**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформаційних систем та технологій**

До захисту допущено:

Завідувач кафедри

_____ Олександр РОЛІК

«___»_____2025 р.

# Дипломний проєкт

**на здобуття ступеня бакалавра**
**за освітньо-професійною програмою «Інформаційні управляючі системи та технології»**
**спеціальності 126 «Інформаційні системи та технології»**
**на тему: «Система побудови багатоточкових маршрутів для повсякденних поїздок»**

Виконала:
студентка IV курсу, групи ІС-13
Скидан Олеся Олександрівна _____

Керівник:
доцент каф. ІСТ, к.т.н
Букасов Максим Михайлович _____

Рецензент:
доцент кафедри ІПІ, к.т.н., доцент
Ліхоузова Тетяна Анатоліївна _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____

Київ – 2025 року

**Національний технічний університет України**
**«Київський політехнічний інститут імені Ігоря Сікорського»**
**Факультет інформатики та обчислювальної техніки**
**Кафедра інформаційних систем та технологій**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олександр РОЛІК
«___»_____2025 р.

# ЗАВДАННЯ
**на дипломний проєкт студентці**
**Скидан Олесі Олександрівні**

1. Тема проєкту «Система побудови багатоточкових маршрутів для повсякденних поїздок», керівник проєкту М. М. Букасов, доцент кафедри ІСТ, к.т.н., затверджені наказом по університету від «23» травня 2025 р. № 1705-с

2. Термін подання студентом проєкту: 9 червня 2025 року

3. Вихідні дані до проєкту: мова програмування Swift, середовище розробки Xcode, реляційна база даних, фреймворки – Google Maps SDK, EventKit. Графічний інтерфейс реалізовано у вигляді iOS-застосунку. Запропоновані системою маршрути та пов’язані з ними дані (точки, координати, відстань, час) виводяться у мобільному застосунку у зручній для користувача формі – на карті або у вигляді структурованого списку точок маршруту.

4. Зміст пояснювальної записки: аналіз предметної області, аналіз існуючих готових рішень, формування вимог до системи, вибір технологій розробки, розробка інформаційної системи, математичне забезпечання.

5. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо): діаграма варіантів використання, структурна схема, діаграма класів, діаграма послідовностей.

6. Дата видачі завдання: 8 березня 2025 року

Календарний план

| № з/п | Назва етапів виконання дипломного проєкту | Термін виконання етапів проєкту | Примітка |
|---|---|---|---|
| 1 | Аналіз предметної області | 01.04.2025 | |
| 2 | Аналіз існуючих рішень | 13.04.2025 | |
| 3 | Формування вимог до системи | 20.04.2025 | |
| 4 | Вибір технологій розробки | 30.04.2025 | |
| 5 | Розробка інформаційної системи | 19.05.2025 | |
| 6 | Розробка математичного забезпечення | 30.05.2025 | |
| 7 | Оформлення пояснювальної записки | 02.06.2025 | |
| | | | |
| | | | |

Студентка Олеся СКИДАН

Керівник Максим БУКАСОВ

# АНОТАЦІЯ

Система побудови багатоточкових маршрутів для повсякденних поїздок.

Проєкт містить 64 с. тексту, 10 рисунків, 5 таблиць, посилання на 17 літературних джерел, додатки та 4 конструкторських документів.

АЛГОРИТМ ДЕЙКСТРИ, БАГАТОТОЧКОВІ МАРШРУТИ, КАЛЕНДАР, КАРТОГРАФІЧНИЙ СЕРВІС, ПЛАНУВАННЯ ПОЇЗДОК.

Об’єктом розробки є система побудови багатоточкових маршрутів.

Мета розробки – підвищення ефективності планування щоденних поїздок.

У дипломному проєкті було розроблено систему автоматизованого формування маршрутів з кількома зупинками. У системі передбачено можливість додавання точок інтересу вздовж прокладеного маршруту, інтеграцію з системним календарем для зручного планування поїздок, а також функцію збереження створених маршрутів для повторного використання. Значну увагу було приділено вибору алгоритму знаходження оптимального шляху знаходження оптимального шляху та його впровадженню у систему.

Отримані результати можуть бути використані для покращення повсякденних пересувань користувачів, оптимізації витрат їх часу та ресурсів під час щоденних поїздок.

## SUMMARY

Multi-point route planning system for daily trips.

The project contains 64 pages. text, 10 figures, 5 tables, links to 17 literary sources, annexes and 4 design documents.

Keywords: Dijkstra algorithm, multi-point routes, calendar, cartographic service, trip planning.

The object of development is a system for constructing multi-stop routes.

The purpose of the development – improving the efficiency of daily trip planning.

The graduation project developed a system for automated generation of routes that include multiple waypoints selected by the user. The system also provides functionality for adding points of interest along the constructed route, integration with the system calendar for pre-scheduling trips, and a subsystem for saving routes for repeated use. Special attention was paid to the selection and implementation of the optimal pathfinding algorithm for route construction.

The results obtained can be applied to improve the everyday mobility of users and to optimize their time and resource usage during routine travel.

| № рядка | Формат | Позначення | Найменування | Кіл. аркушів | № екз. | Примітка |
|---|---|---|---|---|---|---|
| 1 | | | Документація загальна | | | |
| 2 | | | | | | |
| 3 | | | Знову розроблена | | | |
| 4 | | | | | | |
| 5 | А4 | ІС13.230БАК.005 ПЗ | Система побудови багатоточкових | 64 | | |
| 6 | | | маршрутів для повсякденних | | | |
| 7 | | | поїздок. Пояснювальна записка | | | |
| 8 | | | | | | |
| 9 | А3 | ІС13.230БАК.005 Д1 | Система побудови багатоточкових | 1 | | |
| 10 | | | маршрутів для повсякденних | | | |
| 11 | | | поїздок. Діаграма варіантів | | | |
| 12 | | | використання | | | |
| 13 | | | | | | |
| 14 | А3 | ІС13.230БАК.005 Э1 | Система побудови багатоточкових | 1 | | |
| 15 | | | маршрутів для повсякденних | | | |
| 16 | | | поїздок. Структурна схема | | | |
| 17 | | | | | | |
| 18 | А3 | ІС13.230БАК.005 Д2 | Система побудови багатоточкових | 1 | | |
| 19 | | | маршрутів для повсякденних | | | |
| 20 | | | поїздок. Діаграма класів | | | |
| 21 | | | | | | |
| 22 | А3 | ІС13.230БАК.005 Д3 | Система побудови багатоточкових | 1 | | |
| 23 | | | маршрутів для повсякденних | | | |
| 24 | | | поїздок. Діаграма послідовностей | | | |
| 25 | | | | | | |
| 26 | | | | | | |
| 27 | | | | | | |
| 28 | | | | | | |

| | | | | | ІС13.230БАК.005 ТП | | |
|---|---|---|---|---|---|---|---|
| Зм. | Лист | № докум. | Підпис | | | | |
| Розробив | | Скидан О.О. | | | Система побудови багатоточкових маршрутів для повсякденних поїздок. Відомість дипломного проєкту | Літ. | Арк. 1 / Аркушів 1 |
| Перевірив | | Букасов М.М. | | | | Т | |
| | | | | | | КПІ ім. Ігоря Сікорського Група ІС-13 | |
| Затв. | | | | | | | |

# Пояснювальна записка
## до дипломного проєкту
# на тему: «Система побудови багатоточкових маршрутів для повсякденних поїздок»

Київ – 2025 року

**ЗМІСТ**

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

ARC – Automatic Reference Counting

GPS – Global Positioning System

JSON – JavaScript Object Notation

POI – Point of Interest

SDK – Software Development Kit

SPM – Swift Package Manager

UI – User Interface

UUID – Universally Unique Identifier

# ВСТУП

У наші дні розвиток технологій дає можливість автоматизувати та оптимізувати велику кількість рутинних процесів. Системи, завдяки яким це відбувається, стають усе доступнішими, швидшими, простішими у використанні, але водночас і «розумнішими». Однією із задач оптимізації є задача побудови маршрутів, яка має широке застосування у всіх сферах життя.

У сучасному світі питання побудови маршрутів набуває дедалі більшого значення, особливо коли йдеться про побудову маршрутів із декількома зупинками для повсякденних поїздок. Майже кожного дня люди стикаються з потребою відвідати декілька місць – пошту, школу, садок, роботу тощо, в рамках однієї поїздки. При цьому оптимально використовувати власний час, пальне та інші ресурси. Ці аспекти й роблять питання побудови багатоточкових маршрутів актуальним у контексті повсякденного життя.

Наразі сучасні мобільні пристрої мають вбудовані GPS-модулі, що відкриває великі можливості для реалізації різних навігаційних інструментів, що сприяє появі різноманітних додатків для прокладання маршрутів. Але треба зазначити, що більшість наявних на ринку навігаційних застосунків пропонують прокладання шляху лише між двома точками – з точки А до точки В, що не розв’язує проблему відвідування декількох локацій у межах однієї поїздки. У результаті користувач змушений будувати декілька окремих маршрутів. Такий підхід подовжує процес планування маршруту, прокладений шлях не є оптимальним, і, як наслідок, часові та ресурсні витрати користувача збільшуються.

Наразі сучасні мобільні пристрої мають вбудовані GPS-модулі, що відкриває великі можливості для реалізації різних навігаційних інструментів, що сприяє появі різноманітних додатків для прокладання маршрутів. Але треба зазначити, що більшість наявних на ринку навігаційних застосунків пропонують прокладання шляху лише між двома точками – з точки А до точки В, що не розв’язує проблему відвідування декількох локацій у межах однієї поїздки. У результаті користувач змушений будувати декілька окремих маршрутів. Такий підхід подовжує процес

планування маршруту, прокладений шлях не є оптимальним, і, як наслідок, часові та ресурсні витрати користувача збільшуються.

Іноді також виникає потреба у плануванні маршруту наперед, для того, щоб коректно спланувати свої справи на день, тому додатковий функціонал із плануванням маршруту через системний календар дозволить створювати події із прив’язкою до конкретної дати й часу. Це забезпечить можливість не лише зафіксувати заплановану поїздку, а й швидко перейти до відповідного маршруту безпосередньо з календаря. Такий підхід підвищує зручність користування та дозволяє інтегрувати процес планування маршрутів у щоденний розклад користувача.

Об’єктом дослідження є процес побудови багатоточкових маршрутів для повсякденних поїздок.

Предметом дослідження є алгоритм пошуку найкоротшого шляху на графі та його реалізація у системі побудови маршруту, що проходить через декілька проміжних точок.

Мета розробки полягає у підвищенні ефективності планування багатоточкових маршрутів для щоденних поїздок.

Для досягнення мети необхідно:

– провести аналіз предметної області та програмних рішень;

– визначити вимоги до функціоналу застосунку;

– обрати технології розробки;

– проаналізувати алгоритми та обрати найбільш ефективний для розв’язання поставленої задачі;

– розробити зручний інтерфейс;

– реалізувати систему побудови багатоточкових маршрутів з використанням обраного алгоритму пошуку найкоротшого шляху;

– реалізувати можливість додавання “точок по дорозі” у вже прокладений маршрут;

– реалізувати можливість зберігання прокладених шляхів;

– реалізувати функціонал планування маршруту через взаємодію з системним календарем;

– провести тестування системи.

Практична значущість отриманих результатів полягає в тому, що застосунок значно спрощує планування щоденних подорожей, заощаджуючи час та ресурси користувачів. Наявність можливості збереження маршрутів, додавання “точок інтересу” та інтеграції з календарем для планування поїздок робить застосунок універсальною системою для виконання щоденних справ у контексті повсякденних поїздок. Зрештою, користувач витрачає менше часу на рутинне планування поїздок і має можливість звільнити час для важливих життєвих справ.

У майбутньому система може отримати нові можливості – зокрема, синхронізацію між пристроями, підтримку голосового введення, адаптивні маршрути залежно від трафіку, а також інтеграцію з картами громадського транспорту. Це дозволить розширити її застосування та підвищити комфорт користування. Наприклад, застосунок може стати частиною більш комплексних систем навігації, смартміст чи корпоративних логістичних платформ.

# 1 ОПИС ПРЕДМЕТНОЇ ОБЛАСТІ

## 1.1 Опис процесу діяльності

Процес діяльності у сфері планування багатоточкових маршрутів містить вибір точок для відвідування, аналіз їх розташування відносно одна одної та побудову шляху, який охоплює всі обрані локації. Користувачу необхідно самостійно визначати послідовність проходження точок, що потребує значних витрат часу і не завжди дозволяє сформувати маршрут з раціональним порядком відвідування точок, що, у свою чергу, призводить до неефективного розподілу ресурсів.

З урахуванням поставленої задачі в межах предметної області логічним кроком є розробка системи, яка автоматизує побудову маршрутів для повсякденних поїздок, максимально ефективних щодо витрат часу та пального. Вона повинна забезпечувати зручне додавання точок відвідування, включаючи можливість швидкого пошуку та додавання «точки інтересу» не за конкретною адресою, а орієнтуючись на маршрут, яким користувач планує рухатися. До того ж має бути реалізована можливість збереження маршрутів, які часто повторюються або використовуються на регулярній основі. Таким чином, система підвищує зручність планування поїздок і скорочує часові та ресурсні витрати користувача.

## 1.2 Постановка задачі

### 1.2.1 Призначення системи

Система призначена для автоматизації маршрутного планування поїздок із можливістю додавання проміжних точок до вже побудованого шляху, а також з функцією збереження обраного маршруту до списку улюблених для подальшого використання. Крім того, інтеграція з системним календарем дозволяє планувати поїздки заздалегідь на визначену дату й час. До об’єктів, що підлягають автоматизації, належать:

– багатоточкові маршрути, сформовані користувачем;

– точки маршруту, що додаються через текстовий пошук або вибір на карті;

– провести аналіз предметної області та програмних рішень;

– додаткові локації (так звані «точки інтересу»), що додаються до вже побудованого маршруту;

– збереження маршрутів до списку улюблених із можливістю їх перегляду у вигляді списку або на карті;

– планування поїздок через створення подій у системному календарі.

Автоматизація цих об’єктів дає змогу зменшити час, який необхідний для планування поїздок, підвищити ефективність управління маршрутами та зробити процес їх формування більш зручним і комфортним для користувача.

### 1.2.2 Цілі та задачі розробки

Головне завдання розробки системи – оптимізувати процес створення маршрутів з кількома зупинками, забезпечення зручної взаємодії з ними, можливість збереження, перегляду та повторного використання, а також планування поїздок за допомогою інтеграції із системним календарем.

Для досягнення визначеної мети слід реалізувати комплекс завдань, що включають аналітичну, проєктну та практичну частини розробки системи.

Спочатку здійснюється огляд наявних рішень, що реалізують функціонал побудови маршрутів, з метою виявлення їхніх переваг і недоліків у контексті повсякденного використання.

Далі визначаються функціональні та нефункціональні вимоги до системи, що враховують потреби користувачів, особливості взаємодії з картою та маршрутом, а також вимоги до збереження даних.

Після цього здійснюється огляд алгоритмів, що можуть бути застосовані для розв’язання задачі побудови багатоточкових маршрутів. На основі порівняльного аналізу обґрунтовується доцільність вибору конкретного, найефективнішого методу для реалізації логіки побудови шляху.

Відповідно до визначених вимог підбираються програмні засоби, здатні задовольнити технічні потреби реалізації системи – зокрема, забезпечити інтеграцію з картою, підтримку функцій геопозиціювання та локального зберігання даних.

Наступним кроком є створення інтерфейсу користувача, який має бути легким у використанні, адаптивним та зрозумілим. Він повинен забезпечувати повноцінну взаємодію користувача з функціоналом прокладання маршрутів.

Реалізація функціональності побудови маршрутів передбачає виконання наступних підзадач:

– додавання локацій за допомогою пошуку за адресою або вибору на карті;

– редагування маршруту (видалення точок, встановлення статусу початкової або кінцевої точки);

– додавання точок інтересу до маршруту;

– перегляд сформованого маршруту з відображенням загальної довжини та тривалості;

– планування маршруту через створення відповідної події у системному календарі;

– збереження маршруту до списку улюблених;

– перегляд збережених маршрутів у вигляді списку або на карті.

Завершальним етапом є тестування застосунку з метою виявлення та усунення недоліків, перевірки стабільності функціонування системи, а також відповідності реалізованого функціоналу поставленим вимогам.

Висновки до розділу 1

У цьому розділі проаналізовано специфіку діяльності, пов’язаної з плануванням багатоточкових маршрутів, а також виявлено ключові проблеми, з якими стикається користувач під час самостійного визначення послідовності відвідування локацій. Обґрунтовано доцільність автоматизації цього процесу для підвищення ефективності планування поїздок, економії часу та ресурсів.

Визначено призначення майбутньої системи, яка має забезпечити не лише побудову маршрутів з кількома зупинками, а й можливістю додавання “точок інтересу” до маршруту, збереженням маршрутів для подальшого використання, а також плануванням маршрутів для майбутніх поїздок за допомогою подій у системному календарі. Також сформульовано основні цілі та задачі розробки, що охоплюють аналіз предметної області, вибір алгоритму, розробку інтерфейсу та реалізацію функціональності системи, необхідної для ефективної роботи з маршрутами.

Окрім цього, детально розглянуто об’єкти, які підлягають автоматизації, зокрема – точки маршруту, способи їх додавання та редагування. Такий підхід дозволяє створити цілісну систему, яка враховує практичні потреби користувача і забезпечує масштабованість і адаптивність у майбутньому. Значна увага приділена зручності користування, адже для щоденного застосування системи в реальних умовах це є ключовим фактором.

У результаті проведеного аналізу була створена комплексна модель вимог, яка охоплює функціональні та нефункціональні аспекти. Це дає змогу перейти до наступних етапів проєктування з чітким розумінням структури майбутнього застосунку, його цілей і засобів реалізації.

# 2 АНАЛІЗ ІСНУЮЧИХ РІШЕНЬ

У даному розділі розглянуто популярні наявні рішення для навігації та побудови маршрутів.

## 2.1 Google Maps

Google Maps – це картографічний сервіс, розроблений компанією Google, вперше представлений у 2005 році як вебдодаток і з часом став доступним як мобільний додаток [1]. Сервіс поєднує GPS-дані з пристроїв користувачів, супутникові зображення, інформацію зі служби Street View та краудсорсингові доповнення, забезпечуючи точну навігацію, що оновлюється в реальному часі.

Мобільний застосунок дає змогу орієнтуватися на місцевості й прокладати маршрути для різних видів транспорту – пішки, велосипедом, автомобілем і громадським транспортом. Голосові підказки та підтримка доповненої реальності полегшують навігацію на незнайомій місцевості, а панорамний огляд Street View [2] допомагає візуально впізнати об’єкти довкола.

Користувач також може знаходити певні місця за категоріями («Їжа», «Покупки», «Послуги» тощо). Додаток показує найближчі локації з рейтингами, відгуками й графіком роботи, даючи змогу самостійно вибрати, куди завітати. Таку точку інтересу можна додати на поточний маршрут як проміжну зупинку.

Хоча Google Maps дозволяє вставляти проміжні точки між початком і фінішем, першочергово застосунок не створювався як повноцінний планувальник багатоточкових поїздок. Додавати багато зупинок доволі трудомістко, а максимальна кількість точок обмежена.

Окремі локації можна додавати до «Уподобаних» чи інших персональних списків; дані синхронізуються через обліковий запис Google і зберігаються у хмарі.

Застосунок взаємодіє з Gmail і Google Calendar – адреси з електронних листів та подій автоматично пропонуються до навігації.

Алгоритмічна основа сервісу базується на модифікованих графових методах (Дейкстра, A*) з евристичними доробками для трафіку та прогнозування швидкості руху.

Хоча Google Maps є потужним інструментом для прокладання маршрутів, він не розв'язує проблему побудови саме багатоточкових маршрутів для щоденних поїздок. Як уже згадувалося, цей додаток був розроблений передусім як планувальник маршруту від точки А до точки Б, тому налаштування багатоточкових маршрутів є досить складним і трудомістким процесом. Також у застосунку відсутня можливість зберігати побудовані маршрути, що є доволі корисною функцією з погляду зручності повторного використання, якщо поїздка стосується щоденних рутинних завдань або звичайного планування. Користувацький інтерфейс застосунку представлений на рисунку 2.1.

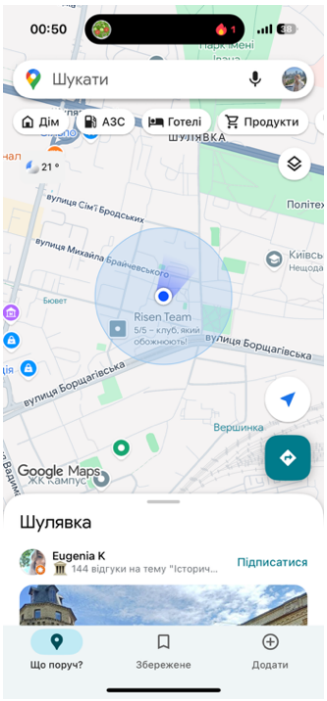


Рисунок 2.1– Користувацький інтерфейс застосунку Google Maps

## 2.2 Apple Maps

Apple Maps – це навігаційний сервіс, який компанія Apple представила у 2012 році як альтернатива Google Maps для пристроїв операційної системи iOS. На

початковому етапі сервіс зазнав суттєвої критики через неточні супутникові знімки, помилки в маршрутах і невідповідність картографічної інформації. Але з часом якість застосунку помітно зросла, а функціональність була розширена завдяки постійним оновленням.

Мобільний застосунок дає змогу будувати автомобільні, пішохідні, велосипедні та маршрути громадським транспортом, відображає актуальний трафік і повідомляє про аварії чи перекриття доріг. Так само як і Google Maps, Apple Maps надає можливість переглядати інформацію про об'єкти інфраструктури: фотографії, графік роботи, вебсайт (якщо доступний) тощо. Користувач може додавати точки в улюблене, щоб мати швидкий доступ до часто відвідуваних місць. Крім того, сервіс підтримує голосову навігацію та інтеграцію з іншими застосунками Apple, що підвищує зручність використання в межах екосистеми.

Додаток дозволяє користувачу будувати багатоточкові маршрути, але з обмеженням у 14 точок [3]. Зупинки можна додавати через пошук або зі списку категорій («Їжа», «Паливо», «Медицина» тощо). Точка на карті додається за допомогою довгого натискання. Додані локації впорядковуються простим перетягуванням, але автоматичного визначення порядку проходження немає, тобто користувач сам має вирішувати, у якій послідовності відвідувати точки. У процесі навігації можна вставити нову зупинку «на ходу».

На моделях iPad із підтримкою Wi-Fi + Cellular під час автомобільної навігації можна додавати проміжні зупинки до основного маршруту. Крім того, сервіс Apple Maps підтримує 3D-перегляд об’єктів, а також функцію Look Around – аналог Google Street View [4], що надає панорамні знімки вулиць у форматі 360°, зібрані за допомогою автомобілів із камерами. Ці можливості роблять навігацію більш наочною та зручною для планування поїздок у міському середовищі.

Попри значний прогрес, Apple Maps залишається сервісом загального призначення, а не спеціалізованим планувальником багатоточкових повсякденних маршрутів. Ліміт у додаванні точок, відсутність алгоритму оптимізації порядку, необхідність ручного вибору точок інтересу не закривають потреб користувачів у плануванні шляхів із декількома локаціями. Так само відсутня функція зберігання

цілих маршрутів для повторного перегляду та використання. На рисунку 2.2 представлено користувацький інтерфейс застосунку.

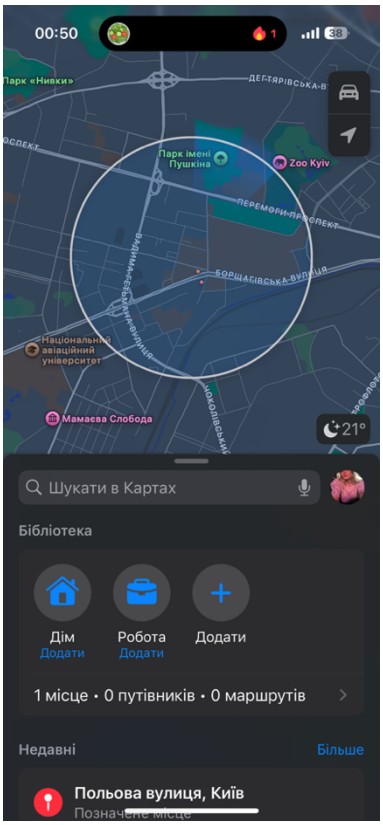


Рисунок 2.2 – Користувацький інтерфейс застосунку Apple Maps

## 2.3 Waze

Waze – це безкоштовний мобільний навігаційний додаток, що дозволяє відстежувати дорожню ситуацію в реальному часі, дізнаватися про розміщення камер фіксації швидкості, а також прокладати оптимальні маршрути. Завдяки звітам користувачів, Waze швидко оновлює маршрути, пропонуючи альтернативні шляхи, щоб уникнути заторів і зекономити час у дорозі.

Водії можуть залишати повідомлення про аварії, автомобілі, які припарковані на краю дороги, дорожній рух, закриття, вибоїни та інші небезпеки на дорозі, навіть де знаходиться поліція, що робить сервіс цінним для щоденних водіїв.

Завдяки своїй інтерактивності та великій кількості користувачів, інформація оновлюється майже миттєво, і маршрути можуть змінюватися автоматично. Waze

пропонує голосову навігацію, інтеграцію з календарем подій та можливість вибору стилю інтерфейсу [5]. Також доступна система балів і значків яка спонукає користувачів активно долучатися до спільноти.

Застосунок також вирізняється простотою та зручністю, що робить процес побудови маршрутів легким та швидким. Інтерфейс зрозумілий: користувач може без зусиль додати пункт призначення, змінити маршрут, переглянути попередження на дорозі або переглядати потенційні перешкоди в реальному часі.

Попри широкий функціонал, який пропонує застосунок, він не вирішує завдання побудови саме багатоточкових маршрутів, оскільки дозволяє додати лише три точки: початкову, кінцеву та одну проміжну. Це значно обмежує можливості користувача у плануванні більш складних поїздок з кількома зупинками. Як і в попередніх прикладах, у Waze відсутня функція збереження та перегляду раніше створених маршрутів, що унеможливлює повторне використання типових поїздок або планування на майбутнє. Користувацький інтерфейс застосунку представлений на рисунку 2.3.

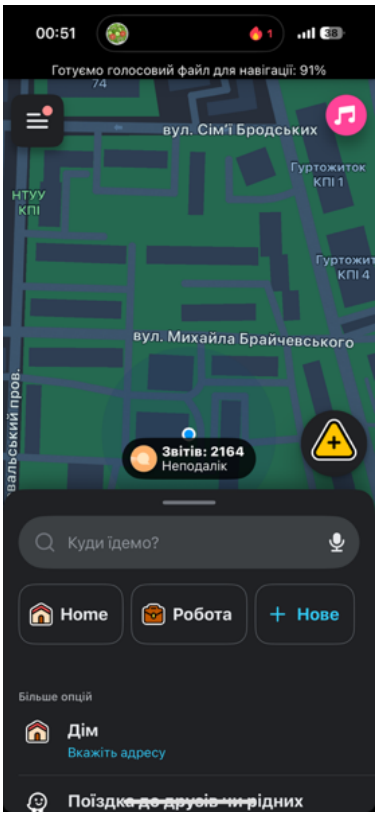


Рисунок 2.3 – Користувацький інтерфейс застосунку Waze

## 2.4 Circuit Route Planner

Circuit Route Planner – це мобільний додаток, призначений для побудови багаточкових маршрутів для платформ iOS та Android. Цільова аудиторія застосунку – кур'єри та водії доставки, оскільки багато функціоналу пов'язано саме із щоденним обслуговуванням великої кількості адрес: позначення статусу виконання доставки, автоматичне сортування зупинок, підтримка часових вікон, відстеження посилки в режимі реального часу та автоматичне збереження маршрутів для повторного використання. Для кожної зупинки можна вказати тип завдання (наприклад, «залишити пакунок», «забрати посилку»), додати замітки або зробити позначку про виконання. Головна мета роботи застосунку полягає в тому, щоб зробити планування маршруту з багатьма зупинками менш складним, скоротити час подорожі та підвищити ефективність доставки. Додаток автоматично визначає маршрут між заданими користувачем точками та будує шлях. Також у застосунку є можливість налаштувати час старту подорожі.

Головні переваги Circuit – це оперативна оптимізація маршруту та швидкий пошук бажаних адрес із можливістю повторно використовувати збережені шаблони маршрутів, що за відгуками кур'єрів економить до години щодня. Але система має логістичний напрямок і розрахована для кур'єрів та служб доставки. Більшість функцій недоступна у безкоштовному тарифі: наприклад, оптимізація маршруту можлива лише для 10 точок, а встановлення часових вікон взагалі відсутнє [6]. Користувач не може встановити кінцеву точку прибуття, якщо йому така необхідна, що є важливим аспектом у побудові буденних подорожей. Також немає можливості додати маршрут в список збережених або улюблених, всі маршрути автоматично зберігаються у застосунку. Через це користувач може витратити деяку кількість часу на пошуки необхідного маршруту. Таким чином, хоча застосунок і є хорошим інструментом для побудови маршрутів із декількома локаціями, він орієнтований переважно на вирішення логістичних завдань і лише частково відповідає потребам щоденних користувачів. Саме тому, під час проєктування системи для щоденних поїздок, важливо врахувати гнучкість

сценаріїв використання, простоту збереження, можливість перегляду улюблених маршрутів і інтеграцію з особистими планувальниками, такими як календар.

Головну сторінку застосунку можна побачити на рисунку 2.4

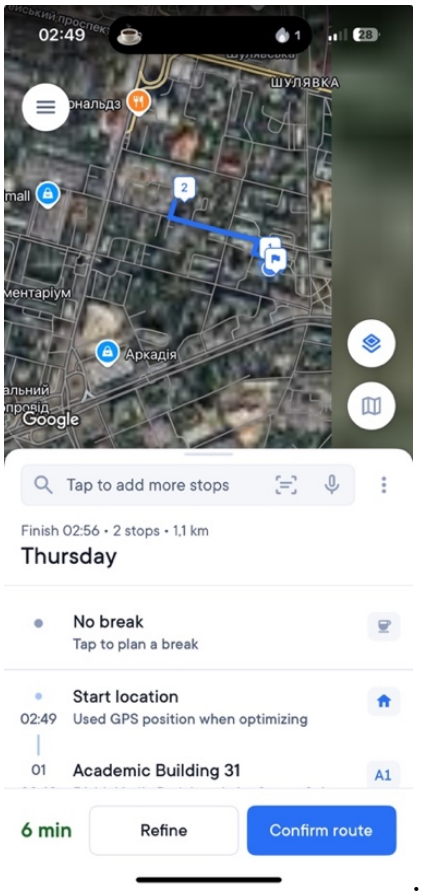


Рисунок 2.4 – Головна сторінка застосунку Circuit

## 2.5 HERE WeGo

HERE WeGo – це безкоштовний мобільний застосунок і планувальник подорожей, розроблений для підтримки подорожей, від простих поїздок на роботу до складних мультимодальних поїздок. Застосунок пропонує функціонал для побудови багатоточкових маршрутів із можливістю збереження цих подорожей до списку “Ваші колекції” [7]. Застосунок підтримує побудову маршрутів для пішоходів, автомобілістів, велосипедистів і користувачів громадського транспорту. HERE WeGo дозволяє завантажувати карти регіонів для офлайн-використання, що

може бути зручним функціоналом в умовах поганого інтернет-з'єднання або в поїздках за кордон.

Крім основної навігації, HERE WeGo надає інформацію про трафік у реальному часі, перекриття доріг, обмеження швидкості, та інші фактори, які можуть вплинути на вибір маршруту, у додатку також реалізовано голосову навігацію. Зручний інтерфейс мобільного застосунку дозволяє легко додавати або змінювати пункти призначення.

Водночас суттєвим недоліком додатку є відсутність автоматичної оптимізації маршруту: система не визначає оптимальний порядок проходження точок, тому вся відповідальність за визначення порядку проходження точок маршруту покладається на користувача. У застосунку це відбувається шляхом перетягування локацій у списку. Інтерфейс застосунку продемонстровано на рисунку 2.5.

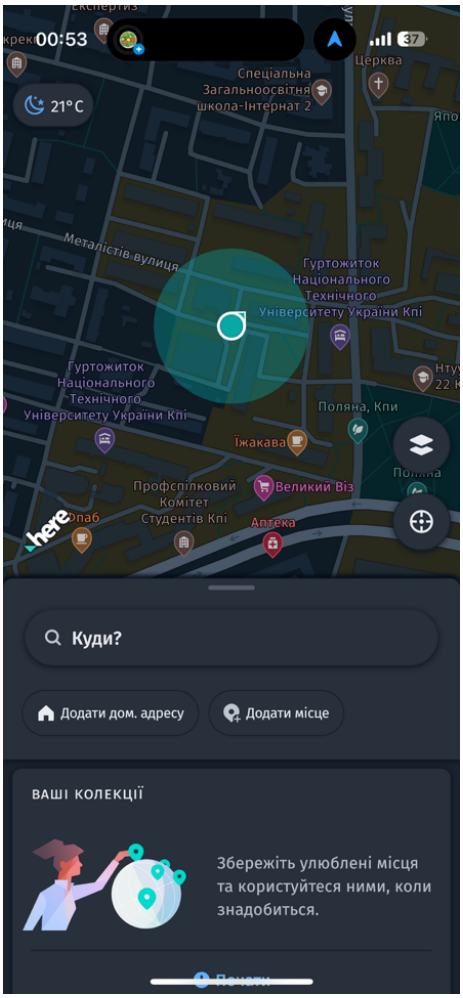


Рисунок 2.5 – Користувацький інтерфейс застосунку HERE WeGo

## 2.6 Порівняння наявних рішень

У таблиці 2.1 наведено порівняння розглянутих застосунків за їхніми функціональними можливостями.

Таблиця 2.1 – Порівняння наявних рішень

| Застосунок / Функціонал | Google Maps | Apple Maps | Waze | Circuit Route Planner | HERE WeGo | Система побудови багатоточкових маршрутів для повсякденних поїздок |
|---|---|---|---|---|---|---|
| Багатоточкові маршрути | + | + | – | + | + | + |
| Збереження маршрутів | – | – | – | + | + | + |
| Визначення порядку проходження зупинок | + | – | – | + | – | + |
| Інтеграція із календарем | + | + | + | – | + | + |
| Додавання точок інтересу | + | + | + | – | – | + |

Отже, жоден із представлених застосунків не охоплює повного спектра функцій, які є корисними для планування щоденних поїздок із багатьма зупинками, тоді як реалізована система поєднує їх в одному рішенні.

## Висновки до розділу 2

Перераховані вище застосунки є потужними інструментами для побудови маршрутів, однак жоден із них не вирішує повною мірою проблему планування багатоточкових маршрутів саме в контексті щоденних поїздок.

Крім того, не в усіх існуючих рішеннях передбачено функціоналу інтеграції з календарем, що дозволив би користувачу створити подію з маршрутом на конкретну дату та час – зручна функція для щоденного планування поїздок.

Для прикладу, Circuit Route Planner підтримує збереження маршрутів, але ця функція реалізована як автоматичне збереження всіх поїздок користувача без можливості самостійно обрати, які саме маршрути слід зберігати. Крім того, у цьому застосунку відсутня можливість встановлення кінцевої точки маршруту – вона визначається системою автоматично.

Функція збереження маршрутів присутня в HERE WeGo, однак додаток не виконує автоматичну оптимізацію порядку проходження точок – користувач має самостійно визначати послідовність зупинок. У деяких із розглянутих застосунків також передбачено додавання точок інтересу, проте у межах дипломного проекту ця можливість буде реалізована на більш гнучкому рівні: користувач зможе або самостійно знайти потрібну локацію і додати її до маршруту, або дозволити системі автоматично знайти найближчу точку інтересу вздовж уже побудованого маршруту.

Таким чином, розроблена система поєднує ключові переваги існуючих рішень і усуває їхні основні обмеження, зосереджуючись на зручності щоденного використання, гнучкому управлінні маршрутом, його збереженні, а також плануванні поїздок.

# 3 ФОРМУВАННЯ ВИМОГ ДО СИСТЕМИ

## 3.1 Вимоги до системи в цілому

Інформаційна система, що розробляється, має відповідати наступним вимогам: забезпечення комфортної взаємодії користувача з картою, ефективне планування маршрутів з багатьма зупинками, можливість зберігання та перегляду побудованих маршрутів, а також інтеграція із вбудованим iOS календарем для планування маршрутів із швидким переходом із календарної події до застосунку.

Система повинна бути простою у використанні, забезпечуючи при цьому стабільну роботу при повсякденному використанні та мінімізуючи кількість потенційно ймовірних помилок під час взаємодії з користувачем.

### 3.1.1 Вимоги до структури та функціонування системи

Система повинна складатися з таких основних підсистем:

– підсистема роботи з картою, яка відповідає за відображення карти, встановлення маркерів точок маршруту, а також візуалізацію побудованого шляху;

– підсистема планування маршруту реалізує логіку побудови оптимального маршруту за допомогою алгоритму знаходження найкоротшого шляху, визначає (за необхідності) початкову та кінцеву точки, формує маршрут між доданими користувачем, точками;

– підсистема пошуку точок інтересу забезпечує можливість знаходження найближчих до користувача об’єктів, наприклад кав’ярень чи магазинів, за ключовим словом і автоматичного додавання знайденої локації до маршруту;

– підсистема збереження та перегляду маршрутів, яка забезпечує зберігати дані у локальному сховищі, надає можливість переглядати маршрути у вигляді списку та на карті, а також видаляти обраний маршрут;

– підсистема взаємодії з календарем надає можливість створення подій у системному iOS календарі із вкладеним маршрутом та можливістю переходу до застосунку за посиланням з календаря;

– інтерфейс користувача забезпечує взаємодію з усіма функціями застосунку: додавання, редагування, збереження маршрутів, перегляд та планування.

Режими функціонування системи описують способи її застосування та визначають, які дії може виконувати користувач під час взаємодії із додатком.

Для розробленої системи можна виокремити такі основні режими роботи:

– режим побудови маршруту, в ньому користувач додає точки на карті або за допомогою пошуку, вибирає початкову та кінцеву точки або залишає цей вибір системі, після чого відбувається процес побудови маршруту;

– режим редагування маршруту, в якому користувач може змінювати статус кінцевої та початкової точки, видаляти локації або очищати маршрут повністю;

– режим пошуку точки інтересу, де після побудови маршруту користувач може знайти точку інтересу за ключовим словом, а система автоматично обирає локацію, яка географічно найближча до побудованого раніше маршруту, і додає її як додаткову зупинку у відповідне місце;

– режим перегляду збережених маршрутів, в якому користувач може переглядати збережені в список “Улюблені” маршрути, як у вигляді списку, так і на карті, а також видаляти маршрути зі списку;

– режим планування маршруту в календарі, користувач створює подію з побудованим маршрутом у системному календарі з можливістю відкриття відповідного маршрут безпосередньо в мобільному застосунку з події в календарі;

– у разі аварійного завершення застосунку останній побудований маршрут або щойно додані точки автоматично зберігаються, і при наступному запуску застосунку маршрут буде відновлений у тому стані, в якому він був до збою.

Перспективи розвитку застосунку можуть бути наступні:

– інтеграція із зовнішніми службами (погода, трафік);

– додаткові алгоритми оптимізації маршруту;

– розширення функцій календарного планування (автоматичне нагадування, прогнозування часу в дорозі тощо);

– реалізація функцій соціальної взаємодії (обмін маршрутами між користувачами);

– інтеграція механізмів монетизації, зокрема впровадження реклами в інтерфейсі застосунку або додавання платних функцій, таких як маршрути з урахуванням платних доріг або заторів, розширені параметри планування.

### 3.1.2 Вимоги до збереження інформації

Система має гарантувати правильне та стабільне збереження даних, пов’язаних із маршрутом користувача. Усі маршрути, які будуть додані до улюблених, повинні зберігатись коректно і залишатись доступними при повторному запуску застосунку. Крім того, система повинна зберігати поточний стан маршруту: якщо користувач лише додав точки, але ще не побудував маршрут, або вже побудував повний маршрут – усі дані мають залишатись без змін до моменту, поки користувач самостійно не очистить маршрут для створення нового. Збереження даних має відбуватись надійно і стабільно, без ризику втрати або спотворення інформації навіть у разі неочікуваного завершення роботи системи.

Всі дані користувача мають зберігатись локально на пристрої й не передаватись на зовнішні сервери, а доступ до маршрутів матиме лише поточний користувач. Це забезпечує повну приватність даних та усуває ризики несанкціонованого доступу. Також система не вимагатиме від користувача жодних особистих даних для користування. При взаємодії з календарем маршрути передаються через внутрішні захищені механізми, що виключає витік або компрометацію інформації.

У разі виникнення помилки система має інформувати користувача зрозумілим повідомленням із поясненням причини та можливими кроками для її усунення, що сприятиме зручності використання та покращенню загального користувацького досвіду.

Система має забезпечувати передбачувану поведінку: за однакових вхідних даних формується один і той самий маршрут. Час обробки зростатиме зі збільшенням кількості точок, але залишатиметься у прийнятних межах – навіть для

маршруту з великою кількістю точок він не повинен перевищувати 3 секунд, щоб користувач не відчував надмірного очікування.

Всі дані системи зберігатимуться локально на пристрої користувача без передавання на зовнішні сервери. Система повинна фіксувати актуальний стан маршруту – додані точки або вже побудований шлях – одразу після кожної суттєвої зміни, щоб у разі непередбачуваного закриття чи перезапуску застосунку користувач міг продовжити роботу над створенням маршруту. Маршрути, збережені в розділі «Улюблені», будуть залишатись доступними при кожному наступному запуску, а користувач, за бажанням, зможе в будь-який момент видалити їх чи скинути активний маршрут, щоб сформувати новий. Запис даних здійснюватиметься атомарно з перевіркою цілісності інформації, щоб запобігти пошкодженню або втраті інформації.

## 3.2 Вимоги до функціональних характеристик

У цьому підрозділі детально описано функціональні характеристики системи, зокрема наведено перелік ключових можливостей, які необхідно автоматизувати, критерії якості виконання кожної функції, форму представлення вихідної інформації, а також характеристики точності й швидкодії. Описані вимоги дозволяють чітко встановити, що саме повинна робити система і як вона має працювати для забезпечення максимальної зручності та ефективності використання.

### 3.2.1 Перелік функцій

Система має забезпечувати наступні функції.

Перша функція – побудова маршруту:

– відображення місцеперебування користувача;

– знаходження локацій за допомогою пошуку або безпосередньо на карті;

– обрані користувачем точки повинні автоматично додаватися до списку маршрутних точок, і мають відображатись у вигляді таблиці;

– можливість встановлення або зняття статусу початкової та кінцевої точки для будь-якої обраної точки маршруту;

– автоматичне визначення початкової точки як поточного місцеперебування користувача у разі, якщо вона не була задана вручну;

– автоматичне визначення кінцевої точки системою після побудови маршруту у разі, якщо користувач не задав її вручну;

– побудова маршруту на основі точок, заданих користувачем;

– редагування маршруту шляхом видалення або додавання точок;

– визначення тривалості та довжини побудованого маршруту;

– очищення побудованого маршруту;

– відновлення маршруту;

– автоматичне збереження поточного стану маршруту під час побудови або редагування;

– відновлення маршруту після випадкового закриття або перезапуску застосунку;

Друга функція – пошук точок інтересу (POI):

– пошук найближчого об’єкта до маршруту за ключовим словом;

– автоматичне додавання знайденої точки до маршруту з відображенням як на карті, так і у списку маршрутних точок із відповідною позначкою.

Третя функція – збереження маршрутів:

– додавання маршруту до розділу «Улюблені»;

– надання користувачу вибору щодо збереження маршруту з точкою інтересу або без неї;

– перегляд маршрутів у вигляді списку або на карті;

– видалення обраного маршруту зі списку “Улюблені”.

Третя функція – взаємодія з календарем:

– створення події в системному календарі з можливістю вибору дати, часу та подальшим збереженням маршруту;

– можливість відкриття збереженого маршруту в системі безпосередньо з події в календарі.

3.2.2 Вихідна інформація

Інформація, яка буде отримана в результаті роботи системи, має бути представлена:

– у вигляді інтерактивної карти з побудованим маршрутом і позначеними точками;

– у вигляді списку точок маршруту з адресами або назвами об’єктів;

– у формі стандартних повідомлень та підказок, зрозумілих для користувача.

3.2.3 Точність та час виконання

Характеристики точності та час виконання операцій:

– максимальний час побудови маршруту з кількістю точок до 20 не має перевищувати 2 секунд;

– максимальний час побудови маршруту з кількістю точок від 20 не має перевищувати 4 секунд;

– час пошуку точки інтересу за ключовим словом не повинен перевищувати 2 секунд;

– усі стандартні операції (збереження, редагування, видалення маршрутів) повинні виконуватись майже миттєво, без помітної затримки для користувача.

3.3 Вимоги до видів забезпечення

У цьому підрозділі сформульовано вимоги до математичного, інформаційного, програмного та технічного забезпечення системи побудови багатоточкових маршрутів для повсякденних поїздок.

### 3.3.1 Математичне забезпечення

Для побудови маршруту між точками, заданими користувачем, у системі має застосовуватись алгоритм пошуку найкоротшого шляху, який забезпечує визначення оптимального маршруту з урахуванням ваг ребер – дистанцій між точками на карті.

Алгоритми пошуку найближчих точок інтересу – система повинна визначати географічно найближчий об’єкт до побудованого маршруту за допомогою аналізу відстані між лінією маршрута та знайденими точками інтересу. На підставі цього обчислення визначається оптимальне місце для розташування нової точки в наявному маршруті.

### 3.3.2 Інформаційне забезпечення

Для збереження маршрутів, які були додані до списку «Улюблені», має використовуватися реляційна база даних, що дозволяє структурувати дані, зберігати інформацію про маршрути зі списками точок, а також додаткову інформацію, таку як назви та координати точок, а також назву маршрутів безпосередньо.

Система повинна забезпечувати збереження поточного стану застосунку, включаючи дані про поточний маршрут та внесені користувачем зміни, що дасть змогу відновити роботу після перезапуску або неочікуваного завершення. Інформаційне забезпечення має гарантувати цілісність і узгодженість даних навіть у разі тимчасових збоїв у роботі додатку.

### 3.3.3 Програмне забезпечення

Розробка системи повинна здійснюватися із застосуванням актуальних інструментів та офіційно підтримуваних середовищ розробки, призначених для мобільної операційної системи iOS.

Під час реалізації необхідно використовувати стабільні програмні компоненти, такі як фреймворки, бібліотеки та сервіси, що мають належну технічну документацію й відповідну ліцензійну підтримку від розробників.

Система повинна бути стабільною, зручною у використанні, ефективно використовувати ресурси пристрою, а її програмний код повинен мати чітку й зрозумілу структуру для забезпечення можливості подальшої підтримки та розвитку.

### 3.3.4 Технічне забезпечення

Система має бути сумісна з мобільними пристроями Apple (iPhone, iPad) та підтримувати актуальні версії операційної системи iOS.

Для коректної роботи системи пристрої повинні мати доступ до GPS (геолокації), достатній обсяг оперативної та постійної пам’яті для локального зберігання даних та стабільну роботу інтерфейсу користувача.

Система має стабільно працювати на пристроях із типово доступними ресурсами (оперативна пам’ять від 2 ГБ, внутрішня пам’ять достатня для зберігання локальних даних маршрутів, стабільна робота з GPS-модулем пристрою).

## Висновки до розділу 3

У розділі було сформульовано ключові вимоги до системи, що охоплюють її архітектурну побудову, поведінку під час використання, надійність роботи та збереження даних. Запропонована структура системи передбачає чітке розмежування функціональних підсистем – кожна відповідає за окрему частину взаємодії з користувачем, від роботи з картою до інтеграції з календарем.

Режими функціонування точно описують можливі сценарії використання, дозволяючи адаптувати логіку системи до потреб користувача на різних етапах

взаємодії. Усі функціональні можливості орієнтовані на забезпечення гнучкості при плануванні маршрутів та їх подальшому використанні.

Окрема увага приділена збереженню даних та конфіденційності – система не передбачає передавання інформації на зовнішні сервери, що гарантує безпеку користувацьких маршрутів. Крім того, впроваджено механізм відновлення поточного стану маршруту після неочікуваного завершення роботи, що позитивно впливає на стабільність і зручність застосунку.

Сформульовані вимоги створюють цілісне уявлення про необхідний функціонал і дозволяють перейти до етапу проектування з чітким розумінням технічних і користувацьких пріоритетів.

Крім основного функціоналу, у вимогах передбачено гнучкість масштабування системи та можливість її подальшого удосконалення відповідно до нових сценаріїв використання.

Такий підхід дозволяє розробити сучасний й перспективний продукт, готовий до подальшого розвитку та інтеграції з новими технологіями. Особлива увага приділена забезпеченню стабільності, збереженню даних та інтуїтивній взаємодії, що в сукупності підвищує якість користувацького досвіду.

# 4 ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

Даному розділі охоплює технології, методології та інструменти, які були обрані для створення мобільного застосунку на платформі iOS.

## 4.1 Клієнтська частина

Swift – це сучасна мова програмування, розроблена компанією Apple у 2014 році для створення застосунків під платформи iOS, macOS, watchOS та tvOS [8]. Вперше її було представлено на конференції розробників Apple у червні того ж року. Swift є наступницею мов C, C++ та Objective-C і поєднує в собі як низькорівневі примітиви – типи, керування потоком, оператори, – так і високорівневі можливості, зокрема об'єктноорієнтоване програмування, використання протоколів та узагальнених типів.

Swift було розроблено для того, щоб зробити процес створення програм у межах екосистеми Apple зручнішим, безпечнішим та більш ефективним. Завдяки простому та чистому синтаксису, Swift мінімізує кількість поширених помилок і підвищує швидкість програмування. Для підтримки стабільної роботи та ефективного використання ресурсів, Swift впроваджує автоматичне управління пам’яттю. Це реалізовано завдяки детермінованому механізму ARC, що забезпечує ефективний розподіл ресурсів без потреби в збиранні сміття.

Swift підтримує написання асинхронного та багатопотокового коду за допомогою вбудованих ключових слів, що робить мову більш зрозумілою та менш схильною до помилок. Окрім цього, Swift забезпечує захист від поширених помилок, таких як використання неініціалізованих змінних, переповнення масивів і числових значень, а також виявляє потенційні конфлікти доступу до даних ще на етапі компіляції.

Серед інших переваг – підтримка модульності та масштабованості, що полегшує організацію як малих проєктів, так і багаторівневих систем. Завдяки суворій типізації, організованій структурі та гнучкості в організації залежностей

Swift сприяє написанню чистого, повторно використовуваного та легко підтримуваного коду.

Окрім технічних переваг, Swift є мовою з відкритим вихідним кодом, що активно підтримується як спільнотою розробників, так і компанією Apple. Вона постійно вдосконалюється, для того, щоб відповідати сучасним вимогам безпеки та залишатися сумісною з новими версіями операційних систем. Така динаміка розвитку гарантує стабільність у довгострокових проєктах.

У межах сучасної екосистеми Swift також пропонує інструменти для зручної інтеграції сторонніх бібліотек. Одним із ключових рішень у цьому напрямку є Swift Package Manager – менеджер пакетів Swift.

Swift Package Manager – це вбудований інструмент від компанії Apple для керування залежностями [9]. Основною метою SPM є надання легкого способу встановлення сторонніх пакетів у проєкті. SPM автоматизує процес завантаження, компіляції та зв’язування зовнішніх пакетів, що значно спрощує підтримку та масштабування програмного коду.

У межах проєкту використовуються такі пакети:

– GoogleMaps і GooglePlaces – для реалізації картографічного інтерфейсу та пошуку геолокацій;

– SwiftyJSON – для зручної обробки JSON-даних, отриманих із зовнішніх API.

Для побудови користувацького інтерфейсу мобільного застосунку було вирішено використовувати фреймворк UIKit – основного інструменту для створення інтерфейсів користувача в середовищі iOS.

UIKit – це фреймворк від Apple, що використовується для розробки користувацьких компонентів (UI) додатків iOS [10]. Він містить широкий набір елементів – таких як кнопки, таблиці, колекції, панелі навігації та інші компоненти – які дозволяють створювати зручні та ефективні інтерфейси. UIKit надає можливість розробникам легко реалізовувати адаптивні інтерфейси, які коректно відображаються на пристроях з різною діагоналлю екрана.

## 4.2 Карти та геолокація

Для реалізації функціоналу побудови маршрутів використовується Google Maps API.

Google Maps API – це набір SDK та вебсервісів, які дозволяють розробникам включати карти, геокодування, маршрутизацію та пошук місць у додатку.

Основними модулями API є:

– SDK Maps (для iOS/Android) для вбудовування та налаштування карт з опціональними елементами, такими як накладені види та маркери;

– Places API для автозаповнення запитів та розрізнення типів місць;

– Directions API для планування маршрутів залежно від виду транспорту;

– Geocoding API для трансляції широти/довготи в адреси та адрес в широту/довготу;

– Distance Matrix API для обчислення часу/відстані подорожі в пакетному режимі.

З самого початку ця система послуг забезпечувала дуже високу точність, релевантність та масштабованість і донині залишається еталоном цифрової картографії на ринку.

## 4.3 Зберігання даних

Для зберігання інформації, необхідної для роботи мобільного застосунку, було обрано два основні підходи: UserDefaults та Core Data.

UserDefaults – це система зберігання невеликого обсягу даних у формі ключ-значення. Це поширене рішення в застосунках на Swift для збереження налаштувань, які мають залишатися доступними між запусками програми. Найчастіше UserDefaults використовують для зберігання користувацьких налаштувань (наприклад, обрані параметри інтерфейсу) або для фіксації стану. Таке рішення є зручним для оперативного доступу до невеликого обсягу даних без

необхідності побудови складної структури збереження, що робить його оптимальним для простих сценаріїв взаємодії з користувачем.

Core Data – це функціональний фреймворк від Apple, призначений для побудови стійкої моделі збереження постійних даних у мобільних застосунках [11]. Завдяки редактору моделі даних (Data Model Editor), розробники мають можливість визначати типи даних і встановлювати зв’язки між ними, а також генерувати відповідні класи. Core Data керує об’єктами під час виконання програми, забезпечуючи ефективне управління пам’яттю, фільтрацію, сортування, валідацію даних та відстеження змін, що робить його універсальним та зручний інструмент для організації локального зберігання даних у мобільних застосунках.

Core Data виступає не тільки як база даних, але і як об’єктно-орієнтований граф, набір активних об’єктів у пам’яті. Таким чином, об’єкти можуть бути логічно пов’язані, і навігація між ними не вимагає складних SQL запитів.

Core Data оптимізована для продуктивності завдяки використанню «лінивого завантаження» (lazy loading). Дані завантажуються лише за необхідності, що особливо корисно при роботі з великими наборами даних.

Core Data безпосередньо взаємодіє з SQLite, обмежуючи потребу розробника взаємодіяти з SQL напряму.

SQLite – це реляційна база даних, що функціонує без необхідності запуску окремого серверного компонента [12]. SQLite підтримує стандарт SQL, транзакцій, індексації та складних запитів, що дозволяє ефективно управляти великими наборами даних. Попри те, що це вбудована та компактна система, SQLite підтримує всі можливості реляційної моделі та стандарту SQL, такі як транзакційність та гарантії ACID. Завдяки відкритому вихідному коду та кросплатформності, вона сумісна з широким спектром сучасних мов програмування. Однією з переваг SQLite є те, що вона ще більш гнучка з погляду типів: хоча вона дозволяє вам вказувати тип даних, ці типи не вимагаються під час початкового схемування, що дозволяє, наприклад, вставити рядок у поле типу INTEGER. Попри компактність, SQLite використовується у великій кількості

мобільних додатків, браузерів, операційних систем і вбудованих систем завдяки своїй стабільності, швидкості роботи та простоті інтеграції.

### 4.4 Середовище розробки

Як основний інструмент розробки було обрано інтегроване середовище Xcode, яке є офіційним середовищем розробки від Apple. Xcode охоплює повний цикл розробки додатків, від написання коду до тестування і подання до App Store. Xcode має потужний редактор коду, засіб автоматичного налагодження, що виявляє помилки коду, а також симулятор iOS та інструменти тестування продуктивності. Середовище містить велику частину офіційної документації для розробників Apple, а також інструмент Interface Builder, який дозволяє створювати графічні інтерфейси візуально. Xcode містить модифіковану версію збору компіляторів GNU Compiler Collection, що забезпечує підтримку мов програмування C, Swift, Objective-C, Objective-C++, AppleScript, Java, Python та Ruby.

### 4.5 Контроль версій

Для ефективного управління кодом застосунку використовується система контролю версій Git. Git полегшує ведення короткого запису змін у коді, координацію роботи кількох розробників та контроль потоку нових функцій. Застосування Git у поєднанні з хостингом репозиторіїв на таких платформах, як GitHub або GitLab, спрощує процеси рев'ю коду, роботи з гілками (branching), злиття (merging), виявлення та усунення конфліктів, а також повернення до попередніх стабільних версій коду в разі потреби.

Для підтримки високої якісті коду, використовуються інструменти статичного аналізу коду, інтегровані в Xcode; вони не лише виявляють помилки під час компіляції, але й підсвічують потенційні помилки та надають поради для покращення коду. Також застосовується ручне тестування функціоналу, що дозволяє перевірити коректність роботи застосунку в реальних сценаріях

використання, включаючи перевірку обробки помилок, логіки інтерфейсу та взаємодії з користувачем. Розширені тестування на декількох пристроях і версіях iOS допомагають усувати помилки та контролювати стабільність.

Висновки до розділу 4

Обраний технологічний стек надає уніфіковану і цілісну екосистему, яка повністю достатня для реалізації мобільного додатку для створення багатоточкових маршрутів.

Swift забезпечує безпечний, швидкий і виразний спосіб розробки, а інструменти UIKit для проєктування інтерфейсів є гнучкими і зрозумілими.

Для досягнення ключової функції карт у проєкті будуть використовуватися пакети Google Maps і Google Places, що означає інтеграцію з API Google Maps. Обробка відповіді у форматі JSON від сервісів Google реалізується через бібліотеку SwiftyJSON.

Для зберігання налаштувань і складних структур даних використано комбінацію UserDefaults та Core Data, причому остання спирається на продуктивність і універсальність SQLite, не вимагаючи від розробника роботи з низькорівневим SQL.

Розробницьке середовище Xcode забезпечує повний цикл створення, налагодження й профілювання застосунку, а Git у поєднанні з платформами GitHub/GitLab гарантує контроль версій і зручне командне співробітництво. Інструменти статичного аналізу, вбудовані в Xcode, та ретельне ручне тестування підсилюють якість коду й стабільність продукту на різних версіях iOS. У комплексі ці рішення забезпечують високу продуктивність, масштабованість і простоту підтримки проєкту, що створює міцний фундамент для подальшого розвитку й удосконалення застосунку.

# 5 РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

## 5.1 Функціональна модель системи

Користувач є ключовою фігурою системи, який взаємодіє з нею і має можливість створювати маршрути з кількома зупинками, додавати точки інтересу, а також зберігати та планувати поїздки.

Користувач має доступ до таких функції системи:

– перегляд карти та визначення місцезнаходження – користувач може переглядати інтерактивну карту, яка показує його поточне місцезнаходження;

– додавання точок маршруту – користувач додає точки маршруту використовуючи пошук за адресою або за допомогою пошуку на карті;

– встановлення статусу точок – користувач може задавати або змінювати статус точок маршруту (початкова, проміжна, кінцева) або залишити це на розсуд системи;

– побудова маршруту – після вибору точок маршруту система автоматично прокладає шлях, який проходить через обрані користувачем точки;

– редагування маршруту – користувач має змогу змінювати маршрут, видаляючи чи додаючи точки;

– пошук та додавання точок інтересу – користувач може шукати та додавати точки інтересу, наприклад, кафе чи магазини, за допомогою ключових слів, після чого система автоматично додає знайдену точку у вже побудований маршрут;

– перегляд інформації про маршрут – користувач переглядає сформований маршрут, включаючи довжину шляху та приблизний час, необхідний для його проходження;

– збереження маршрутів – користувач зберігає створені маршрути до списку «Улюблені», з можливістю перегляду його як на карті, так і у вигляді списку точок шляху;

– планування маршруту – користувач створює подію у системному календарі, прив’язану до маршруту, з можливістю швидкого переходу до застосунку для подальшої навігації.

Таким чином, користувач може взаємодіяти з системою декількома способами:

– користувач може прокласти маршрут, самостійно вказавши початкову та кінцеву точки, або залишити цей вибір системі, яка визначить початкову точку як його поточне місцезнаходження, а кінцеву – як оптимальну з існуючих точок;

– користувач може додавати додаткові точки інтересу, які система автоматично врахує при формуванні маршруту;

– користувач може зберігати маршрути регулярних або повторюваних поїздок, створюючи персональний список, який буде доступний для перегляду;

– користувач може планувати поїздки заздалегідь, створюючи події в календарі, та швидко переходити до системи безпосередньо з календарної події.

На основі вищезазначених функцій була створена діаграма варіантів використання, яка наочно демонструє, як користувач взаємодіє з системою та які функціональні можливості вона надає.

Діаграма варіантів використання представлена у кресленику ІС13.230БАК.005 Д1. Вона ілюструє сценарії взаємодії єдиного актора – користувача – з функціональними можливостями системи. Діаграма побудована з урахуванням вимог до функціонального моделювання і показує, як користувач ініціює основні дії, пов’язані з побудовою, редагуванням, збереженням та плануванням маршрутів. Усі варіанти використання логічно впорядковані та пов’язані між собою відповідно до передбачених сценаріїв.

Користувач – центральна фігура моделі. Він ініціює кожен сценарій взаємодії із системою: шукає місця, додає точки, зберігає готові маршрути, планує поїздки за допомогою системного календаря.

Ключові сценарії використання наведені нижче.

Користувач має можливість здійснювати пошук потрібної точки двома способами: або шляхом введення адреси у текстове поле, або безпосередньо на карті, обираючи локацію вручну.

Користувач може додавати знайдені місця до маршруту, а також має можливість змінювати статус кожної з них – наприклад, позначити як початкову

або кінцеву. Якщо статус не визначено вручну, система автоматично встановлює початкову точку за поточним місцезнаходженням, а кінцеву – на основі логіки оптимізації.

## 5.2 Структура системи

Процес створення системи для побудови багатоточкових маршрутів починається з розробки архітектурної моделі, котра містить в собі основні складові та принципи їх взаємодії. Це закладає підґрунтя для подальшої деталізації функціоналу та розподілу системи на логічно пов’язані блоки. Структурна схема ілюструє зв’язки між компонентами, що дає цілісне уявлення про функціонування та логіку роботи всієї системи. Структурна схема представлена у кресленику ІС13.230БАК.005 Э1.

Перший рівень, з яким взаємодіє користувач – представлення. Тут представлені такі модулі як:

– модуль пошуку локацій - надає текстове поле та інтерактивну карту для вибору координат;

– модуль пошуку точок інтересу – приймає введене користувачем ключове слово – наприклад, «кафе» чи «АЗС» – і передає його далі на рівень обробки даних;

– модуль взаємодії з картою – відображає шлях та маркери, реагує на жести;

– модуль взаємодії з маршрутами – відображає сформований шлях із послідовним з’єднанням усіх вибраних точок, демонструє їх перелік та надає користувачеві можливість одним натисканням зберегти маршрут до списку «Улюблені» або запланувати його у системному календарі.

Другий рівень – це рівень обробки даних. Цей шар концентрує бізнес-логіку застосунку. У ньому присутні наступні модулі:

– модуль точок маршруту – додає, редагує статус і видаляє точки з маршруту, а також виконує логіку для знаходження точок інтересу;

– модуль побудови маршруту – збирає точки, запускає оптимізацію маршруту, знаходить шлях, який охоплює всі точки, передає створений маршрут на рівень представлення;

– модуль геокодування – конвертує адреси у координати та навпаки, кешує результати запитів;

– модуль планування маршруту у календарі – інтегрується з системним календарем, створює події та нагадування;

– модуль збереження маршруту приймає маршрути із відповідними точками, фіксує дані у сховищі й повертає підтвердження успішної операції.

Рівень зовнішніх сервісів відповідає за інтеграцію застосунку зі сторонніми платформами. Цей рівень не реалізує бізнес-логіку сам по собі, натомість надає інструменти та дані, які використовуються модулями системи для виконання основних функцій.

У системі реалізовано взаємодію з двома основними зовнішніми сервісами:

– Google Maps API – відповідає за картографічні сервіси, включаючи геокодування адрес, відображення побудованого маршруту між точками, а також пошук точок інтересу (POI) на основі ключових слів;

– CalendarKit – забезпечує створення подій у системному календарі пристрою iOS.

Окремим шаром виділено реляційну базу даних, де зберігаються всі маршрути й відповідні їм точки.

Описана структура дозволяє створити стабільну, просту та ефективну систему для щоденного планування маршрутів.

## 5.3 Модель бази даних

У цьому проєкті застосовується локальна база даних, яка реалізована за допомогою вбудованого фреймворку Core Data. Цей фреймворк дозволяє зберігати структуровані дані на пристрої та надає розробникам зручний спосіб взаємодії з базою даних через об'єктно-орієнтовану модель.

База даних містить дві сутності: «RouteModel» та «WaypointModel».

Таблиця RouteModel містить інформацію про збережені маршрути та їх основні характеристики. Її структура представлена в таблиці 5.1.

Таблиця 5.1 – Структура таблиці RouteModel

| Назва поля | Тип даних | Опис поля |
|---|---|---|
| id | UUID | Унікальний ідентифікатор |
| title | String | Назва маршруту |
| createdAt | Date | Дата та час створення маршруту |
| polyline | String | Закодоване представлення маршруту для відображення на карті |
| scheduled | Boolean | Ознака того, чи було маршрут заплановано через системний календар |

Таблиця WaypointModel містить інформацію про точки маршруту їх координати та додаткові характеристики. Її структура представлена в таблиці 5.2.

Таблиця 5.2 – Структура таблиці WaypointModel

| Назва поля | Тип даних | Опис поля |
|---|---|---|
| id | UUID | Унікальний ідентифікатор |
| title | String | Назва точки маршруту |
| lat | Double | Географічна широта точки |
| lng | Double | Географічна довгота точки |
| order | Integer | Порядковий номер точки в маршруті |

| Назва поля | Тип даних | Опис поля |
|---|---|---|
| isPOI | Boolean | Вказує, чи є точка об’єктом інтересу |
| placeId | String | Ідентифікатор місця |
| routeModelId | UUID | Ідентифікатор маршруту |

Між RouteModel і WaypointModel існує зв’язок один до багатьох: один маршрут може мати кілька точок маршруту, а кожна з цих точок належить лише одному маршруту. Схема моделі бази даних представлена на рисунку 5.2.

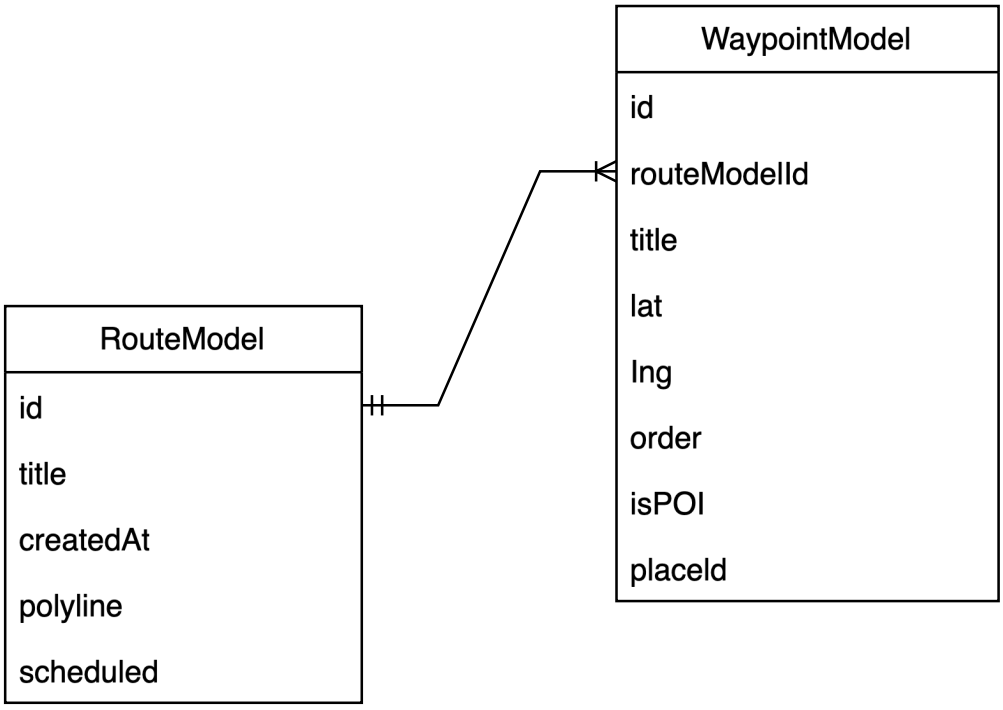


Рисунок 5.2 – Схема бази даних

5.4 Передавання та обробка даних

5.4.1 Вхідні дані

У системі планування багатоточкових маршрутів користувач може самостійно вводить дані через інтерфейс мобільного застосунку. Серед цих даних:

– координати точок маршруту, які користувач може вказати вручну (через пошук або натискання на карту);

– тип точки маршруту – початкова, проміжна або кінцева;

– ключові слова для пошуку точок інтересу;

– назви маршрутів – для збереження у список «Улюблені»;

– дата та час подорожі – під час створення події в системному календарі.

5.4.2 Вихідні дані

Після обробки вхідних даних система формує вихідні дані у зручній для користувача формі:

– побудований маршрут, який можна побачити на інтерактивній карті;

– інформація про протяжність маршруту та приблизний час його проходження;

– список точок маршруту із відповідними статусами;

– маршрути в розділі “Улюблені” з можливістю перегляду у вигляді списку точок маршруту або на карті;

– подія в календарі, що містить дату, час і посилання на маршрут.

5.4.3 Обробка даних

Основні процеси обробки в системі:

– обчислення оптимального маршруту між усіма обраними точками шляху за допомогою алгоритму Дейкстри;

– визначення початкової/кінцевої точки, якщо вони не були задані вручну;

– пошук найближчої точки інтересу (POI) до вже побудованого маршруту;

– оновлення відображення карти, таблиці з точками та загальної інформації про маршрут.

### 5.4.4 Передавання даних

Оскільки система повністю локальна, усі процеси обробки та передавання даних відбуваються безпосередньо на пристрої користувача. Зовнішні сервери не використовуються, що гарантує конфіденційність інформації. Комунікація між модулями застосунку реалізується засобами iOS – через локальні API та сховища (UserDefaults, Core Data).

Дані для події в календарі передаються до системного календаря iOS за допомогою фреймворку EventKit.

### 5.4.5 Зберігання даних

Маршрути та пов’язані точки зберігаються у структурованому вигляді у локальної бази даних.

Інформація про останній активний маршрут, а також дії користувача під час сесії, фіксуються у локальному тимчасовому сховищі для забезпечення відновлення роботи після випадкового закриття або перезавантаження застосунку.

## 5.5 Архітектура програмного забезпечення

При створенні системи для побудови багатоточкових маршрутів використовується сучасний модульний підхід, заснований на шаблоні проєктування MVC (Model-View-Controller). Вибір цієї архітектури зумовлений потребою чітко розділити зони відповідальності між логікою представлення, бізнес-логікою та опрацюванням даних, що суттєво полегшує підтримку, розширення та тестування застосунку.

Перший рівень – рівень представлення. Цей рівень відповідає за взаємодію з користувачем та користувацький інтерфейс. Він реалізується через класи контролерів, які забезпечують функціональність перегляду карти, відображення та

вибору точок маршруту, показу оптимізованих маршрутів, а також взаємодію з різними вікнами та елементами керування.

Другий рівень – бізнес-логіка. На цьому рівні відбувається обробка даних, які ввів користувач, валідація, формування та оптимізація маршрутів, а також реалізація логіки взаємодії між компонентами інтерфейсу та сервісами. До цього рівня належать класи такі як WaypointsManager, RouteOptimizer, PopupManager, які відповідають за внутрішню логіку оптимізації маршрутів, управління точками маршруту та додатковими елементами інформаційного інтерфейсу.

Наступний рівень – рівень моделі даних. Цей рівень відповідає за управління і представлення даних, які зберігаються на пристрої. Тут використовується технологія CoreData, представлена класами CoreDataManager, RouteModel+CoreDataProperties, WaypointModel+CoreDataProperties. Вони забезпечують зберігання маршрутів, точок маршруту, взаємодію з локальним сховищем.

Ще один рівень – сервісний рівень. Цей шар відповідає за взаємодію з зовнішніми ресурсами, такими як Google Maps API, Places API та Directions API. На цьому рівні розташовані класи: PlacesService, DirectionsService та POIFinder, котрі виконують мережеві запити та одержують дані для подальшої обробки в застосунку. Цей рівень цілковито приховує подробиці роботи з API від бізнес-логіки, що забезпечує стабільність застосунку навіть при змінах у зовнішніх API.

### 5.5.1 Діаграма класів

Діаграма класів системи побудови багатоточкових маршрутів представлена у кресленику ІС13.230БАК.005 Д2. Вона показує чітко структуровану архітектуру, що складається з чотирьох основних рівнів: представлення, бізнес-логіка, сервісний та рівень даних. Це забезпечує модульність, що значно спрощує підтримку та тестування застосунку.

Рівень представлення відповідає за інтерфейс користувача. До нього належать контролери:

– MapViewController, що відповідає за динамічне оновлення інтерфейсу карти та обробку жестів користувача;

– FavoritesHomeViewController, що показує список збережених маршрутів;

– FavoritesMapViewController та FavoritesListViewController, які демонструють маршрути відповідно у вигляді карти або списку точок.

Другий шар – бізнес-логіка. Центральним класом тут є RouteBuilder, який виконує важливі функції такі як clearRoute, buildOptimisedRoute, drawRoute, findPOIAndRebuild, saveCurrentRoute та showRoute. Посередниками між інтерфейсом користувача і бізнес-логікою виступають класи MapViewModel та FavoritesViewModel. FavoritesViewModel WaypointsManager відповідає за управління точками маршруту (додавання, видалення, зміна порядку). RouteOptimizer займається розрахунком оптимального маршруту. Для роботи з інтерфейсом задіяні допоміжні класи MarkerManager, PopupManager, MapStyleManager та AlertManager.

Наступним є сервісний рівень, який відповідає за комунікацію з зовнішніми ресурсами. Класи PlacesService, DirectionsService, GeocoderService, POIFinder та CalendarService приховують деталі взаємодії з API Google Maps та EventKit, забезпечуючи стабільну роботу бізнес-логіки.

Рівень даних представлений класами CoreDataManager, RouteModel та WaypointModel. CoreDataManager управляє створенням, читанням та видаленням маршрутів і точок маршруту, а RouteModel має зв'язок «один-до-багатьох» із WaypointModel, який реалізує методи для отримання впорядкованого списку точок.

Поділ системи на окремі модулі забезпечує ізоляцію логіки обробки даних від інтерфейсної частини, що забезпечує надійність та стабільність роботи застосунку.

### 5.5.2 Специфікація функцій

Цей підпункт узагальнює роботу ключових методів, що забезпечують повний цикл від дії користувача до відображення результату. Описано призначення кожної

функції, походження її вхідних даних та наслідок виконання – тобто зміни у стані моделі, інтерфейсу чи сховища. В таблиці 5.3 наведений перелік основних функцій, які використовуються в проєкті.

Таблиця 5.3 – Специфікація функцій

| Назва функції | Вхідні параметри | Опис функції |
|---|---|---|
| clearRoute | – | Скидає побудований маршрут, чистить маршрут на карті, зупиняє мережу, чистить таблицю із точками, які обирав користувач |
| buildOptimisedRoute | – (waypoints, які додав користувач) | Виконує обчислення оптимального порядку проходження точок та викликає метод побудови маршруту на карті. |
| drawRoute | polyline (лінія маршруту), order (оптимізований порядок проходження точок) | Формує вид маршруту на карті: лінія та маркери. |
| addWaypoint | title (назва точки), placeId (унікальний ідентифікатор точки), coord (координати точки) | Створює новий об’єкт Waypoint та додає його до списку в WaypointsManager. Оновлює таблицю з точками маршруту |
| findPOIAndRebuild | query (точка, задана користувачем) | Знаходить точку інтересу по назві, яка була задана користувачем вздовж побудованого маршруту, додає точку та запускає переобчислення |

| Назва функції | Вхідні параметри | Опис функції |
|---|---|---|
| saveCurrentRoute | – (waypoints, які додав користувач, які були збережені у локальному сховищі) | Зберігає маршрут до списку улюблених, визначає GPS-координату користувача у випадку, якщо той не задав початкову точку, а також викликає функцію clearRoute |
| showRouteFromCalendar | saved (об’єкт RouteModel), fromCalendar | Відновлює та візуалізує маршрут, відкритий користувачем із системного календаря: малює полілінію на карті, відображає відповідні маркери, синхронізує таблицю точок та встановлює відповідний стан інтерфейсу. |
| addEvent | date, routeID | Створює EKEvent, додає у календар користувача |
| humanAddress | GPS-координата | Виконує зворотне геокодування через GMSGeocoder, щоб перетворити координати на адресу місцезнаходження користувача |
| loadWaypoints | (Дані беруться з кешу) | Після запуску підтягує збережені точки, відновлює маркери й стани. |
| load | (Дані беруться з кешу) | Десеріалізує Waypoints у пам’ять при старті. |
| save | (Дані беруться з кешу) | Зберігає поточний стан точок маршруту |
| remove | index : Int (індекс точки маршруту) | Видаляє точку, оновлює кеш, викликає save(). |

| Назва функції | Вхідні параметри | Опис функції |
|---|---|---|
| updateUIForCurrentState | - | Керує видимістю й доступністю елементів керування залежно від стану маршруту |

Такий структурований перелік дає змогу простежити ланцюжок викликів і зрозуміти, як окремі модулі поєднуються у єдину логіку. Посилання на вихідний код зображено на рисунку А.1 Додатку А.

5.5.3 Діаграма послідовності

Діаграма послідовностей, представлена у кресленику ІС13.230БАК.005 Д3, вона демонструє взаємодію компонентів системи побудови багатоточкових маршрутів. На початку користувач надає доступ до геопозиції і бачить інтерфейс системи, після чого він може знаходити й додавати локації до маршруту. Інтерфейс відображає додані точки як на карті, так і у вигляді списку, а користувач може встановити статус кожної з них (початкова, проміжна або кінцева).

Наступним етапом є побудова маршруту, яка відбувається через модуль формування маршруту. Після формування маршруту система пропонує користувачеві додати POI. Якщо користувач погоджується, то система отримує запитує користувача назву точки і знаходить найближчу POI та додає її до маршруту; у випадку відмови маршрут не змінюється. Система повертає маршрут для подальшого використання.

Після створення маршруту, він може бути збережений у реляційній базі даних. Модуль, що відповідає за взаємодію з базою, підтверджує, що операція пройшла успішно. Користувач також може переглядати збережені маршрути.

Останній етап – це можливість спланувати маршрут у системному календарі. Після того, як маршрут збережено, користувач отримує підтвердження, що все пройшло успішно.

Така схема взаємодій забезпечує зрозумілий і чіткий процес роботи користувача з системою, забезпечуючи ефективність і зручність у повсякденному плануванні маршрутів.

Висновки до розділу 5

У розділі була розроблена та описана функціональна модель системи, котра дозволяє користувачам створювати маршрути, додавати точки інтересу, а також планувати й зберігати поїздки. Користувач є центральною фігурою системи і має кілька різних способів взаємодії з нею. Запропоновані сценарії включають перегляд карти, додавання нових локацій, зміну статусів точок та редагування створеного маршруту. Система визначає оптимальну послідовність проходження точок і дозволяє знаходити та включати додаткові об’єкти інтересу.

Було розглянуто архітектуру програмного забезпечення, побудована на основі модульного підходу із застосуванням шаблону MVC, що передбачає чіткий розподіл відповідальності між різними складовими застосунку. Вона включає рівні представлення, бізнес-логіки, збереження даних, а також інтеграції із зовнішніми сервісами. Це дозволяє легко підтримувати та розширювати застосунок, що забезпечує його стабільність та зручність тестування.

Також була розроблена модель бази даних з використанням технології Core Data, що дозволяє зберігати, оновлювати та отримувати інформацію про маршрути та їхні точки.

Крім того, було проведено детальний аналіз процесів передачі, обробки та зберігання інформації. Система обробляє всі дані безпосередньо на пристрої користувача, що гарантує високий рівень конфіденційності. Дані передаються між модулями через внутрішні механізми та локальні сховища, що забезпечує надійну взаємодію між компонентами застосунку.

# 6 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

## 6.1 Змістовна постановка задачі

Припустімо, є набір точок на місцевості ($n$ локацій, котрі планує відвідати користувач). Їх доцільно представити у вигляді вершин $V = \{1,2,\dots,n\}$ неорієнтованого зв'язного графа $G\,(V,U)$, де $U$ – множина ребер (доріг) між вершинами, а вага кожного ребра – відстань. Потрібно визначити маршрут, який починається у заданій початковій точці, проходить через усі проміжні пункти, визначені користувачем, і, за потреби, завершується у вказаній кінцевій вершині.

Нехай множина обраних точок, котрі потрібно обов'язково відвідати, є підмножиною $S \subseteq V$, де $S = \{s_1, s_2, \dots, s_k\}$. Приклад графа дорожньої мережі для побудови маршруту подано на рисунку 6.1.

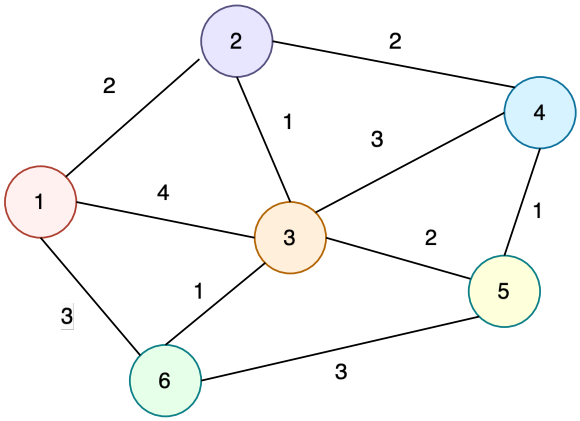


Рисунок 6.1 – Приклад графа дорожньої мережі для побудови маршруту

Початковою вершиною маршруту $v_{start}$ розглядають або локацію, вказану користувачем, або, у разі її відсутності, використовують актуальне геолокаційне положення користувача. Кінцева вершина $v_{end}$ опціональна – якщо її не вказано, маршрут може завершитися на тій вершині з підможини $S$, яку було досягнуто останньою.

Задача зводиться до визначення шляху, що включає всі точки з множини $S$ та мінімізує загальну вагу маршруту. Поставлена задача нагадує проблему

найкоротшого шляху, яку потрібно пройти декілька разів задля досягнення кожної з необхідних вершин.

У підсумку формується послідовність вершин, котра починається у початковій точці, проходить через усі визначені користувачем точки (множина *S*) та завершується у кінцевій точці, якщо така передбачена. Водночас маршрут має мінімізувати сумарну довжину, що є критерієм оптимальності.

Приклад маршруту, який може бути отримано наведено на рисунку 6.2.

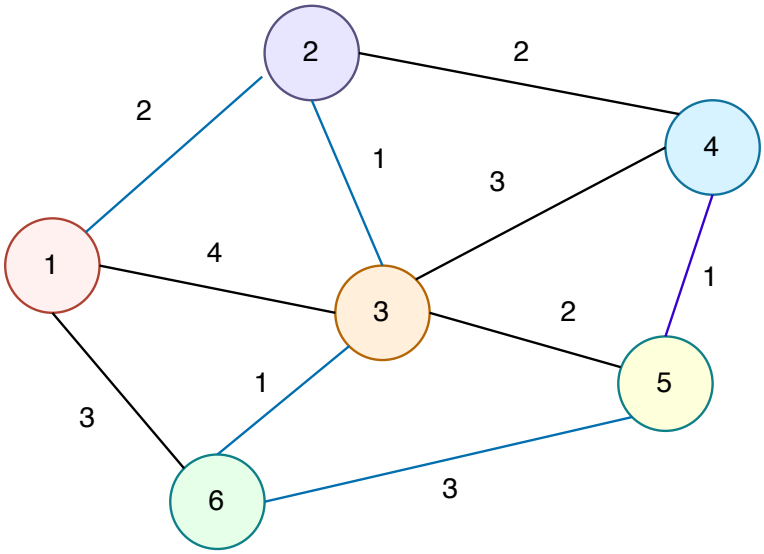


Рисунок 6.2 – Приклад побудованого маршруту в графі за заданими точками

## 6.2 Математична постановка задачі

Розглядається неорієнтований граф $G = (V, U, w)$, де $V = \{v_1, v_2, \dots, v_n\}$ – множина вершин, а $U \subseteq V \times V$ – множина ребер графа та $w: U \to \mathbb{R}^+$ – функція ваг, яка кожному ребру $(u, v) \in U$ ставить у відповідність додатне число $w(u, v)$, що означає відстань між вершинами $u$ і $v$.

Також задана $v_{start}$ – стартова вершина маршруту та опціонально $v_{end}$ – кінцева вершина шляху. $S \subseteq V$ – множина обраних користувачем точок, які потрібно обов'язково відвідати.

Необхідно побудувати маршрут $P = (v_{start} = p_0, p_1, p_2 \dots p_k, p_{k+1} = v_{end})$, який задовольняє умови, наведені нижче. Якщо ж кінцева вершина $v_{end}$ не вказана,

маршрут вважається завершеним на останній відвіданій вершині з множини *S*.

Кожна пара послідовних вершин у маршруті з'єднана ребром у графі

$$\forall i, (p_i, p_{i+1}) \in U. \tag{6.1}$$

Кожна вершина з множини S входить у маршрут хоча б один раз

$$S \subseteq \{p_1, p_2, \dots, p_k\}. \tag{6.2}$$

Серед усіх маршрутів, які задовольняють умови 1 та 2, шукається той, що мінімізує сумарну вагу

$$min \sum_{i=0}^{k} w(p_i, p_{i+1}). \tag{6.3}$$

## 6.3 Обґрунтування методу розв’язання

Впродовж тривалого часу питання побудови найкоротших шляху залишається актуальним, тому було запропоновано значну кількість алгоритмічних рішень, кожне з яких має свої особливості застосування. Залежно від структури графа та вимог до точності чи швидкодії, кожен метод має свої переваги та недоліки.

### 6.3.1 Алгоритм Дейкстри

Алгоритм Дейкстри був розроблений Едсгером Дейкстрою. Принцип роботи алгоритму полягає у поступовому виборі найближчої вершини та перегляді шляхів через неї до суміжних вершин, послідовно оновлюючи відстані [13]. Завдяки використанню пріоритетної черги алгоритм демонструє обчислювальну складність $O((V + E)\ log\ V)$, що забезпечує високу продуктивність навіть у випадку великих

графів. Крім того, потреби щодо пам’яті залишаються порівняно невисокими – $(O(V))$, що є важливим критерієм для використання у мобільних застосунках. Враховуючи особливості поставленої задачі – побудову маршрутів через множину проміжних точок – цей алгоритм може бути повторно використаний кілька разів, що є допустимим з огляду на швидкодію мобільного застосунку.

### 6.3.2. Алгоритм Беллмана–Форда

Алгоритм Беллмана–Форда був створений для вирішення задач пошуку шляхів у графах з негативними вагами ребер, проте він не може коректно працювати, якщо у графі є цикли з негативною сумарною вагою [14]. Принцип його дії ґрунтується на поступовому оновленні відстаней між вершинами шляхом багаторазового проходу по всіх ребрах графа. Обчислювальна складність алгоритму складає $O(V \cdot E)$, що значно вище порівняно з алгоритмом Дейкстри.

### 6.3.3 Алгоритм A*

Алгоритм A* належить до класу евристичних, що використовуються для пошуку найкоротших шляхів у графах із додатними вагами ребер. Алгоритм завжди знаходить оптимальний шлях, якщо евристика є допустимою.

Оцінка кожної вершини x виконується за формулою:

$$f(x) = g(x) + h(x), \tag{6.4}$$

де $g(x)$ – вартість маршруту від початкової вершини до вершини $x$, а $h(x)$ – евристична оцінка відстані від x до цільової вершини. Таким чином, A* намагається збалансувати реальні витрати з передбачуваними [15].

Однак, треба зазначити, що для багатоточкової маршрутизації даний метод потребує багаторазового застосування для кожної пари точок, що може спричинити

збільшення обчислювальних витрат і зниження загальної ефективності у порівнянні з іншими методами.

### 6.3.4 Алгоритм Флойда – Воршелла

Алгоритм Флойда – Воршелла – це класичний динамічний алгоритм, призначений для знаходження найбільш коротких шляхів між усіма парами вершин у графі [16]. Він може бути використаний як для орієнтованих, так і для неорієнтованих графів, за умови, що значення ваг на ребрах є невід'ємними або нульовими.

Основний механізм полягає у поетапному вдосконаленні поточних оцінок найкоротших шляхів використовуючи кожну вершину як можливий проміжний пункт. На кожній ітерації перевіряється, чи можна дістатися з вершини i до вершини j швидше через третю вершину k, тобто чи виконується:

$$d(i,j) > d(i,k) + d(k,j). \tag{6.5}$$

Якщо так – відстань оновлюється. Цей процес повторюється для всіх можливих трійок вершин, що гарантує врахування всіх можливих маршрутів у графі.

Незважаючи на універсальність, його кубічна обчислювальна складність $O(V^3)$ та високі вимоги до пам'яті $O(V^2)$ роблять цей метод малопридатним для мобільного додатку, особливо при великих графах дорожньої мережі.

### 6.3.5 Алгоритм Джонсона

Алгоритм Джонсона поєднує складові алгоритмів Беллмана–Форда та Дейкстри, що дозволяє ефективно працювати з від'ємними вагами, якщо у графі немає негативних циклів. Його складність $O(V^2 \, log \, V + VE)$ та складна реалізація роблять його використання неефективним для задачі без негативних ваг [17].

Порівняння алгоритмів пошуку найкоротших маршрутів для задачі багатоточкових маршрутів представлено у таблиці 6.1.

Таблиця 6.1 – Порівняння алгоритмів пошуку найкоротших шляхів для задачі багатоточкових маршрутів

| Алгоритм | Обчислювальна складність | Використання пам’яті | Наявність негативних ваг | Відповідність до поставленої задачі |
|---|---|---|---|---|
| Дейкстри | $O((V + E) \log V)$ | $O(V)$ | Ні | Висока |
| Беллмана–Форда | $O(V \cdot E)$ | $O(V)$ | Так | Низька |
| A* | $O(E)$ або експоненційна | $O(V)$ | Ні | Середня |
| Флойда–Воршелла | $O(V^3)$ | $O(V^2)$ | Ні | Низька |
| Джонсона | $O(V^2 \log V + VE)$ | $O(V^2)$ | Так | Низька |

Отже, з урахуванням проаналізованих характеристик алгоритмів, найбільш доцільним для впровадження у мобільному застосунку, призначеному для побудови маршрутів через кілька точок, є саме підхід Дейкстри. Це рішення виправдане його стабільною ефективністю, помірним споживанням оперативної пам’яті та відповідністю особливостям структури дорожнього графа, що виключає потребу обробки від’ємних ваг.

## 6.4 Опис методу розв’язання

Враховуючи результати дослідження, для вирішення вирішення поставленої задачі обрано покроковий підхід із застосуванням алгоритму Дейкстри. Алгоритм

Дейкстри використовується для обчислення найкоротших шляхів від однієї вершини до всіх інших у графі з додатними вагами на ребрах.

Основна ідея реалізована таким чином:

– початкову вершину $v_{start}$ користувач або сам обирає, або ця точка задається автоматично як поточне геолокаційне положення;

– множина обов'язкових для відвідування точок $S$ визначається користувачем;

– кінцева вершина $v_{end}$ може бути зафіксованою користувачем або відсутньою (тоді маршрут завершується в останній відвіданій точці).

Алгоритм на кожному кроці:

– проводиться пошук найкоротшого шляху за допомогою алгоритму Дейкстри, від поточної вершини до всіх вершин множини *S*, що ще не були відвідані;.

– вибирається точка з множини *S*, до якої визначений найменший шлях;

– формується проміжний маршрут до цієї точки, після чого обрана точка стає поточною;

– прибирає її зі списку невідвіданих точок множини *S*.

Процес триває, поки всі вершини з множини *S* не будуть відвідані. Якщо вказано кінцеву вершину $v_{end}$, виконується останній пошук від поточної вершини до $v_{end}$. Таким чином, формується повний маршрут, що послідовно проходить усі визначені точки множини *S* і мінімізує загальну довжину маршруту.

Для роз'яснення алгоритму необхідно ввести наступні визначення:

– $v_{start}$ – початкова вершина;

– $v_i$ – довільна вершина графа;

– $A$ – множина вершин, для котрих вже знайдено найкоротший маршрут від початкової вершини $v_{start}$, тобто вже оброблені вершини;

– $B$ – множина ще не опрацьованих вершин;

– $dist(v_i)$ – поточне значення (оцінка) довжини мінімального шляху від $v_{start}$ до $v_i$, актуальне на момент поточної ітерації;

– $dist(v_{start}) = 0$ – ініціалізація, початкове значення $dist$ дорівнює 0.

– *parent($v_i$)* – попередня вершина на шляху від поточної вершини до $v_i$, що дозволяє відновити маршрут.

Перейдемо до початкової ініціалізації:

– $A := \emptyset$ – жодна вершина ще не була опрацьована;

– $B := \{ v_{tart} \}$ – спочатку в черзі на обробкулише стартова вершина s;

– для всіх $v_i$ встановити $dist(v_i) := +\infty$;

– $dist(v_{tart}) := 0$.

Обирається така вершина $u$ із $B$, що:

$$dist(u) = min_{u \epsilon B} dist(v_i). \quad (6.6)$$

Тобто визначається вершина, у якої значення *dist* найменше. Видаляється *u* з *B* та додається *u* до *A*. Далі розглядаються усі вершини w для яких існує ребро *(u,w)* у графі:

– якщо $w \in A$ не виконуються жодні дії;

– якщо $w \in B$ перевіряється, чи можна покращити (зменшити) значення $dist(w)$ через вершину *u*

$$dist(u) + w(u, w) < dist(w); \quad (6.7)$$

$$dist(w) := dist(u) + w(u, w); \quad (6.8)$$

$$parent(w) := u; \quad (6.9)$$

– якщо w не міститься ні в $A$, ні в $B$ треба додати її до $B$ та встановити

$$dist(w) := dist(u) + w(u, w), parent(w) := u. \quad (6.10)$$

Коли $B = \emptyset$ – алгоритм завершується. На цьому моменті множина A містить всі вершини графа.

Значення $dist(v_i)$ містить довжину шляху від $v_{start}$ до $v_{end}$– кінцевої точки маршруту.

Розглядається наступний граф, який представлено на рисунку 6.3.

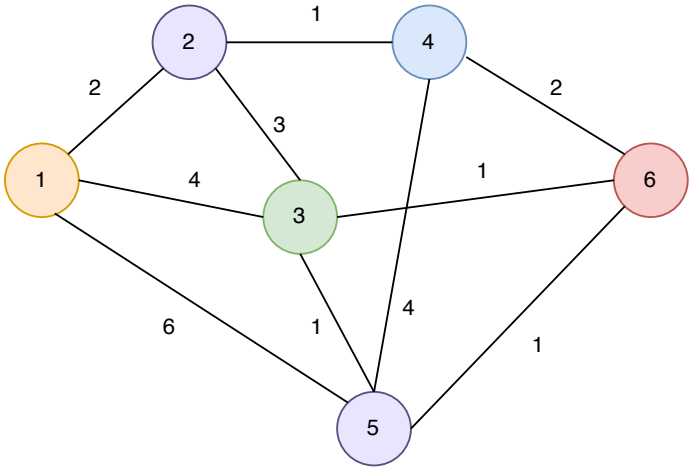


Рисунок 6.3 – Граф для побудова оптимального маршруту

Початкова вершина маршруту – $v_{start} = 1$. Задано кінцеву вершину – $v_{end} = 5$. Необхідно побудувати маршрут, який проходить через всі вершини графа і завершується у вершині 5. Тобто є наступна задача: розпочати шлях у вершині 1, обійти всі вершини $(2, 3, 4, 6)$ і завершити маршрут у вершині 5, при цьому загальна довжина шляху має бути мінімальною.

Тому маємо:

– $current = 1$ – початкова вершина;

– $targets = \{2, 3, 4, 6\}$ – усі вершини, які потрібно охопити;

– $route = 1$ – на початку маршрут складається тільки з однієї вершини.

Тепер необхідно почати пошук найкоротший маршрут від вершини 1:

а) $dist(2) = 2$ (шлях $1 \rightarrow 2$);

б) $dist(3) = 4$ (шлях $1 \rightarrow 3$);

в) $dist(4) = 3$ ($1 \rightarrow 2 \rightarrow 4$, бо $2 + 1 = 3$);

г) $dist(6) = 5$ – найменша з можливих відстаней до вершини 6:

1) $1 \rightarrow 2 \rightarrow 4 \rightarrow 6$: $2 + 1 + 2 = 5$;

2) $1 \rightarrow 3 \rightarrow 6$: $4 + 1 = 5$;

3) $1 \to 3 \to 2 \to 4 \to 6$: $4 + 3 + 1 + 2 = 10$;

4) $1 \to 5 \to 6$: $6 + 1 = 7$.

Отже, серед $\{2, 3, 4, 6\}$ найменше значення *dist*: $dist(2) = 2$, $dist(3) = 4$, $dist(4) = 3$, $dist(6) = 5$. Відповідно, найближчою вважається вершина дорівнює 2 (з відстанню 2).

Наступною розглядається вершина 2. Відновлюється локальний шлях (1→2) і додається до *Route*: тепер $Route = [1, 2]$. $Current = 2$, $Targets = \{3, 4, 6\}$, оскільки 2 вже відвідана, вона має бути видалена зі списку вершин, які потрібно охопити.

На другому кроці здійснюється пошук мінімального (найкоротшого) шляху від вершини 2:

– $dist(3) = 3$ (шлях $2 \to 3$);

– $dist(4) = 1$ ($2 \to 4$);

– $dist(6) = 3$ ($2 \to 4 \to 6$: $1 + 2 = 3$ – найкоротший шлях, але також існує шлях 2→4→6: $3 + 1 = 4 > 3$, тому оптимальний варіант саме 3);

Найменше – $dist(4) = 1$. Далі відбувається перехід до вершини 4, локальний шлях стає ($1 \to 2 \to 4$), $Route = [1, 2, 4]$. $Current = 4, Targets = \{3, 6\}$.

На третьому кроці відбувається пошук найкоротшого шляху від вершини 4. Існують наступні варіанти:

– $dist(3) = 4$ (шлях $2 \to 3$: $1 + 3 = 4$);

– $dist(6) = 2$ (прямий шлях вагою 2).

Найменше значення – $dist(6) = 2$, тому відбувається перехід до вершини 6, шлях стає ($1 \to 2 \to 4 \to 6$), $Route = [1, 2, 4, 6]$. $Current = 6, Targets = \{3\}$.

На даному етапі здійснюється пошук шляху у вершину 3, який буде дорівнювати $dist(3) = 1$, тому новий шлях ($1 \to 2 \to 4 \to 6 \to 3$), $Route = [1, 2, 4, 6, 3]$. $Current = 3$. Так як всі проміжні вершини вже відвідано залишається лише остання вершина – 5 до якої визначається найкоротший шлях. В даному графі існує ребро з вершини 3 до вершини 5, яке має вагу 1 і є оптимальним варіантом. Також існує можливість пройти через вершину 6, але вага буде $2 > 1$, тому обирається кращий для розв'язку варіант – 1.

Таким чином усі проміжні вершини відвідано, враховано початкову і кінцеву точку і побудовано маршрут ($1 \rightarrow 2 \rightarrow 4 \rightarrow 6 \rightarrow 3 \rightarrow 5$), вага якого $2 + 1 + 2 + 1 + 1 = 7$ – що і є розв'язком задачі. Розв'язок представлений на рисунку 6.4

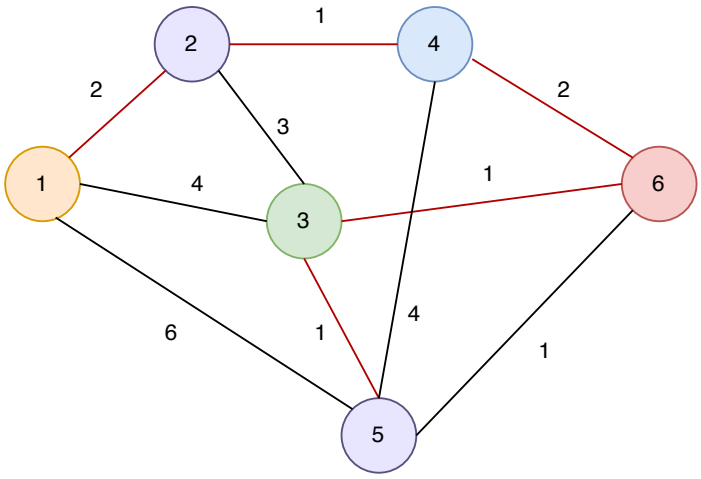


Рисунок 6.4 – Побудований оптимальний маршрут згідно з результатами алгоритму

Висновки до розділу 6

У ході дослідження було детально проаналізовано сучасні алгоритми для знаходження найкоротших шляхів: алгоритми Дейкстри, Беллмана-Форда, A*, Флойда–Воршелла та Джонсона. Визначено ключові характеристики цих алгоритмів, їх переваги, обмеження та особливості використання.

За результатами аналізу було зроблено обґрунтований вибір на користь алгоритму Дейкстри із застосуванням ітеративного підходу для вирішення конкретної задачі. Цей алгоритм демонструє високу швидкодію і точність при визначенні найкоротших маршрутів у графах із невід'ємними вагами, що цілком відповідає специфіці задачі побудови багатоточкових маршрутів дорожньої мережі.

# ВИСНОВКИ

У дипломному проєкті проведено повний цикл досліджень і створено систему, яка автоматично формує багатоточкові маршрути. Її актуальність зумовлена зростанням потреби у швидкому й зручному плануванні щоденних поїздок, а також обмеженнями популярних навігаційних сервісів, які не пропонують дієвого механізму керування кількома зупинками.

Порівняльний аналіз Google Maps, Apple Maps, Waze й інших рішень показав: користувачів стримують ліміти на кількість точок, відсутність автоматичної оптимізації їх послідовності та брак інструментів для зберігання й повторного використання маршрутів. Запропонована система усуває ці недоліки завдяки власному алгоритму оптимізації, модулю «точок інтересу» та підтримці локального каталогу улюблених поїздок.

Функціонально, цей застосунок відповідає всім вимогам технічного завдання: він має зрозумілий інтерфейс, локальне зберігання даних, інтеграцію з системним календарем і швидкий пошук найближчих POI. Навіть при великій кількості точок, час побудови шляху не перевищує прийнятних значень.

Ядром математичного забезпечення став алгоритм Дейкстри, який забезпечує оптимальний результат при невід’ємних вагових коефіцієнтах ребер і демонструє високу продуктивність на мобільних пристроях. Застосовані технології – Swift, UIKit, Google Maps API, Core Data – забезпечують стабільність, масштабованість і простоту обслуговування. Розробка може бути успішно впроваджена в повсякденні сценарії використання, дозволяючи користувачам економити час та ресурси під час планування та проходження маршрутів із кількома зупинками.

Подальші перспективи проєкту включають інтеграцію додаткових сервісів, таких як аналіз трафіку та прогнозування погодних умов. Також планується розширення підтримки різних платформ і впровадження нових функцій, які зроблять застосунок ще більш зручним та універсальним для широкого кола користувачів.

## ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Google Maps. Wikipedia. URL: https://en.wikipedia.org/wiki/Google_Maps (дата звернення: 03.04.2025).

2. Додавання, редагування або видалення місць. Google Support. Google Maps Help. URL: https://support.google.com/maps/answer/3093484 (дата звернення: 04.04.2025).

3. Learn advanced gestures to interact with iPad. Apple Support. iPad User Guide. URL: https://support.apple.com/guide/ipad/ipad590dd01f/ipados (дата звернення: 04.04.2025).

4. Look around places in Maps on iPhone. Apple Support. iPhone User Guide. URL: https://support.apple.com/guide/iphone/look-around-places-iph65703a702/ios (дата звернення: 07.04.2025).

5. Sync your calendar with Help. URL: https://support.google.com/waze/answer/6341187 (дата звернення: 07.04.2025).

6. Miller L. The 8 Best Free Route Planners – That Aren't Google Maps. Routific. Routific Blog. 19.03.2024. URL: https://www.routific.com/blog/free-route-planning-software-with-unlimited-stops (дата звернення: 08.04.2025).

7. Routing & Navigation. HERE WeGo Support. FAQ. URL: https://help.here.com/faq/routing_and_navigation (дата звернення: 09.04.2025).

8. Swift. Apple Developer. Swift. URL: https://developer.apple.com/swift/ (дата звернення: 21.04.2025).

9. The Swift Package Manager. Swift.org. Documentation. URL: https://www.swift.org/documentation/package-manager/ (дата звернення: 22.04.2025).

10. UIKit. Apple Developer. UIKit Documentation. URL: https://developer.apple.com/documentation/uikit (дата звернення: 23.04.2025).

11. Core Data Programming Guide. Apple Developer. Core Data Documentation. URL: https://developer.apple.com/library/archive/documentation/Cocoa/Conceptual/CoreData/index.html (дата звернення: 25.04.2025).

12. Sqlite3 – DB-API 2.0 interface for SQLite databases. Python Documentation. Standard Library. URL: https://docs.python.org/uk/3/library/sqlite3.html (дата звернення: 27.06.2025).

13. Dijkstra E. A note on two problems in connexion with graphs. Numerische Mathematik. 1959. T. 1. C. 269–271.

14. Bellman R. On a routing problem. Quarterly of Applied Mathematics. 1958. T. 16. C. 87–90.

15. Geeksforgeeks. A* is Admissible. GeeksforGeeks. Algorithms. URL: https://www.geeksforgeeks.org/a-is-admissible/ (дата звернення: 20.05.2025).

16. Geeksforgeeks. Floyd Warshall Algorithm | DP-16. GeeksforGeeks. Dynamic Programming. URL: https://www.geeksforgeeks.org/floyd-warshall-algorithm-dp-16/ (дата звернення: 22.05.2025).

17. Baeldung. Johnson’s Algorithm for All-Pairs Shortest Paths. Baeldung. Computer Science. URL: https://www.baeldung.com/cs/all-pairs-shortest-paths-johnsons-algorithm (дата звернення: 27.05.2025).