

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

**«Розробка мобільного додатку для допомоги
переміщення військовим»**

Виконав студент групи ПЗ-20-3
РАДЧЕНКО Данило Вадимович

Керівник роботи:
САВІЦЬКИЙ Роман Святославович

Рецензент:
ТОЛСТОЙ Ігор Анатолійович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«26» лютого 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **РАДЧЕНКО Данило Вадимович**

Керівник роботи: **САВІЦЬКИЙ Роман Святославович**

Тема роботи: **«Розробка мобільного додатку для допомоги переміщення військовим»**,

затверджена Наказом закладу вищої освіти від **«23» лютого 2024 р., №74с**

Вихідні дані для роботи: Об'єктом дослідження є додаток з використанням технології геолокації на iOS, який буде спрямований на полегшення та пришвидшення процесів побудови маршрутів та не тільки. Предметом дослідження є архітектура та компонентна структура iOS додатку з використанням технології GPS. Дослідження спрямоване на вивчення технологій та методів, що дозволяють забезпечити швидке та ефективне використання функцій геолокації та на дослідження роботи технології в рамках iOS додатка.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Савіцький Р.С.	02.03.2024р.	02.03.2024р.
2	Савіцький Р.С.	04.04.2024р.	04.04.2024р.
3	Савіцький Р.С.	05.05.2024р.	05.05.2024р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з iOS додатку допомоги військовим та пояснювальної записки. Пояснювальна записка до випускної роботи містить 87 сторінок, 35 ілюстрацій та 7 таблиць.

Метою роботи є створення додатку допомоги військовим з використанням сучасних технологій в розробці GPS-додатків та інтеграцією нових функцій для полегшення подорожей.

Методи, використані під час створення додатку, включаючи в себе програмування на iOS-орієнтованих мовах програмування, роботу з графічними програмами для дизайну та аналіз потреб майбутніх споживачів.

Результати розробки полягають у створенні інтерактивного та привабливого додатку для військових, який надає зручний доступ до мапи та плавання маршруту.

КЛЮЧОВІ СЛОВА: IOS, ДОДАТОК, SWIFT, GPS, GEOLOCATION, ПОДОРОЖІ, ВІЙСЬКОВІ, SWIFTUI, CORELOCATION, MAPKIT, COMBINE.

					ІПЗ.КР.Б-121-24-ІЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Розробка мобільного додатку для допомоги переміщення військовим	Літ.	Арк.	Аркуші
Розроб.		Радченко Д.В.						
Керівник		Савіцький Р.С.					3	87
Рецензен.		Толстой І.А.				Житомирська політехніка, група ІПЗ-20-3		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The final bachelor's thesis consists of an iOS application to help the military and an explanatory note. The explanatory note to the final thesis contains 87 pages, 35 illustrations and 7 tables.

The purpose of the work is to create an application to help the military using modern technologies in the development of GPS applications and integrating new features to facilitate travel.

The methods used to create the application include programming in iOS-oriented programming languages, working with graphic design software, and analysing the needs of future consumers.

The results of the development are the creation of an interactive and attractive application for the military, which provides convenient access to the map and route planning.

KEYWORDS: IOS, APPLICATION, SWIFT, GPS, GEOLOCATION, TRAVEL, MILITARY, SWIFTUI, CORELOCATION, MAPKIT, COMBINE.

					ИП3.КР.Б-121-24-ИЗ	
						4

ЗМІСТ

РЕФЕРАТ.....	3
ABSTRACT	4
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ДОДАТКУ ДЛЯ ДОПОМОГИ ПЕРЕМІЩЕННЯ ВІЙСЬКОВИМ	10
1.1 ПОСТАНОВКА ЗАДАЧІ	10
1.2 АНАЛІЗ АНАЛОГІВ ПРОГРАМНОГО ПРОДУКТУ	11
1.3 ВИБІР АРХІТЕКТУРИ ДОДАТКУ ДЛЯ ДОПОМОГИ ПЕРЕМІЩЕННЯ ВІЙСЬКОВИХ	20
1.4 ОБҐРУНТУВАННЯ ВИБОРУ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ ТА ВИМОГИ ДО АПАРАТНОГО ЗАБЕЗПЕЧЕННЯ	22
Висновки до першого розділу	25
РОЗДІЛ 2. ПРОЕКТУВАННЯ ДОДАТКУ ДЛЯ ДОПОМОГИ ПЕРЕМІЩЕННЯ ВІЙСЬКОВИМ.....	26
2.1 ВИЗНАЧЕННЯ ВАРІАНТІВ ВИКОРИСТАННЯ ТА ОБ’ЄКТНО- ОРІЄНТОВАНОЇ СТРУКТУРИ СИСТЕМИ	26
2.2 РОЗРОБКА БАЗИ ДАНИХ СИСТЕМИ	33
2.3 РЕАЛІЗАЦІЯ ФУНКЦІОНАЛУ ДОДАТКУ ДОПОМОГИ ПЕРЕМІЩЕННЯ ВІЙСЬКОВИМ	34
Висновки до другого розділу	43
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ	45

3.1 ПОРЯДОК ВСТАНОВЛЕННЯ ТА НАЛАШТУВАННЯ ПАРАМЕТРІВ ДОДАТКУ	45
3.2 СТРУКТУРА ІНТЕРФЕЙСУ ДОДАТКУ ДЛЯ ПЕРЕМІЩЕННЯ ВІЙСЬКОВИХ	47
3.3 ТЕСТУВАННЯ	58
Висновки до третього розділу	63
Висновки	64
Список використаних джерел	66
Додатки.....	69

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс

GPS – Global Positioning System – це сукупність радіоелектронних засобів, що дозволяє визначати положення та швидкість руху об'єкта на поверхні Землі або в атмосфері

IDE – Integrated Development Enviroment

JSON – JavaScript Object Notation

MVVM – Model View ViewModel

SDK – Software Development Kit – набір для розробки програмного забезпечення

SSD – Solid State Drive

БД – база даних

ПЗ – програмне забезпечення

					ІПЗ.КР.Б-121-24-ПЗ	
						7

ВСТУП

Актуальність теми полягає у тому, що додатки з тісним використанням геолокації залишаються актуальними по цей день, завдяки своєму внеску в комфортне планування подорожей/туризму, планування маршрутів не тільки для особистих мінімальних цілей, а й в масштабах економіки країни.

Говорячи про додатки, які тісно використовують технологію GPS, вони на сьогоднішній день стали невід’ємною частиною життя суспільства. Вони дозволяють планувати подорожі, налаштовувати логістику в економічних гігантах для пришвидшення доставки товарів та не тільки. Вони дозволяють цілодобово моніторити місцеперебування важливих для вас людей, навіть дозволяють робити спеціальних запрограмованих роботів, які працюють на фермах та виконують певні функції за заданим маршрутом (комбайни, дрони).

Технологія розвинулася до такого рівня, що існують окремі програми завдяки яким можна замовляти таксі на певне місце та планувати його теж як звичайний маршрут в Google Maps. Або ж можемо відхилитися від звичайної буденності та поринути глибше та дізнатися про те, що навіть існують додатки з GPS спеціалізовані під археологічні роботи, з великим вибором старовинних карт, та різним функціоналом характерним для такого виду діяльності.

Додатки з використанням GPS нині слугують як зручний та корисний інструмент в житті кожної людини, надаючи їм змогу швидко отримати бажаний результат, себто отримати свої координати, чи викликати таксі, або ж дізнатися де зараз знаходиться важлива для вас людина.

Загалом, додатки з використанням технології GPS виконують надзвичайно важливу функцію в сучасному цифровому світі та в реальному житті, що робить їх невід’ємною частиною буденності.

Метою даної роботи є розробка алгоритмів роботи та реалізація додатку для допомоги військовим.

Сформульована мета визначає **наступні завдання:**

					ІПЗ.КР.Б-121-24-ПЗ	
						8

- аналіз теоретичних та практичних вимог щодо розробки додатку на iOS;
- визначення структури та загальної будови системи;
- оцінка існуючих додатків з обраної теми та визначення їх особливостей;
- обґрунтування та вибір інструментів реалізації;
- розробляння компонентів структури та алгоритмів функціонування;
- реалізація додатку з геолокації.

Об’єктом дослідження є додаток з використанням технології геолокації на iOS, який буде спрямований на полегшення та пришвидшення процесів побудови маршрутів та не тільки.

Предметом дослідження є архітектура та компонентна структура iOS додатку з використанням технології GPS. Дослідження спрямоване на вивчення технологій та методів, що дозволяють забезпечити швидке та ефективне використання функцій геолокації та на дослідження роботи технології в рамках iOS додатка.

За темою випускної кваліфікаційної роботи бакалавра було опубліковано статтю: Савіцький Р.С., Радченко Д.В. Java та процес роботи функціоналу додатків для роботи з GPS. Збірник тез XI Всеукраїнської науково–практичної конференції молодих учених, 16 травня 2024. Київ: “Київський столичний університет імені Бориса Грінченка”, 2024. С. 86-88 [1].

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ДОДАТКУ ДЛЯ ДОПОМОГИ ПЕРЕМІЩЕННЯ ВІЙСЬКОВИМ

1.1 Постановка задачі

Додаток з геолокації для військових – це певний додаток, створений з метою полегшити роботу фахівців з питань логістики, розвідки чи будь-якого підрозділу що стосується переміщення. Суть проекту полягає в створенні простого та надійного продукту який буде працювати чітко та швидко для виконання поставлених задач.

Говорячи про такі додатки, там зазвичай є наступні елементи:

- визначення місця знаходження, використання GPS для визначення поточного місця знаходження користувача, автоматичне або мануальне оновлення місця розташування;
- інтерактивні карти з можливістю переміщення та масштабування, різні шари карт (супутниковий, дорожній, гібридний, особистий накладний), маркери і позначки на карті для вказівки певних точок;
- обмін даних, збір, відстеження та збереження історії місць, де побував користувач; обмін місце знаходження з друзями або сім'єю; налаштування приватності для контролю доступу до геоданих.
- оповіщення та повідомлення про події поблизу (заходи, небезпеки, пробки тощо). Повідомлення про прибуття на місце або залишення певної зони.
- безпека та конфіденційність, шифрування даних для захисту приватності користувачів, налаштування доступу до геоданих та інших функцій.

Наведемо технічні завдання реалізації додатку.

Вибір технологій:

- визначення стеку технологій для розробки iOS додатку;
- вибір фреймворку для швидкої розробки та підтримки.

Розробка бази даних:

- проектування та розробка структури бази даних для зберігання координат, маршрутів або просто поміток;
- вибір та налаштування системи управління базами даних.

Користувацький інтерфейс:

- Розробка користувацького інтерфейсу для додатку з використанням Swift, SwiftUI;
- реалізація адаптивного дизайну для оптимального відображення на різних пристроях та екранах.

Функціонал додатку:

- створення та впровадження функцій та особливостей додатку з метою забезпечення його функціональності та ефективності.

Тестування та відладка:

- проведення тестування всіх функцій та компонентів додатку для виявлення та виправлення помилок.

Задачі:

- пошук та аналіз аналогів;
- пошук та збір необхідних даних для створення та оформлення додатку;
- створення БД та таблиць БД для зберігання даних;
- обрання програми та мови програмування для створення;
- визначення структури додатку та кількості модулів;
- тестування додатку.

1.2 Аналіз аналогів програмного продукту

Аналіз аналогів є важливим кроком в розробці будь-якого продукту або ж сервісу, також він може бути корисним при управлінні та вдосконаленні вже існуючих проектів. Дискутуючи про аналіз в цілому, він може бути корисним для розуміння того, що є в цій галузі досліджуваного продукту та які можливості є для покращення програмного продукту. Важливим є те, що це допомагає

виявити нові ідеї, функціональні можливості або підходи, які можна впровадити в продукт для збільшення його конкурентноспроможності та відповідності потребам користувачів.

Google Maps [2] – є безкоштовним картографічним веб-сервісом від компанії Google, що надає можливість переглядати карти, супутникові знімки, панорами вулиць багатьох країн світу.

Серед основних функцій:

- Картографія, як відображення детальних карт міст, доріг, місць та ландшафтів.
- Пошук напрямків. Маршрути для подорожі автомобілем, громадським транспортом, велосипедом або пішки.
- Street View – панорамний огляд вулиць з можливістю віртуальної прогулянки.
- Інформація про місця. Відомості про компанії, ресторани, готелі, пам'ятки та інші важливі об'єкти, включаючи відгуки користувачів.
- Розширені додаткові функції, такі як: реальний час руху транспорту, оцінка часу прибуття, інформація про затори або аварії на дорогах.
- функція подорожі по вулицях, розглядаючи все детально завдяки технології фото-360 градусів.

Переваги:

- Google Maps покриває більшість країн світу.
- Постійно оновлюється, що забезпечує актуальність інформації.
- Підтримка навігації для автомобілів громадського транспорту і не тільки та надає їм же точні напрямки та альтернативні маршрути з урахуванням заторів, аварій, дорожніх робіт у режимі реального часу.
- Легко інтегрується з іншими сервісами, такими як Google Search, Google Assistant та Google Calendar, що максимально полегшує планування подорожей та синхронізацію даних.

- Можливість завантажувати карти для використання без доступу до інтернету, що дуже зручно при подорожах у місцях зі слабким або повністю відсутнім покриттям.
- API для інтеграції Google Maps у сторонні додатки та веб-сайти, що робить сервіс гнучким для розробників.

Недоліки:

- Більшість функцій, таких як оновлення даних у реальному часі та пошук місць, потребують активного підключення до інтернету.
- Використання сервісу вимагає надання доступу до місця розташування, що може викликати занепокоєння щодо конфіденційності та збору даних.
- У віддалених або малонаселених регіонах карти неточні, а інформація про місця та дороги обмеженою.
- Рекламні оголошення або просування бізнесів, що відволікає
- Неточність або застарілість даних, стосується особливо годин роботи бізнесів, адрес або телефонних номерів.

Загальний вид сервісу можна побачити на фото нижче (рис. 1.1) та (рис.

1.2).

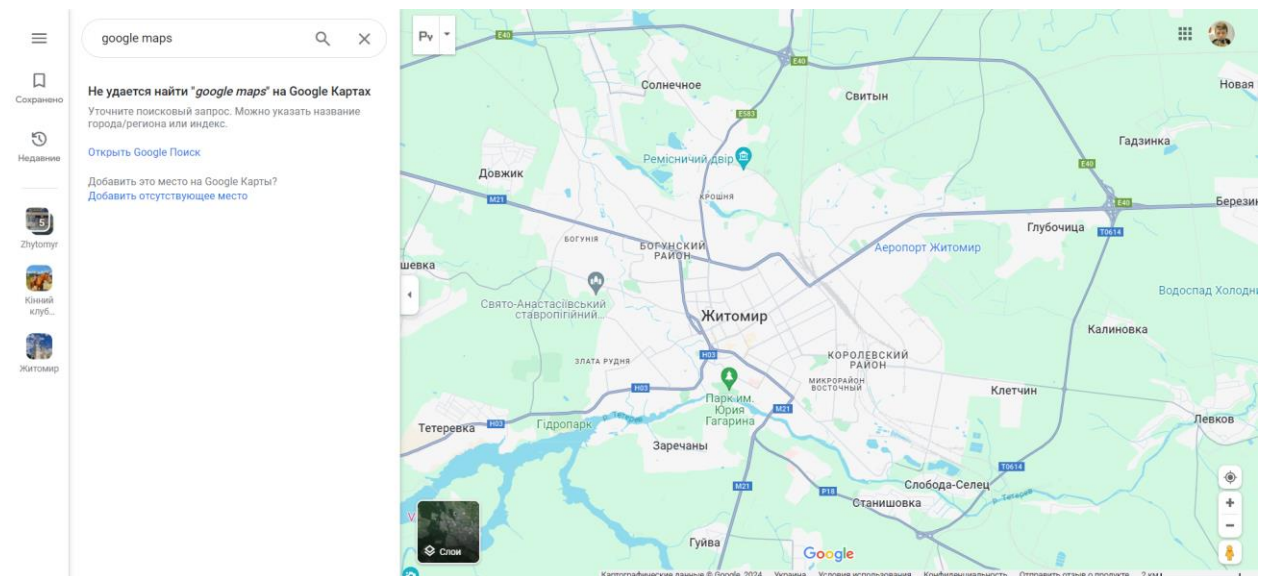


Рис. 1.1 Інтерфейс Google Maps



Рис. 1.2 Перегляд від першої особи

Waze [3] – є безкоштовним мобільним додатком для навігації, який розроблений ізраїльською компанією Waze Mobile і був придбаний Google у 2013 році.

Серед основних функцій:

- Картографія, як відображення детальних карт міст, доріг, місць та ландшафтів.
- Пропонує оптимальні маршрути до місця призначення, та вони можуть змінюватися в реальному часі залежно від даних про затори чи інші події.
- Інформація про місця. Відомості про компанії, ресторани, готелі, пам'ятки та інші важливі об'єкти, включаючи відгуки користувачів.

Переваги:

- Користувачі можуть у реальному часі ділитися інформацією про дорожні умови, затори, трупи тварин, ремонтні роботи та інші події, що впливають на дорожній рух. Що дозволяє компанії забезпечувати актуальні точні дані для навігації.
- Додаток попереджає про потенційні небезпеки на дорозі, такі як нерівності, автомобілі, що стоять на узбіччі, та інші перешкоди.

- Має можливість інтеграції з музичними додатками та іншими сервісами, що забезпечує зручність використання під час водіння.
- Користувачі можуть підключатися до своїх друзів та синхронізувати свої поїздки.

Недоліки:

- Якість інформації залежить від активності користувачів. У регіонах з меншою кількістю користувачів дані можуть бути менш точними або відсутніми.
- Додаток використовує GPS та інтернет з'єднання, що може швидко розряджати батарею смартфона, особливо під час тривалих поїздок.
- У додатку з'явилися рекламні оголошення, які відволікають під час водіння.
- Частина користувачів вважають інтерфейс складним та перевантаженим великою кількістю іконок та повідомлень на екрані.
- Іноді додаток працює нестабільно в умовах слабого покриття або використання на старіших моделях телефонів.
- Може пропонувати маршрути через житлові зони чи інші місця, де не має бути автомобільний рух.

Загальний вид додатку можна побачити на фото нижче (рис. 1.3), (рис. 1.4) та (рис. 1.5).

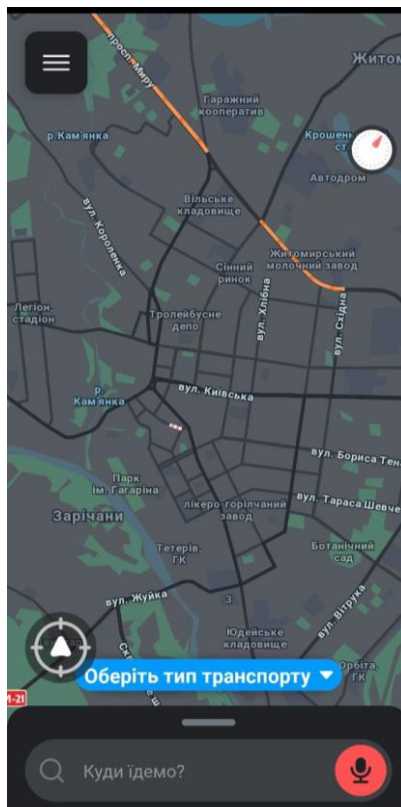


Рис. 1.3 Перегляд головного меню Waze

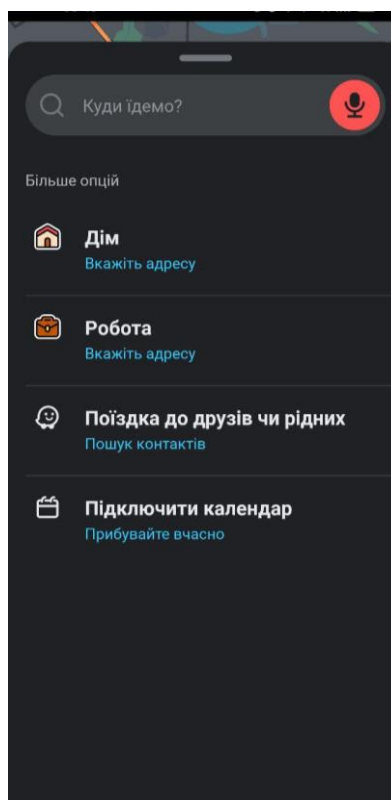


Рис. 1.4 Перегляд головного меню Waze

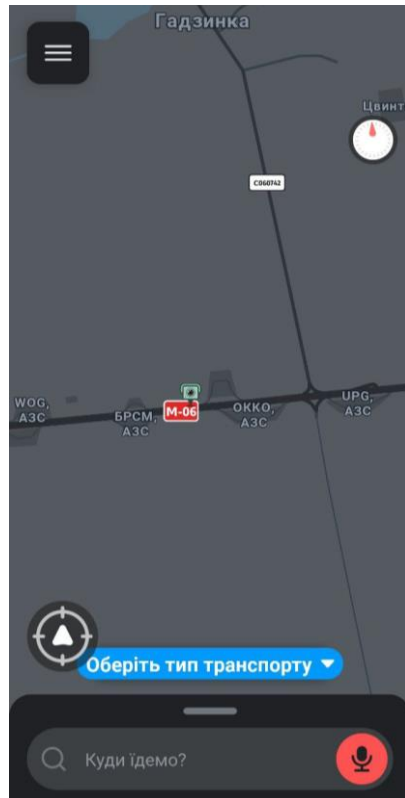


Рис. 1.5 Перегляд головного меню Waze

Vetus Maps [4] – створений спеціально для полегшення роботи краєзнавцям та любителям старовини.

Серед основних функцій:

- Картографія, як відображення детальних карт міст, доріг, місць та ландшафтів.
- Інформація про місця, місцевості, регіони.
- Інструменти для вимірювання відстані.
- Інструменти для графічної роботи з картами.
- Інструменти для завантаження особистих карт.
- Трекінг система та збереження трекерів.

Переваги:

- Програма надає вченим і дослідникам інструменти для аналізу старовинних карт.
- Оцифровані копії старовинних карт які можуть бути доступні через Інтернет.

- Розвинута трекінг система з можливостями зберігання та поширення.

Недоліки:

- Якість інформації залежить від активності користувачів. У регіонах з меншою кількістю користувачів дані можуть бути менш точними або відсутніми.
- Якщо не надати всі дозволи повністю, трекінг система працює некоректно.
- Не знаючим людям важко розібратися з графічними функціями для обраховування.
- Додаток використовує GPS та інтернет з'єднання, що може швидко розряджати батарею смартфона, особливо під час тривалих поїздок.

Загальний вид додатку можна побачити на фото нижче (рис. 1.6), (рис. 1.7) та (рис. 1.8).



Рис. 1.6 Загальний вид додатку



Рис. 1.7 Загальний вид додатку



Рис. 1.8 Загальний вид додатку

Виходячи з наявних переваг та недоліків, складемо порівняльну таблицю вищезгаданих програмних аналогів (табл. 1.1).

Таблиця порівняння додатків з технологією геолокації

Характеристика	Google Maps	Waze Maps	Vetus Maps	SaveMaps
Поточне місцезнаходження	+	+	+	+
Кросплатформенність	+	-	-	-
Додавання власних карт	-	-	+	-
Інструменти для вимірювання	-	-	+	+
Офлайн карти	+	-	+	-
Розвинуте ком'юніті	+	+	+	-
Доступ для незареєстрованих	+	+	+	+

Всі представлені додатки мають свої особливості та переваги. Google Maps та Waze Maps підходять більше для людей, які подорожують або хочуть побудувати маршрут з кимось, незалежно туристичний чи ні. Vetus Maps, підкований більше для дослідницької роботи, розкопок та історії. Додатки основа яких є технологія GPS – це потужний інструмент, який може використовуватися для різних цілей. Для дозвілля, або для якихось професійних цілей.

1.3 Вибір архітектури додатку для допомоги переміщення військових

Архітектура MVVM є сучасним підходом до розробки програмного забезпечення, який широко використовується в розробці додатків для iOS, зокрема за допомогою SwiftUI [5]. В цій архітектурі кожен компонент має чітко визначені ролі, що сприяє кращому розподілу відповідальностей і спрощує тестування та підтримку коду.

Схему архітектури наведено на рисунку 2.2

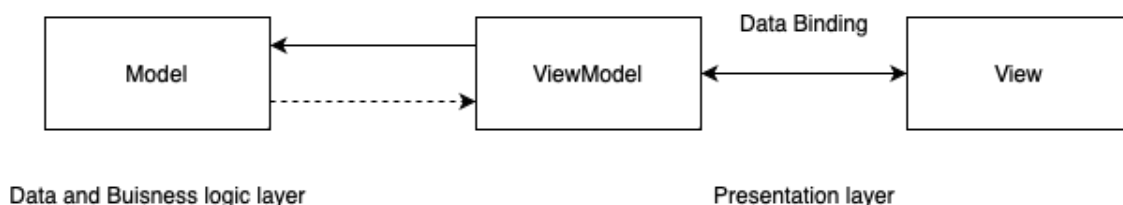


Рис. 2.2 Візуальна репрезентація архітектури MVVM

Компонент Model відповідає за управління даними та бізнес-логікою додатку. Він містить структури даних і методи для доступу до цих даних, такі як запити до бази даних або виклики API [6]. Model є незалежним від інших компонентів і не взаємодіє безпосередньо з користувацьким інтерфейсом, що забезпечує чітке розмежування даних та їхнього представлення.

Компонент View представляє користувацький інтерфейс і відповідає за відображення даних, які надаються ViewModel [7]. Він використовує декларативний підхід, особливо в SwiftUI, де інтерфейс описується у вигляді коду. View не містить бізнес-логіки, що дозволяє легко змінювати зовнішній вигляд додатку без впливу на його функціональність [8].

Компонент ViewModel слугить посередником між Model і View. Він отримує дані з Model, обробляє їх і надає у вигляді, зручному для відображення у View. ViewModel також містить логіку для обробки користувацьких дій, передаючи відповідні команди Model. Це дозволяє View залишатися простим і зосередженим виключно на рендерингу даних.

Архітектура MVVM (Model-View-ViewModel) забезпечує високий рівень модульності та підтримуваності коду. Завдяки чіткому розділенню обов'язків між компонентами, кожна частина додатку може бути розроблена, протестована та підтримана окремо, що значно спрощує процес розробки. Це особливо важливо для великих проектів, де багато розробників працюють над різними аспектами додатку.

Ще однією перевагою MVVM є можливість легко впроваджувати зміни в користувацький інтерфейс [9]. Оскільки View не містить бізнес-логіки, дизайнери та розробники інтерфейсу можуть працювати незалежно від

розробників бізнес-логіки. Це сприяє швидшому впровадженню нових функцій та поліпшень без ризику порушити основний функціонал додатку.

Крім того, використання ViewModel дозволяє реалізувати двостороннє зв'язування даних (data binding), що спрощує синхронізацію даних між Model і View. Це означає, що будь-які зміни в даних автоматично відображаються в інтерфейсі користувача, а зміни в інтерфейсі користувача передаються назад до моделі даних.

У результаті, архітектура MVVM сприяє підвищенню продуктивності розробників, знижує кількість помилок та спрощує процес тестування і підтримки додатку. Це робить її ідеальним вибором для створення складних та масштабованих мобільних та настільних додатків.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Swift — це потужна і сучасна мова програмування, розроблена Apple для створення додатків на платформах iOS, macOS, watchOS та tvOS [10]. Вона поєднує високу продуктивність з простотою та зручністю використання. Swift забезпечує безпечне управління пам'яттю та захист від помилок завдяки статичній типізації, що робить розробку надійнішою та зменшує кількість багів. Мова також підтримує всі сучасні парадигми програмування, такі як об'єктно-орієнтоване та функціональне програмування, що дозволяє розробникам вибирати найбільш підходящий підхід для вирішення конкретних задач. Завдяки чіткому та лаконічному синтаксису, Swift робить код легким для читання та підтримки [11].

SwiftUI — це інтерфейсний фреймворк від Apple, який дозволяє створювати користувацькі інтерфейси для всіх платформ Apple за допомогою декларативного синтаксису. Використовуючи SwiftUI, розробники можуть швидко і легко створювати складні інтерфейси, які автоматично адаптуються до різних розмірів екранів і орієнтацій пристроїв [12]. Завдяки інтеграції з Xcode,

розробники можуть бачити результати своїх змін у режимі реального часу, що значно скорочує час на розробку та тестування. SwiftUI також підтримує анімації та переходи, що дозволяє створювати привабливі та динамічні інтерфейси без великої кількості додаткового коду [13].

MapKit — це фреймворк, який дозволяє додавати карти та пов'язані з ними функціональності в додатки [14]. З його допомогою можна відображати карти, додавати анотації, малювати маршрути та багато іншого. MapKit забезпечує доступ до актуальних даних карт та дозволяє інтегрувати додаткові сервіси, такі як пошук місць та прокладання маршрутів. CoreLocation, в свою чергу, забезпечує доступ до сервісів визначення місцеположення, дозволяючи додаткам отримувати географічні координати, відстежувати переміщення користувача та орієнтуватися на місцевості [15]. Разом ці фреймворки дозволяють створювати потужні геолокаційні додатки, які можуть використовуватися в різних сферах, від навігації до моніторингу.

Combine — це фреймворк для обробки асинхронних подій, що дозволяє працювати з потоком даних у реальному часі [16]. Він підтримує реактивне програмування, що робить його ідеальним для створення сучасних додатків з високою продуктивністю та інтерактивністю. Combine дозволяє об'єднувати кілька джерел даних, трансформувати та фільтрувати дані, а також керувати помилками та завершенням потоків [17]. Це значно спрощує роботу з асинхронними завданнями, дозволяючи розробникам фокусуватися на логіці додатку замість боротьби з асинхронними викликами. З допомогою Combine можна легко інтегрувати мережеві запити, роботу з базами даних та інші асинхронні операції в одне цілісне рішення.

Xcode — це інтегроване середовище розробки (IDE) від Apple, яке використовується для створення додатків на платформах Apple. Xcode включає в себе всі необхідні інструменти для розробки, налагодження та тестування додатків, а також забезпечує інтеграцію з фреймворками Swift, SwiftUI, Combine та іншими [18]. Завдяки Xcode розробники можуть створювати високоякісні

додатки з мінімальними зусиллями. Середовище надає потужні інструменти для аналізу коду, виявлення та виправлення помилок, профілювання продуктивності та багато іншого. Крім того, Xcode підтримує інтеграцію з системами контролю версій, такими як Git, що робить процес командної розробки більш зручним та ефективним.

SwiftData — це фреймворк для управління даними, який використовується для зберігання та отримання даних локально на пристрої [19]. Він забезпечує простий і ефективний спосіб роботи з даними, використовуючи об'єктно-реляційну модель. SwiftData підтримує автоматичне керування версіями, що дозволяє легко вносити зміни до схеми бази даних без ризику втрати інформації. Це особливо важливо для додатків, які постійно еволюціонують та потребують внесення змін до структури даних. Фреймворк також підтримує синхронізацію даних між різними пристроями, що дозволяє створювати зручні та надійні рішення для користувачів.

Для розробки та тестування додатків з використанням вищезгаданих інструментальних засобів необхідно мати Mac-комп'ютер з операційною системою macOS. Рекомендована конфігурація включає процесор Intel Core i5 або новіший, 8 ГБ оперативної пам'яті, та SSD для швидкого доступу до даних [20]. Це забезпечує оптимальну продуктивність та швидкість роботи під час розробки.

Swift, SwiftUI, MapKit, CoreLocation, Combine та Xcode — це потужні інструменти від Apple, які забезпечують високу продуктивність, надійність та зручність розробки додатків на платформах Apple. Використання цих інструментів дозволяє створювати сучасні, інтерактивні та масштабовані додатки з мінімальними витратами часу та зусиль. SwiftData забезпечує ефективне управління даними, що є критично важливим для додатків, які працюють з конфіденційною інформацією. Завдяки цьому комплексу інструментів розробники можуть зосередитися на створенні інноваційних

функцій та покращенні користувацького досвіду, забезпечуючи високу якість та надійність своїх додатків.

Цей набір інструментів дозволяє розробникам легко адаптувати свої додатки до різних платформ та екранних розмірів, забезпечуючи відмінний користувацький досвід на всіх пристроях Apple [21]. Використання Swift та SwiftUI робить процес розробки швидким та ефективним, дозволяючи створювати сучасні та привабливі інтерфейси користувача. MapKit та CoreLocation забезпечують потужні можливості для розробки геолокаційних додатків, що можуть використовуватися в різних сферах, від навігації до соціальних мереж. Combine спрощує роботу з асинхронними завданнями, забезпечуючи високу продуктивність та реактивність додатків. Завдяки Xcode, розробники мають доступ до всіх необхідних інструментів для створення, тестування та оптимізації своїх додатків, що робить процес розробки більш зручним та продуктивним.

Висновки до першого розділу

Проведено аналіз предметної області за допомогою якого були визначені основні вимоги до додатку для допомоги переміщення військовим, основною задачею якого є створення додатку з технологією геолокації та відповідних інструментів що дозволять чітко переміщуватись по місцевості.

В процесі дослідження аналогічних і схожих за концепцією продуктів були проаналізовані та порівняні різні додатки та сервіси з різними принципами роботи, основним функціоналом яких є карти та сервіси для побудови маршрутів. Крім цього, були сформовані основні вимоги для власного додатку.

На основі відповідних вимог та критеріїв був сформований набір основних технологій, таких як, Swift, SwiftUI і засобів реалізації для розробки мобільного додатку.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ДОДАТКУ ДЛЯ ДОПОМОГИ ПЕРЕМІЩЕННЯ ВІЙСЬКОВИМ

2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи

Аналіз вимог включає визначення та деталізацію функціональних і нефункціональних вимог до програмного забезпечення, оцінку часу реакції системи на дії користувачів, специфікацію можливих сценаріїв використання, а також розробку загальної схеми елементів системи з описом їхньої структури, взаємодії та методів. Цей процес дозволяє чітко зрозуміти потреби користувачів та технічні характеристики, необхідні для створення ефективного та зручного програмного продукту [22].

Функціональні вимоги

Для мобільного додатку було визначено наступні функціональні вимоги:

- відсутність потреби в автентифікації;
- можливість заміру відстані між двома точками по прямій, та довжини маршруту в цілому;
- при приближенні чи віддаленні карті, повинен відображатись масштаб;
- при встановленні мітки на карті, мають відображатись координати мітки;
- можливість перегляду історії пройдених маршрутів;
- по закінченню навігації, користувач має отримати сповіщення що він знаходиться неподалік від кінцевого пункту призначення;
- навігація має відбуватись як за допомогою GPS так і без неї, за допомогою введення координат вручну.

Також для розробленого мобільного додатку було визначено наступні нефункціональні вимоги:

- операційна система: підтримка iOS версії 16.0 і вище;

- мова інтерфейсу користувача: інтерфейс користувача має бути англійською мовою;
- авторизація та автентифікація: авторизація та автентифікація повинні виконуватися за допомогою системи, яка забезпечує надійне зберігання клієнтських даних і паролів, із використанням шифрування та захисту від несанкціонованого доступу;
- продуктивність: система повинна забезпечувати час відгуку на дії користувача не більше 2 секунд у 95% випадків;
- надійність: система повинна мати час безвідмовної роботи не менше 99.9% протягом місяця;
- безпека: повинні бути реалізовані засоби захисту від атак типу SQL-ін'єкцій, XSS (міжсайтових скриптів) та CSRF (підробки міжсайтових запитів);
- сумісність: додаток повинен бути сумісним з усіма сучасними версіями веб-браузерів на iOS;
- масштабованість: система повинна підтримувати масштабування для обслуговування збільшеної кількості користувачів без зниження продуктивності;
- узгодженість: всі користувацькі інтерфейси та взаємодії повинні відповідати рекомендаціям Apple Human Interface Guidelines;
- доступність: система повинна бути доступною для користувачів з обмеженими можливостями;
- документованість: всі функції системи повинні бути детально документовані для забезпечення легкого обслуговування та оновлення;
- моніторинг і логування: повинні бути реалізовані засоби моніторингу і логування для відстеження роботи системи та швидкого виявлення і вирішення проблем.

Схема варіантів використання системи

Діаграма варіантів використання (Use Case Diagram) — це тип діаграми в UML (Unified Modeling Language), яка показує взаємодію між користувачами (акторами) та системою для виконання конкретних функцій (варіантів використання). Вона допомагає описати функціональні вимоги до системи і зобразити, як різні користувачі взаємодіють з нею [23]. На рисунку 2.1 наведено діаграму варіантів використання для додатку.

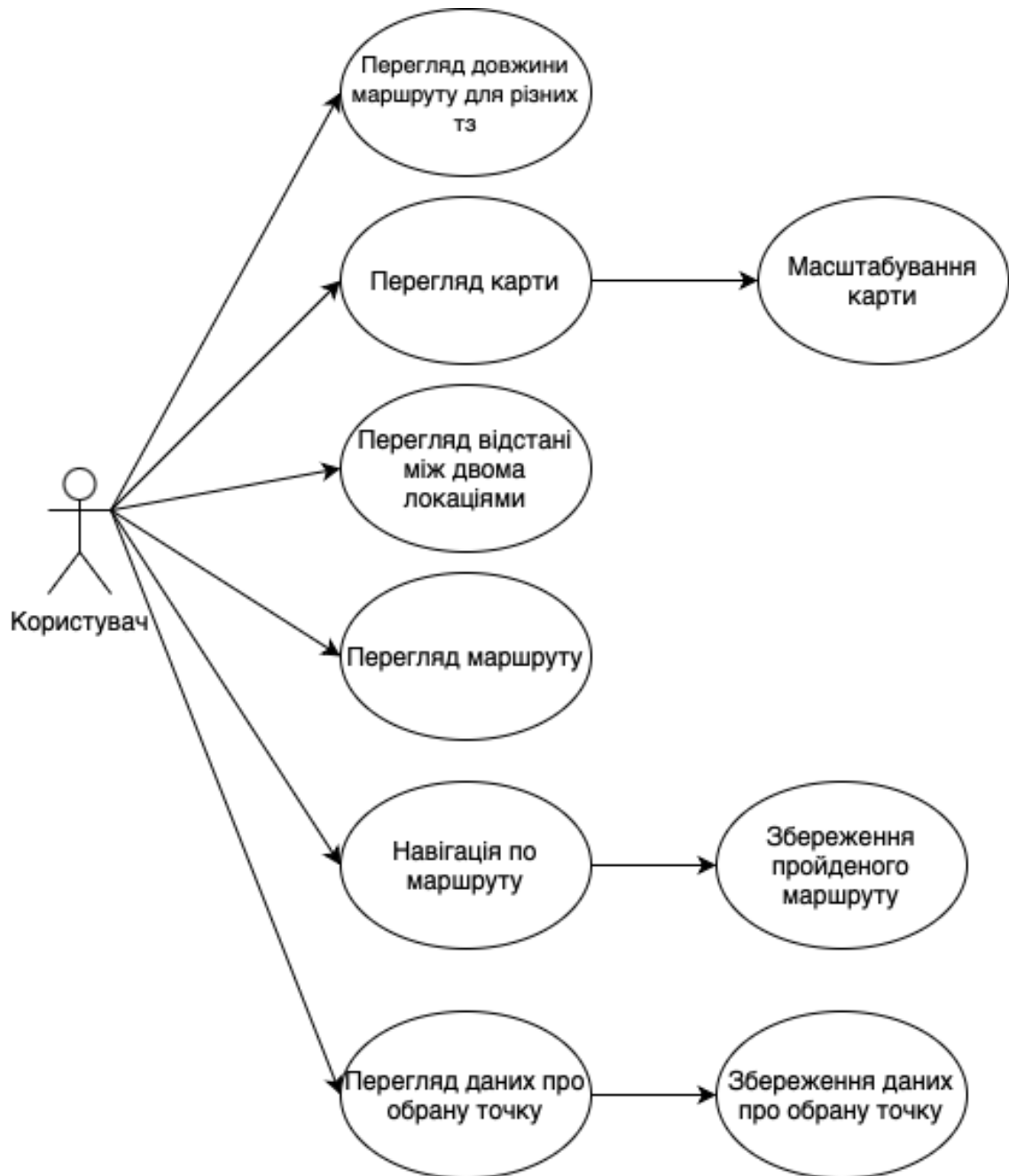


Рисунок 2.1 – Схема варіантів використання

Нижче наведено основні сценарії, які будуть ілюструвати типові випадки використання додатку (табл. 2.1).

Таблиця 2.1

Типові випадки використання

Актори	Найменування	Опис
Користувач	Перегляд довжини маршруту для різних тз	Користувач може обрати тип транспортного засобу (наприклад, автомобіль, велосипед, пішохід) і переглянути довжину маршруту, відповідно до вибраного типу. Це допоможе користувачам планувати поїздки з урахуванням специфіки свого транспорту.
Користувач	Перегляд карти	Користувач може переглядати інтерактивну карту, яка показує різні географічні об'єкти, маршрути та інші важливі деталі. Карта може включати в себе деталі місцевості, дороги, водні шляхи та інші елементи, які допомагають користувачу орієнтуватися.
Користувач	Масштабування карти	Користувач може збільшувати або зменшувати масштаб карти для детального перегляду конкретних областей або для отримання загального уявлення про місцевість. Це дозволяє користувачу більш точно визначити своє місцезнаходження та спланувати маршрут.

Користувач	Перегляд відстані між двома локаціями	Користувач може обрати дві точки на карті та переглянути відстань між ними. Це зручно для планування подорожей та визначення часу, необхідного для пересування між двома локаціями.
Користувач	Перегляд маршруту	Користувач може переглянути повний маршрут від початкової до кінцевої точки, включаючи всі повороти, зупинки та інші важливі деталі. Це допомагає зрозуміти шлях і підготуватися до поїздки.
Користувач	Навігація по маршруту	Користувач може скористатися функцією навігації, яка надає покрокові інструкції для слідування по маршруту. Це включає голосові підказки та візуальні індикатори, які допомагають користувачу не збитися з шляху.
Користувач	Збереження пройденого маршруту	Після завершення поїздки користувач може зберегти пройдений маршрут для подальшого використання або аналізу. Це може бути корисно для відстеження подорожей, планування повторних маршрутів або ділення інформацією з іншим

Користувач	Перегляд даних про обрану точку	Користувач може обрати конкретну точку на карті та переглянути детальну інформацію про неї, таку як адреса, координати, цікаві місця поблизу тощо. Це допомагає краще зрозуміти контекст обраної локації.
Користувач	Збереження даних про обрану точку	Користувач може зберегти дані про конкретну точку на карті, щоб швидко знайти її в майбутньому. Це може включати створення закладок, додавання нотаток та інші способи персоналізації.

Таким чином, діаграма варіантів використання є важливою складовою процесу проектування системи. Вона допомагає краще зрозуміти потреби користувачів. Аналізуючи різні сценарії взаємодії між користувачами та системою, ця діаграма дозволяє визначити ключові функціональні можливості, які повинна мати система. Вона зосереджує увагу на реальних потребах та очікуваннях користувачів, що допомагає уникнути непорозумінь і забезпечити відповідність функціональності системи вимогам користувачів.

Протокол-орієнтована структура системи

Протокол-орієнтоване програмування — це парадигма, яка наголошує на використанні протоколів, схожих на інтерфейси в інших мовах. У POP протоколи визначають план методів, властивостей та інших вимог, які відповідають певному завданню чи частині функціональності [24]. Такий підхід дозволяє розробникам створювати гнучкі та багаторазово використовувані компоненти.

У Swift протоколи використовуються для досягнення поліморфізму, тобто здатності різних типів розглядатися як один тип через загальний інтерфейс. Це особливо корисно для написання загального та модульного коду. Замість того,

щоб покладатися на успадкування, коли один клас походить від іншого, Swift заохочує використання протоколів для визначення спільної поведінки та можливостей.

Ключовою перевагою протоково-орієнтованого програмування є те, що воно сприяє композиції над успадкуванням. Композиція передбачає побудову складних типів шляхом поєднання простіших, які можуть бути більш гнучкими та легшими в обслуговуванні, ніж традиційне успадкування класів. Це добре узгоджується з наголосом Swift на безпеці та продуктивності, оскільки зменшує ймовірність створення тісно пов'язаних і схильних до помилок ієрархій класів [25].

Протоколи в Swift можна розширити, щоб забезпечити реалізацію своїх вимог за замовчуванням. Це означає, що ви можете визначити протокол із набором методів, а потім надати спосіб виконання цих методів за замовчуванням. Типи, які відповідають протоколу, можуть використовувати ці реалізації за замовчуванням або надавати власні власні реалізації. Ця функція допомагає зменшити дублювання коду та підвищити можливість повторного використання коду.

Стандартна бібліотека Swift значною мірою побудована на принципах, орієнтованих на протокол. Багато потужних функцій Swift, наприклад його структура колекцій, покладаються на протоколи для визначення загальних інтерфейсів для масивів, словників, наборів та інших типів. Це полегшує написання загальних алгоритмів, які можуть працювати з будь-яким типом, що відповідає певному протоколу.

Загалом, протокол-орієнтоване програмування в Swift пропонує сучасний підхід до створення програмного забезпечення. Він надає спосіб написання чистого, модульного та багаторазового коду, зосереджуючись на тому, що об'єкт робить, а не на тому, що він є. Ця зміна парадигми від традиційного об'єктно-орієнтованого програмування до програмування, орієнтованого на протоколи,

стало одним із найвпливовіших внесків Swift у сучасну практику розробки програмного забезпечення.

2.2 Розробка бази даних системи

SwiftData — це фреймворк від Apple, розроблений для зберігання та управління даними в додатках на платформі iOS. Основна мета SwiftData — забезпечити зручний і ефективний спосіб зберігання даних локально на пристрої, що є критично важливим для додатків, які працюють з конфіденційною інформацією або потребують високої швидкості доступу до даних без залежності від мережевих з'єднань. Фреймворк SwiftData відзначається простотою використання завдяки інтуїтивно зрозумілому API, який дозволяє розробникам швидко інтегрувати його у свої проєкти. Це робить його ідеальним вибором навіть для тих, хто має обмежений досвід роботи з базами даних.

SwiftData підтримує об'єктно-реляційну модель для зберігання даних, що дозволяє розробникам описувати свої дані у вигляді класів. Ці класи автоматично відображаються у відповідні таблиці бази даних, спрощуючи процес зберігання та отримання інформації.

Однією з ключових переваг SwiftData є автоматичне керування версіями бази даних. Це означає, що при внесенні змін до моделей даних фреймворк автоматично оновлює схему бази даних, що допомагає уникнути проблем з міграцією даних між різними версіями додатку.

Крім того, SwiftData надає потужні інструменти для виконання запитів до бази даних, включаючи можливості фільтрації, сортування та обмеження результатів. Це дозволяє легко отримувати тільки ті дані, які потрібні для конкретної операції, забезпечуючи високу продуктивність та ефективність роботи додатку.

Інтеграція SwiftData з мовою програмування Swift забезпечує всі переваги цієї мови, такі як типова безпека, висока продуктивність та зручність читання

коду. Це робить розробку додатків з використанням SwiftData ще більш привабливою для розробників.

Оскільки наша задача зробити додаток максимально зручним та безпечним, кількість даних та таблиць необхідних для збереження даних є мінімальною.

Нижче приводиться опис двох таблиць (табл. 2.2), (табл. 2.3):

Таблиця 2.2

Опис таблиці Pin

Назва	Тип	Опис
id	UUID	Унікальний ідентифікатор
Location	CLLocationCoordinate2D	Координата збереженої локації

Таблиця 2.3

Опис таблиці PinInfoModel

Назва	Тип	Опис
id	UUID	Унікальний ідентифікатор
Route	MKRoute	Збережений маршрут
startingPoint	CLLocationCoordinate2D	Початкова точка маршруту
finishingPoint	CLLocationCoordinate2D	Кінцева точка маршруту

2.3 Реалізація функціоналу додатку допомоги переміщення військовим

Спочатку розглянемо реалізацію основних функцій додатку. Для цього було використано вбудовану в MapKit структуру під назвою Map. Map – це

представлення який відображає цілісний інтерфейс карти. Реалізація даного представлення наведена на (рис. 2.3).

```
Map(position: $cameraPosition) {  
    UserAnnotation()  
}
```

Рис. 2.3 Реалізація Map

Для того щоб успішно відобразити геолокацію користувача на мапі було використано метод `startLocationServices` (рис. 2.4) який перевіряє чи надав користувач дозвіл на відстеження геопозиції і якщо відповідь позитивна, починає відстежувати геолокацію користувача

```
func startLocationServices() {  
    if manager.authorizationStatus == .authorizedAlways || manager.authorizationStatus == .authorizedWhenInUse {  
        manager.startUpdatingLocation()  
        isAuthorized = true  
    } else {  
        isAuthorized = false  
        manager.requestWhenInUseAuthorization()  
    }  
}
```

Рис. 2.4 Реалізація методу `startLocationServices`

Для того, щоб відобразити вікно, в якому користувач має вибрати чи хоче він, щоб додаток відслідковував його геопозицію, було створено метод `locationManagerDidChangeAuthorization` (рис. 2.5).

```
func locationManagerDidChangeAuthorization(_ manager: CLLocationManager) {  
    switch manager.authorizationStatus {  
    case .authorizedAlways, .authorizedWhenInUse:  
        isAuthorized = true  
        manager.requestLocation()  
    case .notDetermined:  
        isAuthorized = false  
        manager.requestWhenInUseAuthorization()  
    case .denied:  
        isAuthorized = false  
        print("access denied")  
    default:  
        isAuthorized = true  
        startLocationServices()  
    }  
}
```

Рис. 2.5 Реалізація методу `locationManagerDidChangeAuthorization`

Відображення таких параметрів як: дистанція по прямій, назва вулиці, координати і так далі, реалізовано за допомогою вбудованого в SwiftUI представлення Sheet. На (рис. 2.6) зображено реалізацію sheet, яка відповідає за відображення результатів вимірювання відстані між двома точками по прямій.

```
.sheet(isPresented: $vm.isShowingRullerResults, onDismiss: {
    isInMeasuringMode = false
    vm.locations = []
}) {
    VStack {
        Text("Дистанція між обраними точками становить:")
        Text("\(vm.distanceBeetween.components(separatedBy: (","))[0]) метрів")
    }.presentationDetents([.fraction(0.3)])
}
```

Рис. 2.6 Реалізація sheet для відображення результатів вимірювання

На (рис. 2.7) зображено реалізацію sheet, яка відповідає за відображення інструкцій для маршруту. Цей код використовується для створення інтерфейсу, який показує інструкції маршруту в модальному вікні. Він написаний на SwiftUI і включає функцію показу модального вікна (.sheet), яке відображається, коли змінна vm.isShowingRouteDirections встановлена в true. Після закриття цього вікна змінна повертається в false.

Всередині модального вікна використовується горизонтальна прокрутка (ScrollView), що дозволяє користувачеві переглядати інструкції маршруту, якщо їх багато. Усі інструкції розташовані горизонтально в HStack, який містить елементи для кожного напрямку. Для кожної інструкції створюється VStack, що вертикально розміщує іконку, текст інструкції та відстань. Іконка вибирається на основі тексту інструкції: якщо в тексті є слово "ліво", використовується іконка зі стрілкою назад, якщо "право" — стрілка направо. Під іконкою знаходиться текст інструкції та відстань у метрах.

Кожен елемент у VStack має фіксовані розміри (ширина 120, висота 200), що робить відображення інструкцій упорядкованим і зручним для читання. Додатково, модальне вікно оснащено індикатором, що показує можливість його перетягування, а також встановлено на висоту 30% від екрану, що забезпечує компактний вигляд вікна і зручність використання.

```

.sheet(isPresented: $vm.isShowingRouteDirections, onDismiss: {
    vm.isShowingRouteDirections = false
}) {
    ScrollView(.horizontal) {
        HStack {
            ForEach(vm.routeDirections) { direction in
                VStack {
                    Image(systemName: direction.direction.instructions.contains("ліво")
                        ? "arrow.backward.square" : direction.direction.instructions.contains("право")
                        ? "arrow.right.square" : "")
                    Text(direction.direction.instructions)
                    Text(String(direction.direction.distance).components(separatedBy: (","))[0]) + Text("м")
                }
            }
        }
        .frame(width: 120, height: 200)
    }
    .presentationDragIndicator(.visible)
    .presentationDetents([.fraction(0.3)])
}

```

Рис. 2.7 Реалізація sheet для відображення напрямків

На (рис. 2.8) зображено реалізацію sheet, яка відповідає за відображення основної інформації про обрану локацію. Цей фрагмент коду SwiftUI відображає модальний аркуш, що містить географічну інформацію, і пропонує варіанти для спільного використання чи збереження цієї інформації.

Коли змінна стану `vm.showSheet` має значення true, аркуш відображається. В середині аркуша для організації вмісту використовується вертикальний стек (`VStack`).

Перший `VStack`, вирівняний за переднім краєм, містить три частини інформації. Назва вулиці відображається за допомогою `vm.geocodingReversedValue.name`. Під ним за допомогою властивостей `vm.geocodingReversedValue` відображаються деталі розташування, такі як країна, місцевість, адміністративна область і субадміністративна область. Цей текст оформлено шрифтом виноски та сірим кольором. Крім того, текстові компоненти використовують `presentationDetents([.fraction(0.3)])` для встановлення висоти аркуша та `presentationDragIndicator(.visible)` для додавання індикатора перетягування. Географічні координати, широта та довгота показані далі.

Після текстової інформації горизонтальний стек (`HStack`) містить дві кнопки. Перша кнопка – це кнопка спільного доступу, натискання якої викликає

`vm.didTapOnRoute(destination: coordinateOfTap)`, щоб ініціювати дію спільного доступу. Кнопка має рамку. Друга кнопка — це кнопка збереження, призначена для збереження деталей розташування. Хоча логіка збереження закоментована, вона пропонує створити об'єкт `PointDataToSave` і вставити його в контекст. Ця кнопка має помітну рамку.

Навколо вертикального стосу, що містить текстову інформацію, і навколо всього вмісту аркуша додається відступ. `HStack`, що містить кнопки, фіксований горизонтально, але гнучкий вертикально.

Модель перегляду (`vm`) керує даними та діями, а `coordinateOfTap` зберігає координати широти та довготи вибраного місця. Цей код створює ефективний аркуш SwiftUI, який відображає детальну інформацію про місцезнаходження та дозволяє користувачам ділитися цими даними або зберігати їх із добре структурованим і візуально привабливим макетом.

```
.sheet(isPresented: $vm.showSheet) {
    VStack {
        VStack(alignment: .leading) {
            Text("Byname: \${vm.geocodingReversedValue.name}")
                .font(.title)
            Text("\${vm.geocodingReversedValue.country}, \${vm.geocodingReversedValue.locality}, \${vm.geocodingReversedValue.admistrativeArea}, \${vm.geocodingReversedValue.subAdministrativeArea}")
                .font(.footnote)
                .foregroundColor(.gray)
                .presentationDetents([.fraction(0.35)])
                .presentationDragIndicator(.visible)
            Text("Longitude: \${coordinateOfTap.longitude}, \${vm.geocodingReversedValue.longitude}")
            Text("Latitude: \${coordinateOfTap.latitude}, \${vm.geocodingReversedValue.latitude}")
            Text("Bearing = \${vm.bearing(from: coordinateOfTap, to: coordinateOfTap)}")
        }
        .padding(.bottom)
        HStack {
            Button(action: {
                vm.didTapOnRoute(destination: coordinateOfTap)
            }, label: {
                Text(Image(systemName: "square.and.arrow.up")) + Text(" ") + Text("Navigate")
            })
            .padding(.horizontal)
            .buttonStyle(.bordered)
            Button(action: {
                let location = PointDataToSave.init(descriptionOfPoint: vm.geocodingReversedValue, longitude: coordinateOfTap.longitude, latitude: coordinateOfTap.latitude)
                context.insert(location)
                print(query)
            }, label: {
                Text(Image(systemName: "square.and.arrow.down")) + Text(" ") + Text("Save")
            })
            .padding(.horizontal)
            .buttonStyle(.borderedProminent)
        }
        .fixedSize(horizontal: true, vertical: false)
        Button {
            label: {
                Text(Image(systemName: "paperplane")) + Text(" ") + Text("Share")
            }
        }
        .padding(.horizontal)
        .buttonStyle(.borderedProminent)
    }
    .padding()
}
```

Рис. 2.8 Реалізація sheet для відображення інформації про локацію

На (рис. 2.9) зображено реалізацію методу, який призначений для обчислення відстані між двома географічними точками, які зберігаються у масиві `locations`. Спочатку він отримує координати двох точок з цього масиву. Далі координати кожної точки перетворюються в об'єкти типу `CLLocation`, що представляють конкретне місце на Землі з заданою широтою та довготою.

Після цього метод використовує вбудовану функцію `distance(from:)` класу `CLLocation`, щоб обчислити відстань між двома точками в метрах. Отриманий результат зберігається у змінній `distanceBetween` після форматування, що перетворює числове значення на зручний для читання рядок. Нарешті, змінна `isShowingRullerResults` встановлюється в `true`, що використовується для індикації того, що результат готовий і може бути відображений користувачу.

Таким чином, цей метод забезпечує обчислення відстані між двома точками та підготовку даних для їх подальшого відображення у користувацькому інтерфейсі, що може бути корисно для різних цілей, таких як планування подорожей або вимірювання відстаней на карті.

```
func calculateDistanceBetweenPoints() {
    let firstValue = locations[0].coordinate
    let secondValue = locations[1].coordinate

    let firstValueConverted = CLLocation(latitude: firstValue.latitude, longitude: firstValue.longitude)
    let secondValueConverted = CLLocation(latitude: secondValue.latitude, longitude: secondValue.longitude)

    let result = firstValueConverted.distance(from: secondValueConverted)
    distanceBetween = result.formatted()

    isShowingRullerResults = true
}
```

Рис. 2.9 Реалізація методу `calculateDistanceBetweenPoints`

На (рис. 2.10) зображено реалізацію методу `didTapOnRoute`, який виконується при натисканні на маршрут з вказаним місцем призначення. Він спочатку очищає масив `routeDirections` і перевіряє, чи є поточне місцезнаходження користувача. Якщо його немає, метод завершується. Далі метод ініціалізує запит `MKDirections.Request`, встановлюючи вихідну точку як поточне місцезнаходження користувача і точку призначення як задані координати.

Потім метод використовує асинхронне завдання `Task`, щоб створити об'єкт `MKDirections` і обчислити маршрут. Якщо обчислення вдається, результат зберігається у змінній `route`. Анімація виконується для оновлення маршруту в інтерфейсі користувача. Після цього, якщо маршрут успішно обчислено, метод асинхронно оновлює головний потік, де додає кроки маршруту до масиву `routeDirections`, якщо вони містять інструкції.

Нарешті, метод змінює кілька змінних стану: вимикає показ результатів вимірювання лінійки (`isShowingRullerResults`), закриває вікна (`showSheet` і `isShowingRouteDirections`), і встановлює режим відображення маршрутів (`isInRoutesMode`). Цей метод забезпечує обчислення маршруту між поточним місцезнаходженням користувача та заданим місцем призначення, відображення інструкцій для маршруту та оновлення інтерфейсу користувача відповідно до нового маршруту.

```
func didTapOnRoute(destination: CLLocationCoordinate2D) {
    routeDirections = []
    guard let userLocation else { return }
    route = nil
    let request = MKDirections.Request()
    request.source = MKMapItem(placemark: .init(coordinate: userLocation.coordinate))
    request.destination = MKMapItem(placemark: .init(coordinate: destination))
    Task {
        let directions = MKDirections(request: request)
        let response = try? await directions.calculate()

        withAnimation {
            route = response?.routes.first
        }

        guard let route else { return }

        DispatchQueue.main.async {
            = route.steps.map { [weak self] in
                guard $0.instructions != "" else { return }
                self?.routeDirections.append(.init(direction: $0))
            }

            self.isShowingRullerResults = false
            self.showSheet = false
            self.isShowingRouteDirections = false
            self.isInRoutesMode = true
        }
    }
}
```

Рис. 2.10 Реалізація методу `didTapOnRoute`

На (рис. 2.11) зображено реалізацію методу, який призначений для оновлення напрямків до заданого місця призначення. Він починається з перевірки, чи є у програмі поточне місцезнаходження користувача (`userLocation`). Якщо місцезнаходження відсутнє, метод припиняє свою роботу.

Далі створюється об'єкт `MKDirections.Request`, в якому вказуються вихідна точка (поточне місцезнаходження користувача) і точка призначення (координати, передані в метод).

Після цього метод використовує асинхронне завдання (Task), щоб створити об'єкт MKDirections з цим запитом і спробувати обчислити напрямки. Виклик directions.calculate() виконується асинхронно, щоб не блокувати основний потік.

Якщо обчислення вдається, результат зберігається в змінній route. Оновлення маршруту виконується з анімацією, щоб зробити процес плавним і приємним для користувача.

Цей метод забезпечує отримання та оновлення інформації про маршрут між поточним місцезнаходженням користувача та заданим місцем призначення, дозволяючи додатку динамічно реагувати на зміни та відображати актуальні маршрути.

```
func updateDirections(destination: CLLocationCoordinate2D) {  
    guard let userLocation else { return }  
    let request = MKDirections.Request()  
    request.source = MKMapItem(placemark: .init(coordinate: userLocation.coordinate))  
    request.destination = MKMapItem(placemark: .init(coordinate: destination))  
    Task {  
        let directions = MKDirections(request: request)  
        let response = try? await directions.calculate()  
  
        withAnimation {  
            route = response?.routes.first  
        }  
    }  
}
```

Рисунок 2.11 Реалізація методу updateDirections

На (рис. 2.12) зображено реалізацію методу, didTapOnLocation, який виконується при натисканні на конкретну точку на карті. Він призначений для отримання детальної інформації про обрану географічну локацію за допомогою зворотного геокодування.

Метод починається зі створення об'єкта CLGeocoder, який використовується для перетворення координат місця в зрозумілу для користувача адресу. Потім створюється об'єкт CLLocation з використанням переданих у метод координат.

Далі виконується зворотне геокодування за допомогою методу reverseGeocodeLocation. Цей метод асинхронно надсилає запит на сервери Apple для отримання інформації про місцезнаходження. У разі успіху,

він повертає масив placemarks, який містить різні деталі про місце, такі як назва, країна, код країни, адміністративна область, субадміністративна область, населений пункт і поштовий індекс.

У циклі for кожен об'єкт placemark перетворюється на модель PinInfoModel, яка включає в себе всю зібрану інформацію або повідомлення "No data", якщо якесь поле порожнє. Потім ця інформація зберігається у властивості geocodingReversedValue для подальшого використання в додатку.

Таким чином, метод didTapOnLocation забезпечує зворотне геокодування координат обраного місця і зберігає детальну інформацію про це місце, що дозволяє додатку надавати користувачеві зрозумілі дані про обрану локацію.

```
func didTapOnLocation(location: CLLocationCoordinate2D) {
    let coder = CLGeocoder()
    print(location)

    let myLocation = CLLocation(latitude: location.latitude, longitude: location.longitude)

    coder.reverseGeocodeLocation(myLocation) { [weak self] (placemarks, error) in
        for placemark in placemarks! {
            let result: PinInfoModel = .init(name: placemark.name ?? "No data", country: placemark.country ?? "No data", isoCountryCode:
            placemark.isoCountryCode ?? "No data", administrativeArea: placemark.administrativeArea ?? "No data", subAdministrativeArea:
            placemark.subAdministrativeArea ?? "No data", locality: placemark.locality ?? "No data", postalCode: placemark.postalCode ?? "No data")

            self?.geocodingReversedValue = result
        }
    }
    showSheet = true
}

func didTapOnSavingLocation(location: CLLocationCoordinate2D) {
}
}
```

Рисунок 2.12 Реалізація методу didTapOnLocation

На (рис. 2.13) зображено реалізацію структури, яка допомагає відображенню основних даних про геолокацію. PinInfoModel - це структура, яка використовується для зберігання інформації про певне географічне місце. Вона складається з декількох полів, кожне з яких містить специфічну інформацію про місцезнаходження. Нижче наведено опис кожного з полів структури:

- name: String - це поле містить назву місця. Назва може бути, наприклад, назвою вулиці, будівлі або іншого об'єкта, який ідентифікує конкретне місце на карті.
- country: String - це поле містить назву країни, де знаходиться місце. Наприклад, "Україна", "США" або "Німеччина".

- isoCountryCode: String - це поле містить код країни згідно з ISO стандартом. ISO-код країни складається з двох літер, наприклад, "UA" для України, "US" для США або "DE" для Німеччини.
- administrativeArea: String - це поле містить інформацію про адміністративну область, в якій знаходиться місце. Це може бути назва області, штату або іншого адміністративного поділу країни.
- subAdministrativeArea: String - це поле містить інформацію про підпорядковану адміністративну область. Це може бути назва району, муніципалітету або іншого меншого адміністративного підрозділу.
- locality: String - це поле містить назву населеного пункту, такого як місто або селище, де знаходиться місце.
- postalCode: String - це поле містить поштовий індекс місця. Поштовий індекс використовується для полегшення сортування та доставки пошти.

```

struct PinInfoModel: Codable, Comparable {
    static func < (lhs: PinInfoModel, rhs: PinInfoModel) -> Bool {
        return lhs.dateOfSave < rhs.dateOfSave
    }
    static func > (lhs: PinInfoModel, rhs: PinInfoModel) -> Bool {
        return lhs.dateOfSave > rhs.dateOfSave
    }
    static func == (lhs: PinInfoModel, rhs: PinInfoModel) -> Bool {
        return lhs.dateOfSave == rhs.dateOfSave
    }
}

let name: String
let country: String
let isoCountryCode: String
let administrativeArea: String
let subAdministrativeArea: String
let locality: String
let postalCode: String
var dateOfSave: Date = Date.now

enum CodingKeys: String, CodingKey {
    case name
    case country
    case isoCountryCode
    case administrativeArea
    case subAdministrativeArea
    case locality
    case postalCode
    case dateOfSave
}

```

Рисунок 2.13 Реалізація структури PinInfoModel

Висновки до другого розділу

У другому розділі було виконано проектування мобільного додатку, зокрема визначено варіанти використання, розроблено протокол-орієнтовану структуру системи та спроектовано базу даних. Розроблено діаграми варіантів використання, таблиці з описом основних варіантів та реалізовано архітектуру

MVVM (Model-View-ViewModel), яка забезпечує чітке розподілення обов'язків між різними частинами додатку.

Процес проектування включав детальний аналіз функціональних і нефункціональних вимог до системи, що дозволило зрозуміти потреби кінцевих користувачів та визначити необхідний функціонал. Було розроблено діаграму варіантів використання (Use Case Diagram), яка показала взаємодію між користувачами та системою для виконання конкретних функцій. Цей процес допоміг описати функціональні вимоги до системи і зобразити, як різні користувачі взаємодіють з нею.

Особливу увагу приділено архітектурі додатку, для чого було обрано архітектуру MVVM. Цей підхід дозволив чітко розділити бізнес-логіку, управління даними та користувацький інтерфейс, що значно спрощує тестування і підтримку коду. Використання MVVM сприяло підвищенню продуктивності розробників, зниженню кількості помилок та спрощенню процесу тестування і підтримки додатку.

Окрім цього, було спроектовано базу даних для зберігання необхідних даних, таких як координати маршрутів та збережені локації. Використання фреймворку SwiftData дозволило забезпечити ефективне управління даними, їх безпечне зберігання та швидкий доступ. Це особливо важливо для додатків, що працюють з конфіденційною інформацією або потребують високої швидкості доступу до даних без залежності від мережевих з'єднань.

Таким чином, у другому розділі було закладено надійну основу для функціонування додатку, що забезпечує його зручність і ефективність у використанні. Реалізація основних методів таких як `startLocationServices`, `locationManagerDidChangeAuthorization`, `calculateDistanceBetweenPoints`, `didTapOnRoute`, `updateDirections` та `didTapOnLocation` дозволила створити додаток, який відповідає вимогам користувачів і забезпечує високий рівень функціональності.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ДОДАТКОМ

3.1 Порядок встановлення та налаштування параметрів додатку

Діаграма розгортання (deployment diagram) є одним із типів діаграм у моделюванні UML (Unified Modeling Language), яка використовується для відображення фізичної архітектури системи [26]. Вона показує, як програмні компоненти розгортаються на апаратних компонентах, таких як сервери, робочі станції та інші пристрої.

Основні елементи діаграми розгортання включають:

- Вузли (Nodes): Вузли представляють апаратні пристрої або інші обчислювальні ресурси. Вони можуть бути фізичними (сервери, комп'ютери) або віртуальними (віртуальні машини).
- Артефакти (Artifacts): Артефакти представляють фізичні файли або програмні компоненти, які розгортаються на вузлах. Це можуть бути виконувані файли, бібліотеки, конфігураційні файли тощо.
- Зв'язки (Associations): Зв'язки між вузлами показують, як ці вузли взаємодіють один з одним, а також як артефакти розподіляються між ними.
- Контейнери: В деяких випадках діаграма може містити контейнери, що представляють середовища виконання, такі як веб-сервери або бази даних, на яких виконуються артефакти.

Діаграма розгортання корисна для розуміння і планування фізичного розгортання системи, виявлення потенційних точок відмови, забезпечення відповідності вимогам до обладнання та мережі, а також для координації між розробниками і адміністраторами систем. Подальший опис розгортання додатку візуалізований нижче (рис. 3.1).

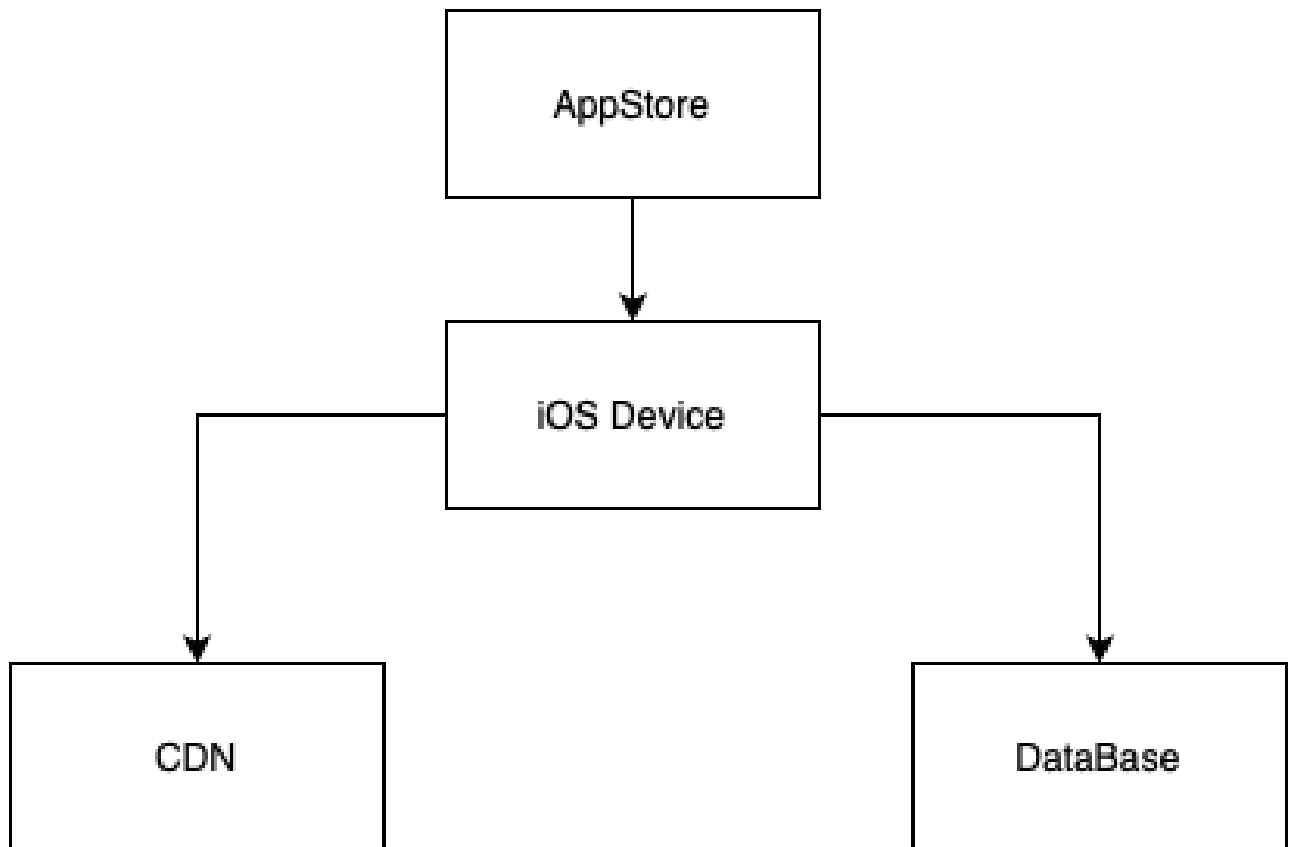


Рис. 3.1 Діаграма розгортання додатку

Діаграма розгортання для iOS застосунку демонструє наступні компоненти:

Пристрій (Клієнт):

- iOS Пристрій (iPhone/iPad): Основний пристрій, через який користувач взаємодіє із застосунком.
- iOS Застосунок: Мобільний застосунок, встановлений на iOS пристрої, який надає інтерфейс для користувача та взаємодіє з серверними компонентами.

App Store:

- App Store: Платформа, через яку користувачі завантажують та оновлюють застосунок.

Сервер БД (База даних):

- База Даних: Зберігає дані, необхідні для роботи застосунку, обробляє запити на зберігання і отримання даних від сервера.

Сховище файлів:

- CDN (Content Delivery Network): Мережа доставки контенту, що забезпечує швидке та ефективне завантаження статичних ресурсів, таких як зображення з супутника.

Опис взаємодії:

- Користувач за допомогою свого iOS пристрою взаємодіє із застосунком, який встановлений на цьому пристрої.
- Якщо запит стосується файлів з супутника Процесор Керування Файлами обробляє файли та використовує Інтерфейс Сховища для доступу до CDN, яке забезпечує доставку файлів до кінцевого користувача.
- App Store використовується для завантаження та оновлення застосунку на iOS пристроях користувачів.
- База даних зберігає необхідні дані.

Ця діаграма допомагає зрозуміти, як основні функції системи розподілені між різними компонентами та як ці компоненти взаємодіють один з одним, забезпечуючи належну роботу iOS застосунку.

3.2 Структура інтерфейсу додатку для переміщення військових

Коли користувач запускає додаток, його зустрічає головний екран, на якому знаходиться карта та головні кнопки. Зовнішній вигляд головного екрану виглядатиме таким чином (рис. 3.2).

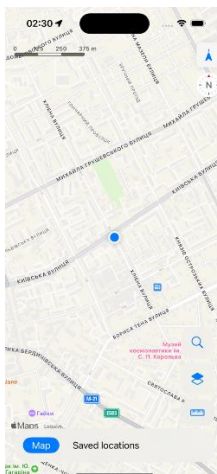


Рис. 3.2 Загальний вигляд застосунку

На головному екрані можна побачити: карту, кнопку зміни шарів карти, кнопку ввімкнення режиму вимірювання, кнопку пошуку, масштаб, компас, та кнопку відслідковування власної геопозиції. Для того щоб змінити шар, натискаємо на кнопку зміни шару та бачимо перед собою екран вибору шарів. Це виглядатиме наступним чином (рис. 3.3).

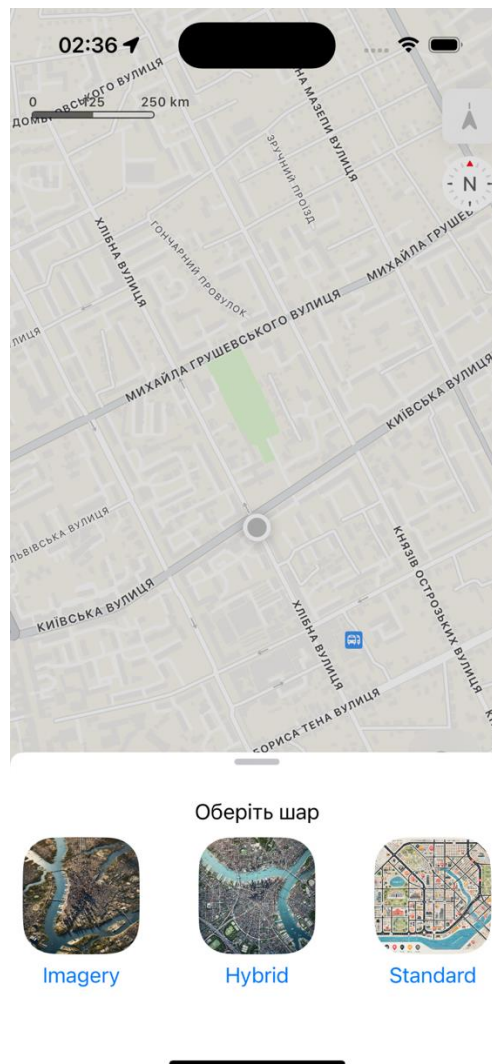


Рис. 3.3 Екран вибору шарів

Після того, як обрано шар під назвою «Hybrid», на карті можна спостерігати поєднання супутникових зображень з додатковими мітками та підписами, які зазвичай присутні на звичайній карті. Це забезпечує зручність перегляду територій, дозволяючи бачити як детальні зображення місцевості, так і важливу інформацію про дороги, будівлі та інші об'єкти інфраструктури.

Видозмінений вигляд головного меню можна побачити далі (рис.3.4).

Додатково, шар «Hybrid» є корисним інструментом для навігації та аналізу територій, оскільки він поєднує в собі переваги обох режимів – супутникового та картографічного. Це особливо зручно для користувачів, які хочуть отримати якнайбільше інформації про місцевість, яку вони досліджують.

Рис. 3.4 ілюструє, як виглядає карта у режимі «Hybrid», демонструючи чіткі супутникові зображення разом з важливими мітками, такими як назви вулиць, будівель та інші орієнтири. Це дозволяє легко орієнтуватися на місцевості та швидко знаходити потрібні об'єкти.

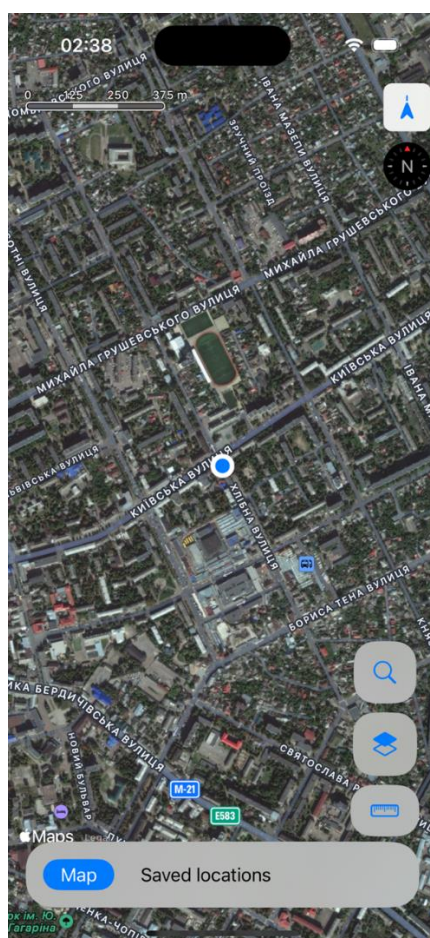


Рис. 3.4 Зовнішній вигляд додатку після того як було обрано шар під назвою «Hybrid»

Далі натиснемо на кнопку вимірювання відстаней. Як можна побачити на (рис. 3.5), додаток дає користувачеві зрозуміти, що він знаходиться в режимі вимірювання відстані, за допомогою підсвічування кнопки фіолетовим

кольором. Це підсвічування є корисним візуальним індикатором, що допомагає уникнути помилок і забезпечує зручність у використанні інструменту.

В режимі вимірювання відстаней користувач може вибрати будь-які дві точки на карті, після чого додаток автоматично розрахує відстань між ними. Це особливо корисно для планування маршрутів, оцінки відстаней між об'єктами та інших задач, що потребують точних вимірювань.

Рис.3.5 ілюструє активний режим вимірювання відстаней, де видно фіолетову підсвітку кнопки. Після натискання на карту, користувач може почати вимірювання, просто вибираючи потрібні точки. Відстань буде розрахована автоматично та відображена на екрані, що дозволяє швидко отримувати необхідну інформацію.

Використання цього інструменту є простим і зручним, забезпечуючи високу точність вимірювань та інтуїтивно зрозумілий інтерфейс. Це робить його незамінним для різних видів діяльності, які потребують знання точних відстаней на місцевості.

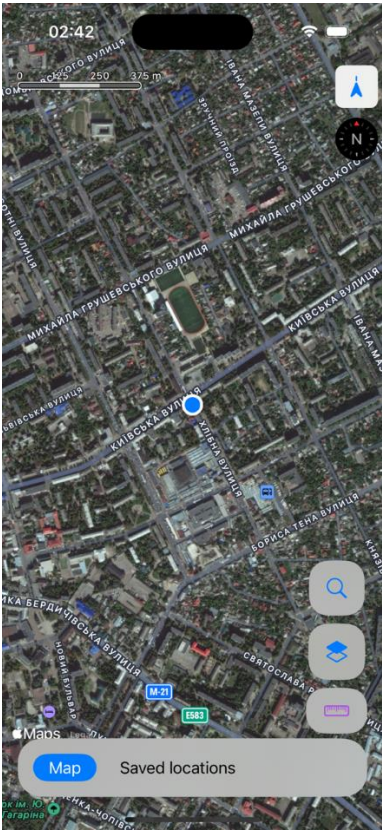
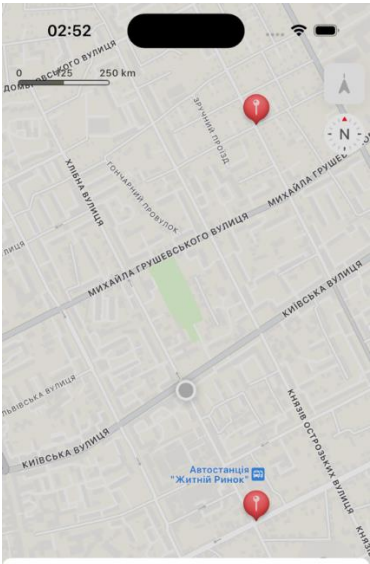


Рис. 3.5 Зовнішній вигляд додатку в режимі вимірювання

Далі, натискаємо та тримаємо на точці, яку хочемо зробити початковою. Після того як з'явився індикатор, натискаємо на точці на карті, яку хочемо зробити кінцевою. Як видно на (рис.3.6), додаток відображає де саме знаходяться початкова та кінцева точки, та відстань між ними по прямій.

Після вибору початкової точки індикатор, зазвичай у вигляді маркера або точки, з'являється на карті, що підтверджує успішний вибір. Потім, користувач може вибрати кінцеву точку, і додаток автоматично розрахує відстань між обраними точками по прямій лінії.

Рис.3.6 ілюструє цей процес, показуючи початкову та кінцеву точки, а також лінію, яка з'єднує їх, з позначенням відстані. Цей інтерфейс забезпечує зрозуміле та наочне відображення вимірної відстані, що є дуже корисним для різних задач, таких як планування подорожей, оцінка територій, чи навігація.



Дистанція між обраними точками становить:
1 292 метрів

Рис. 3.6 Зовнішній вигляд додатку при відображенні дистанції між двома точками

Далі просто натиснемо та потримаємо на випадковій точці на карті. Як видно на (рис.3.7), додаток відображає інформацію про місцезнаходження точки, та її координати, а також пропонує зберегти цю точку або почати навігацію до

неї. Це дозволяє користувачеві легко отримати детальну інформацію про будь-яку обрану точку на карті та використовувати ці дані для подальших дій.

Після натискання та утримання на обраній точці, на екрані з’являється інформаційне вікно. У цьому вікні вказується назва місця (якщо така доступна), точні географічні координати (широта і довгота), а також декілька опцій для подальших дій. Користувач може обрати зберегти цю точку для майбутніх використань або ж почати навігацію до неї, що робить цей інструмент дуже практичним для планування маршрутів і подорожей.

Рис.3.7 ілюструє цей процес, показуючи інформаційне вікно з деталями про обрану точку. Завдяки цьому інтерфейсу, користувачі можуть легко і швидко взаємодіяти з картою, отримуючи необхідну інформацію та плануючи свої дії без зайвих зусиль.

Ця функція додає додаткової гнучкості та функціональності додатку, роблячи його корисним інструментом для різноманітних задач, від простого ознайомлення з місцевістю до детального планування маршрутів і навігації.

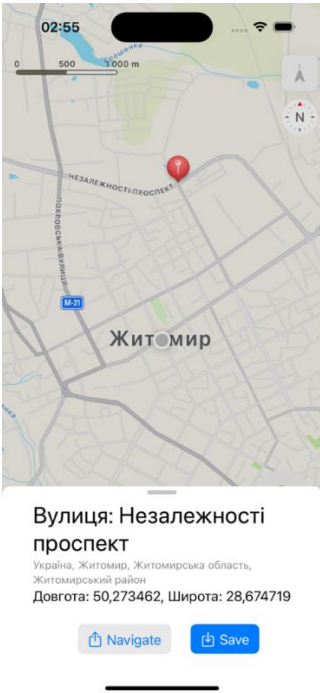


Рис. 3.7 Зовнішній вигляд додатку при відображенні інформації про локацію

Далі ми натискаємо на кнопку "почати навігацію" і бачимо перед собою маршрут та дві нові кнопки (рис. 3.8). Цей крок дозволяє користувачеві перейти від перегляду карти до активного використання навігаційних функцій додатку.

Після натискання кнопки "почати навігацію", додаток автоматично розраховує оптимальний маршрут від поточного місцезнаходження до вибраної точки. На карті з'являється лінія маршруту, що показує шлях, який необхідно пройти. Це допомагає користувачеві легко орієнтуватися та слідувати запропонованому шляху.

Рис.3.8 ілюструє цей етап, де на екрані видно маршрут і дві нові кнопки, які з'являються після запуску навігації. Ці кнопки можуть мати різні функції, залежно від додатку. Наприклад, вони можуть дозволяти зупинити навігацію, переглянути деталі маршруту, змінити налаштування навігації або скористатися іншими корисними функціями, такими як голосові підказки.

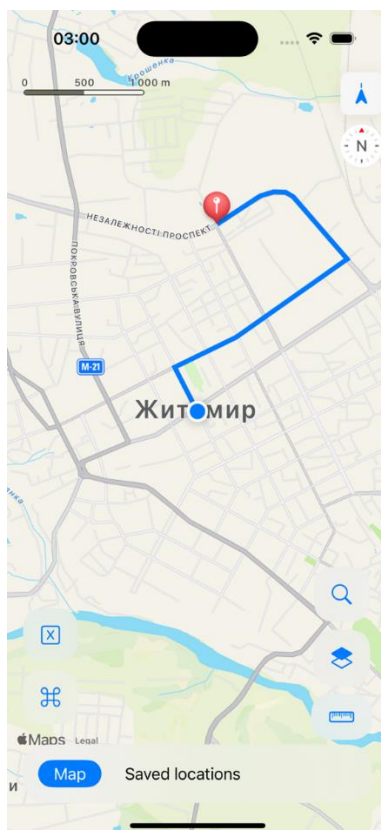


Рис. 3.8 Зовнішній вигляд додатку в режимі навігації

Далі, натиснувши на одну із кнопок, ми бачимо перед собою детальні кроки, які необхідно виконувати, щоб успішно дійти до місця призначення. Цей

етап забезпечує користувача покроковими інструкціями, що робить процес навігації ще більш зрозумілим і зручним.

Після натискання відповідної кнопки, на екрані з'являється список кроків маршруту. Кожен крок включає точні інструкції, такі як повороти, орієнтири та інші важливі деталі, що допомагають користувачеві не збитися з шляху. Крім текстових підказок, часто додаються візуальні ілюстрації або мапи, які полегшують орієнтування на місцевості.

Рис.3.9 ілюструє цей процес, показуючи екран з детальними інструкціями для кожного кроку маршруту. Це дозволяє користувачеві впевнено слідувати запропонованому шляху, знаючи, що він отримує точні та надійні підказки.

Ця функція є надзвичайно корисною для тих, хто подорожує незнайомою місцевістю або потребує точної навігації для досягнення місця призначення. Завдяки детальним покроковим інструкціям, користувачі можуть уникнути помилок і ефективно досягати своїх цілей, що робить навігаційний додаток незамінним інструментом у повсякденному житті.

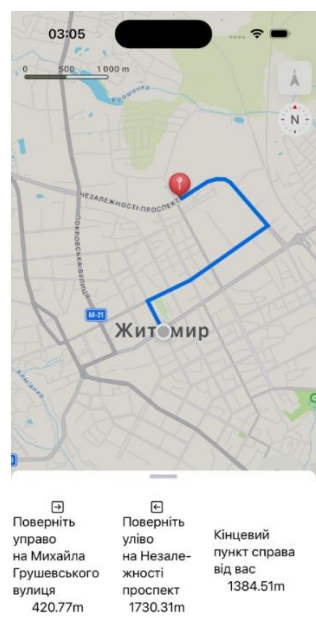


Рис. 3.9 Зовнішній вигляд додатку в режимі відображення кроків

Якщо ж ми натиснемо на іншу кнопку, то, як можна побачити на (рис.3.10), навігація просто закінчиться. Ця функція дозволяє користувачеві легко завершити навігаційний процес у будь-який момент, коли це необхідно.

Після натискання цієї кнопки, додаток припиняє показувати маршрут і повертається до звичайного режиму перегляду карти. Це зручно, коли користувач більше не потребує навігаційних підказок або вирішив змінити маршрут.

Рис.3.10 ілюструє цей процес, показуючи екран після завершення навігації. На екрані більше не видно лінії маршруту чи покрокових інструкцій, і додаток повертається до свого звичайного вигляду.

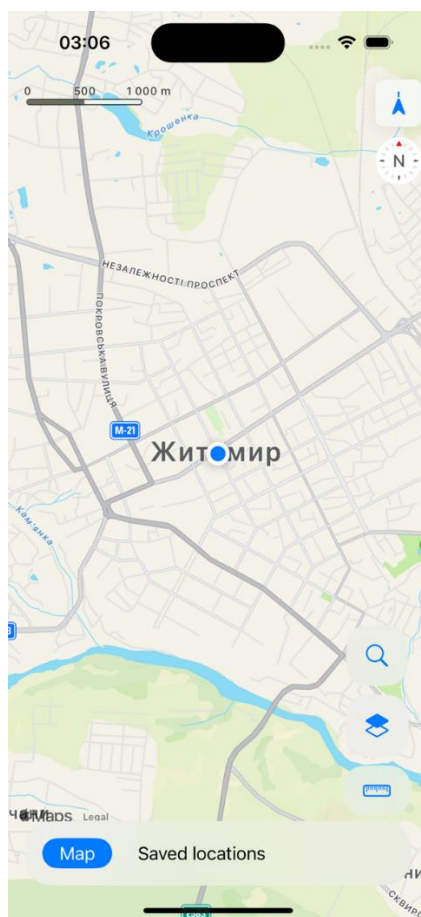


Рис. 3.10 Зовнішній вигляд додатку при завершенні навігації

Далі протестуємо функцію збереження даних. Для цього оберемо точку на карті та, у спливаючому меню, яке зображено на (рис.3.11), натиснемо на кнопку

збереження локації. Ця функція дозволяє користувачеві зберігати важливі місця для майбутнього використання, роблячи їх легкодоступними в будь-який момент

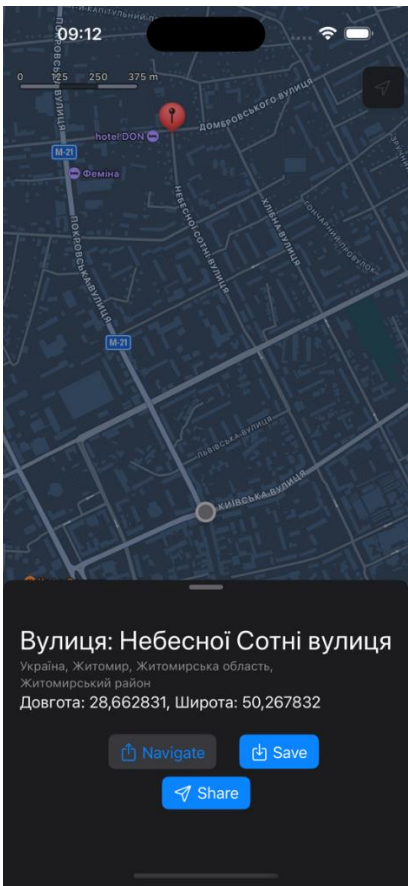


Рис. 3.11 Зовнішній вигляд екрану з кнопкою збереження локації

Після цього натискаємо на кнопку пошуку та переглядаємо збережені локації. Як видно на (рис.3.12), ця функція дозволяє користувачеві швидко знаходити і переглядати всі збережені місця, що значно спрощує процес управління важливими точками на карті.

Кнопка пошуку розташована на основному екрані додатку і дозволяє отримати доступ до списку збережених локацій. Після натискання на цю кнопку відкривається інтерфейс пошуку, де користувач може бачити всі збережені місця, відсортовані за різними критеріями, такими як дата збереження або відстань від поточного місця.

Рис.3.12 ілюструє інтерфейс зі списком збережених локацій. Користувач може обрати будь-яку з цих локацій для перегляду деталей, початку навігації до неї або видалення зі списку збережених.

Ця функція забезпечує зручність у використанні додатку, дозволяючи швидко та ефективно керувати важливими місцями, до яких користувач може повертатися в майбутньому.

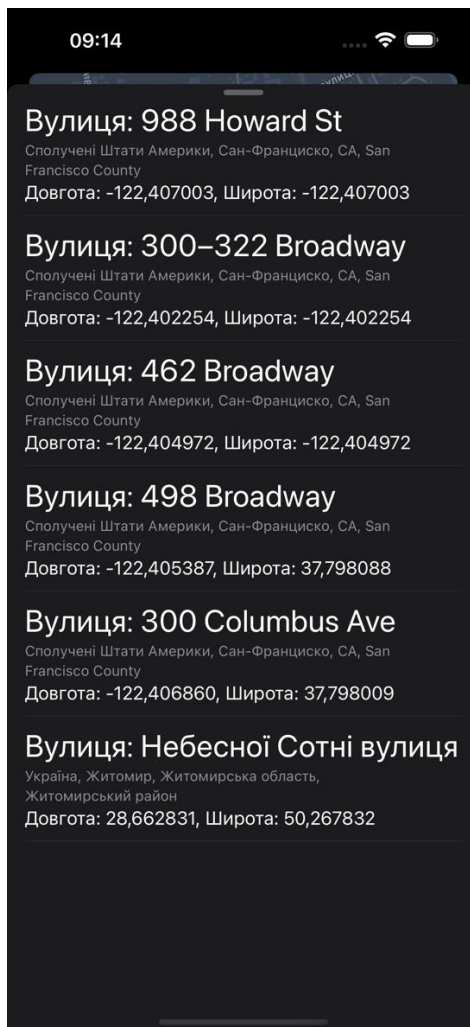


Рис. 3.12 Зовнішній вигляд екрану зі списком збережених локацій

Тепер, натискаємо на будь-яку локацію, яка нас цікавить, та бачимо, що додаток успішно відобразив геолокацію та інформацію про неї. Як видно на (рис.3.13), ця функція дозволяє користувачеві швидко отримати доступ до деталей збереженої локації та її точного розташування на карті.

Після вибору збереженої локації зі списку, додаток автоматично переносить користувача до цієї точки на карті, відображаючи всю необхідну інформацію про місце. Це може включати назву локації, її географічні координати, та додаткові примітки або деталі, які були збережені раніше.

Рис.3.13 ілюструє цей процес, показуючи, як виглядає карта з обраною локацією та інформаційним вікном, що містить всі необхідні дані про цю точку.

Ця функція забезпечує зручність і ефективність у використанні додатку, дозволяючи користувачам легко керувати своїми збереженими місцями та переглядати їх у будь-який момент.

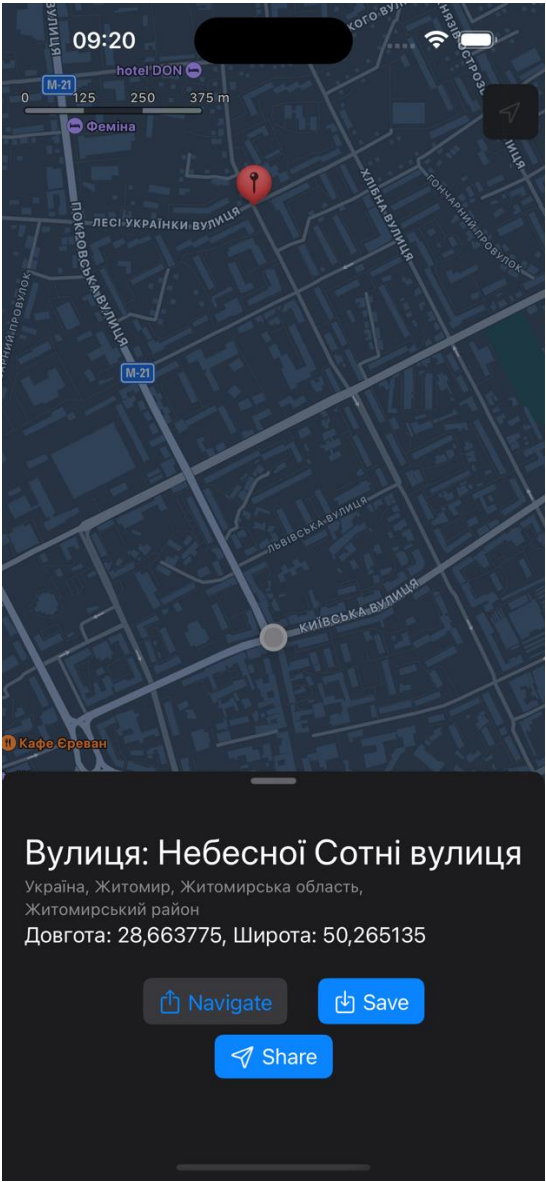


Рис. 3.13 Зовнішній вигляд екрану з картою обраної локації
Таким чином було оглянуто основну функціональність додатку.

3.3 Тестування

Перед тим як фіналізувати розробку програмного забезпечення, необхідно оцінити його якість. Зробити це можна проаналізувавши код і

протестувавши роботу застосунку

Для того щоб провести аналіз коду в даній роботі було використано застосунок embold. Результати аналізу (рис.3.14)

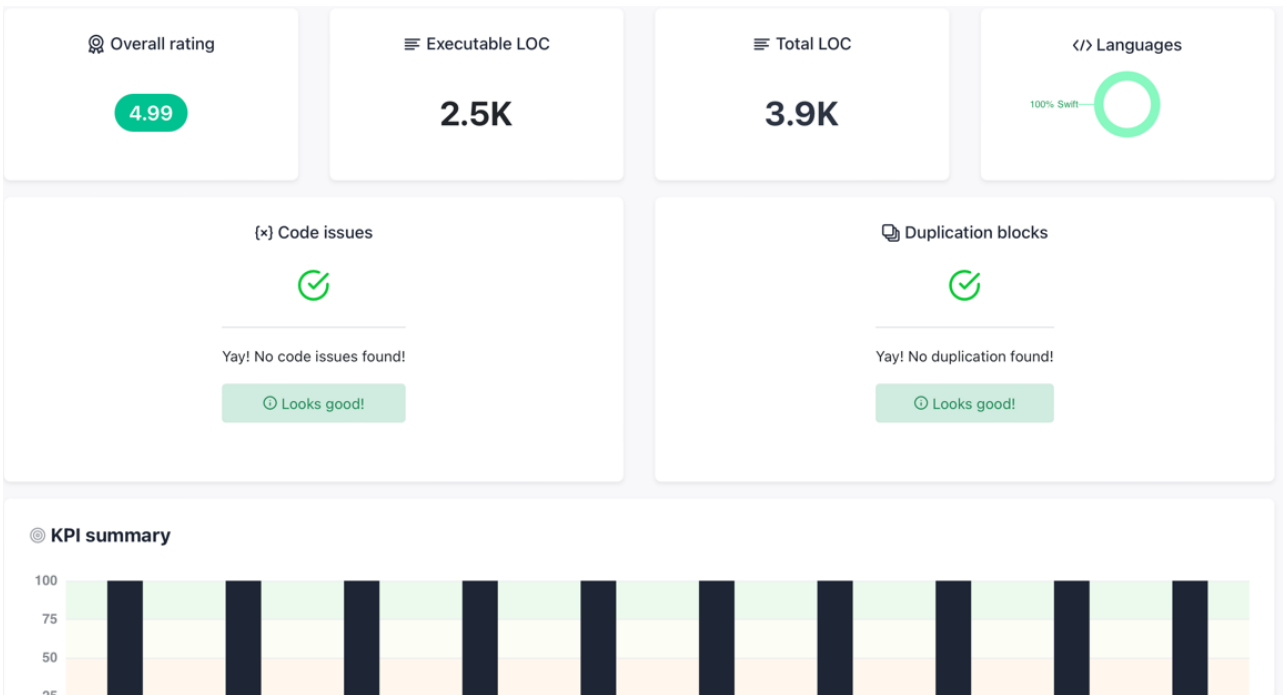


Рис. 3.14 Результати аналізу коду

Даний застосунок проводить аналіз коду за наступними параметрами:

- Пошук проблеми з кодом та вразливих місць
- Повторювання коду
- Використання антипаттернів

Мануальне тестування є критично важливим етапом розробки програмного забезпечення, особливо для мобільних додатків, які взаємодіють з кінцевими користувачами в реальному часі. Метою мануального тестування є виявлення багів, недоліків інтерфейсу, а також перевірка функціональності додатку в різних умовах використання. У цьому розділі ми розглянемо підхід до мануального тестування розробленого мобільного додатку для допомоги військовим.

Перед початком тестування було визначено основні функціональні вимоги додатку, які підлягатимуть перевірці. Серед них:

- Відсутність потреби в автентифікації.

- Можливість заміру відстані між двома точками.
- Відображення масштабу при наближенні чи віддаленні карти.
- Відображення координат міток.
- Перегляд історії пройдених маршрутів.
- Оповіщення при наближенні до кінцевого пункту призначення.

Тест-кейс 1

У цьому тест кейсі, я не дам дозвіл на відстеження геолокації, та спробую скористатися можливостями додатку, які залежать від геолокації користувача.

Таблиця 3.1

Тест-кейс 1

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Визначення геолокації користувача	Крапка, яка показує де знаходиться користувач не відобразиться на карті	Як і очікувалось	Виконано
2	Визначення маршруту від користувача до точки призначення	Маршрут не відобразиться	Як і очікувалось	Виконано
3	Вимірювання дистанції між двома точками	Дистанція успішно вимірюється	Як і очікувалось	Виконано

4	Повернення карти на локацію користувача	Карта не повернеться, оскільки локація користувача невідома	Як і очікувалося	Виконано
5	Навігація користувача за маршрутом	Навігація не відбудеться, оскільки невідомо де знаходиться користувач	Як і очікувалося	Виконано

Тест-кейс 2

У цьому тест кейсі, я спробую побудувати маршрути, переглянути відстані між точками та переглянути покрокові інструкції до маршруту.

Таблиця 3.2

Тест-кейс 2

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Натиснути кнопку вимірювання відстані	Відображення можливості встановлення початкової та кінцевої точок	Як і очікувалося	Виконано
2	Встановити початкову та кінцеву точки на карті	Відображення відстані між точками по прямій	Як і очікувалося	Виконано

3	Перейти до історії маршрутів	Відображення збережених маршрутів	Як і очікувалося	Виконано
4	Відкрити збережений маршрут	Відображення маршруту на карті	Як і очікувалося	Виконано

Тест-кейс 3

Таблиця 3.3

Тест-кейс 3

№	Кроки	Очікуваний результат	Отриманий результат	Статус
1	Запустити навігацію	Відображення покрокових інструкцій для навігації	Як і очікувалося	Виконано
2	Слідкувати за маршрутом	Оновлення поточного місцезнаходження на карті	Як і очікувалося	Виконано
3	Отримати сповіщення при наближенні до кінцевого пункту	Відображення сповіщення на екрані	Як і очікувалося	Виконано
4	Увімкнути GPS на пристрої	Автоматичне визначення поточного місцезнаходження	Як і очікувалося	Виконано

Проведення тест-кейсів є невід'ємною частиною розробки якісного програмного забезпечення. Тест-кейси дозволяють систематично перевіряти функціональність додатку, забезпечуючи його надійну та стабільну роботу. В процесі розробки мобільного додатку для допомоги військовим було виконано кілька ключових тест-кейсів, які продемонстрували свою високу ефективність та корисність.

Висновки до третього розділу

У третьому розділі було розглянуто інтерфейс та порядок роботи з додатком. Детально описано процес встановлення та налаштування додатку, представлено діаграму розгортання, яка відображає взаємодію між компонентами системи. Було описано структуру інтерфейсу додатку, включаючи головний екран, кнопки, режими роботи (зміна шарів карти, вимірювання відстані, перегляд інформації про локацію, навігація). Проведено мануальне тестування додатку, виявлено та виправлено критичні баги, що забезпечило стабільну та надійну роботу додатку.

ВИСНОВКИ

У рамках даної дипломної роботи було розроблено мобільний додаток для допомоги військовим у переміщенні, який використовує сучасні технології геолокації. Основною метою було створення зручного та ефективного інструменту, що дозволяє військовим планувати маршрути, отримувати інформацію про своє місцезнаходження та максимально використовувати можливості GPS.

На початковому етапі роботи було проведено ретельний аналіз аналогічних додатків, таких як Google Maps, Waze та інші. Це дозволило виявити їхні переваги та недоліки, а також визначити можливості для покращення. На основі цього аналізу були сформульовані основні вимоги до додатку, що забезпечило чітке розуміння цілей проекту та необхідних функціональних можливостей.

Проектування архітектури додатку включало визначення компонентної структури, вибір технологій та інструментів розробки. Було обрано сучасні та надійні технології, такі як Swift, SwiftUI, MapKit, CoreLocation, Combine та Xcode, які забезпечують високу продуктивність та зручність у розробці. Архітектура додатку базувалася на моделі MVVM (Model-View-ViewModel), що дозволило досягти чіткого розподілу відповідальностей та спрощення процесу розробки і тестування.

Реалізація основних функцій додатку включала створення інтерфейсу користувача, інтеграцію з сервісами геолокації, розробку алгоритмів для побудови маршрутів та їхнього відображення на карті. Особливу увагу було приділено зручності користування додатком, що досягалося шляхом інтуїтивно зрозумілого дизайну та наявності корисних функцій, таких як збереження маршрутів, перегляд відстаней між точками, навігація з голосовими підказками та багато іншого.

Проведено детальне мануальне тестування додатку, яке дозволило виявити та виправити критичні баги. Тестування проводилося за декількома сценаріями

					ІПЗ.КР.Б-121-24-ПЗ	
						64

використання, що охоплювали основні функціональні можливості додатку. Це забезпечило стабільну та надійну роботу додатку в різних умовах експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Савіцький Р.С., Радченко Д.В. Java та процес роботи функціоналу додатків для роботи з GPS. Збірник тез XI Всеукраїнської науково–практичної конференції молодих учених, 16 травня 2024. Київ: “Київський столичний університет імені Бориса Грінченка”, 2024. С. 86-88.
2. Google Maps [Електронний ресурс] – Режим доступу до ресурсу: <https://www.google.com/maps>
3. Waze [Електронний ресурс] – Режим доступу до ресурсу: <https://www.waze.com/uk/live-map/>
4. Vetus Maps [Електронний ресурс] – Режим доступу до ресурсу: <https://apps.apple.com/ua/app/vetus-maps/id1241471670?l=uk>
5. MVVM in iOS Swift [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@abhilash.mathur1891/mvvm-in-ios-swift-aa1448a66fb4>
6. Introducing MVVM into your Swift Project [Електронний ресурс] – Режим доступу до ресурсу: <https://www.hackingwithswift.com/books/ios-swiftui/introducing-mvvm-into-your-swiftui-project>
7. Swift Tutorial: An Introduction to the MVVM Design Pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://adevait.com/ios/swift-tutorial-mvvm-design-pattern>
8. Tutorial: An Introduction to the MVVM Design Pattern [Електронний ресурс] - Режим доступу до ресурсу: <https://www.toptal.com/ios/swift-tutorial-introduction-to-mvvm>
9. iOS MVVM Tutorial: Refactoring from MVC [Електронний ресурс] - Режим доступу до ресурсу: <https://www.kodeco.com/6733535-ios-mvvm-tutorial-refactoring-from-mvc>
10. Swift Docs [Електронний ресурс] – Режим доступу до ресурсу: <https://www.swift.org>
11. Swift [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.apple.com/swift/>

12. SwiftUI [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.apple.com/xcode/swiftui/>
13. Introducing SwiftUI [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.apple.com/tutorials/swiftui>
14. MapKit [Електронний ресурс] - Режим доступу до ресурсу: <https://developer.apple.com/documentation/mapkit/>
15. CoreLocation [Електронний ресурс] - Режим доступу до ресурсу: <https://portal.thirdweb.com/react/react.useaddress>
16. Combine [Електронний ресурс] - Режим доступу до ресурсу: <https://developer.apple.com/documentation/combine>
17. Hello, Combine [Електронний ресурс] - Режим доступу до ресурсу: <https://www.kodeco.com/books/combine-asynchronous-programming-with-swift/v1.0/chapters/1-hello-combine>
18. Xcode [Електронний ресурс] - Режим доступу до ресурсу: <https://developer.apple.com/xcode/>
19. SwiftData [Електронний ресурс] - Режим доступу до ресурсу: <https://developer.apple.com/xcode/swiftdata/>
20. Minimum requirements and supported SDKs [Електронний ресурс] - Режим доступу до ресурсу: <https://developer.apple.com/support/xcode/>
21. What are minimum system requirements for Xcode? [Електронний ресурс] - Режим доступу до ресурсу: <https://www.quora.com/What-are-the-minimum-hardware-requirements-for-Xcode>
22. Що таке функціональні вимоги: приклади, визначення, повний посібник [Електронний ресурс] - Режим доступу до ресурсу: <https://www.quora.com/What-are-the-minimum-hardware-requirements-for-Xcode>
23. UML Use Case Diagram Tutorial [Електронний ресурс] - Режим доступу до ресурсу: <https://www.lucidchart.com/pages/uml-use-case-diagram>

24. Protocol Oriented Programming in Swift [Электронный ресурс] - Режим доступа до ресурсу: <https://www.pluralsight.com/resources/blog/guides/protocol-oriented-programming-in-swift>

25. Protocol-oriented programming [Электронный ресурс] - Режим доступа до ресурсу: <https://www.hackingwithswift.com/sixty/9/5/protocol-oriented-programming>

26. What is deployment diagram? [Электронный ресурс] – Режим доступа до ресурсу: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/>

ДОДАТКИ

					ІІЗ.КР.Б-121-24-ІІЗ	
						69

Лістинг PointDataToSave.swift

```
import Foundation
import CoreLocation
import SwiftData

@Model
class PointDataToSave: Codable {
    enum CodingKeys: CodingKey {
        case descriptionOfPoint, longitude, latitude, dateOfSave
    }

    var descriptionOfPoint: PinInfoModel
    var longitude: Double
    var latitude: Double
    let dateOfSave: Date = Date.now

    init(descriptionOfPoint: PinInfoModel, longitude: Double, latitude: Double) {
        self.descriptionOfPoint = descriptionOfPoint
        self.longitude = longitude
        self.latitude = latitude
    }

    required init(from decoder: Decoder) throws {
        let container = try decoder.container(keyedBy: CodingKeys.self)
        descriptionOfPoint = try container.decode(PinInfoModel.self, forKey:
.descriptionOfPoint)
        longitude = try container.decode(Double.self, forKey: .longitude)
        latitude = try container.decode(Double.self, forKey: .latitude)
        dateOfSave = try container.decode(Date.self, forKey: .dateOfSave)
    }

    func encode(to encoder: Encoder) throws {
        var container = encoder.container(keyedBy: CodingKeys.self)
        try container.encode(descriptionOfPoint, forKey: .descriptionOfPoint)
        try container.encode(longitude, forKey: .longitude)
        try container.encode(latitude, forKey: .latitude)
        try container.encode(dateOfSave, forKey: .dateOfSave)
    }

    var getCLL: CLLocationCoordinate2D {
        return CLLocationCoordinate2D(latitude: latitude, longitude: longitude)
    }
}
```

```
}
```

Лістинг MainMapViewModel.swift

```
import Foundation
import MapKit
import SwiftUI
import SwiftData
```

```
struct RouteDirections: Identifiable {
    let id = UUID()
    var direction: MKRoute.Step

    // var getDirectionImage: String {
    //     switch direction.instructions
    // }
}
```

```
class MainMapViewModel: NSObject, ObservableObject,
CLLocationManagerDelegate {
    let manager = CLLocationManager()
    @Published var userLocation: CLLocation? {
        didSet {
            print(userLocation?.coordinate)
        }
    }
    var isAuthorized = false

    override init() {
        super.init()
        manager.delegate = self
        startLocationServices()
    }
```

```
    func startLocationServices() {
        if manager.authorizationStatus == .authorizedAlways ||
manager.authorizationStatus == .authorizedWhenInUse {
            manager.startUpdatingLocation()
            isAuthorized = true
        } else {
            isAuthorized = false
            manager.requestWhenInUseAuthorization()
        }
    }
}
```

```

func locationManager(_ manager: CLLocationManager,
didUpdateLocations locations: [CLLocation]) {
    userLocation = locations.last
}

func locationManagerDidChangeAuthorization(_ manager:
CLLocationManager) {
    switch manager.authorizationStatus {
    case .authorizedAlways, .authorizedWhenInUse:
        isAuthorized = true
        manager.requestLocation()
    case .notDetermined:
        isAuthorized = false
        manager.requestWhenInUseAuthorization()
    case .denied:
        isAuthorized = false
        print("access denied")
    default:
        isAuthorized = true
        startLocationServices()
    }
}

func locationManager(_ manager: CLLocationManager, didFailWithError
error: Error) {
    print(error.localizedDescription)
}

//MARK: Published things
@Published var isInRoutesMode: Bool = false
@Published var isShowingRullerResults: Bool = false
@Published var isShowingRouteDirections: Bool = false {
    didSet {
        print(isShowingRouteDirections)
    }
}
@Published var showSheet: Bool = false

@Published var routeDirections: [RouteDirections] = .init()
@Published var selectedResult: MKMapItem?
@Published var route: MKRoute?

```

```

@Published var distanceBetween: String = ""
@Published var geocodingReversedValue: PinInfoModel = .init(name: "",
country: "", isoCountryCode: "", administrativeArea: "", subAdministrativeArea:
"", locality: "", postalCode: "")
@Published var locations: [Pin] = .init() {
    didSet {
        if locations.count == 2 {
            calculateDistanceBetweenPoints()
        }
    }
}

```

```

@Query(sort: \PointDataToSave.descriptionOfPoint) var query:
[PointDataToSave]

```

```

func calculateDistanceBetweenPoints() {
    let firstValue = locations[0].coordinate
    let secondValue = locations[1].coordinate

    let firstValueConverted = CLLocation(latitude: firstValue.latitude,
longitude: firstValue.longitude)
    let secondValueConverted = CLLocation(latitude: secondValue.latitude,
longitude: secondValue.longitude)

    let result = firstValueConverted.distance(from: secondValueConverted)
    distanceBetween = result.formatted()

    isShowingRullerResults = true
}

```

```

func dismissRoute() {
    routeDirections = []
    route = nil
    islnRoutesMode = false
}

```

```

func didTapOnRoute(destination: CLLocationCoordinate2D) {
    routeDirections = []
    guard let userLocation else { return }
    route = nil
    let request = MKDirections.Request()

```

```

    request.source = MKMapItem(placemark: .init(coordinate:
userLocation.coordinate))
    request.destination = MKMapItem(placemark: .init(coordinate:
destination))

```

```

    Task {
        let directions = MKDirections(request: request)
        let response = try? await directions.calculate()

        withAnimation {
            route = response?.routes.first
        }

        guard let route else { return }

        DispatchQueue.main.async {
            _ = route.steps.map { [weak self] in
                guard $0.instructions != "" else { return }
                self?.routeDirections.append(.init(direction: $0))
            }

            self.isShowingRulerResults = false
            self.showSheet = false
            self.isShowingRouteDirections = false
            self.isInRoutesMode = true
        }
    }
}

```

```

func updateDirections(destination: CLLocationCoordinate2D) {
    guard let userLocation else { return }
    let request = MKDirections.Request()
    request.source = MKMapItem(placemark: .init(coordinate:
userLocation.coordinate))
    request.destination = MKMapItem(placemark: .init(coordinate:
destination))
    Task {
        let directions = MKDirections(request: request)
        let response = try? await directions.calculate()

        withAnimation {
            route = response?.routes.first
        }
    }
}

```

```

func didTapOnLocation(location: CLLocationCoordinate2D) {
    let coder = CLGeocoder()
    print(location)

    let myLocation = CLLocation(latitude: location.latitude, longitude:
location.longitude)

    coder.reverseGeocodeLocation(myLocation) { [weak self] (placemarks,
error) in
        for placemark in placemarks! {
            let result: PinInfoModel = .init(name: placemark.name ?? "No data",
country: placemark.country ?? "No data", isoCountryCode:
placemark.isoCountryCode ?? "No data", administrativeArea:
placemark.administrativeArea ?? "No data", subAdministrativeArea:
placemark.subAdministrativeArea ?? "No data", locality: placemark.locality ??
"No data", postalCode: placemark.postalCode ?? "No data")

            self?.geocodingReversedValue = result
        }
    }
    showSheet = true
}

func didTapOnSavingLocation(location: CLLocationCoordinate2D) {
}
}

```

Лістинг MainMapView.swift

```

import SwiftUI
import MapKit
import SwiftData

struct MainMapView: View {
    @StateObject private var vm = MainMapViewModel()
    @Environment(\.modelContext) var context
    @Query(sort: \PointDataToSave.dateOfSave) var query:
[PointDataToSave]

    @State var isShowingChoosingLayer: Bool = false
    @State var showDirections: Bool = false
    @State var isShowingSaved: Bool = false
    @State var coordinates = ""
}

```

```

    @State private var cameraPosition: MapCameraPosition =
        .userLocation(followsHeading: true, fallback: .automatic)
    @State var coordinateOfTap: CLLocationCoordinate2D = .init()
    @State var mapStyle: MapStyle = .standard

    @State var isInMeasuringMode: Bool = false

    var body: some View {
        NavigationStack {
            ZStack(alignment: .bottomTrailing) {
                MapReader { proxy in
                    Map(position: $cameraPosition) {
                        UserAnnotation()
                        if vm.showSheet || vm.isShowingRouteDirections ||
vm.isInRoutesMode {
                            Marker("", coordinate: coordinateOfTap)
                        }
                        if isInMeasuringMode {
                            ForEach(vm.locations) { location in
                                Marker("", coordinate: location.coordinate)
                            }
                        }

                        if let route = vm.route {
                            MapPolyline(route)
                                .stroke(.blue, lineWidth: 5)
                        }
                    }

                    .mapStyle(mapStyle)
                    .gesture(DragGesture())
                    .gesture(
                        LongPressGesture(minimumDuration: 0.5, maximumDistance:
0)
                            .sequenced(before: SpatialTapGesture(coordinateSpace:
.local))
                            .onEnded { value in
                                switch value {
                                    case let .second(_, tapValue):
                                        guard let point = tapValue?.location else {
                                            print("Unable to retrieve tap location from gesture
data.")
                                        }
                                        return
                                }
                            }
                    )
                }
            }
        }
    }

```

```

else {
    guard let coordinate = proxy.convert(point, from: .local)
    print("Unable to convert local point to coordinate on
map.")
    return
}

coordinateOfTap = coordinate
if !isInMeasuringMode {
    vm.didTapOnLocation(location: coordinate)
} else if isInMeasuringMode {
    let pin: Pin = .init(coordinate: coordinateOfTap)
    vm.locations.append(pin)
}

print("Long press occurred at: \(coordinate)")
default: return
}
}
)
.mapControls {
    MapCompass()
    MapScaleView()
    MapUserLocationButton()
}
.mapControlVisibility(.visible)
}

HStack(alignment: .bottom) {
    if vm.isInRoutesMode {
        VStack {
            Button {
                vm.dismissRoute()
            } label: {
                Image(systemName: "x.square").imageScale(.large)
            }
            .padding()
            .background(.thinMaterial)
            .cornerRadius(20)

            Button {
                vm.isShowingRouteDirections = true
            } label: {

```

```

        Image(systemName: "command").imageScale(.large)
    }
    .padding()
    .background(.thinMaterial)
    .cornerRadius(20)
}
.padding(.bottom)
}

Spacer()

VStack {
    Button {
        isShowingSaved = true
    } label: {
        Image(systemName: "magnifyingglass").imageScale(.large)
    }
    .padding()
    .background(.thinMaterial)
    .cornerRadius(20)

    Button(action: {
        isShowingChoosingLayer = true
    }, label: {
        VStack {
            Image(systemName:
"square.2.layers.3d.top.filled").imageScale(.large)
        }
    })
    .padding()
    .background(.thinMaterial)
    .cornerRadius(20)

    Button(action: {
        isInMeasuringMode.toggle()
    }, label: {
        Image(systemName: "ruler").imageScale(.large)
        .foregroundColor(isInMeasuringMode ? Color.purple :
Color.blue)
    })
    .padding()
    .background(.thinMaterial)
    .cornerRadius(20)
}

```

```

    }
    }.padding()
  }
  .sheet(isPresented: $vm.isShowingRullerResults, onDismiss: {
    isInMeasuringMode = false
    vm.locations = []
  }) {
    VStack {
      Text("Дистанція між обраними точками становить:")
      Text("\((vm.distanceBeetween.components(separatedBy: (",""))[0])
метрів")
    }.presentationDetents([.fraction(0.3)])
  }
  .sheet(isPresented: $vm.isShowingRouteDirections, onDismiss: {
    vm.isShowingRouteDirections = false
  }) {
    ScrollView(.horizontal) {
      HStack {
        ForEach(vm.routeDirections) { direction in
          VStack {
            Image(systemName:
direction.direction.instructions.contains("ліво") ? "arrow.backward.square" :
direction.direction.instructions.contains("право") ? "arrow.right.square" : "")
            Text(direction.direction.instructions)

Text(String(direction.direction.distance).components(separatedBy: (",""))[0]) +
Text("m")

          }
        }.frame(width: 120, height: 200)
      }
    }
  }
  }.presentationDragIndicator(.visible)
  }.presentationDetents([.fraction(0.3)])
}

.sheet(isPresented: $isShowingChoosingLayer) {
  VStack {
    Text("Оберіть шар")
    HStack(spacing: 20) {
      Button {

```

```

        mapStyle = .imagery
      } label: {
        VStack {
          Image(.imagery)
            .resizable()
            .scaledToFit()
            .cornerRadius(25)
          Text("Imagery")
        }
      }
    }.frame(width: 120, height: 120)
    .contentShape(.rect(cornerRadius: 20))

    Button {
      mapStyle = .hybrid
    } label: {
      VStack {
        Image(.hybrid)
          .resizable()
          .scaledToFit()
          .cornerRadius(25)
        Text("Hybrid")
      }
    }
    }.frame(width: 120, height: 120)
    .contentShape(.rect(cornerRadius: 20))

    Button {
      mapStyle = .standard
    } label: {
      VStack {
        Image(.standard)
          .resizable()
          .scaledToFit()
          .cornerRadius(25)
        Text("Standard")
      }
    }
    }.frame(width: 120, height: 120)
    .contentShape(.rect(cornerRadius: 20))
  }
  .presentationDetents([.fraction(0.3)])
  .presentationDragIndicator(.visible)
}

```

```

    }
    .sheet(isPresented: $isShowingSaved) {
        SavedLocationsView()
        .environmentObject(vm)
        .modelContainer(for: [PointDataToSave.self])
    }
    .sheet(isPresented: $vm.showSheet) {
        VStack {
            VStack(alignment: .leading) {
                Text("Вулиця: \\\(vm.geocodingReversedValue.name)")
                    .font(.title)
                Text("\\(vm.geocodingReversedValue.country),
\\(vm.geocodingReversedValue.locality),
\\(vm.geocodingReversedValue.admistrativeArea),
\\(vm.geocodingReversedValue.subAdministrativeArea)")
                    .font(.footnote)
                    .foregroundColor(Color.gray)
                    .presentationDetents([.fraction(0.35)])
                    .presentationDragIndicator(.visible)
                Text("Довгота: \\\(coordinateOfTap.longitude), Широта:
\\(coordinateOfTap.latitude)")
            }
            .padding(.bottom)
            HStack {
                Button(action: {
                    vm.didTapOnRoute(destination: coordinateOfTap)
                }, label: {
                    Text(Image(systemName: "square.and.arrow.up")) + Text("
") + Text("Navigate")
                })
                .padding(.horizontal)
                .buttonStyle(.bordered)
                Button(action: {
                    let location = PointDataToSave.init(descriptionOfPoint:
vm.geocodingReversedValue, longitude: coordinateOfTap.longitude, latitude:
coordinateOfTap.latitude)
                    context.insert(location)
                    print(query)
                }, label: {
                    Text(Image(systemName: "square.and.arrow.down")) +
Text(" ") + Text("Save")
                })
                .buttonStyle(.borderedProminent)
            }
        }
    }

```



```

@EnvironmentObject var vm: MainMapViewModel
@Query(sort: \PointDataToSave.dateOfSave) var locations:
[PointDataToSave]
@Environment(\.dismiss) var dismiss
var body: some View {
    ScrollView {
        ForEach(locations) { location in
            SavedLocationsCell(name: location)
                .onTapGesture {
                    dismiss()
                    vm.didTapOnLocation(location: location.getCLL)
                }
        }
    }.padding()
}

##Preview {
//  SavedLocationsView()
//}

struct SavedLocationsCell: View {
    let name: PointDataToSave
    var body: some View {
        VStack(alignment: .leading) {
            Text("Вулиця: \(name.descriptionOfPoint.name)")
                .font(.title)
            Text("\(name.descriptionOfPoint.country),
\(\name.descriptionOfPoint.locality),
\(\name.descriptionOfPoint.admistrativeArea),
\(\name.descriptionOfPoint.subAdministrativeArea)")
                .font(.footnote)
                .foregroundColor(Color.gray)
                .presentationDragIndicator(.visible)
            Text("Довгота: \(name.longitude), Широта: \(name.latitude)")
            Divider()
        }
    }
}

```

Лістинг TabBarCustom.swift

```

import Foundation
import SwiftUI

struct TabBarItem: View {

```

```

var name: String
var isActive: Bool = false
let namespace: Namespace.ID

var body: some View {
    if isActive {
        Text(name)
        // .applyStyle(font: .ptmono, size: 16)
        .padding(.horizontal)
        .padding(.vertical, 4)
        .foregroundColor(.white)
        .background(Capsule().foregroundColor(.blue))
        .matchedGeometryEffect(id: "highlightmenuitem", in: namespace)
    } else {
        Text(name)
        // .applyStyle(font: .ptmono, size: 16)
        .padding(.horizontal)
        .padding(.vertical, 4)
        .foregroundColor(.black)
    }
}

}

struct TabBarView: View {
    var tabItems: [String]
    @Binding var selectedIndex: Int
    @Namespace private var menuItemTransition
    var body: some View {
        ScrollViewReader { scrollView in
            ScrollView(.horizontal, showsIndicators: false) {
                HStack {
                    ForEach(tabItems.indices, id: \.self) { index in
                        TabBarItem(name: tabItems[index], isActive: selectedIndex ==
index, namespace: menuItemTransition)
                            .onTapGesture {
                                withAnimation(.easeInOut) {
                                    selectedIndex = index
                                }
                            }
                    }
                }
            }
        }
        .padding()
    }
}

```

```

        .background(.thinMaterial)
        .cornerRadius(25)
        .onChange(of: selectedIndex) {
            scrollView.scrollTo(selectedIndex, anchor: .center)
        }
    }
}
}

#Preview {
    TabBarView(tabItems: ["Home", "AI", "Couples", "Settings", "Not Setings",
"Etc."], selectedIndex: .constant(0))
}

```

```
import SwiftUI
```

```

struct MainTabView: View {
    // let store: StoreOf<TabBarFeature>
    @State var selectedIndex = 0
    var body: some View {
        ZStack(alignment: .bottom) {
            TabView(selection: $selectedIndex) {
                MainMapView()
                    .tag(0)
                    .frame(maxWidth: .infinity, maxHeight: .infinity)
                    .toolbarBackground(.hidden, for: .tabBar)
                    .modelContainer(for: [PointDataToSave.self])

                // SavedLocationsView()
                // .tag(1)
                // .frame(maxWidth: .infinity, maxHeight: .infinity)
                // .toolbarBackground(.hidden, for: .tabBar)
                // .modelContainer(for: [PointDataToSave.self])
            }

            .ignoresSafeArea()

            TabBarView(tabItems: ["Map", "Saved locations"], selectedIndex:
$selectedIndex)
                .padding(.horizontal)
        }
    }
}

```

```
}  
}
```

```
#Preview {  
    MainTabView()  
}
```

Лістинг Pin.swift

```
import Foundation  
import CoreLocation
```

```
struct Pin: Identifiable {  
    var coordinate: CLLocationCoordinate2D  
    let id = UUID()  
}
```

Лістинг PinInfoModel.swift

```
import Foundation
```

```
struct PinInfoModel: Codable, Comparable {  
    static func < (lhs: PinInfoModel, rhs: PinInfoModel) -> Bool {  
        return lhs.dateOfSave < rhs.dateOfSave  
    }  
    static func > (lhs: PinInfoModel, rhs: PinInfoModel) -> Bool {  
        return lhs.dateOfSave > rhs.dateOfSave  
    }  
    static func == (lhs: PinInfoModel, rhs: PinInfoModel) -> Bool {  
        return lhs.dateOfSave == rhs.dateOfSave  
    }  
}
```

```
let name: String  
let country: String  
let isoCountryCode: String  
let administrativeArea: String  
let subAdministrativeArea: String  
let locality: String  
let postalCode: String  
var dateOfSave: Date = Date.now
```

```
enum CodingKeys: String, CodingKey {  
    case name  
    case country  
    case isoCountryCode  
    case administrativeArea  
    case subAdministrativeArea
```

```
case locality
case postalCode
case dateOfSave
}
}
```