульний слайд презентації

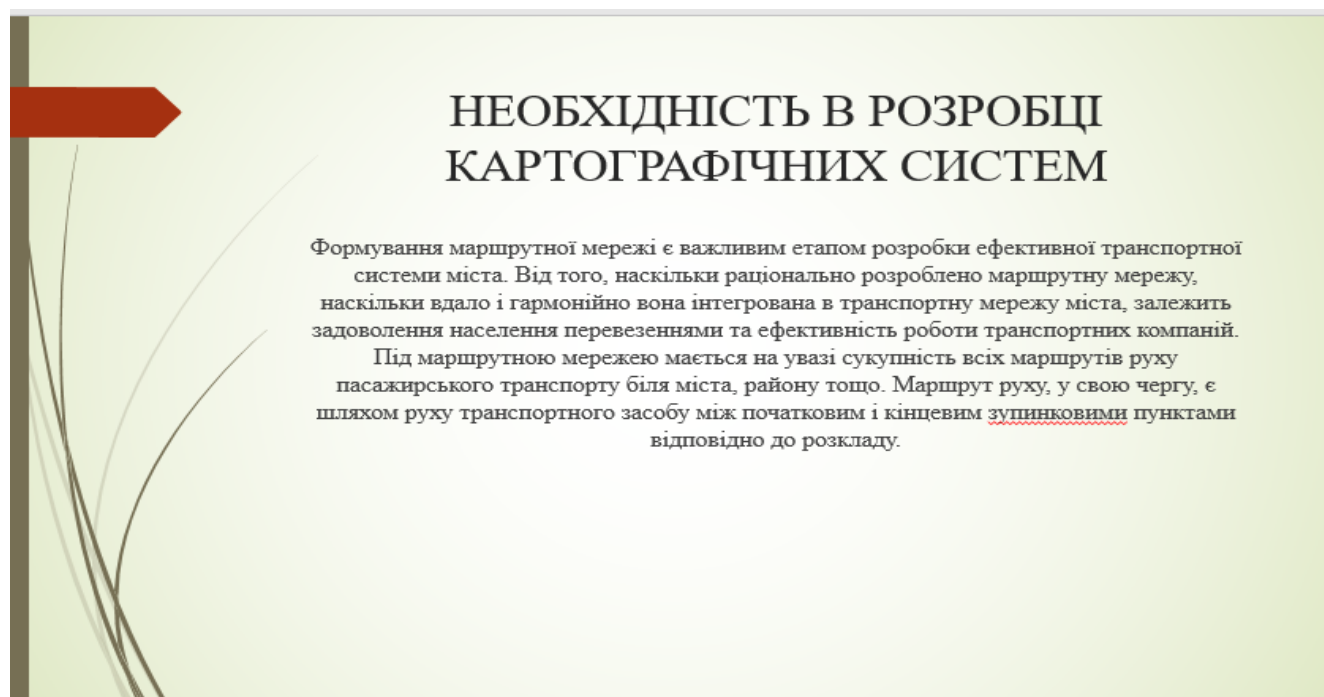
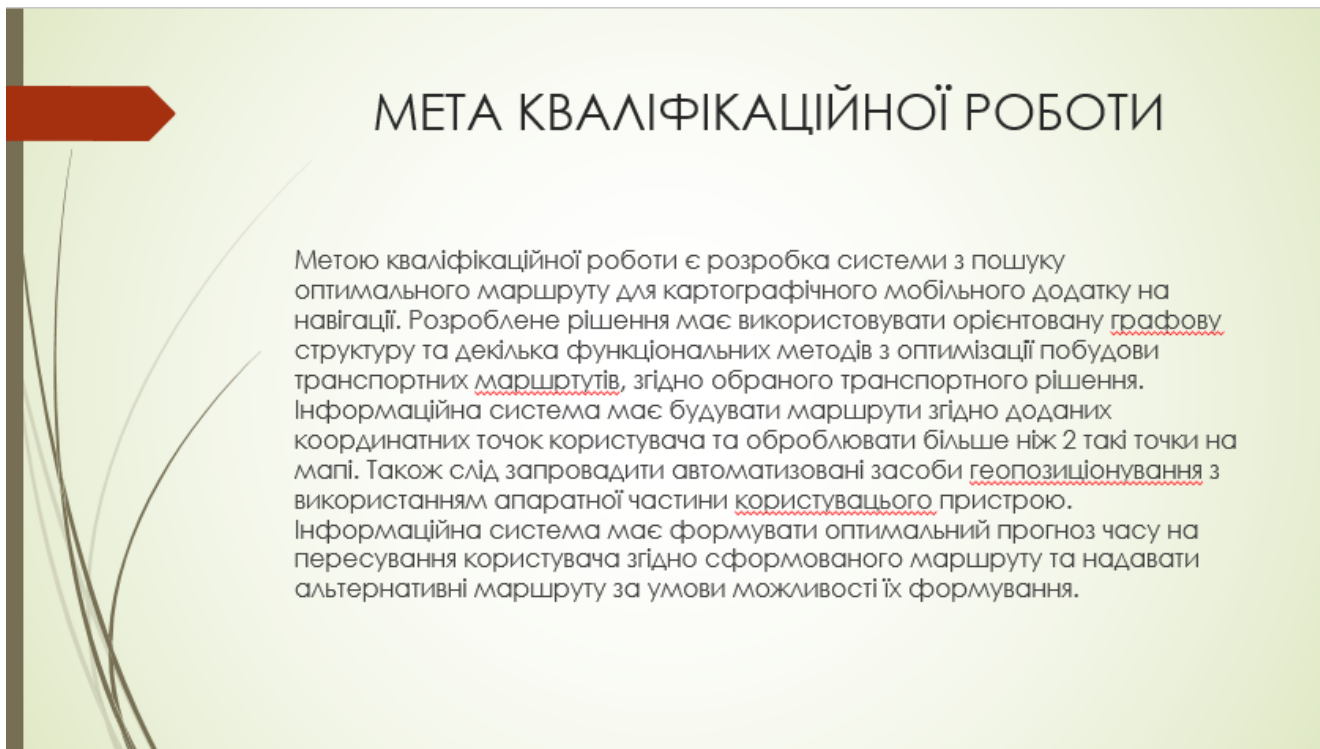


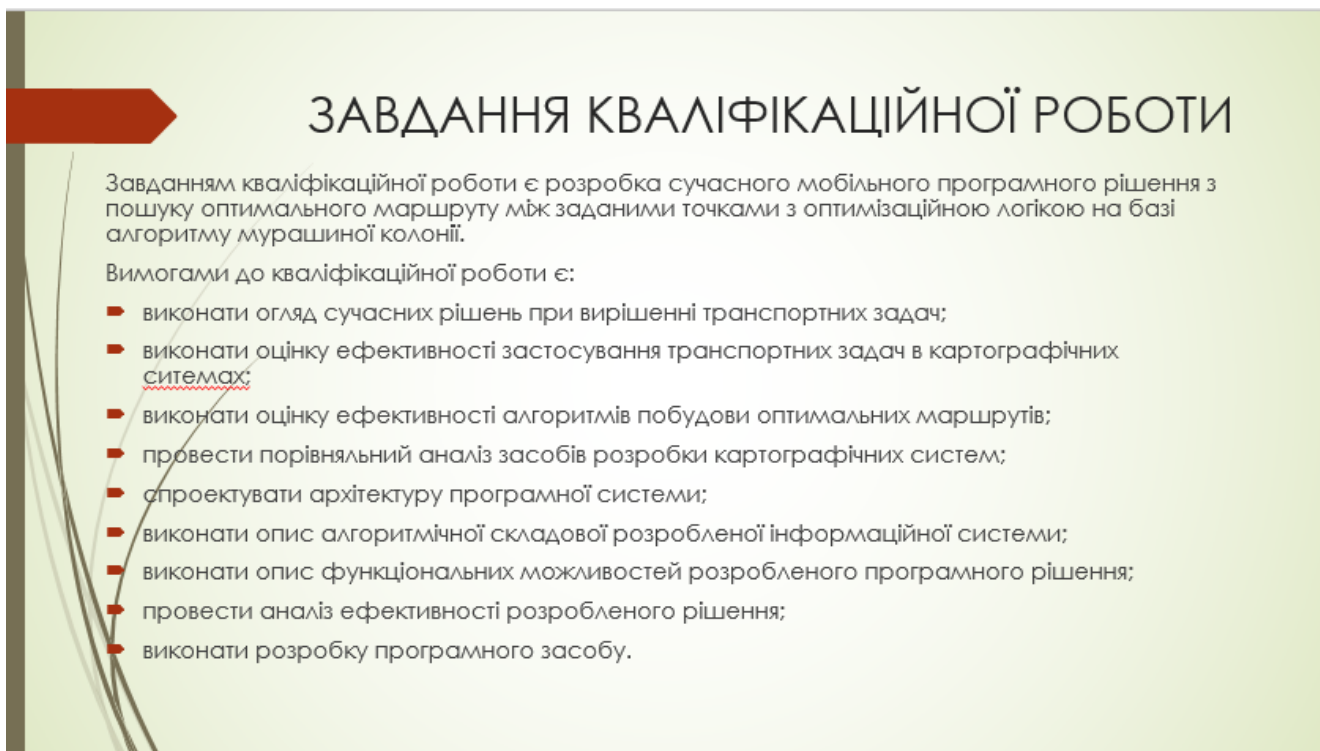
Рисунок Б2 – Слайд з інформацією про необхідність розробки інформаційної системи



МЕТА КВАЛІФІКАЦІЙНОЇ РОБОТИ

Метою кваліфікаційної роботи є розробка системи з пошуку оптимального маршруту для картографічного мобільного додатку на навігації. Розроблене рішення має використовувати орієнтовану графову структуру та декілька функціональних методів з оптимізації побудови транспортних маршрутів, згідно обраного транспортного рішення. Інформаційна система має будувати маршрути згідно доданих координатних точок користувача та оброблювати більше ніж 2 такі точки на мапі. Також слід запровадити автоматизовані засоби геопозиціонування з використанням апаратної частини користувачього пристрою. Інформаційна система має формувати оптимальний прогноз часу на пересування користувача згідно сформованого маршруту та надавати альтернативні маршруту за умови можливості їх формування.

Рисунок Б3 – Слайд з метою роботи



ЗАВДАННЯ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Завданням кваліфікаційної роботи є розробка сучасного мобільного програмного рішення з пошуку оптимального маршруту між заданими точками з оптимізаційною логікою на базі алгоритму мурашиної колонії.

Вимогами до кваліфікаційної роботи є:

- виконати огляд сучасних рішень при вирішенні транспортних задач;
- виконати оцінку ефективності застосування транспортних задач в картографічних системах;
- виконати оцінку ефективності алгоритмів побудови оптимальних маршрутів;
- провести порівняльний аналіз засобів розробки картографічних систем;
- спроектувати архітектуру програмної системи;
- виконати опис алгоритмічної складової розробленої інформаційної системи;
- виконати опис функціональних можливостей розробленого програмного рішення;
- провести аналіз ефективності розробленого рішення;
- виконати розробку програмного засобу.

Рисунок Б4 – Слайд з задачами дипломної роботи

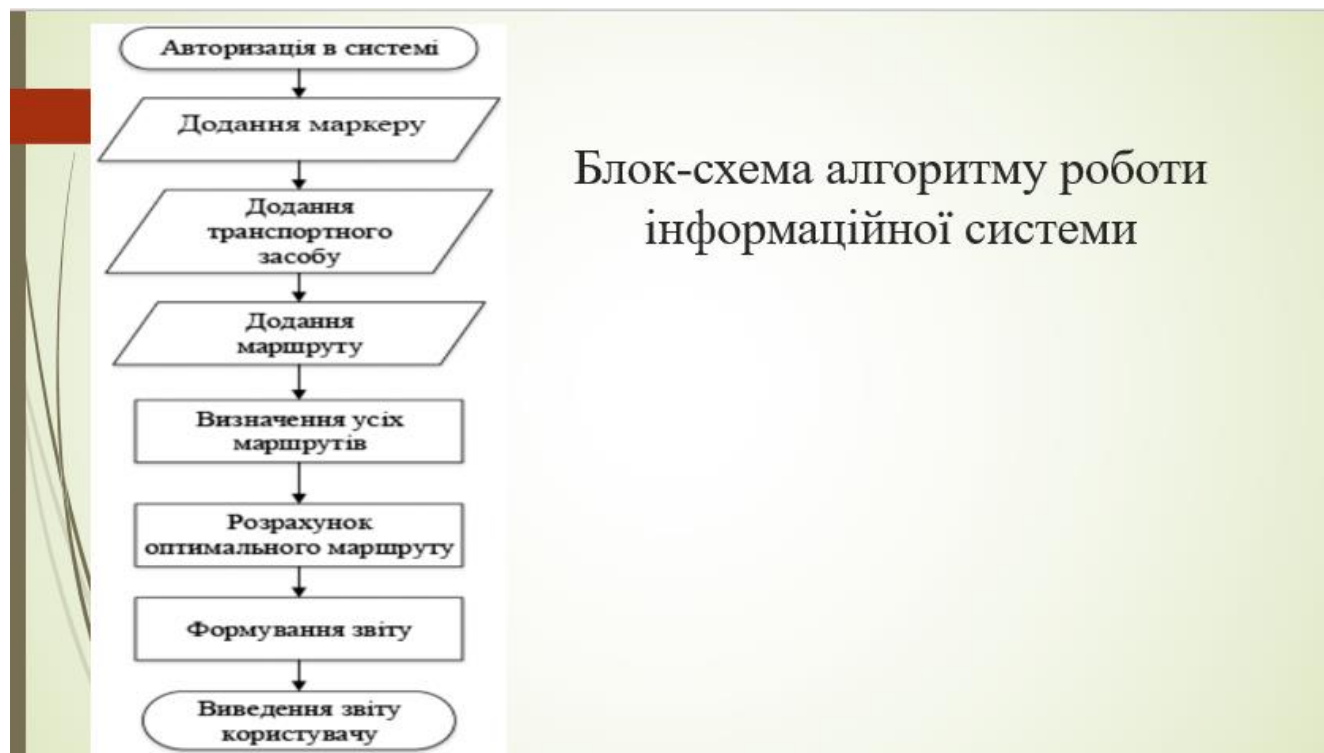


Рисунок Б5 – Слайд із блок-схемою алгоритму роботи



Рисунок Б6 – Слайд з методами вирішення задач пошуку



В роботі застосовано 2 принципово-різних алгоритма пошуку оптимального маршруту:

- Алгоритм розкладання кістяного дерева на маршрути
- Алгоритм мурашиних колоній

Рисунок Б7 – Слайд з описом використаних методів



Діаграма варіантів використання інформаційної системи

Рисунок Б8 – Слайд з зображенням діаграми використання інформаційної системи

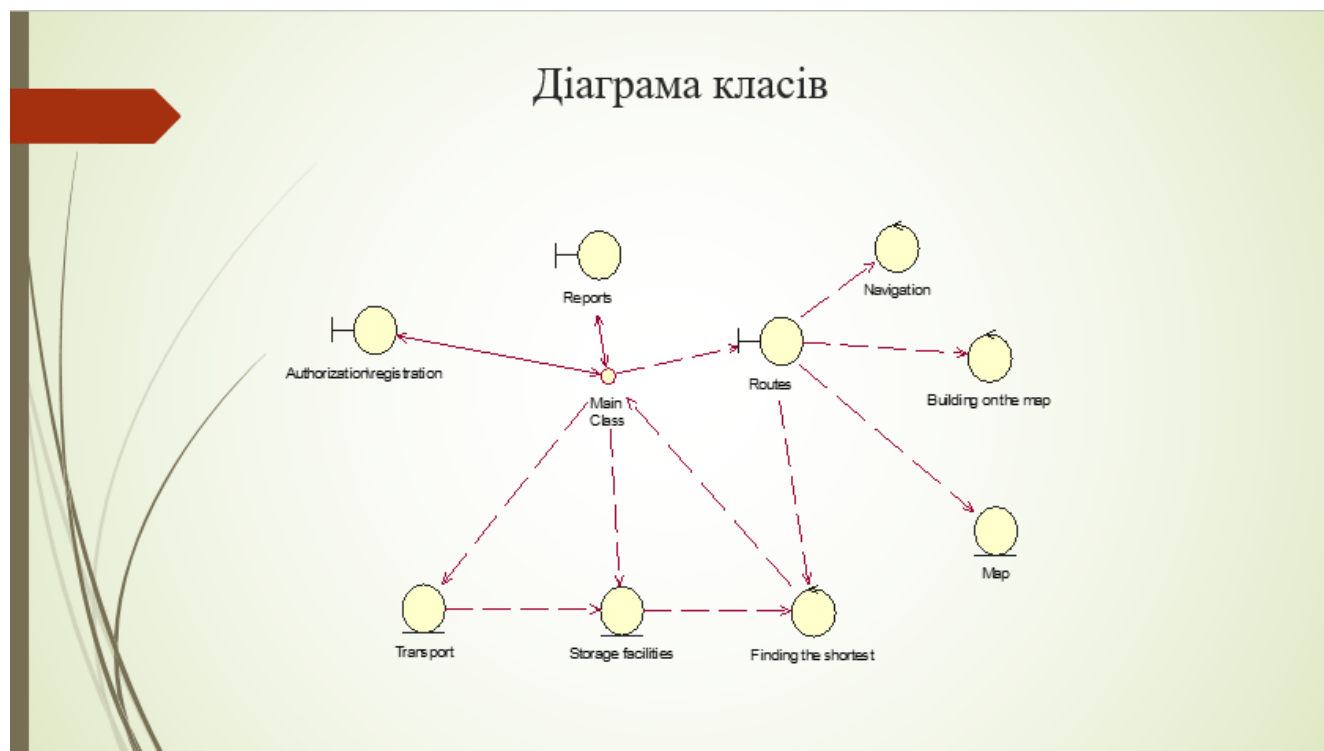


Рисунок Б9 – Слайд з зображенням діаграми класів інформаційної системи

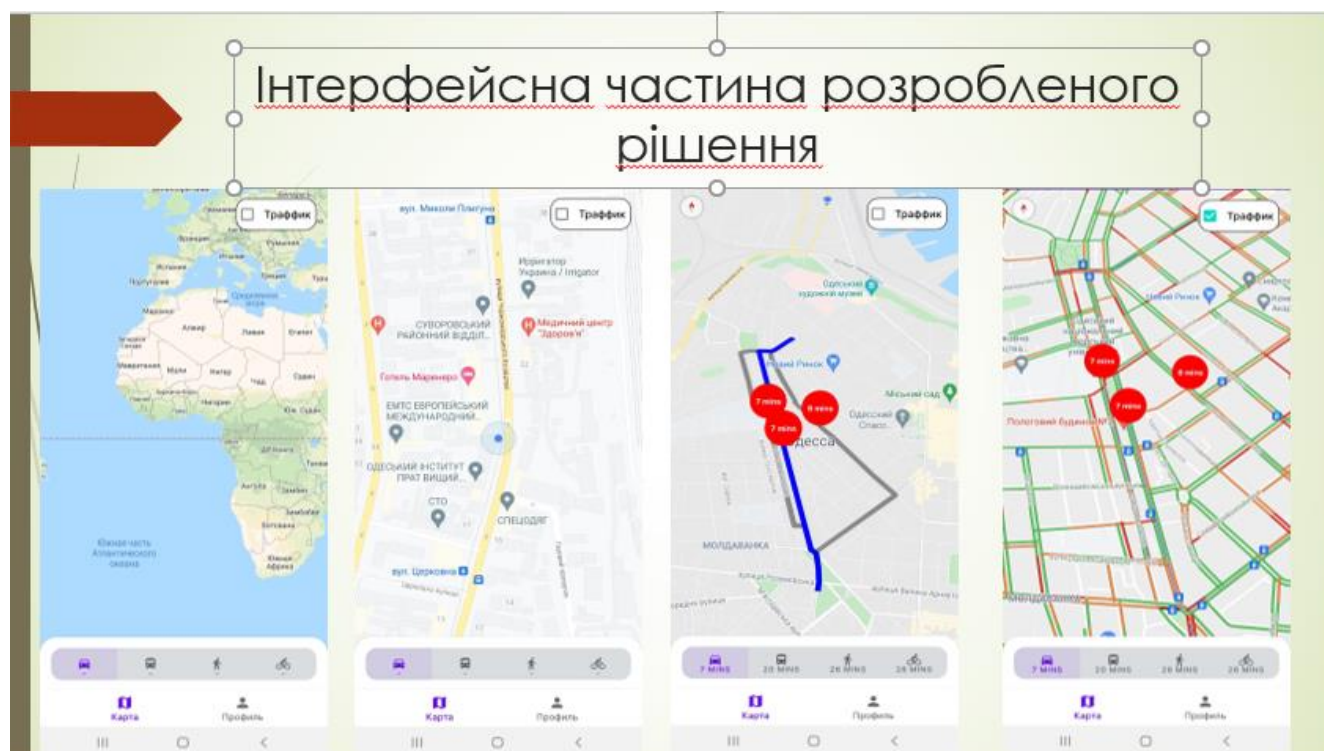


Рисунок Б10 – Слайд з інтерфейсною частиною програмного забезпечення

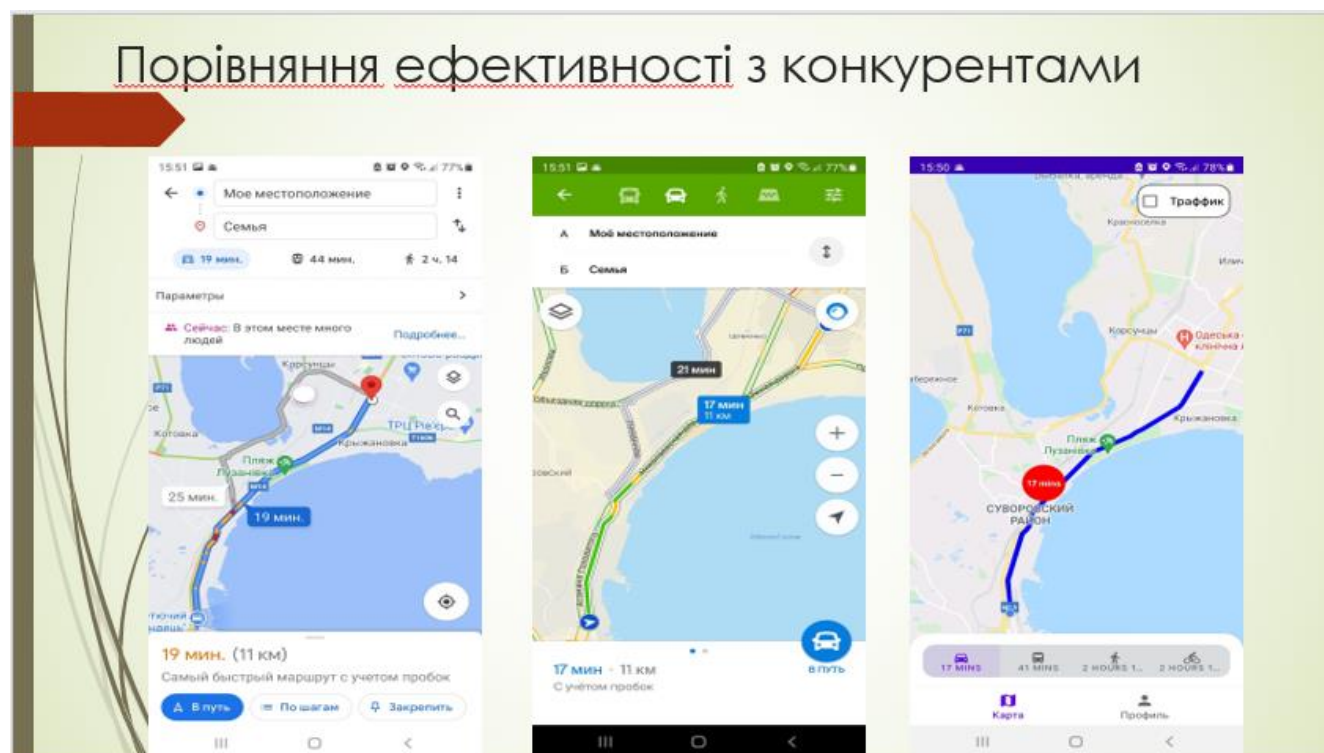


Рисунок Б11 – Слайд з порівнянням ефективності інформаційної системи

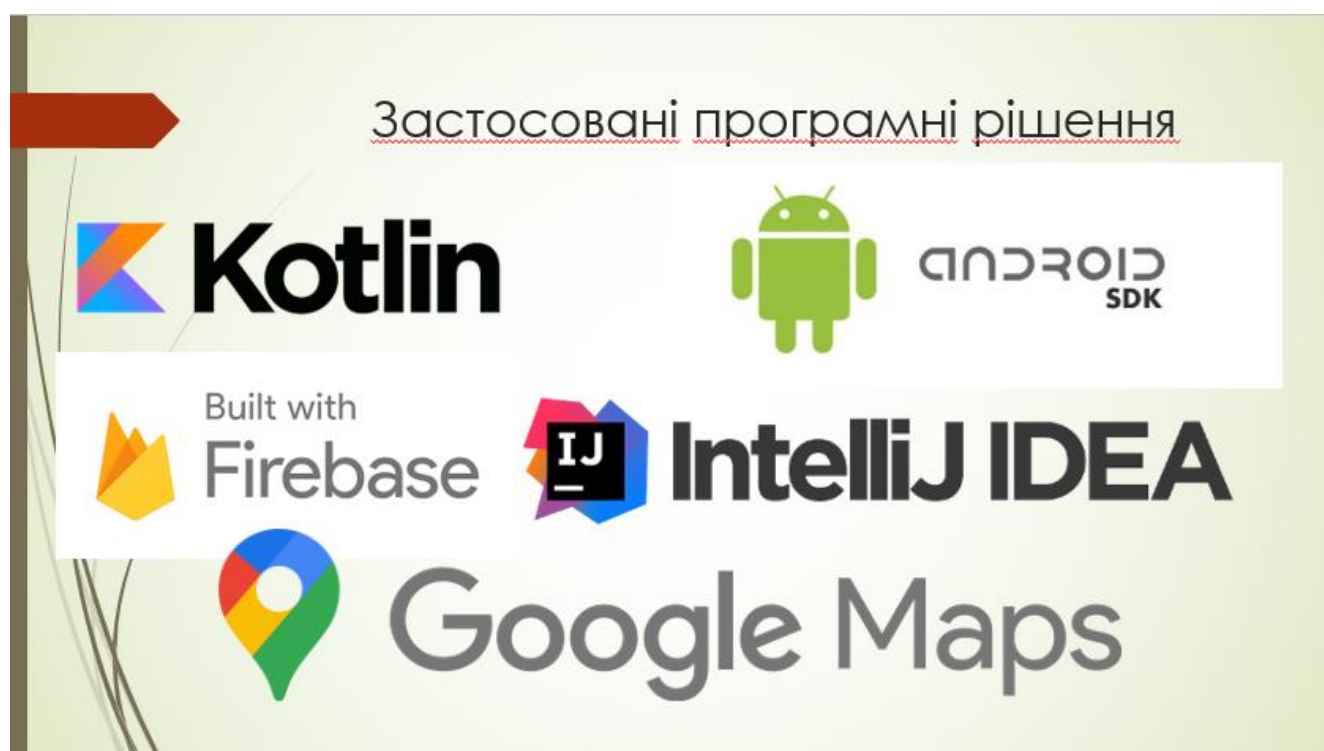


Рисунок Б12 – Слайд з використаними засобами для розробки

Висновки

Результатом кваліфікаційної роботи є розробка програмного засобу з побудови оптимальних маршрутів згідно сформованих початкових умов користувача. Розроблена система використовує велику кількість вхідної інформації, яку аналізує та оброблює в реальному часі та зостосовує в якості додаткових умов при прогнозуванні подій. В кваліфікаційній роботі були виконані наступні задачі:

- виконано огляд сучасних рішень при вирішенні транспортних задач;
- виконано оцінку ефективності застосування транспортних задач в картографічних системах;
- виконано оцінку ефективності алгоритмів побудови оптимальних маршрутів;
- проведено порівняльний аналіз засобів розробки картографічних систем;
- спроектовано архітектуру програмної системи;
- виконано опис алгоритмічної складової розробленої інформаційної системи;
- виконано опис функціональних можливостей розробленого програмного рішення;
- проведено аналіз ефективності розробленого рішення;
- виконано розробку програмного засобу.

Рисунок Б13 – Слайд з висновками

ДОДАТОК В

Апробація результатів роботи

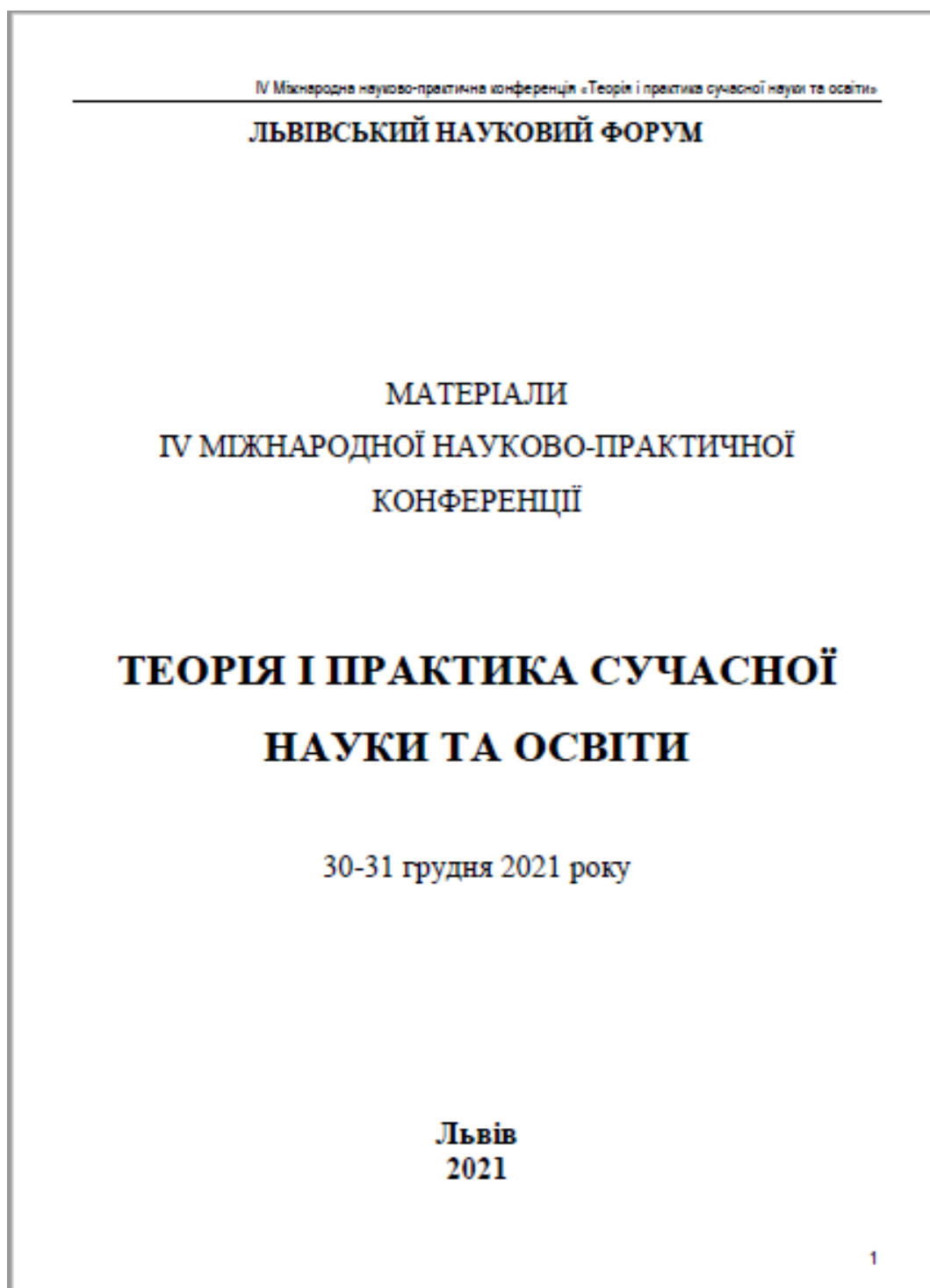


Рисунок В1 – Скановане зображення титульного аркушу збірника тез

Програма інноваційного розвитку процесів автоматизації транспортних компаній, передбачає вдосконалення системи інформаційної підтримки управління технологічних процесів і є невід'ємною частиною комплексу заходів, щодо розробки і впровадження інновацій. Для реалізації успішної діяльності підприємства розробляється і впроваджується комплексна інформаційна система. Ця система може бути представлена у вигляді локальної обчислювальної мережі (ЛОМ), яка об'єднує робочі місця користувачів. Для вирішення зазначених вище завдань прикладне програмне забезпечення (ПЗ) розробленої інформаційної системи (ІС) повинно включати ряд підсистем [2].

Впровадження ІС дозволяє автоматизувати робочі місця диспетчерських служб, відділу механізації, вантажних складів, диспетчерів по залізничним операціям та інших відділів, що призводить до зниження кількості помилок, часу розрахунків і формування звітності, внаслідок чого підвищується оперативність отримання даних для прийняття управлінських рішень. На рис. 1 зображено модель архітектури ІС.

Така система дозволить оптимізувати робочі процеси логістичних та транспортних організацій, за рахунок використання системи динамічної оцінки відстаней між точками маршруту, а також впровадженні автоматизованої функції з прогнозування витрати палива згідно сформованих вхідних умов та обраної техніки.

ЛІТЕРАТУРА:

1. Аниязин Б. А. Коммерческая логистика [Текст] : учебное пособие / Б. А. Аниязин, А. П. Тятлузин. – М.: Проспект, 2014. – 432 с.
2. Галібурад В. Г. Критерии оценки эффективности и качества работы различных видов транспорта [Текст] / В. Г. Галібурад, Д. С. Проксурський // Экономика железных дорог. – 2013. – №3. – С. 86-95.

Горшков Я.В.,

здобувач кафедри комп'ютерних інформаційних систем та технологій

Міжрегіональна академія управління персоналом

ЗАПРОВАДЖЕННЯ АЛГОРИТМІВ ПОШУКУ ОПТИМАЛЬНОГО МАРШРУТУ ДО НАВИГАЦІЙНИХ КАРТОГРАФІЧНИХ СИСТЕМ

Розвиток міст визначається взаємозв'язком зростання міських територій, чисельності населення, планування та розміщення різних функціональних зон. Рівень розвитку міст залежить від організації та технічних можливостей транспортних систем. Створення ефективної транспортної системи міста – це складна комплексна проблема, що включає низку завдань, різних за значущістю, складністю та трудомісткістю, серед яких: визначення маршрутів руху міського пасажирського транспорту, обґрунтування типу, виду та кількості рухомого складу по кожному маршруту, розподіл маршрутів по перевізникам, розробка розкладу та оптимізація режимів руху на маршруті.

Маршрут руху, у свою чергу, є шляхом руху транспортного засобу між початковим і кінцевим пунктами відповідно до розкладу.

Усі існуючі підходи до проектування раціональних маршрутних мереж можна поділити на три групи: автоматизоване проектування маршрутів пасажирського транспорту на основі формалізованих математичних моделей, часткова автоматизація процесу побудови маршрутів пасажирського транспорту та експертна оцінка результатів спеціалістом, прийняття рішень на основі досвіду та неформалізованого аналізу експертів.

Застосування жорстко формалізованих математичних моделей дає оптимальне рішення з погляду суворо закладеного в програму алгоритму, проте за такого підходу неможливо врахувати традиції та звичай пасажирів, що склалися в місті, екологічну обстановку та інші вимоги, що не піддаються формальному опису, рис. 1. Тому найбільш ефективним вважається другий підхід, у якому експерт проводить аналіз отриманих результатів та приймає остаточне рішення.



Рисунок 1 – Діаграма станів роботи алгоритму пошуку оптимального маршруту

Основою всього проектування є визначення величини пасажиропотоків за напрямками транспортних кореспонденцій, своєю чергою визначені з урахуванням транспортних розрахункових районів. Чим правильніше територія міста розділена на транспортні райони, тим правильніше (точніше) величини пасажиропотоків. Вирішення такої задачі дозволить підвищити якість пересування громадян, а також покращити стан на дорогах міста.

ЛІТЕРАТУРА:

1. Герман В.Д. Методология формирования системы городского пассажирского общественного транспорта. - М.: МАДИ. 2001. – 313 с.
2. Самойленко Н.И. Транспортные системы большой размерности / Н.И. Самойленко, А.А. Кобец. Харьков.: «НТМТ», 2010. – 212 с.

Горшков Я.В.,

*здобувач кафедри комп'ютерних інформаційних систем та технологій
Міжрегіональна академія управління персоналом*

РОЗРОБКА КАРТОГРАФІЧНОГО ЗАСОБУ ОПТИМІЗАЦІЇ МАРШРУТУ ЗАСОБАМИ АЛГОРИТМУ МУРАШИНОЇ КОЛОНІЇ

Картографічні інформаційні системи набувають активного використання в різних напрямках діяльності, як головні навігаційні засоби так і як системи додаткової оптимізації маршрутів під час пасажирських та вантажних перевезень.

Найважливішою складовою транспортної інфраструктури, що багато в чому визначає динаміку розвитку сучасного міста, є маршрутна система пасажирського транспорту. У процесі розвитку міста його

маршрутна система потребує періодичного перегляду. Це може бути пов'язано з численними поточними змінами у розкладі маршрутів, зміною розташування місць застосування трици, модернізацією вулично-дорожньої мережі міста [1].

Перепроєктування маршрутної мережі міського транспорту (або розробка нової раціональної маршрутної мережі) – трудомісткий процес, що включає кілька етапів робіт. Найбільш ефективним підходом розв'язання даної задачі є її автоматизація, але за участю експерта, який проводить аналіз отриманих результатів та приймає остаточне рішення. Однак автоматизація завдань даної галузі вимагає проведення наукових досліджень з метою глибокої формалізації та розробки алгоритмів, придатних для використання на практиці. Під міським транспортом слід враховувати різні види транспортних засобів, як громадського використання так і приватного [2].

Одним із методів вирішення завдань пошуку оптимальних маршрутів на графах є алгоритм оптимізації наслідуванням мурашиної колонії. Суть підходу полягає у використанні моделі поведінки мурах, що шукають шлях від колонії до джерела їжі, і є метавирисичною оптимізацією.

Перевагами даного алгоритму можна назвати високу ефективність у порівнянні з іншими методами глобальної оптимізації (наприклад, нейронні мережі, генетичні алгоритми), адаптованість та масштабованість, а також гарантовану збіжність, що дозволяє отримати оптимальне рішення незалежно від розмірності графа.

Мурашиний алгоритм відноситься до категорії алгоритмів розв'язку розв'язку і моделює поведінку мурашиної колонії. Мурахи – це соціальні комахи, здатні утворювати колективи (колонії). Саме колективна система дозволяє ефективно вирішувати завдання динамічного характеру, які могли б бути виконані окремими елементами системи без наявності відповідного зовнішнього управління та координації. Основу поведінки мурашиної колонії становить здатність самоорганізації, що дозволяє швидко адаптуватися до умов навколишнього середовища, що змінюються, і забезпечує досягнення загальної мети колонії на основі низькорівневої взаємодії.

Цей ітеративний процес триває до виконання певної умови завершення: виконано задану кількість ітерацій, вся задана кількість мурах завершила пошук, досягнуто необхідної якості рішення, минув заданий процесорний час.

Розробка інформаційної системи, яка дозволить використовувати такий алгоритм та забезпечить надійне виконання усіх навігаційних умов є актуальною сучасною задачею, яку можливо запровадити в якості головного чи допоміжного засобу картографічної навігації.

ЛІТЕРАТУРА:

1. Герасим В.Д. Методология формирования системы городского пассажирского общественного транспорта. - М.: МАДИ. 2001. – 313 с.
2. Самойленко Н.И. Транспортные системы большой размерности / Н.И. Самойленко, А.А. Кобец. Харьков: «НТМТ», 2010. – 212 с.

Рисунок В4 – Скановане зображення третьої сторінки публікації

ДОДАТОК Г

Електронні матеріали

Зміст	Папка	Ім'я файла
Пояснювальна записка до атестаційної роботи магістра		diploma.docx
Вихідні дані до атестаційної роботи магістра		readme.txt
Докладний проект програми з кодами і поясненнями у середовищі проектування програмного продукту		project.rar
Демонстраційний ролик програмної системи (слайди, анімація тощо)		demo.pptx