Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка мобільного додатку для пошуку найближчих укриттів на платформі Android»

Виконав: студент 4 курсу
групи 2ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Мисловський А.В.
(прізвище та ініціали)

Керівник: PhD, ст. вик. каф. ПЗ Стахов О.Я.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент: к.т.н., проф. каф. ОТ Захарченко С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Допущено до захисту

Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
(ініціали та прізвище)

«09» 06 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
ПЗ О. Н. Романюк
« 21 » березня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Мисловському Антону Вікторовичу

1. Тема роботи – «Розробка мобільного додатку для пошуку найближчих укриттів на платформі Android».

Керівник роботи: Стахов Олексій Ярославович, PHD, старший викладач кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025.

3. Тип програми – нативний мобільний застосунок для Android; Модель розробки – ітеративна, з поетапним розширенням функціональності; Засоби організації даних – база даних SQLite для локального зберігання інформації про укриття; Використані сервіси – Google Maps API для відображення карти, Google Routes API для побудови маршрутів; Вхідні дані – координати поточного місцезнаходження користувача; Вихідні дані – розташування найближчого укриття та маршрут до нього; Середовище розробки – Android Studio; Мова програмування – Java;

4. Зміст текстової частини: вступ; обґрунтування вибору методу розробки та постановка задач дослідження; розробка методів та алгоритмів роботи програмного продукту; розробка програмного забезпечення; тестування програмного застосунку; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична цінність одержаних результатів; порівняльний аналіз аналогів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Стахов О.Я. PHD, старший викладач кафедри ПЗ	24.03.25	04.06.2

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

Ч.ч.	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітк
1	Аналіз стану питання та постановка задач дослідження	25.03.25 – 29.03.25	Вик.
2	Розробка структури та моделі застосунку	30.03.25 – 05.04.25	Вик.
3	Розробка алгоритмів взаємодії з мапою та логіки визначення маршрутів	06.04.25 – 12.04.25	Вик.
4	Вибір технологій, бібліотек і сервісів для реалізації програмного забезпечення	13.04.25 – 15.04.25	Вик.
5	Аналіз та вибір засобів для реалізації програмного продукту	16.04.25 – 05.05.25	Вик.
6	Програмна реалізація	06.05.25 – 12.05.25	Вик.
7	Тестування програмного застосунку	13.05.25 – 19.05.25	Вик.
8	Оформлення матеріалів до захисту БДР	20.04.25 – 30.05.25	Вик.

Студент (підпис) Мисловський А.В (ініціали та прізвище)

Керівник бакалаврської кваліфікаційної роботи (підпис) Стахов О.Я (ініціали та прізвище)

Анотація

Курсовий проєкт присвячено розробці мобільного додатку для пошуку найближчих укриттів із використанням сервісів Google Maps SDK та Google Routes API.

У рамках роботи реалізовано модулі для інтеграції карти в мобільний додаток, відображення спеціальних маркерів укриттів, зміни стилів карти за допомогою кількох варіантів оформлення (через JSON-файли стилізації), пошуку найближчої точки укриття з використанням локальної бази даних SQLite та автоматичного прокладання маршруту через сервіси Google.

Інтерфейс додатку розроблений відповідно до принципів адаптивного дизайну, що забезпечує коректну роботу на різних розмірах екранів мобільних пристроїв.

Додаток написано мовою програмування Java із використанням середовища розробки Android Studio. Для завантаження даних про укриття застосовується локальна база даних SQLite, що забезпечує швидкий доступ до інформації без необхідності постійного інтернет-з'єднання. Взаємодія з мапою реалізована через Google Maps SDK, а побудова маршрутів — за допомогою Google Routes API, що забезпечує точне й динамічне оновлення маршрутної інформації.

Розроблений програмний продукт орієнтований на використання у цивільному захисті, надзвичайних ситуаціях та особистому користуванні, де важливе швидке орієнтування у просторі та знаходження найближчого прихистку.

Annotation

The course project is dedicated to the development of a mobile application for locating the nearest shelters using Google Maps API and Google Routes API services.

As part of the project, modules were implemented for integrating maps into the mobile application, displaying custom shelter markers, dynamically switching between several map styles (through JSON style files), searching for the nearest shelter using a local SQLite database, and automatically building a route via Google services.

The application's interface was developed according to adaptive design principles, ensuring proper functionality across various mobile device screen sizes.

The application was developed using the Java programming language within the Android Studio environment. Shelter data loading is handled through a local SQLite database, enabling fast access to information without the need for a constant internet connection. Interaction with the map is realized via the Google Maps SDK, and route generation is implemented using the Google Routes API, providing accurate and dynamic routing information.

The developed software product is intended for use in civil protection, emergency situations, and for personal use, where rapid orientation and finding the nearest safe place are critical.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження

У сучасних умовах зростання кількості надзвичайних ситуацій, зокрема пов’язаних із воєнними загрозами, питання швидкого та зручного доступу до інформації про найближчі укриття набуває критичної ваги. В умовах обмеженого часу на прийняття рішень забезпечення особистої безпеки населення стає пріоритетним завданням як для державних органів, так і для розробників цифрових інструментів.

Розвиток інформаційних технологій, зокрема мобільних платформ, геоінформаційних систем та сервісів навігації, створює передумови для впровадження ефективних цифрових рішень, які здатні допомогти громадянам зорієнтуватися в критичній ситуації. Важливим є не лише надання доступу до бази даних укриттів, а й можливість врахування геолокації користувача, зручного відображення інформації на карті, динамічного прокладання маршруту та адаптації інтерфейсу до індивідуальних потреб.

У цьому контексті розробка мобільного застосунку ViShelter є актуальною як з наукового, так і з практичного погляду. Застосунок має на меті забезпечити користувачів простим у використанні інструментом, що надає швидкий доступ до важливої інформації у стресових умовах. ViShelter орієнтований на широке коло користувачів: місцевих жителів, туристів, осіб з інших регіонів, які тимчасово перебувають у місті. Впровадження подібного рішення сприяє підвищенню рівня індивідуальної безпеки та готовності до дій у надзвичайних ситуаціях.

Зв’язок роботи з науковими програмами, планами, темами

Дослідження виконано відповідно до тематики наукових досліджень кафедри програмного забезпечення в рамках вивчення мобільних систем інформування населення та цифрових рішень у сфері безпеки.

Мета та завдання дослідження

Мета – підвищення швидкості орієнтації користувача в умовах

надзвичайних ситуацій за рахунок створення мобільного застосунку, що дозволяє оперативно знаходити найближчі укриття та прокладати маршрут до них з урахуванням поточної геолокації.

Завданнями дослідження є:

– провести аналіз існуючих аналогів мобільних застосунків у сфері безпеки;

– розробити архітектуру застосунку з модулем картографії та маршрутизації;

– реалізувати локальну базу укриттів з можливістю офлайн-доступу;

– створити алгоритм пошуку найближчого укриття за координатами;

– забезпечити побудову маршрутів за допомогою Google Maps API;

– розробити інтерфейс з підтримкою налаштувань стилю мапи;

– протестувати функціональність застосунку в умовах, наближених до реальних.

Об’єкт дослідження - Процес розробки мобільного застосунку для пошуку найближчих укриттів.

Предмет дослідження - Методи реалізації мобільних застосунків із використанням картографічних сервісів, геолокації та маршрутизації.

Методи дослідження

У процесі дослідження були використані такі методи:

- аналіз існуючих мобільних застосунків та технічних рішень;

- проєктування та реалізація інтерфейсу користувача (UI/UX);

- використання інструментів Android Studio, Google Maps SDK, Google Routes API, SQLite;

- функціональне тестування, тестування продуктивності та зручності використання застосунку.

Новизна отриманих результатів

- Вперше реалізовано можливість динамічної зміни стилів карти у мобільному застосунку пошуку укриттів, особливість якого полягає у врахуванні уподобань користувача, що дозволяє персоналізувати візуальне відображення

інформації та підвищити зручність взаємодії з додатком.

- Удосконалено метод побудови маршрутів, який, на відміну від існуючих, враховує динамічну зміну геолокації користувача всередині додатку, що дає можливість забезпечити актуальність прокладених маршрутів до укриттів у режимі реального часу та підвищити ефективність навігації.

Практична цінність отриманих результатів

Розроблений застосунок ViShelter може бути впроваджений як засіб підвищення особистої безпеки в умовах надзвичайних ситуацій. Він орієнтований на потреби мешканців міст, туристів і внутрішньо переміщених осіб. Застосунок забезпечує швидкий доступ до інформації про найближчі укриття та маршрути до них, що робить його корисним у практичному застосуванні та перспективним для подальших досліджень у сфері мобільних рішень безпеки.

Особистий внесок здобувача

Усі етапи розробки мобільного застосунку ViShelter – від аналізу предметної області до реалізації функціоналу та тестування – виконані автором самостійно. Результати, отримані в межах роботи, є авторськими та не використовувалися в інших проєктах.

Апробація матеріалів кваліфікаційної роботи

Апробація матеріалів бакалаврської кваліфікаційної роботи. Матеріали з дослідження доповідалися на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)».

Публікації

Результати дослідження були опубліковані в матеріалах конференції [1].

1 АНАЛІЗ РОЗВИТКУ ТЕХНОЛОГІЙ СТВОРЕННЯ МОБІЛЬНИХ ДОДАТКІВ ДЛЯ НАДАННЯ ІНФОРМАЦІЇ ПРО УКРИТТЯ ТА ПОСТАНОВКА ЗАДАЧ ДОСЛІДЖЕННЯ

1.1 Аналіз стану технологій створення мобільних додатків для пошуку укриттів

Сьогоднішній етап розвитку інформаційних технологій потребує впровадження новітніх підходів до гарантування безпеки населення в умовах надзвичайних ситуацій. Особливу роль у цьому процесі відіграють мобільні застосунки, що сприяють оперативній орієнтації користувачів у просторі під час повітряних тривог, евакуацій або інших кризових подій [2].

Традиційні джерела інформування, зокрема друковані карти чи оголошення, мають суттєві обмеження: вони не забезпечують актуальності інформації, не враховують поточне розташування користувача та можуть бути недоступними в момент загрози [3]. На цьому тлі мобільні застосунки, що поєднують геолокаційні технології, інтеграцію з онлайн-базами даних та динамічні картографічні сервіси, виступають важливим інструментом індивідуального захисту та швидкого прийняття рішень [4].

Широке впровадження таких рішень стало можливим завдяки стрімкому розвитку супутникових навігаційних систем, стабільному доступу до мобільного інтернету та зростанню продуктивності сучасних смартфонів [5]. Ці технологічні фактори забезпечують швидке визначення координат користувача, побудову оптимального маршруту до найближчого укриття та надання актуальної інформації в реальному часі.

Мобільні додатки мають істотні переваги над традиційними каналами сповіщення. Головною з них є можливість миттєвого доступу до персоналізованих даних, що адаптуються до конкретного місця перебування користувача. Це дозволяє скоротити час реагування в екстреній ситуації та знизити ризик помилкових рішень через використання недостовірної чи застарілої інформації.

Додатково, сучасні мобільні рішення забезпечують можливість інтеграції розширеного функціоналу: рекомендацій щодо поведінки під час надзвичайних подій, повідомлень про зміну рівня загрози, а також персоналізованих налаштувань вигляду мапи та стилю її відображення.

Отже, аналіз сучасного стану технологій у сфері мобільних застосунків для орієнтації під час надзвичайних ситуацій засвідчує високу актуальність і перспективність цього напряму. Використання передових картографічних сервісів, геолокаційних даних і відкритих API значно підвищує ефективність систем навігації та оповіщення. Для досягнення високої якості та практичної цінності таких застосунків доцільно враховувати найкращі практики розробки, тенденції ринку та реальні потреби користувачів.

1.2 Порівняльний аналіз аналогів

На сучасному етапі розвитку мобільних технологій зростає потреба у створенні зручних та надійних застосунків для орієнтування у надзвичайних ситуаціях. Тема пошуку найближчих укриттів стає дедалі актуальнішою, особливо в умовах загроз безпеці населення. У зв'язку з цим було проведено аналіз основних існуючих рішень на ринку, аби визначити їх переваги та недоліки порівняно із розроблюваним застосунком.

Одним із прикладів є застосунок «Укриття Кременчук» [6], який також працює на платформі Android [7]. Він використовує карту Google Maps для відображення укриттів та поточного місцезнаходження користувача. Додаток має функцію пошуку найближчого укриття, довідкові матеріали щодо користування програмою, а також інформацію про пункти незламності та джерела води. Дизайн додатку залишається досить простим і лаконічним.

Окрім основних функцій, застосунок забезпечує інтуїтивно зрозумілу навігацію та швидкий доступ до важливої інформації. Проте його можливості обмежуються лише базовим відображенням укриттів без розширених функцій прокладання маршрутів у реальному часі. Це створює простір для вдосконалення та розробки більш функціональних рішень.

Інтерфейс застосунку «Укриття Кременчук» наведено на рисунку 1.1.

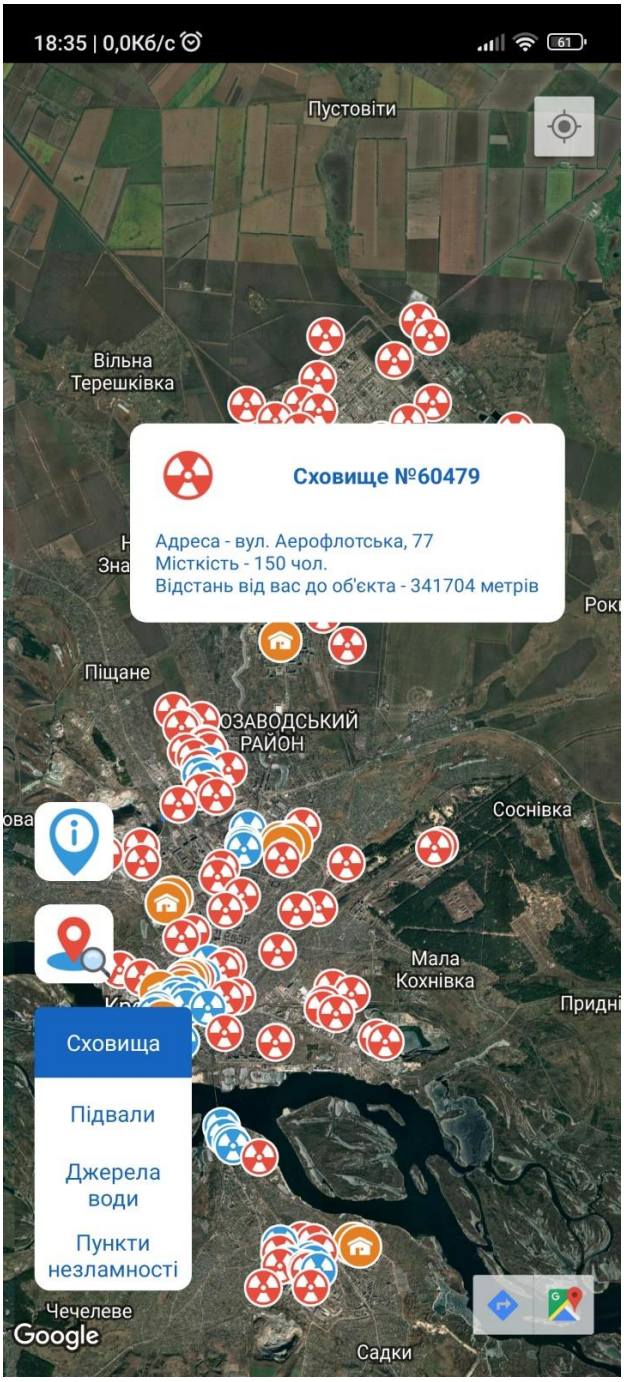

Рисунок 1.1 – Інтерфейс застосунку «Укриття Кременчук»

Іншим аналогом є застосунок «ДеУкриття» [8] для Android-платформи. Він орієнтований на відображення укриттів у місті Тернопіль, з можливістю фільтрації за типами укриттів, перегляду списку укриттів, а також прокладання маршруту до обраного об'єкта через інтеграцію з Google Maps. Користувачу

доступна базова інформація про обране укриття: тип, адреса, наявність туалету чи інтернету. Однак функція пошуку найближчого укриття в цьому застосунку відсутня, що обмежує його оперативність використання у критичних ситуаціях.

Інтерфейс програми «ДеУкриття» наведено на рисунку 1.2.

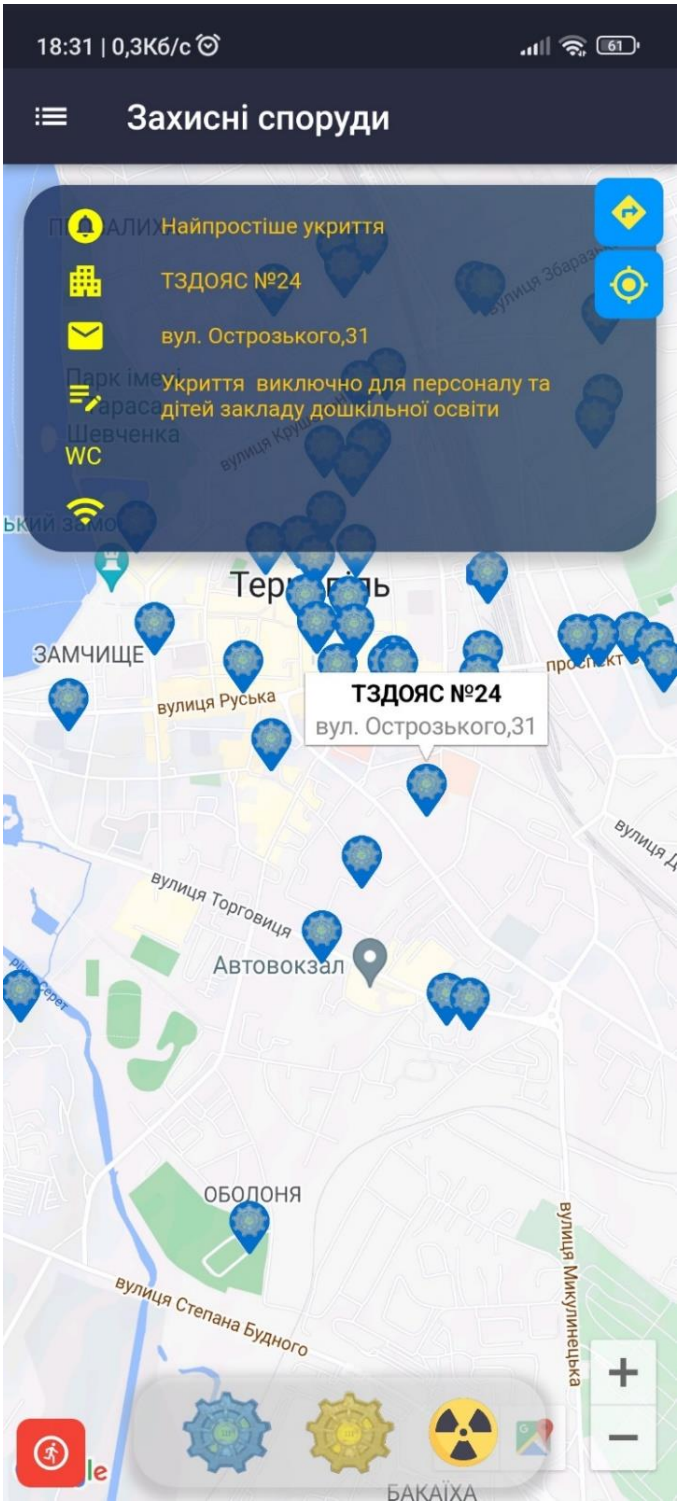

Рисунок 1.2 – Інтерфейс застосунку «ДеУкриття»

Ще одним прикладом є застосунок «Укриття Вінниця» [9], який також працює на платформі Android. Його розробниками є та сама команда, що зробила застосунок «Укриття Кременчук». Програма надає користувачам інформацію про розташування укриттів у місті Вінниця та дозволяє швидко знайти найближче до поточного місцезнаходження. Для відображення укриттів використовується інтеграція з картою Google Maps. Інтерфейс застосунку є простим і інтуїтивно зрозумілим, орієнтованим на оперативний доступ до необхідної інформації в надзвичайних ситуаціях. Розробниками є та ж сама

Інтерфейс програми «Укриття Вінниця» наведено на рисунку 1.3.

Рисунок 1.3 – Інтерфейс застосунку «Укриття Вінниця»

На основі проведеного аналізу було сформовано порівняльну характеристику застосунків-аналогів із розроблюваним продуктом. Результати порівняних аналогів зведено в табл. 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Додатки / Критерії	«Укриття Кременчук»	«ДеУкриття»	«Укриття Вінниця»	«ViShelter»
Місцезнаходження користувача	+	-	+	+
Довідка з користування	+	-	+	+
Прокладання маршруту	-	+	-	+
Вбудований пошук найближчого укриття	+	-	+	+
Довідка про поведінку під час екстрених ситуацій	-	-	-	+
Джерела води та пункти незламності	+	-	+	-
Вибір стилізації мапи	-	-	-	+

Таким чином, результати проведеного аналізу демонструють, що розроблюваний мобільний застосунок ViShelter має ряд суттєвих переваг над існуючими аналогами. Тому актуальною є розробка такого рішення, яке дозволить ефективніше задовольнити потреби користувачів у критичних умовах, і має високий потенціал для подальшого практичного застосування.

1.3 Аналіз методів розв’язання задачі

Проєктування мобільного застосунку для пошуку найближчих укриттів вимагає застосування комплексу технічних рішень, спрямованих на забезпечення точного позиціонування, високої швидкодії та зручності користувача. У межах реалізації було досліджено низку підходів до відображення об’єктів на карті, побудови маршрутів та організації локального збереження інформації про укриття.

Ключовим компонентом реалізації стала інтеграція з картографічною платформою. Для візуалізації мапи та розміщення маркерів укриттів було обрано сервіс Google Maps [10]. Цей інструмент надає широкі функціональні можливості: підтримку кількох режимів відображення (звичайна карта, супутник, гібрид), роботу з маркерами та їх інтерактивними елементами, а також забезпечує стабільність і високу продуктивність у мобільних середовищах.

Однією з найважливіших функцій є визначення найближчого укриття до поточного місцезнаходження користувача. Для реалізації цього механізму було розглянуто два підходи:

1. Використання зовнішніх геолокаційних сервісів, зокрема Google Places API;

2. Застосування локальної бази даних SQLite [11], яка містить координати укриттів.

Оскільки використання зовнішніх API передбачає постійне підключення до мережі, що може бути недоступним у критичних ситуаціях, було прийнято рішення на користь локального зберігання даних. Це дозволяє працювати в офлайн-режимі, мінімізувати затримки та підвищити швидкість доступу до інформації. База укриттів реалізована у вигляді локальної SQLite-бази даних, що дозволяє зберігати координати та іншу інформацію про об’єкти у структурованій формі.

Для побудови маршрутів до вибраного укриття використовується Google Routes API [12] – сервіс, що дозволяє генерувати оптимальні маршрути з урахуванням обраного способу пересування (пішки, автомобілем тощо).

Результати маршрутизації вбудовуються у відображення карти, завдяки чому користувач отримує наочну візуалізацію напряму руху в реальному часі.

Обробка великого обсягу даних укриттів (наприклад, після імпорту з KMZ-файлів [13]) потребує ефективних механізмів оптимізації. Для цього використовується попереднє структурування інформації у базі та індексація координат. Це забезпечує можливість швидкого пошуку найближчих об’єктів за алгоритмами аналізу відстані, зокрема із використанням евклідової метрики.

У результаті проведеного аналізу було обґрунтовано доцільність застосування такої комбінації методів:

- Google Maps API – для інтерактивного відображення карти та маркерів укриттів;
- Google Routes API – для побудови маршрутів з урахуванням типу переміщення;
- SQLite – для локального зберігання бази укриттів і забезпечення роботи в офлайн-режимі;
- Алгоритми визначення найближчих об’єктів – для швидкого пошуку укриттів за координатами користувача.

Застосування зазначених технологій дозволяє створити надійний мобільний додаток, здатний ефективно функціонувати навіть за обмеженого або відсутнього доступу до Інтернету, що критично важливо в умовах надзвичайних ситуацій.

1.4 Постановка задач курсового проєкту

На основі аналізу існуючих рішень у сфері навігаційних мобільних застосунків, а також виявлених їхніх обмежень щодо пошуку безпечних укриттів у надзвичайних ситуаціях, були сформульовані основні завдання, необхідні для успішної реалізації курсового проєкту:

- розробити мобільний застосунок для Android, який дозволяє користувачу переглядати карту місцевості з нанесеними укриттями;

- реалізувати механізм визначення поточного місцезнаходження

користувача за допомогою служб геолокації;

- створити алгоритм пошуку найближчого укриття відповідно до місцезнаходження користувача;

- інтегрувати Google Maps API для відображення карти та маркерів укриттів;

- використовувати Google Routes API для побудови оптимального маршруту до вибраного укриття;

- розробити простий та інтуїтивно зрозумілий графічний інтерфейс користувача, зручний для використання в екстрених умовах;

- забезпечити мінімізацію витрат трафіку та оптимізацію запитів до серверів під час роботи застосунку;

- провести комплексне тестування програмного продукту на відповідність функціональним та якісним вимогам.

1.5 Висновки

Проведений аналіз існуючих мобільних рішень для пошуку укриттів дозволив визначити основні функціональні можливості, які відповідають потребам користувачів в умовах надзвичайних ситуацій. Було встановлено сильні сторони аналогів, такі як використання геолокаційних сервісів та інтеграція з картографічними платформами, а також виявлено їхні обмеження – недостатня інтеграція пошуку найближчих об'єктів, відсутність персоналізованої інформаційної підтримки та обмежені можливості налаштування інтерфейсу.

На основі отриманих результатів визначено доцільність розробки нового мобільного застосунку, орієнтованого на розширення функціональних можливостей, підвищення зручності використання та забезпечення автономності роботи за рахунок локального зберігання даних.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ ПРОГРАМНОГО ПРОДУКТУ

2.1 Аналіз вхідних даних системи

Для реалізації мобільного додатку для пошуку найближчих укриттів буде використано мову програмування Java для Android-розробки, а також сервіси Google Maps API та SQLite для локального збереження даних.

Java — це об'єктно-орієнтована мова програмування, яка історично є основною для розробки Android-додатків. Вона широко використовується завдяки своїй стабільності, великій спільноті розробників та багатому набору бібліотек і фреймворків. Незважаючи на появу новіших мов, таких як Kotlin, Java залишається одним із найпопулярніших виборів для Android-розробки [14, 15].

Google Maps API надає потужні можливості для інтеграції картографічних сервісів у мобільні додатки. З його допомогою можна відображати карти, визначати місцезнаходження користувача, додавати маркери та будувати маршрути. Це дозволяє створювати навігаційні функції, які є критично важливими для додатків, що допомагають знаходити найближчі укриття [16, 17].

SQLite — це легка вбудована база даних, яка дозволяє зберігати структуровані дані локально на пристрої. Вона ідеально підходить для мобільних додатків, оскільки не вимагає окремого сервера та забезпечує швидкий доступ до даних без необхідності постійного підключення до Інтернету. Android надає вбудовану підтримку SQLite через пакет android.database.sqlite, що спрощує її використання в додатках [18].

Розробка мобільного додатку передбачає реалізацію кількох основних модулів. Першим етапом стане створення інтерфейсу користувача, який забезпечить зручну навігацію та доступ до основного функціоналу системи, зокрема перегляд карти, пошук найближчих укриттів та отримання інформації про них.

Наступним кроком буде розробка модуля для імпорту та обробки даних про укриття. Цей модуль здійснюватиме збереження координат, опису та іншої необхідної інформації у локальній базі даних SQLite, а також оновлення даних за необхідності.

Далі планується реалізувати модуль для пошуку найближчих укриттів відносно поточного розташування користувача, використовуючи геолокаційні сервіси та алгоритми обчислення відстані між точками на карті.

Також буде створений модуль для відображення результатів на карті за допомогою маркерів, побудови маршрутів до обраного укриття та візуалізації детальної інформації про кожен об'єкт.

Розробка вищезазначених модулів вимагає знань у сфері Android-програмування, роботи з базами даних, картографічними сервісами та геолокаційними технологіями. Для реалізації проекту будуть використані такі інструменти, як Android Studio, Google Maps SDK for Android, а також бібліотеки для роботи із базами даних на Android.

Таким чином, розробка мобільного додатку для пошуку укриттів є комплексним завданням, що передбачає інтеграцію кількох технологій та модулів. Завдяки правильній архітектурі, ретельному проєктуванню та тестуванню, можливо створити надійний і зручний програмний продукт, який ефективно допомагатиме користувачам орієнтуватися у надзвичайних ситуаціях.

2.2 Розробка інтерфейсу програмного додатку

Під час розробки інтерфейсу мобільного додатку було приділено особливу увагу його простоті, інтуїтивності та зручності для користувача. Основним кольором інтерфейсу обрано синій, що асоціюється із безпекою та надійністю. Допоміжними кольорами використовуються білий та сірий для створення візуального контрасту і кращої читабельності.

Після запуску додатку користувача одразу зустрічає екран з мапою та кнопками для взаємодії з мапою (див. рисунок 2.1), ніякого екрану завантаження

не було імплементовано через швидке завантаження мапи при мінімальному інтернет-підключенні.

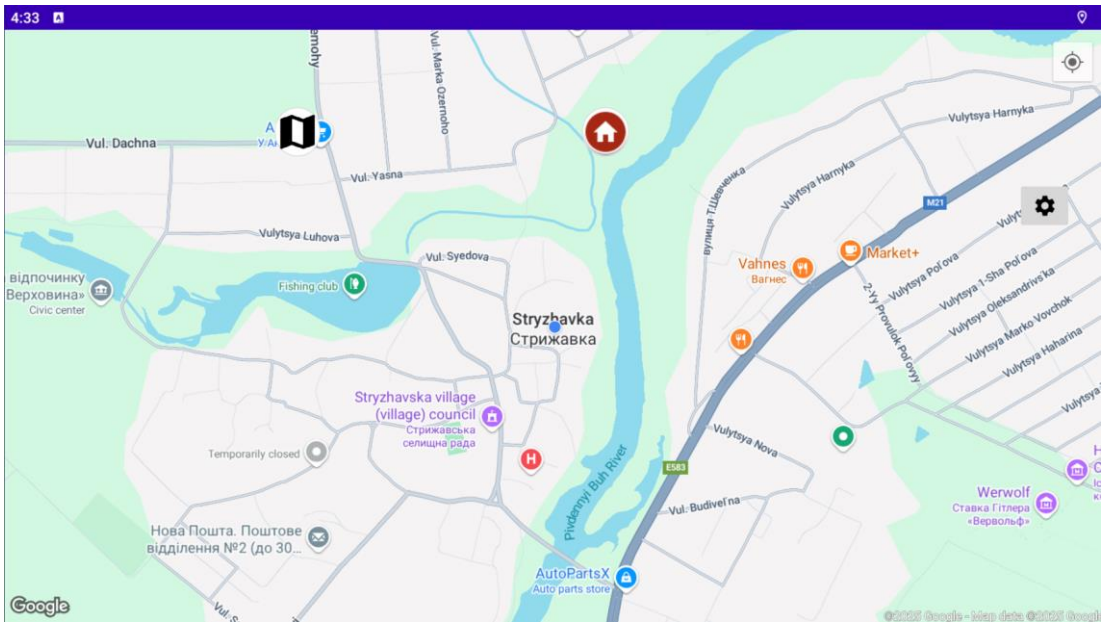

Рисунок 2.1 – Вигляд програми

Інтерфейс додатку передбачає основний робочий екран із картою Google Maps, на якій відображаються місця розташування укриттів, а також панель навігації для доступу до основних функцій системи.

Меню переключення режимів пошуку наведено на рисунку 2.2

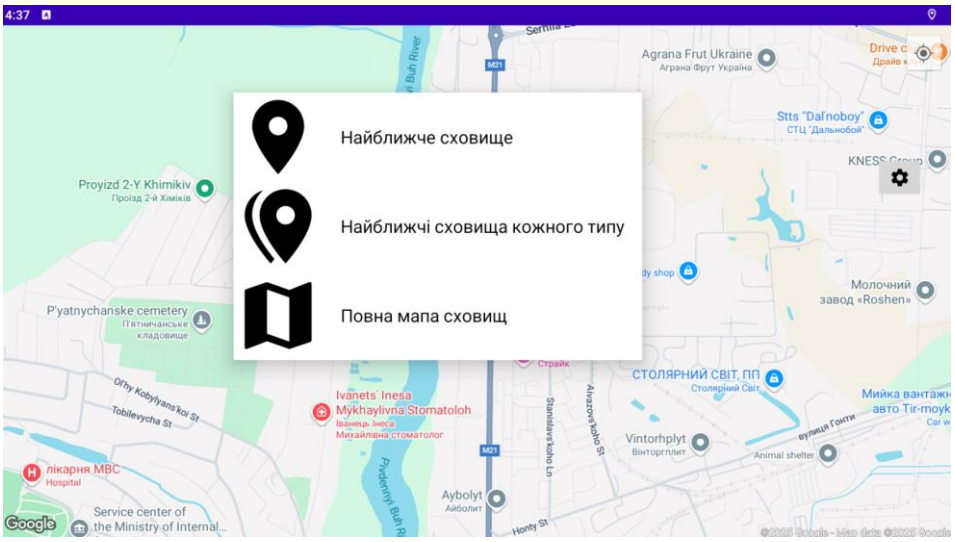

Рисунок 2.2 – Основне меню додатку

На початковому екрані додатку відображається Вінницька область на карті та доступне найближче укриття у вигляді маркера (див. рисунок 2.3). Це забезпечує швидкий доступ до важливої інформації у надзвичайних ситуаціях, не чекаючи завантаження геолокації користувача для наведення мапи на його координати, після встановлення геолокації користувача можна натиснути на інтегровану кнопку в інтерфейсі для наведення на користувача на мапі.

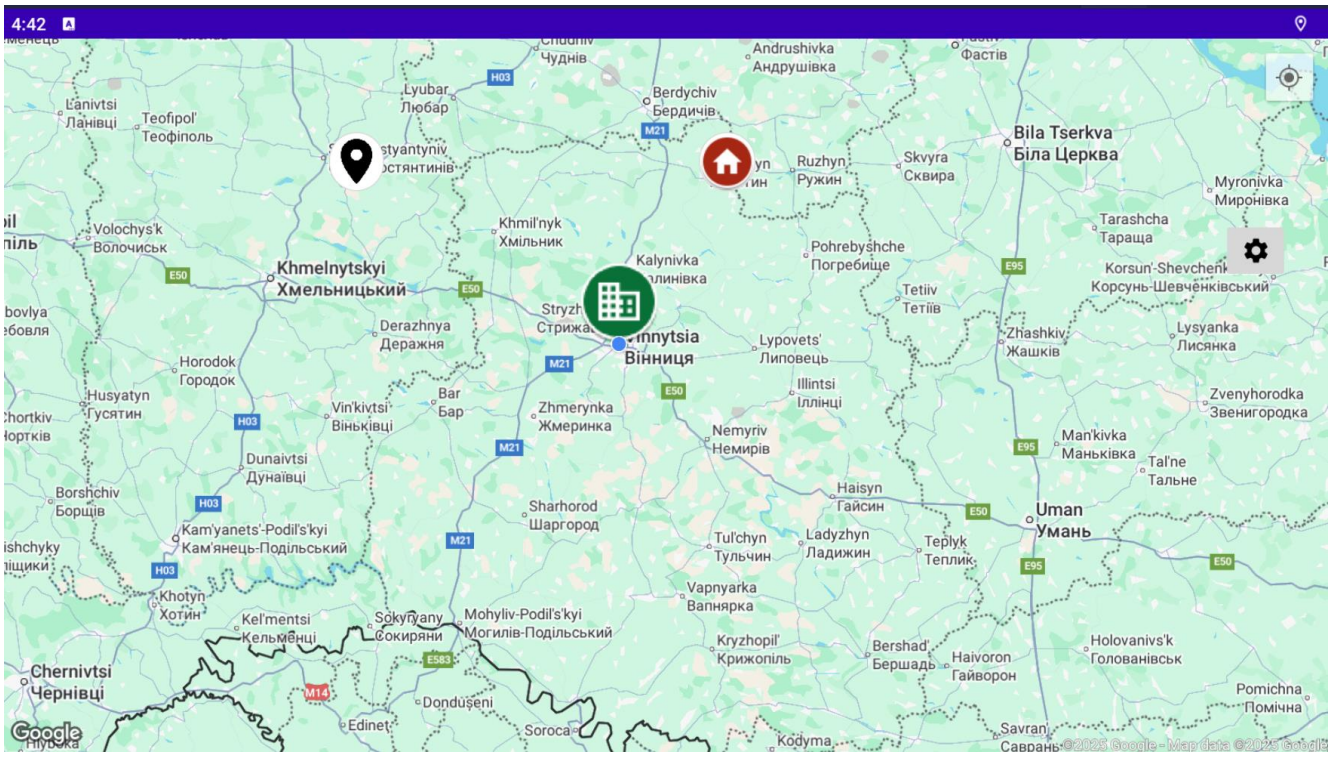

Рисунок 2.3 – Початковий вигляд робочого екрану

Після натискання на маркер укриття користувач може переглянути детальну інформацію про нього: адресу, доступність, а також за потреби прокласти маршрут до обраного об'єкта.

Приклад виведення інформації про сховище при натисканні наведено на рисунку 2.4.

Такий підхід дозволяє швидко отримати необхідні дані без зайвих дій, що є критично важливим у стресових ситуаціях. Однак функціонал прокладання маршрутів реалізовано досить базово, без урахування змін трафіку чи оновлення

шляху в режимі реального часу. Це обмежує зручність користування у динамічних умовах.

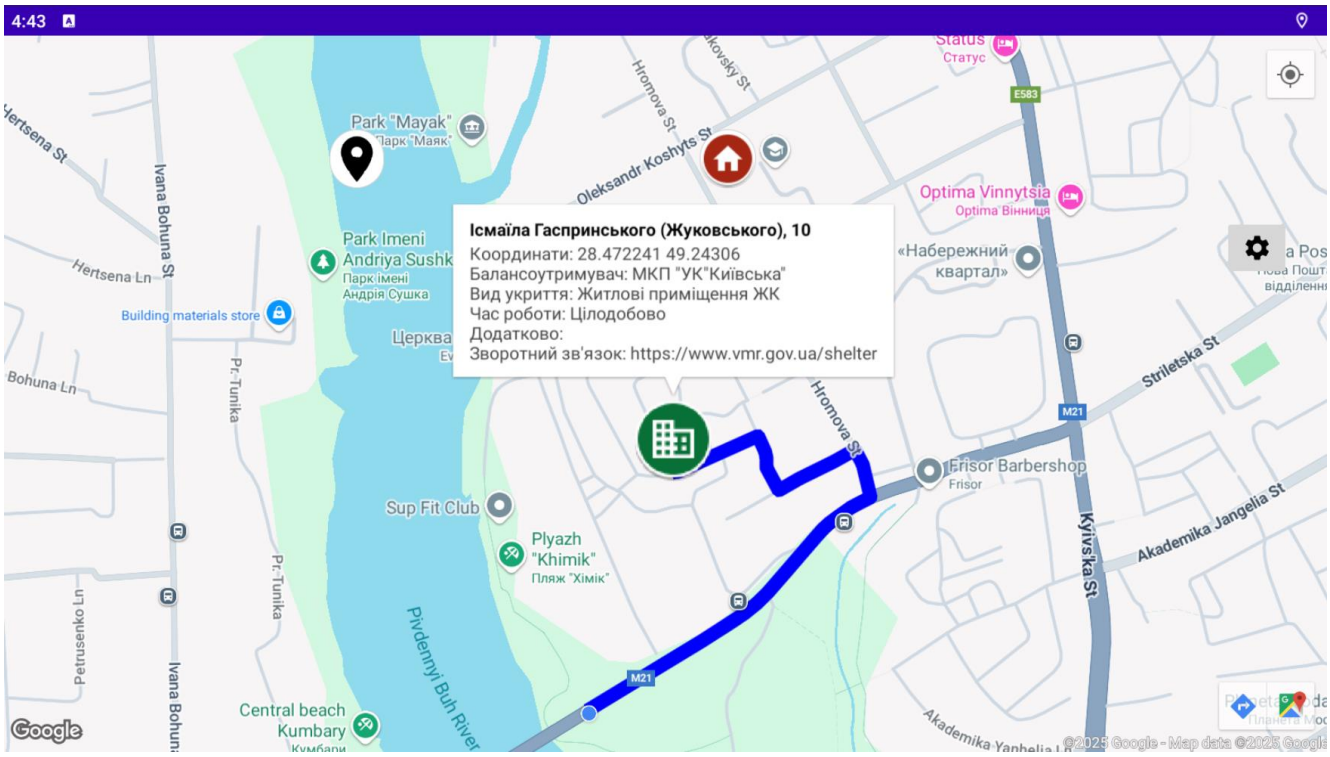

Рисунок 2.4 – Відображення інформації про укриття

Окрім основного функціоналу, додаток має розділ «Довідка» всередині налаштувань (див. рисунок 2.5), де користувач може ознайомитися з інструкціями щодо використання застосунку та отримати відповіді на найпоширеніші запитання.

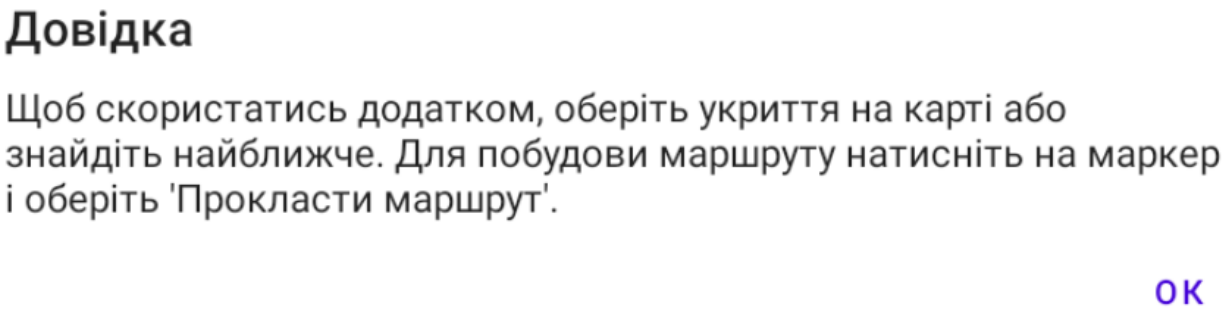

Рисунок 2.5 – Екран «Довідка»

Також реалізовано екран «Про програму» (див. рисунок 2.6), де розміщена загальна інформація про додаток, його призначення, контактні дані розробника та відомості про версію застосунку.

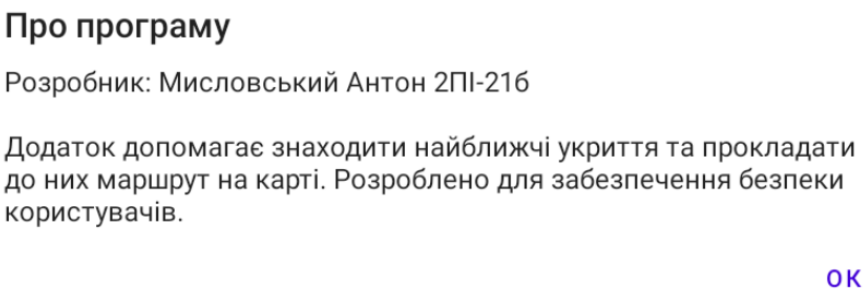

Рисунок 2.6 – Екран «Про програму»

Розробка графічного інтерфейсу мобільного додатку була здійснена у середовищі Android Studio з використанням мови програмування Java та XML для розробки макетів інтерфейсу.

2.3 Розробка моделі системи

Для кращого розуміння роботи основних функціональних режимів додатку було розроблено UML діаграму діяльності (див. рисунок 2.7). Згідно з цією діаграмою, користувач починає взаємодію з додатком через виведення найближчого сховища. Далі настає вибір — якщо користувач хоче змінити стиль відображення сховищ, він може це зробити. Якщо ж він вирішує залишити поточний стиль, наступним кроком буде пропонований маршрут, який користувач може прокласти, натискаючи на обране сховище на карті. У разі, якщо користувач хоче змінити режим пошуку, він може перейти до опції пошуку інших сховищ, що дозволяє вивести найближче сховище кожного типу.

Також передбачено кілька варіантів для випадків нестабільного з'єднання чи відсутності доступу до геолокації. У таких випадках додаток виводить повну карту сховищ, де користувач може вручну вибирати потрібне сховище для прокладання маршруту. Ці режими дають можливість вибору найближчого

сховища або перегляду усіх сховищ на карті з подальшим прокладанням маршруту.

Наприкінці процесу роботи з додатком є можливість перейти до налаштувань, де користувач може ознайомитись з довідкою, а також переглянути основну інформацію про програму, її призначення та доступні функції. Така організація режимів роботи додатку дозволяє користувачеві зручним і інтуїтивно зрозумілим способом здійснювати пошук та навігацію, а також отримувати детальну інформацію про поточний стан сховищ та маршрути.

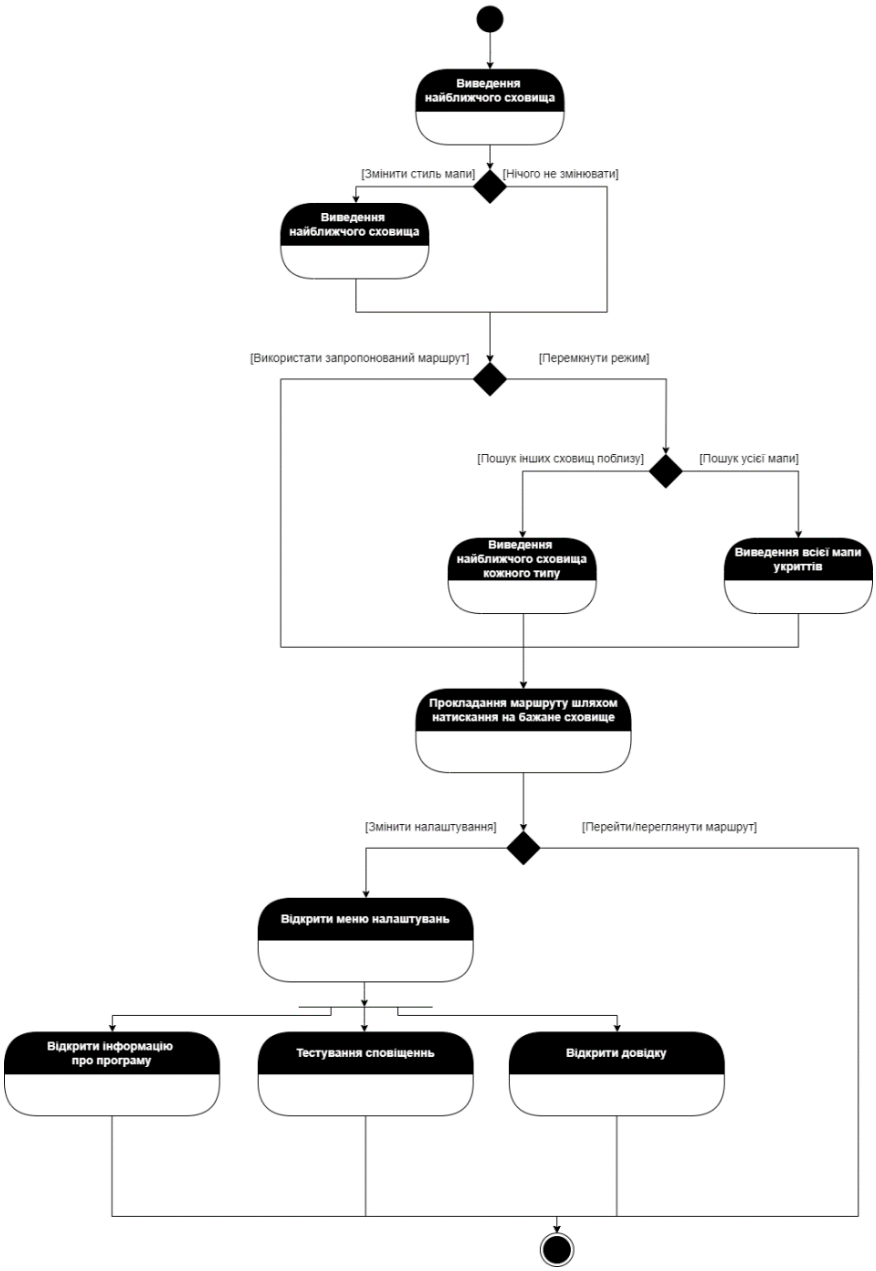

Рисунок 2.7 – Модель роботи системи у вигляді діаграми діяльності

Розробка діаграми діяльності була виконана у програмному забезпеченні Draw.io [19].

2.4 Розробка алгоритмів роботи системи

Для розробки програмного застосунку, який здійснює пошук найближчих сховищ на карті та забезпечує прокладання маршруту до них, необхідно розробити загальний алгоритм, згідно з яким буде працювати цей модуль (див. рисунок 2.8). Важливою складовою є зручний і ефективний процес пошуку сховищ, який враховує поточні умови і можливості користувача.

Згідно цього алгоритму, на початку роботи додатку користувачу необхідно дозволити доступ до геолокації. Після отримання геолокаційних даних або вводу координат, програмний застосунок здійснює пошук найближчого сховища на основі розташування користувача. У разі успішного знаходження сховища, у мапі додатку виводиться найближчий маркер сховища. Користувач може обрати відображення найближчого сховища конкретного типу або переглянути всі доступні сховища на мапі. Додаток також має можливість відображати додаткову інформацію про сховище, таку як його тип, адреса або доступні ресурси. У разі, коли геолокація недоступна або є проблеми зі з'єднанням, додаток можна використовувати для відображення всієї карти сховищ, що дозволяє користувачеві вільно вибрати будь-яке сховище на мапі без прив'язки до поточного місцезнаходження.

Наступним етапом є можливість користувача вибрати конкретне сховище для прокладання маршруту. Програмний застосунок надає користувачу можливість прокласти маршрут до вибраного сховища шляхом натискання на нього на карті. Після цього система використовує алгоритм маршрутизації, щоб обчислити найкоротший шлях до сховища з урахуванням поточних умов на маршруті, таких як обмеження по дорогах або пішохідні доріжки. Якщо користувач бажає змінити стиль відображення мапи, програма надає таку можливість через меню налаштувань. За замовчуванням додаток використовує стандартний стиль відображення мапи та маршруту, але користувач може обрати

інші варіанти візуалізації, наприклад, зміни кольорів маршрутів або позначень сховищ, для зручності сприйняття.

Для кращого розуміння алгоритмів роботи з картами та покращення ефективності пошуку, у додатку реалізовано алгоритм, який відповідає за пошук найближчого сховища на основі геолокації. Спочатку додаток здійснює завантаження карти і перевірку доступу до геолокації. У разі наявності доступу, система отримує координати користувача, а потім виконується пошук найближчого сховища за допомогою алгоритму обчислення відстані між точками на карті. Алгоритм враховує відстань до кожного сховища і надає найближчий результат. Якщо доступ до геолокації відсутній або з'єднання нестабільне, система все одно дозволяє відобразити повну карту всіх сховищ, щоб користувач міг обрати сховище без прив'язки до географічної прив'язки.

Після знаходження найближчого сховища або всіх доступних сховищ, користувач може вибрати бажане сховище та прокласти до нього маршрут. Для цього використовується алгоритм маршрутизації, який обчислює оптимальний шлях з урахуванням доступних шляхів і місць. Алгоритм маршрутизації може використовувати різні способи обчислення маршруту, наприклад, через пішохідні доріжки, або ж оптимізувати маршрут з точки зору часу або безпеки. Після цього програма надає користувачеві відображення маршруту на карті, що дозволяє змінювати масштаб, кут огляду та інші параметри візуалізації для покращення навігації. Також, користувач може мати можливість переглядати деталі маршруту, такі як довжина, час, або інші важливі дані.

Розроблений алгоритм також включає можливість перемикання між режимами пошуку різних типів сховищ, таких як найближчі сховища певного типу або загальний перегляд всіх сховищ. Крім того, додаток дозволяє користувачам інтерактивно взаємодіяти з іншими елементами карти через інтерфейс додатку. Наприклад, користувач може вибирати сховища для прокладання маршруту, змінювати вид мапи або переглядати додаткові дані про сховища, що відображаються на карті.

Завдяки цьому алгоритму, додаток надає користувачу гнучкий і зручний інтерфейс для пошуку сховищ, прокладання маршрутів та взаємодії з картами, що підвищує ефективність пошуку та допомагає забезпечити безпечний та зручний доступ до сховищ.

Блок-схема алгоритму роботи застосунку наведена на рисунку 2.8.

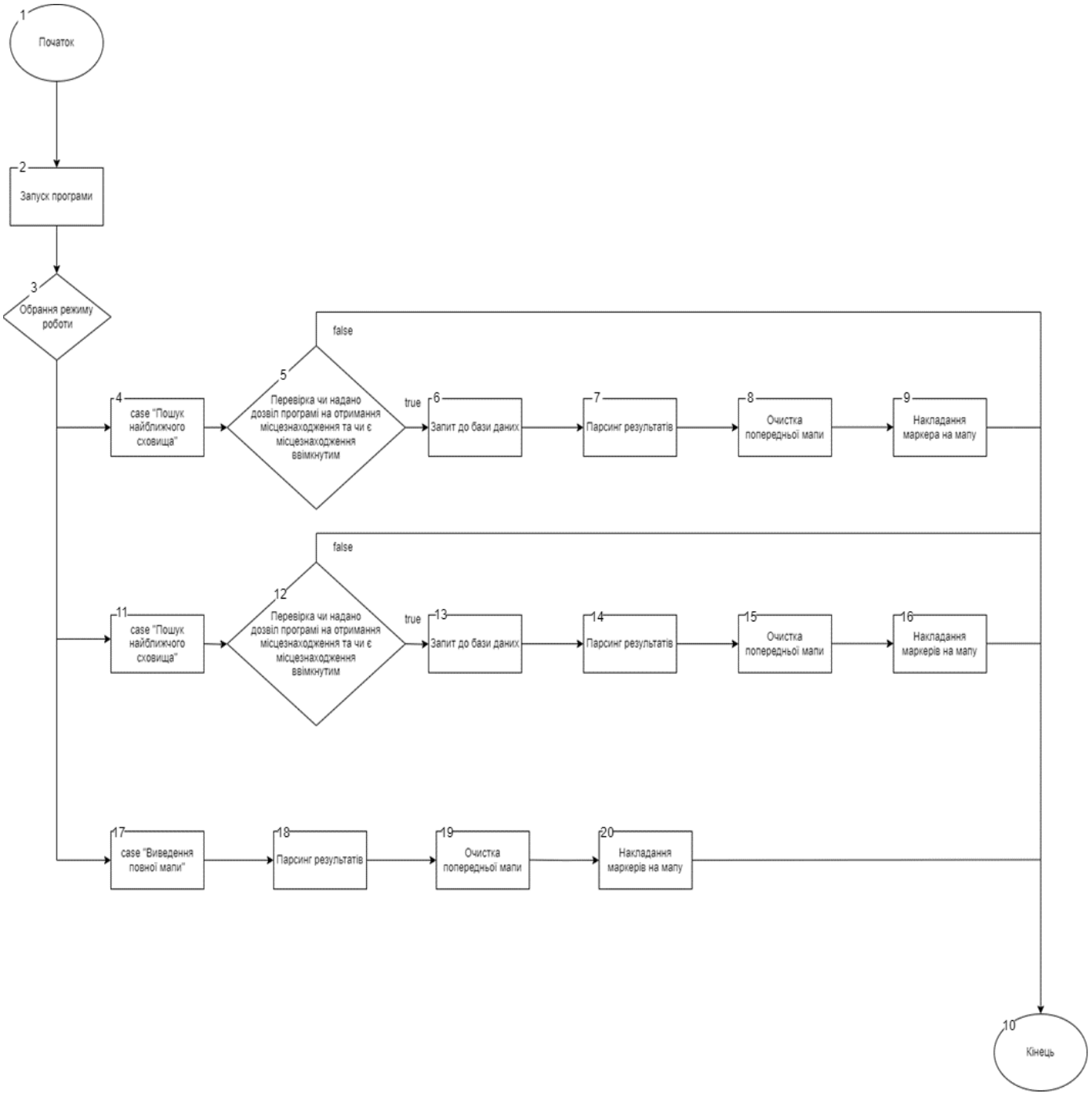

Рисунок 2.8 – Алгоритм роботи програмного застосунку

Завершенням взаємодії з додатком є можливість перейти до налаштувань, де користувач може ознайомитись із довідкою або інформацією про програму. Також у налаштуваннях передбачено зміну стилю відображення карти та інших параметрів для підвищення зручності користування додатком.

Особливу увагу в роботі системи приділено механізму прокладання маршруту. Після вибору бажаного сховища, додаток формує спеціальний URL-запит до зовнішнього маршрутизатора за допомогою сервісу Routes API. Цей запит містить координати поточного місця розташування користувача та обраного сховища. Routes API обробляє запит і повертає оптимальний маршрут, враховуючи реальні умови на дорогах, дозволені напрямки руху, наявність пішохідних маршрутів тощо. Отримана відповідь містить набір координат для побудови лінії маршруту на мапі, а також супутню інформацію — приблизну тривалість пересування, довжину маршруту та можливі попередження. Програмний застосунок обробляє ці дані та відображає маршрут на карті в режимі реального часу, забезпечуючи точне і зручне візуальне супроводження користувача до обраного укриття.

Таким чином, розроблений алгоритм забезпечує користувачу гнучкість у виборі режимів взаємодії з додатком, дозволяючи не лише ефективно знаходити найближче укриття та прокладати маршрути, але й адаптувати відображення карти відповідно до своїх потреб і умов навколишнього середовища.

Діаграма алгоритму визначення точок між користувачем та маркером наведена на рисунку 2.9.

Для підвищення надійності роботи додатку реалізовано механізм обробки можливих помилок під час побудови маршруту. Крім того, для оптимізації навігації передбачено можливість автоматичного оновлення маршруту у разі зміни поточного місцезнаходження користувача під час руху. Окрему увагу також приділено адаптивності інтерфейсу: застосунок коректно відображається на різних розмірах екранів, що дозволяє використовувати його як на смартфонах, так і на планшетах. Реалізовані функціональні можливості спрямовані на те, щоб

забезпечити максимальну швидкість доступу до укриттів, підвищуючи рівень безпеки користувачів у надзвичайних ситуаціях.

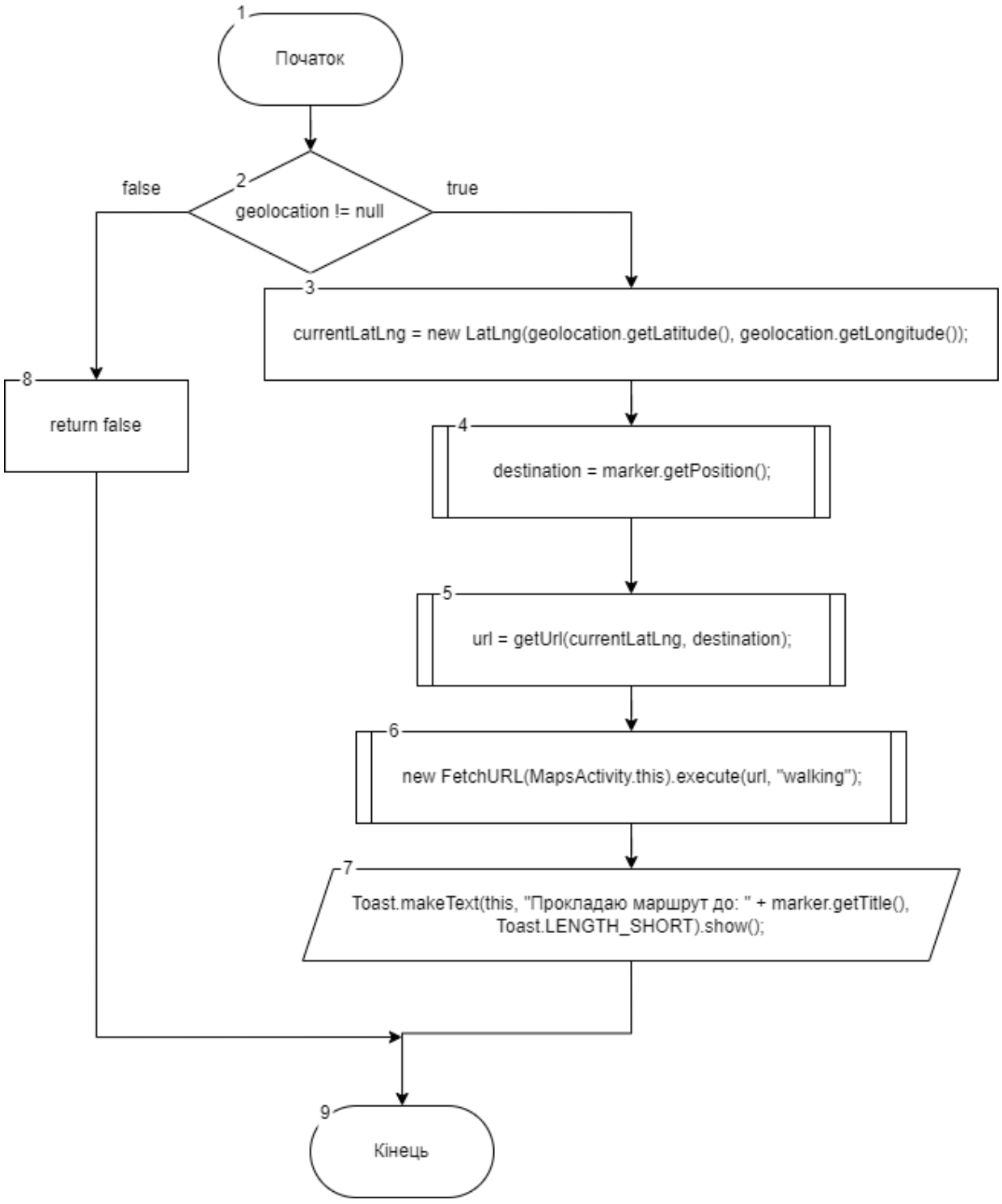

Рисунок 2.9 – Алгоритм визначення маршруту між користувачем та маркером

2.5 Висновки

У другому розділі було здійснено детальний аналіз вхідних і вихідних даних, що використовуються у процесі функціонування мобільного застосунку, призначеного для пошуку найближчих укриттів та побудови маршрутів до них. Особливу увагу приділено обробці геолокаційної інформації користувача, а також взаємодії із зовнішніми сервісами, що надають відомості про розташування захисних споруд.

Було сформовано базову структуру інтерфейсу користувача, яка забезпечує інтуїтивну навігацію та доступ до ключових функцій додатку: перегляд карти, пошук укриттів, прокладання маршрутів і зміна стилю відображення мапи.

Також розроблено основний алгоритм роботи застосунку, що описує послідовність дій користувача від моменту запуску програми до отримання маршруту або зміни параметрів карти. Окремо розглянуто алгоритм маршрутизації, який дозволяє визначити оптимальний шлях до вибраного укриття з урахуванням поточної геопозиції користувача за допомогою сервісу Route API.

Для наочного представлення внутрішніх процесів функціонування системи створено UML-діаграму діяльності. Вона відображає основні етапи виконання операцій, альтернативні сценарії використання та взаємозв’язки між структурними компонентами застосунку.

Здійснена робота дала змогу сформувати цілісне уявлення про функціональність розроблюваного програмного продукту та заклала основу для подальшої реалізації програмної частини.

3. РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ РЕАЛІЗАЦІЇ ПОШУКУ УКРИТТІВ

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

У процесі створення програмного забезпечення для відображення найближчих укриттів на мапі та побудови маршрутів до них особливу увагу було приділено ретельному вибору відповідних інструментів і технологій. Від обґрунтованості цих рішень залежала не лише якість кінцевого продукту, а й зручність його експлуатації для користувачів.

Як основне середовище для реалізації проєкту було обрано Android Studio [20] — одну з найпотужніших платформ для розробки мобільних застосунків під операційну систему Android [7]. Цей інструмент надає широкі можливості для інтеграції з Google-сервісами, зокрема з Google Maps API [10], що дало змогу ефективно реалізувати відображення мапи, розміщення маркерів укриттів, обробку взаємодій користувача з картою та побудову маршрутів через використання сервісів маршрутного планування.

Для реалізації допоміжних процесів були використані додаткові інструменти, що суттєво спростили певні етапи розробки. Зокрема, середовище PyCharm [21], орієнтоване на мову Python [22], застосовувалося для створення невеликих скриптів автоматизації. За допомогою цих скриптів було здійснено конвертацію даних з формату .kmz у структуру бази SQLite [11], що забезпечило зручне зберігання та обробку інформації про сховища.

Також важливу роль відіграла утиліта Google Map Style Wizard [23], яка дала змогу гнучко налаштувати зовнішній вигляд мапи відповідно до візуальних вимог проєкту. Створені за допомогою цього сервісу JSON-файли стилів редагувалися та перевірялися у середовищі Visual Studio Code [24], що дозволило швидко вносити необхідні зміни без ризику порушення структури конфігурації.

Центральним елементом технічної реалізації стала інтеграція з Google Maps API, яка охоплювала відображення поточного місцезнаходження

користувача, пошук найближчих укриттів за координатами та побудову маршрутів за допомогою Directions API. Система генерувала відповідні URL-запити, обробляла відповіді у форматі JSON та візуалізувала маршрути безпосередньо на мапі. Це вимагало узгодженої взаємодії між різними логічними модулями застосунку.

Варто зазначити, що навіть обмежене використання таких допоміжних інструментів, як PyCharm і Visual Studio Code, суттєво оптимізувало процес розробки. Це дало змогу зосередити основні зусилля на створенні якісної функціональної частини проєкту без зайвих витрат часу на технічну рутину. Такий підхід підтверджує, що вдало підібрані інструменти важливі не лише для написання основного коду, а й для реалізації супутніх операцій.

Отже, завдяки поєднанню Android Studio як основної платформи розробки з додатковими інструментами обробки даних і налаштування мапи вдалося створити стабільний, функціональний і користувацько-дружній застосунок. Вибрані технології забезпечили ефективну інтеграцію з сервісами Google, адаптацію до різних умов використання та високу зручність у щоденній експлуатації. Результатом стала повноцінна система, що поєднує простоту використання, надійність та оперативність у ситуаціях, пов’язаних із загрозами безпеці.

3.2 Розробка модуля визначення найближчого укриття у мобільному додатку

Під час реалізації модуля визначення найближчого укриття у складі мобільного додатку необхідно було врахувати низку технічних аспектів, пов’язаних із точністю розрахунків, ефективністю вибору та інтеграцією з іншими компонентами системи. Основна мета цього модуля полягає у визначенні найбільш зручного та безпечного маршруту до найближчого укриття, що особливо актуально в ситуаціях, коли користувачу потрібно швидко ухвалити рішення.

На першому етапі було розроблено структуру локальної бази даних SQLite, у якій зберігається вся необхідна інформація про доступні укриття. До основних полів цієї бази входять координати об’єкта (широта і довгота), його назва, короткий опис, а також поле styleUrl, яке використовується для ідентифікації типу укриття та подальшої візуальної стилізації відповідного маркера на мапі. Це дозволяє створити гнучкий механізм відображення укриттів у залежності від їх призначення чи характеристик.

Така структура даних забезпечує ефективну роботу з об’єктами під час побудови маршруту та покращує зручність навігації. На рисунку 3.1 наведено структуру бази даних SQLite з таблицею укриттів, яка використовується в додатку. Її проєктування дозволило швидко реалізувати механізм визначення найкоротшої відстані до укриття шляхом аналізу координат, що надходять від поточного місцезнаходження користувача.

У поєднанні з іншими модулями додатку, цей компонент забезпечує автоматизований вибір найближчого укриття та підготовку вихідних даних для побудови маршруту за допомогою Google Maps API. У підсумку реалізований підхід дозволяє ефективно вирішити задачу орієнтації користувача на місцевості, підвищуючи рівень безпеки та зменшуючи час реагування у разі виникнення загрози.

У підсумку реалізований підхід дозволяє ефективно вирішити задачу орієнтації користувача на місцевості, підвищуючи рівень безпеки та зменшуючи час реагування у разі виникнення загрози.

Додатково була реалізована перевірка коректності даних у базі на етапі ініціалізації додатку, що мінімізує ризик виникнення помилок при обробці координат.

Модуль також легко масштабувати для роботи з більшими обсягами інформації, що дозволяє адаптувати систему до потреб інших міст або регіонів.

Наступним важливим кроком було визначення принципу пошуку найближчого маркера. Для цього в мобільному додатку реалізовано алгоритм, який отримує поточну геолокацію пристрою, після чого здійснює обхід усіх

записів у базі даних та обчислює відстань до кожного укриття за допомогою стандартної функції Location.distanceBetween(). Об'єкт із найменшою відстанню вибирається як найближчий.

Structure | Data | Constraints | Indexes | Triggers | DDL

Grid view | Form view

Filter data | Total rows loaded: 1087

	name	description	lat	lng	styleUrl
1	ДНЗ №71	Адреса: 20-го Березня, 32 Кількість місць: 320 Час роботи: Балансоутримувач: Комунальний...	49.22716	28.48363	1
2	Ліцей №21	Адреса: 600-річчя, 16 Кількість місць: 333 Час роботи: Балансоутримувач: Комунальний ...	49.23141	28.42257	2
3	ДНЗ №45	Адреса: 600-річчя, 6 Кількість місць: 78 Час роботи: Балансоутримувач: Комунальний закла...	49.22394	28.42345	1
4	ДНЗ №75	Адреса: 600-річчя, 62 Кількість місць: 67 Час роботи: Балансоутримувач: Комунальний закл...	49.22324	28.42319	1
5	ДНЗ №30	Адреса: 600-річчя, 8 Кількість місць: 300 Час роботи: Балансоутримувач: Комунальний закл...	49.23302	28.42292	1
6	Клуб	Адреса: Агатангела Кримського, 46 Кількість місць: 200 Час роботи: ...	49.2243	28.43758	3
7	ДНЗ №3	Адреса: Академіка Ющенка, 14 Кількість місць: 180 Час роботи: Балансоутримувач: ...	49.21884	28.442	1
8	Спорт школа	Адреса: Академіка Янгеля, 33 Кількість місць: 200 Час роботи: 09:00-17:00 Балансоутримувач...	49.24579	28.49435	4
9	ДЮСШ №3 (Аква-Він)	Адреса: Академіка Янгеля, 48 Кількість місць: 500 Час роботи: 09:00-17:00 Балансоутримувач...	49.2446408	28.4846164	4
10	Клуб	Адреса: Академіка Янгеля, 65 Кількість місць: 100 Час роботи: 09:00-18:00 Балансоутримувач...	49.2428	28.48162	3
11	Гімназія №24	Адреса: Анатолія Бортняка, 3 Кількість місць: 1230 Час роботи: 18.00-08.00, Вихідні дні: ...	49.21499	28.44511	2
12	Ліцей №10	Адреса: Андрія Первозванного, 22 Кількість місць: 920 Час роботи: Балансоутримувач: ...	49.2204	28.41553	2
13	Барвінок	Адреса: Андрія Первозванного, 44 Кількість місць: 300 Час роботи: Балансоутримувач: ...	49.22371	28.40661	2
14	ДНЗ №74	Адреса: Андрія Первозваного, 68 Кількість місць: 500 Час роботи: Балансоутримувач: ...	49.22483	28.40208	1
15	Музична школа №1	Адреса: Архітектора Артинова, 21 Кількість місць: 125 Час роботи: Балансоутримувач: Закла...	49.2343	28.4666	2
16	Бібліотека	Адреса: Барвиста, 23 Кількість місць: 150 Час роботи: Балансоутримувач: Заклад «Вінницька...	49.21203	28.46659	5
17	Музей Коцюбинського	Адреса: Бевза, 15 Кількість місць: 15 Час роботи: Балансоутримувач: Заклад «Вінницький ...	49.2372081	28.4879244	6
18	Бібліотека	Адреса: Білоконя (Баженова), 32 Кількість місць: 150 Час роботи: ...	49.24699	28.53361	5
19	Ліцей №31	Адреса: Богдана Ступки, 13 Кількість місць: 998 Час роботи: 18.00-08.00, Вихідні дні: ...	49.24533	28.47438	2
20	Школа №5	Адреса: Богдана Ступки, 18 Кількість місць: 315 Час роботи: Балансоутримувач: Комунальни...	49.2461769	28.4739237	2
21	Центр культури	Адреса: Ботанічна, 13-А Кількість місць: 50 Час роботи: Балансоутримувач: Вінницький ...	49.25261	28.44975	6
22	Школа мистецтв	Адреса: Брацлавська, 85 Кількість місць: 120 Час роботи: Балансоутримувач: Заклад ...	49.2330606	28.4916867	2
23	ДЮСШ №2	Адреса: Бузький Узвіз, 33 Кількість місць: 246 Час роботи: Балансоутримувач: Заклад «Міськ...	49.22473	28.46636	4
24	Торгівельний інститут, гуртожиток	Адреса: В'ячеслава Чорновола, 3 Кількість місць: 676 Час роботи: 08.00-18.00, Вихідні дні: ...	49.23899	28.46534	7
25	ДНЗ №52	Адреса: Василя Порика, 17 Кількість місць: 320 Час роботи: Балансоутримувач: Комунальни...	49.22823	28.41746	1
26	Бібліотека	Адреса: Василя Порика, 28Б Кількість місць: 75 Час роботи: 09:00-17:00 Балансоутримувач: ...	49.23202	28.40426	2

Рисунок 3.1 – Структура таблиці укриттів у базі даних SQLite

Після знаходження найближчого укриття, додаток автоматично формує маршрут до нього, використовуючи Google Routes API, що дозволяє враховувати реальні дорожні умови. Крім того, передбачено обробку випадків, коли доступ до геолокації тимчасово недоступний або користувач відключив його вручну – у таких ситуаціях застосунок відображає відповідне повідомлення з проханням активувати служби локації.

У стандартному режимі роботи на карті відображається лише один найближчий маркер. Такий підхід був обраний свідомо, щоб у випадках надзвичайних ситуацій користувач не був перевантажений великою кількістю варіантів і міг сконцентруватися на найзручнішому та найближчому укритті. Для користувачів, які бажають переглянути більше варіантів, у меню налаштувань

реалізовано можливість активації режиму розширеного пошуку, який дозволяє відображати кілька найближчих укриттів одночасно.

Для забезпечення візуальної інформативності маркерів у застосунку використовується параметр styleUrl. Він визначає як вигляд маркера, так і стиль відображення карти. Наприклад, маркери для різних типів укриттів можуть мати різний колір чи іконку, що полегшує візуальне сприйняття інформації. Стилізація карти реалізована за допомогою спеціального JSON-файлу конфігурації, створеного через Google Map Style Wizard. Стиль дозволяє адаптувати карту під особливості завдань додатку: зробити її менш деталізованою, зменшити кількість візуальних об'єктів, залишаючи тільки ті елементи, що важливі в надзвичайних ситуаціях.

Вигляд фрагменту структури файлу з елементами стилю на мапі наведений на рисунку 3.2.

Завдяки реалізованому механізму, модуль пошуку працює стабільно як у щільній міській забудові, так і на відкритій місцевості. Особливу увагу було приділено оптимізації продуктивності при великій кількості записів у базі, щоб уникнути затримок при обчисленні відстаней. Для цього запити до бази даних реалізовано асинхронно, що дозволяє зберігати плавність роботи інтерфейсу користувача. Також додано логіку повторного обчислення маршруту у разі зміни поточного місцезнаходження користувача під час руху. Це підвищує точність побудованого маршруту та забезпечує динамічну навігацію в режимі реального часу.

Сама мапа з укриттями була отримана з офіційного джерела — сайту Вінницької міської ради [25]. Використання цього джерела було логічним, адже воно містить актуальну, офіційно підтверджену інформацію. Водночас використання QR-коду як основного інструменту доступу до карти укриттів у надзвичайних ситуаціях має низку недоліків. Зокрема, у випадку повітряної тривоги користувачу потрібно або заздалегідь зберігати посилання у браузері, або сканувати QR-код, що часто розміщується лише у фізичних місцях (наприклад, на зупинках громадського транспорту). Такий підхід вимагає

додаткових дій у критичних умовах, що може суттєво знижувати ефективність реагування. Натомість інтеграція даних безпосередньо в мобільний застосунок дозволяє забезпечити швидкий, зручний і автономний доступ до інформації про укриття без необхідності в зовнішніх переходах чи скануванні кодів у стресових ситуаціях.

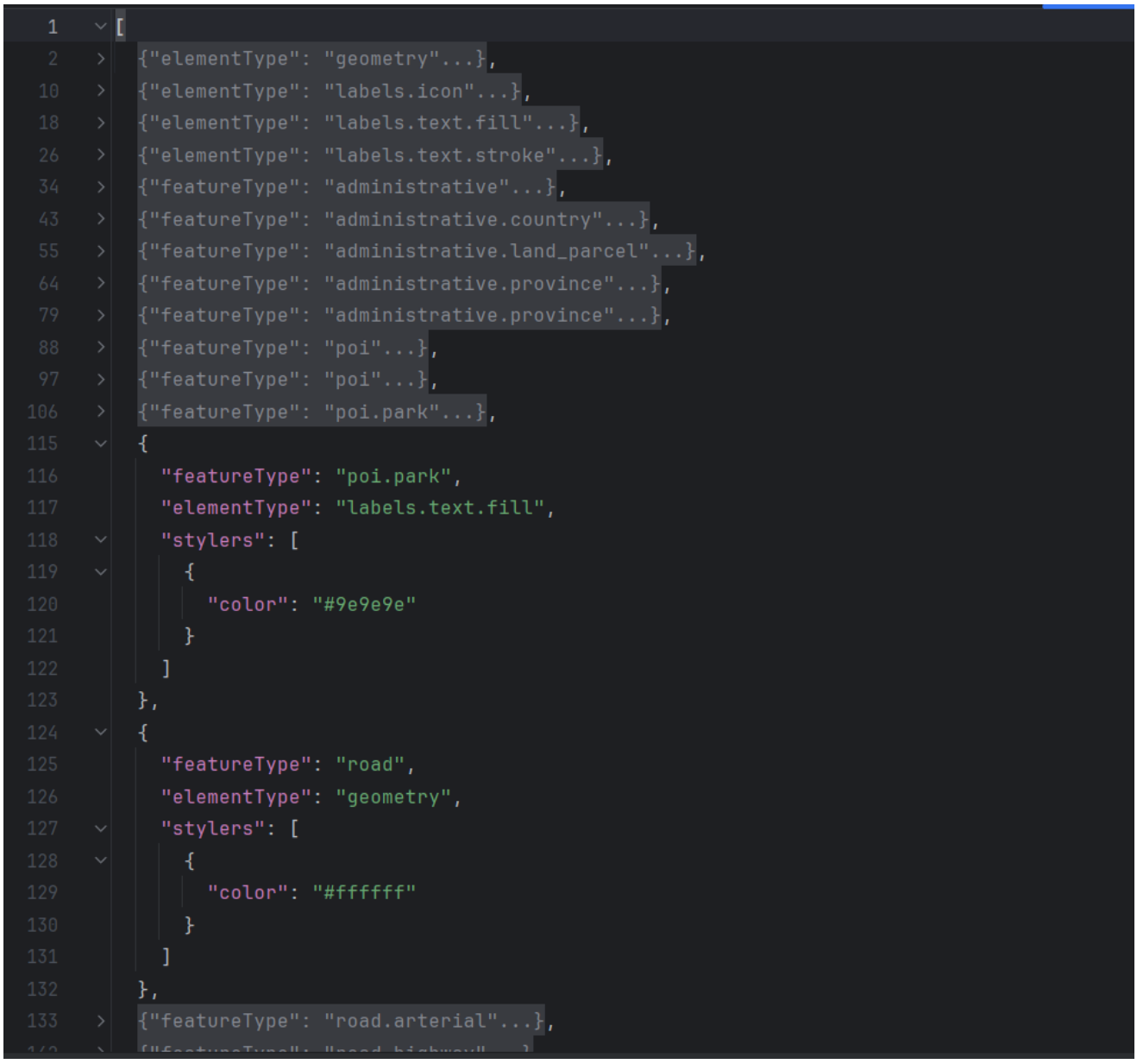

Рисунок 3.2 – Фрагмент JSON-файлу стилізації карти

На початкових етапах розробки, для наповнення бази даних інформацією про укриття, використовувався KMZ-файл, який містив географічні дані та опис кожного об'єкта. Файл містив усю необхідну інформацію для подальшої

обробки, а саме координати, опис укриття та класифікацію за типами. Формат KMZ є стисненою версією KML (Keyhole Markup Language) і використовується для обміну просторовими даними між геоінформаційними системами. Зазвичай він містить як текстові дані (у форматі XML), так і вкладені ресурси, такі як іконки або стилі. Такий формат зручний для імпорту в Google Maps або Google Earth, що значно полегшило процес попереднього перегляду даних перед інтеграцією. На основі цих даних були створені скрипти на Python для конвертації інформації у формат, сумісний із SQLite базою даних. Структура KMZ-файлу показана на рисунку 3.3.

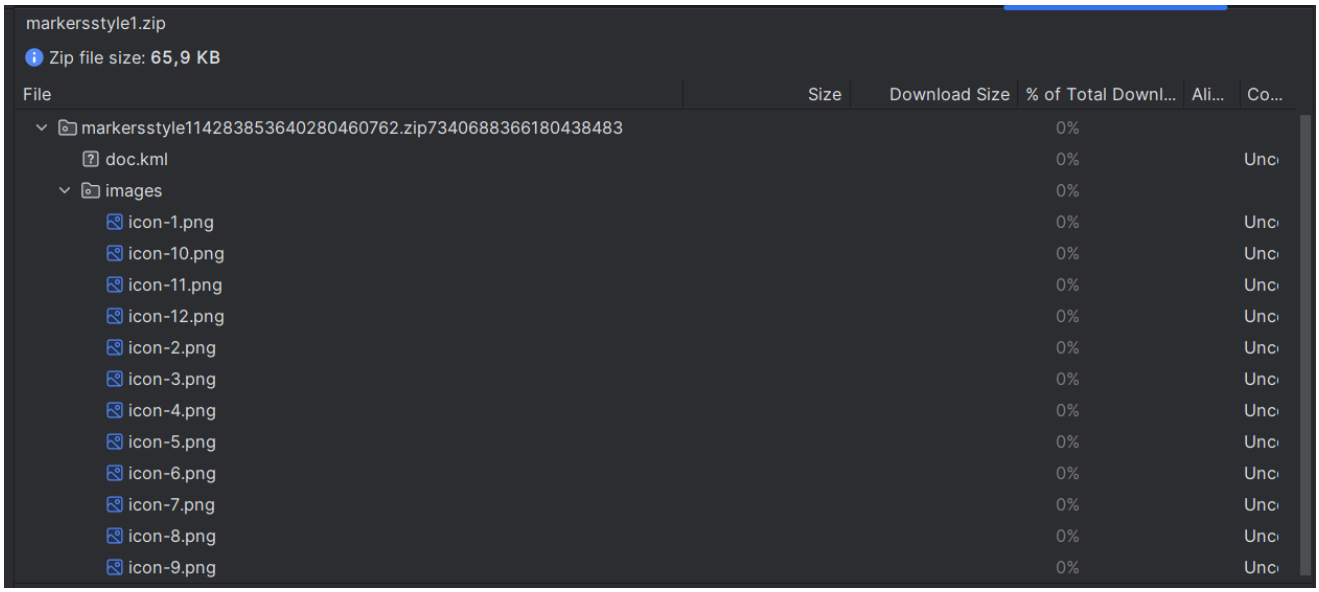

Рисунок 3.3 – Структура KMZ-файлу укриттів

Окремий акцент був зроблений на організації взаємодії користувача з маркерами укриттів на карті. Щоб зберегти мапу візуально чистою та уникнути перевантаження інтерфейсу зайвою інформацією, було реалізовано підхід, при якому детальні відомості про укриття відображаються лише після натискання на відповідний маркер. Така реалізація дозволяє користувачу швидко орієнтуватися у просторі без надлишкових елементів на екрані, що особливо важливо в умовах обмеженого часу або стресової ситуації.

У результаті реалізований модуль дозволяє здійснювати оперативний, зручний і логічно зрозумілий пошук найближчого укриття. Поєднання

кастомізованої візуалізації карти, грамотно побудованої структури бази даних і коректної обробки вхідних даних із KMZ-файлів зробило можливим створення надійного, функціонального та орієнтованого на користувача застосунку.

3.3 Розробка модуля прокладання маршруту до найближчого укриття

Основною функцією модуля побудови маршруту є надання користувачеві можливості швидко та зручно отримати оптимальний шлях до обраного укриття в режимі реального часу. Ключова мета полягає в тому, щоб зменшити час переміщення і підвищити ефективність навігації у надзвичайних ситуаціях шляхом інтеграції сучасних картографічних інструментів.

Для реалізації цієї функціональності було використано сервіс Google Routes API, який дозволяє будувати маршрути з урахуванням актуальної дорожньої інфраструктури, обмежень у русі та реального трафіку. Інтеграція з картографічними сервісами здійснювалася через офіційну бібліотеку Google Maps SDK для Android, що забезпечує тісний зв’язок між додатком та інструментами Google Maps.

Підключені API, які використовуються у додатку, відображені в інтерфейсі Google Cloud на рисунку 3.4.

Для забезпечення стабільної роботи функції побудови маршрутів було належним чином налаштовано середовище Google Cloud Console [26], де було активовано всі необхідні API та створено унікальний API-ключ для доступу додатку до сервісів Google. Цей етап є критично важливим для забезпечення коректної взаємодії між додатком і зовнішніми сервісами, а також для гарантування точності побудови маршрутів у реальних умовах.

На рівні логіки додатку прокладання маршруту починається автоматично після вибору або визначення найближчого укриття. Після отримання координат поточного місця знаходження користувача та координат укриття формується HTTP-запит до сервісу Google Directions API. У цьому запиті вказуються параметри початкової та кінцевої точки, спосіб пересування (пішки або транспортом), а також бажані налаштування маршруту.

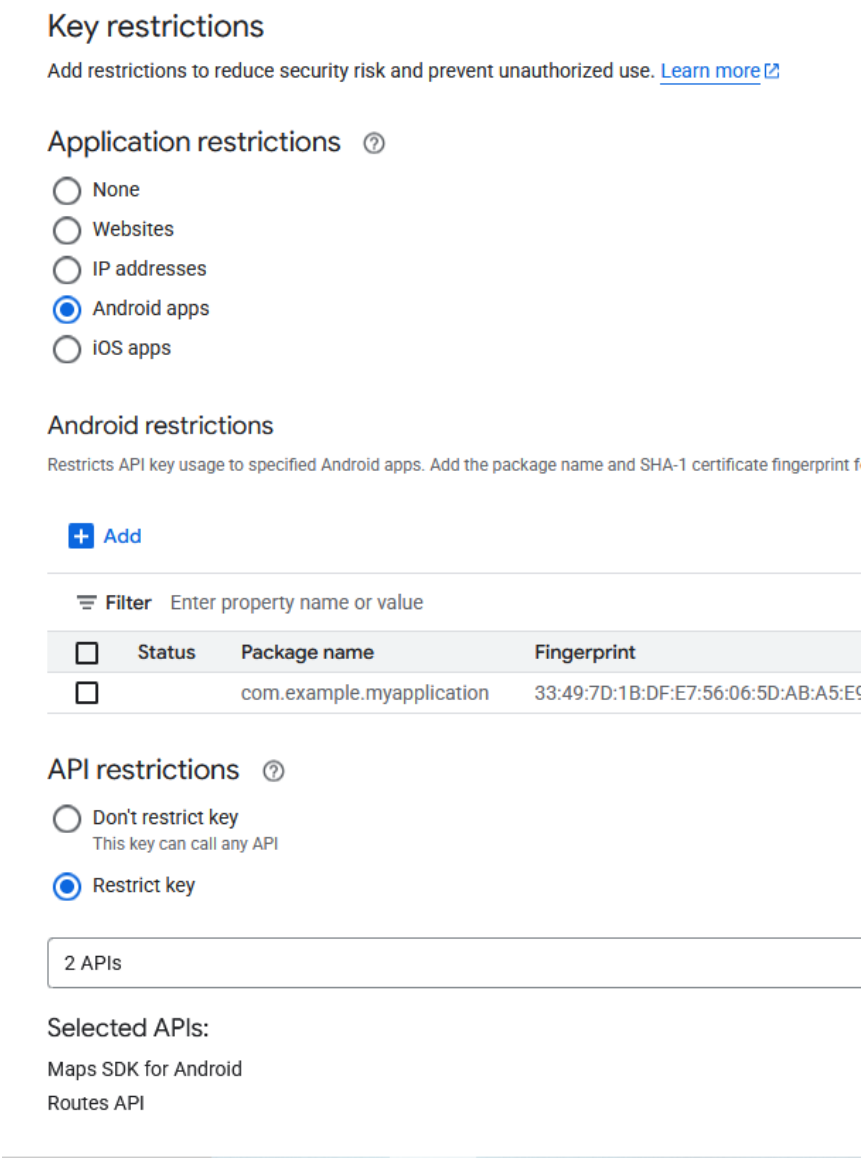

Рисунок 3.4 – Використання Google Cloud для підключення Google Maps та Routes API

Після відправлення запиту додаток обробляє відповідь у форматі JSON, який містить перелік точок маршруту (polyline). Polyline – це закодована лінія, що складається з серії координат, що визначають шлях маршруту [27]. Щоб відобразити цей маршрут на карті, необхідно декодувати ці закодовані дані у список координат.

Фрагмент коду для побудови запиту до Google Directions API наведено на рисунку 3.5. Функція downloadURL() виконується через клас FetchURL. Цей клас використовує асинхронне завдання для виконання HTTP-запиту до сервісу та отримання відповіді у форматі JSON.

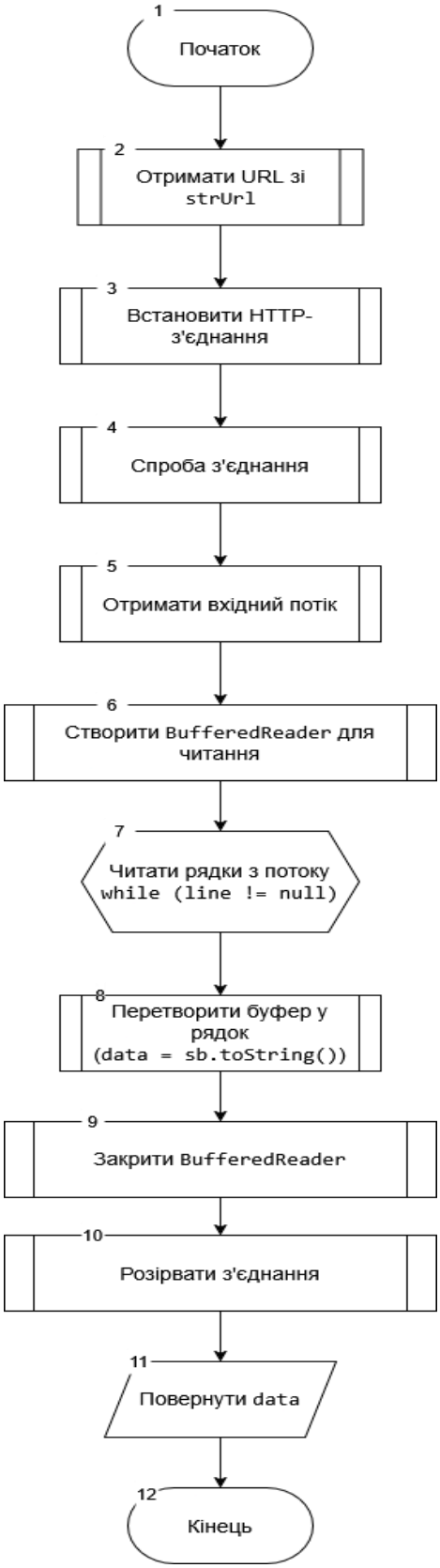

Рисунок 3.5 – Блок-схема функції з завантаження маршруту через запит в Routes API

Для декодування використовується вбудована функція, яка розпаковує закодовану polyline у форматі списку точок (LatLng). Даний процес працює наступним чином:

1. Кожна точка в polyline зберігається у вигляді закодованого значення, яке проходить через серію бітових операцій для стискання даних. Цей формат дозволяє зберігати координати з високою точністю, але в компактному вигляді.

2. Функція декодування перебирає закодований рядок, відновлюючи значення координат через бітові операції, включаючи зсуви та маскування байтів. В результаті ми отримуємо точні координати для кожної точки маршруту.

3. Оскільки координати в закодованому вигляді зберігаються у цілих числах, що означають соті тисячні градусів, після декодування вони діляться на 1E5 для отримання точних значень широти та довготи.

Ці декодовані координати потім додаються до списку LatLng, який може бути використаний для малювання лінії маршруту на карті.

Функція decodePoly виконує декодування закодованого рядка encoded, який містить серію координат маршруту у форматі Google Polyline. Цей формат використовується для компактного представлення маршруту, що передається через Google Maps Directions API. Результатом функції є список об'єктів LatLng, які відображають реальні точки маршруту.

Ключові етапи роботи функції:

1. Ініціалізація змінних: створюється порожній список координат і початкові значення широти та довготи.

2. Декодування широти:

- Послідовно читаються символи рядка.

- Символи перетворюються в числа, які обробляються побітово.

- Отримується дельта-значення широти, яке додається до поточного значення.

3. Декодування довготи: відбувається за аналогічним принципом, як і для широти.

4. Обчислення координати: широта та довгота перетворюються у формат

LatLng шляхом ділення на 1E5.

5. Додавання точки до списку: створений об'єкт LatLng додається до списку координат.

6. Цикл повторюється: поки не буде оброблено весь закодований рядок.

7. Повернення результату: функція повертає список декодованих координат маршруту.

Крім того, важливим етапом розробки було забезпечення зручності відображення маршруту на карті. Було розроблено спеціальні стилі відображення маршруту, які виділяють його яскравим кольором, а також додають маркери початкової та кінцевої точок з інформативними підписами. Приклад стилізованого відображення маршруту на карті наведений на рисунку 3.6.

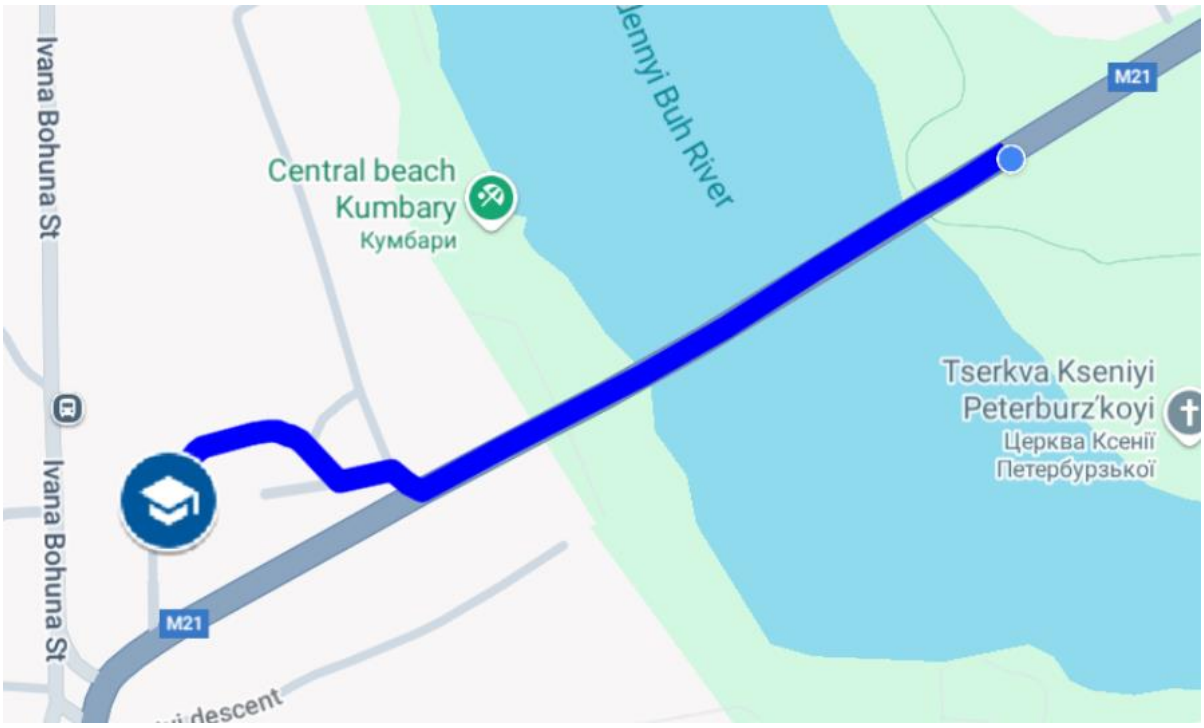

Рисунок 3.6 – Стилізоване відображення маршруту на карті

Процес реалізації прокладання маршруту до найближчого укриття в мобільному додатку здійснюється через інтеграцію з Google Maps. Коли користувач взаємодіє з маркером укриття на карті – зокрема, натискає на нього – і система активує відповідну подію, яка запускає механізм побудови маршруту.

Після натискання на маркер програма ініціює формування спеціального запиту до Google Directions API. Для цього автоматично генерується URL-адреса, яка містить координати поточного розташування користувача, координати вибраного укриття, а також вказівку на бажаний режим пересування (наприклад, автомобілем). Цей запит передається до окремого фону процесу, що відповідає за отримання та обробку маршруту.

В результаті обробки запиту система отримує дані про шлях, які візуалізуються на карті у вигляді маршруту від користувача до обраного укриття. Такий підхід забезпечує зручність та швидкість орієнтації під час надзвичайної ситуації, дозволяючи користувачу оперативно дістатися до безпечного місця.

3.4 Висновки

У цьому розділі було всебічно розглянуто процес вибору технологій, інструментів та методів, що стали основою для створення мобільного додатку, призначеного для пошуку найближчих укриттів та побудови маршрутів до них. Аналіз функціональних вимог до системи дав змогу визначити найбільш доцільні технічні рішення, які забезпечують високу продуктивність, надійність та зручність використання навіть за умов стресових ситуацій.

Основною середовищем розробки обрано Android Studio, що надало можливість повноцінної інтеграції з картографічними сервісами Google Maps API. Це забезпечило реалізацію ключових функцій: відображення карти, розміщення маркерів укриттів і побудову маршрутів через Directions API. Результатом стало створення інтуїтивно зрозумілого та простого у використанні інтерфейсу, що дозволяє користувачеві оперативно отримати життєво важливу інформацію.

Додаткові інструменти, такі як PyCharm і Visual Studio Code, значно спростили етапи обробки вихідних даних, підготовку стилів карти та організацію структури проєкту. Особливо корисним виявився Google Map Style Wizard, який дав змогу адаптувати візуальне оформлення карти згідно з потребами додатку —

акцентування на критично важливих елементах та уникнення інформаційного перевантаження.

У межах реалізації функціональних модулів була створена локальна база даних SQLite для зберігання інформації про укриття, реалізовано алгоритм пошуку найближчої точки за координатами користувача, а також налагоджено побудову маршруту з урахуванням поточної ситуації на дорогах. Застосування асинхронних HTTP-запитів до сервісу Google Directions API забезпечило оперативне оновлення маршрутів під час пересування користувача.

Окрему увагу приділено організації взаємодії з користувачем. Було реалізовано компактне подання інформації — деталі про укриття відкриваються лише після взаємодії з відповідним маркером. Такий підхід забезпечує чистоту інтерфейсу та сприяє швидкому сприйняттю даних у критичних умовах. Крім того, система автоматично пропонує лише найближче укриття, що дозволяє уникнути зайвого вибору та зменшує час реагування.

У підсумку, грамотне поєднання обраних технологій, чітко структурована архітектура додатку та орієнтація на потреби кінцевого користувача дозволили створити ефективний, надійний та адаптивний інструмент для навігації в умовах надзвичайних ситуацій. Реалізовані рішення забезпечують як технічну досконалість, так і практичну користь, що відповідає основним цілям розробки цього програмного продукту.

4 ТЕСТУВАННЯ ПРОГРАМИ

4.1 Тестування системи

Тестування є критично важливим етапом у процесі розробки програмного забезпечення, адже воно дозволяє виявити недоліки у роботі додатку, перевірити його стабільність та переконатися у відповідності функціональності встановленим вимогам. Основна мета тестування мобільного застосунку — гарантувати його надійне функціонування в умовах реального використання.

Перед початком перевірок у середовищі Android Studio було зібрано релізну версію APK-файлу, який використовувався для встановлення програми на різноманітні пристрої, як фізичні, так і віртуальні. Основними платформами, де проводилося тестування, стали:

- фізичний смартфон Xiaomi Redmi Note 8T, підключений через USB у режимі налагодження;

- хмарний сервіс Firebase Remote Device із використанням віртуального пристрою Google Pixel 8a;

- емулятор BlueStacks 5 [28], де також застосовувалася можливість зміни геопозиції для тестування функцій, пов’язаних із місцеположенням.

Переважна частина тестування здійснювалася вручну. Для діагностики використовувався режим налагодження (debug mode) та інструмент Logcat [29], що дозволив виявити помилки, нестабільну поведінку або порушення логіки роботи додатку.

Процес перевірки складався з кількох ключових етапів:

Функціональне тестування: на цьому етапі перевірялися основні функції додатку – завантаження карти, пошук найближчих укриттів, побудова маршруту до вибраного об’єкта та перегляд інформації про сховище. Особлива увага приділялася точності геолокації, коректності маршрутизації, а також поведінці програми у випадку некоректних даних або запитів.

На фізичному пристрої Xiaomi Redmi Note 8T було виконано наступні дії:

- запуск додатку при різних типах з’єднання (Wi-Fi, мобільна мережа, відсутність інтернету);

- імітація втрати доступу до мережі;

- перевірка роботи геолокації через GPS;

- тестування функціональності кнопок інтерфейсу — пошуку, побудови маршруту, оновлення карти.

На рисунку 4.1 представлено приклад інтерфейсу додатку під час відображення найближчого укриття в умовах реального тестування.

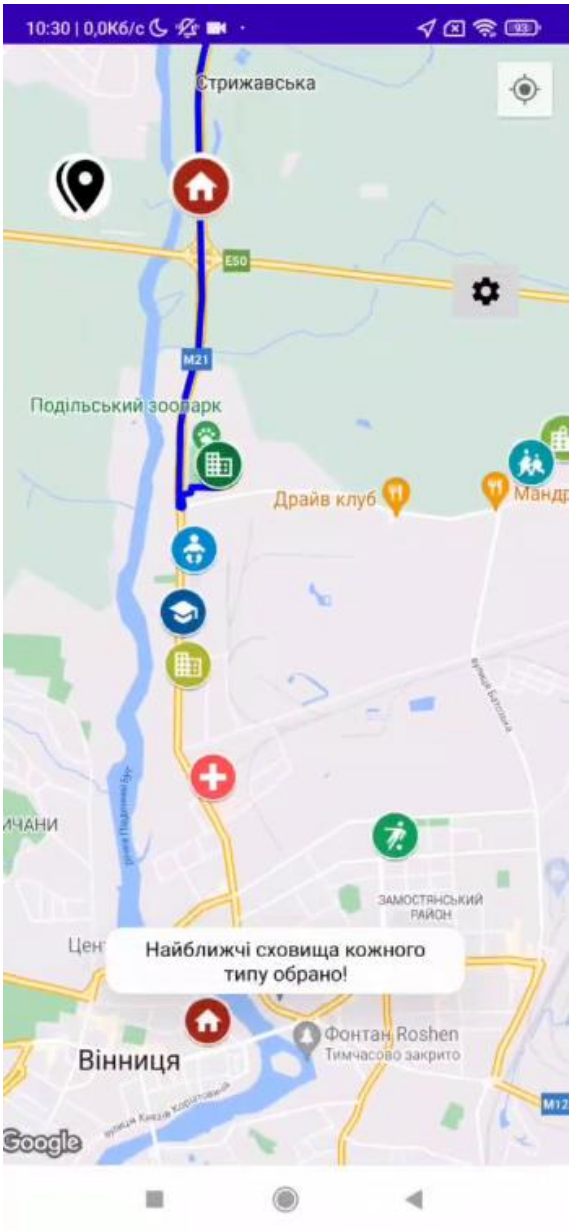

Рисунок 4.1 – Відображення найближчих укриттів на фізичному пристрої

У процесі тестування в реальному часі активно використовувався інструмент Logcat, за допомогою якого здійснювався моніторинг обробки запитів до геолокаційних сервісів та побудови маршрутів. Це дозволяло швидко реагувати на помилки, аналізувати внутрішню логіку роботи додатку та оперативно вносити необхідні коригування.

Перевірка роботи в умовах відмов та нестабільності стала другим важливим етапом тестування. Вона передбачала моделювання ситуацій, за яких система працює з обмеженими або недоступними ресурсами. Зокрема, було протестовано:

- ситуацію втрати інтернет-з’єднання під час побудови маршруту;
- відключення функцій геолокації на самому пристрої;
- запуск пошуку укриттів без надання дозволу на визначення місцезнаходження;
- ручну зміну координат користувача через інструменти налаштування у віртуальному середовищі.

На рисунку 4.2 показано приклад повідомлення, яке з’являється у випадку спроби побудувати маршрут без активованої служби GPS.

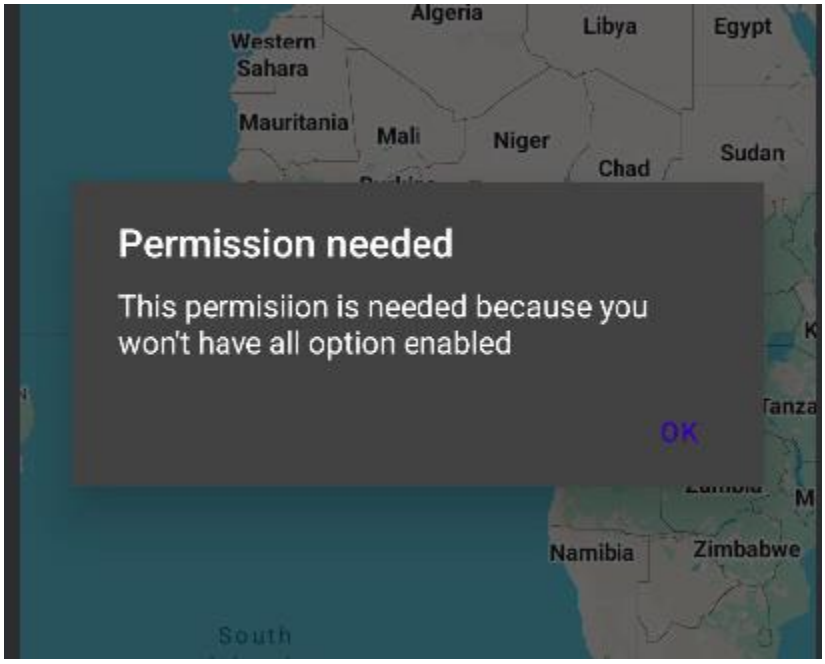

Рисунок 4.2 – Повідомлення про відсутність геолокації користувача

Для моделювання таких ситуацій активно застосовувався емулятор BlueStacks 5, що дозволив змінювати координати вручну, імітуючи переміщення користувача в регіони, де немає жодного укриття. Це дало змогу перевірити, наскільки коректно програма реагує на подібні обставини, а також чи виводить вона відповідні попередження або повідомлення.

На початковому етапі запуску застосунку, у разі відсутності автоматичної передачі геолокаційних даних, реалізовано механізм запиту дозволу на доступ до місцезнаходження. У програмному коді передбачена перевірка на наявність як приблизної, так і точної геолокації.

Діалогове вікно з відповідним запитом з'являється автоматично та відображається мовою, яка відповідає системним налаштуванням пристрою. Приклад цього вікна представлено на рисунку 4.3.

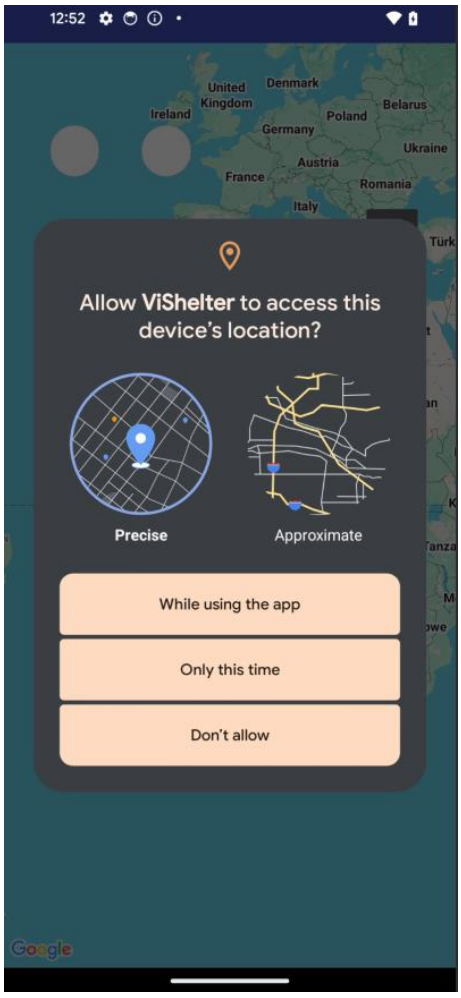

Рисунок 4.3 – Вікно запиту на геолокацію користувача

Під час тестування особливий акцент було зроблено на перевірці стабільності роботи додатку у взаємодії з Google Maps API, а також на ефективності обробки виняткових ситуацій.

Для додаткової перевірки функціонування застосунку використовувався сервіс Firebase Test Lab [30], який надає можливість запуску тестів на реальних пристроях у хмарному середовищі. Зокрема, було проведено тестування на моделі Google Pixel 8a з актуальною версією Android.

Використання Firebase дозволило автоматизувати симуляцію дій користувача та оцінити поведінку програми в умовах, коли розробник не має прямого доступу до пристрою. Звіти, сформовані після проходження тестів, включали знімки екрана, журнали подій та інформацію про можливі збої, що значно полегшувало виявлення прихованих проблем у роботі додатку.

Автоматично сформований звіт про результати тестування наведено на рисунку 4.4.

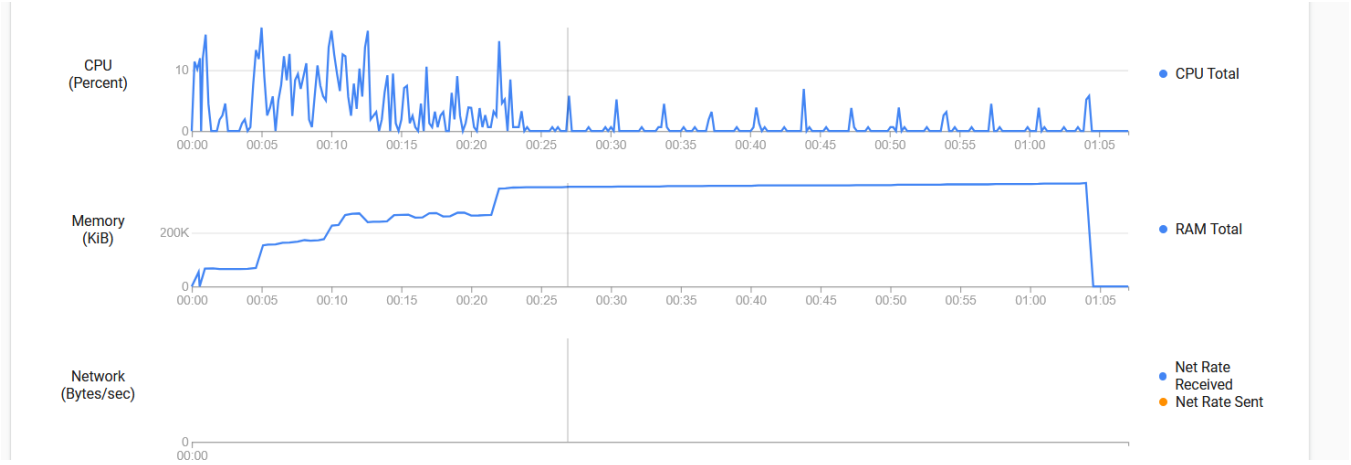

Рисунок 4.4 – Звіт Firebase Test Lab про тестування додатку

Суттєвих збоїв або критичних помилок виявлено не було, оскільки тестування проводилося на завершальних етапах розробки застосунку. Серед зафіксованих незначних зауважень — один випадок використання застарілого елементу Android SDK та п’ять попереджень щодо доступності, пов’язаних із недостатнім розміром деяких кнопок інтерфейсу. Аналіз продуктивності

засвідчив низьке навантаження на процесор і оперативну пам’ять. Усі виявлені недоліки були усунуті до формування фінальної версії APK-файлу.

Оцінка продуктивності та швидкодії. На наступному етапі було проведено тестування основних параметрів швидкодії, зокрема:

- час запуску карти під час першого відкриття застосунку;
- тривалість пошуку найближчого укриття;
- середній час побудови маршруту;
- швидкість реакції інтерфейсних елементів на дії користувача.

Результати тестування на фізичному пристрої засвідчили, що карта завантажується за приблизно 2.3 секунди при стандартному з'єднанні з Інтернетом, а побудова маршруту займає від 1.5 до 2 секунд — залежно від відстані до укриття.

На рисунку 4.5 продемонстровано приклад побудованого маршруту до найближчого укриття під час тестування на реальному пристрої.

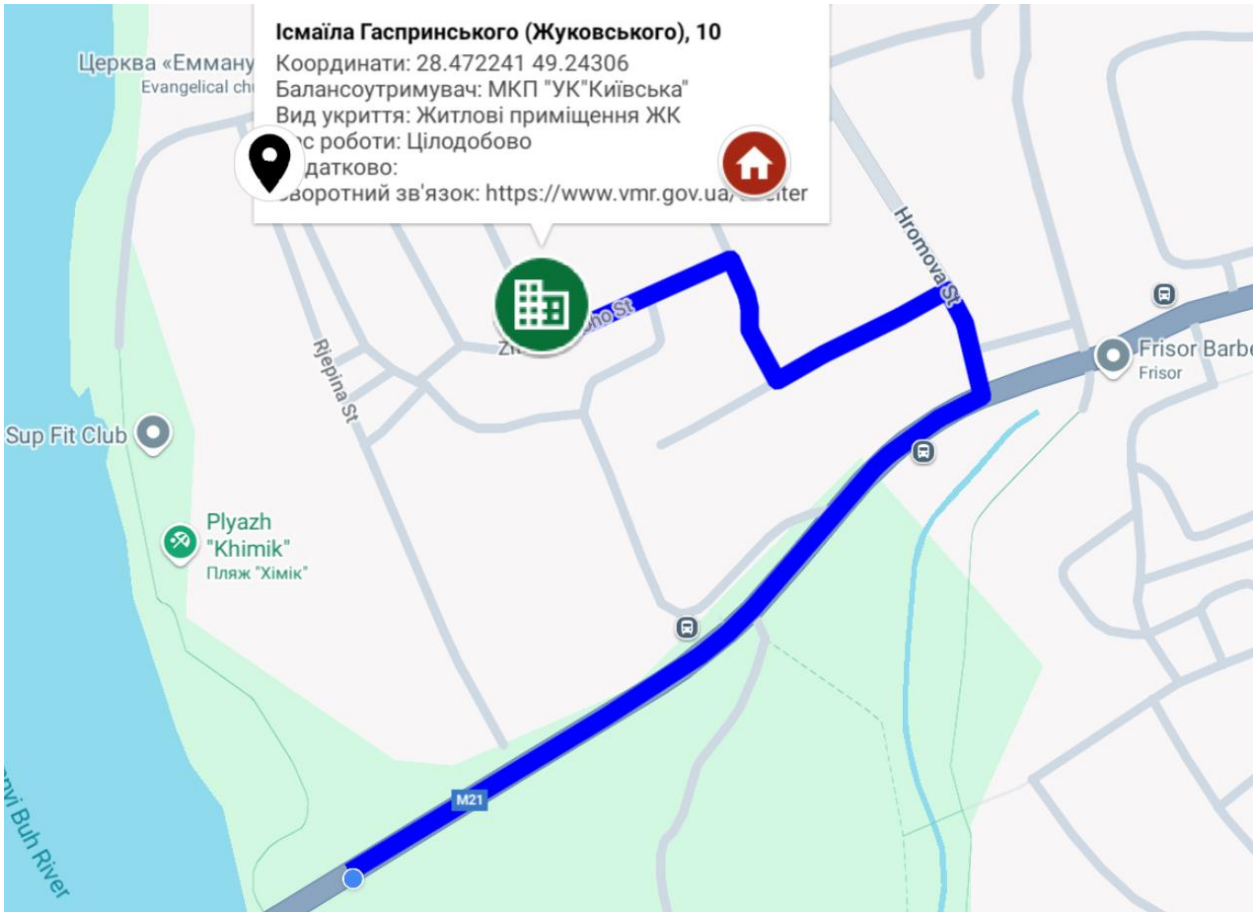

Рисунок 4.5 – Побудова маршруту до найближчого укриття

У середовищі емулятора BlueStacks час відгуку був дещо більшим через обмеження ресурсів, однак залишався у допустимих межах і не перевищував 3 секунд.

Тестування сумісності. Окремий етап був присвячений перевірці роботи додатку на різних версіях операційної системи Android. Тестування охоплювало такі конфігурації:

- Android 11 (Redmi Note 8T);
- Android 14 (Google Pixel 8a у середовищі Firebase);
- Android 10 (емулятор BlueStacks).

На всіх перевірених версіях операційної системи застосунок демонстрував стабільну роботу без критичних збоїв чи помилок. Особлива увага була зосереджена на стабільності роботи геолокаційного функціоналу та прокладання маршрутів, оскільки саме ці модулі можуть мати специфічні особливості реалізації в залежності від версії Android.

Перевірка геолокації та функції пошуку. Тестування змін координат користувача в середовищі емулятора BlueStacks 5 дало змогу змоделювати різні сценарії використання додатку. Зокрема, через вбудований інструмент ручної зміни місцеположення перевірялося, як система реагує на нові координати та чи коректно визначає найближчі укриття.

Сценарії включали:

- зміну координат у межах одного району;
- переміщення в інші міста України;
- встановлення місцеположення за межами країни для перевірки реакції додатку на відсутність укриттів.

На рисунку 4.6 представлено приклад зміни координат в BlueStacks і результат пошуку найближчих укриттів відповідно до нових даних.

Додаток коректно обробляв зміну місця розташування та оновлював список доступних укриттів у відповідності до нових координат.

Аналіз логів: під час усього тестування активно використовувався Logcat для:

- моніторингу стану підключення до служб Google Maps API;
- відстеження запитів на побудову маршрутів;
- обробки повідомлень про помилки;
- аналізу продуктивності додатку.

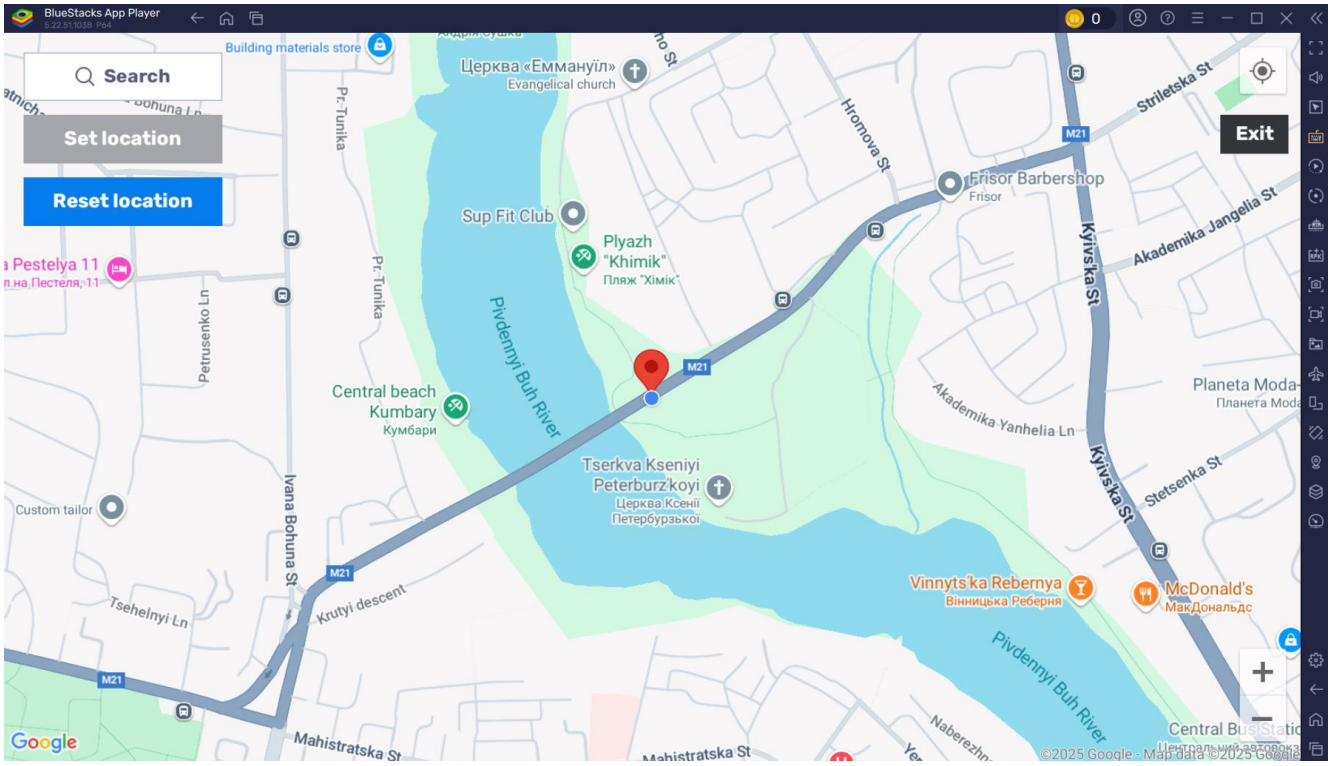

Рисунок 4.6 – Зміна місцеположення через емулятор BlueStacks 5

Завдяки детальному аналізу логів вдалося своєчасно виявити такі проблеми, як некоректна обробка дозволів на доступ до геолокації та неефективні запити до картографічного сервісу, що дало змогу оптимізувати ці процеси.

Застосування Logcat стало одним із ключових інструментів налагодження: воно дало змогу відстежувати всі критичні етапи виконання, швидко локалізовувати джерела помилок і перевіряти коректність взаємодії з API.

Цей підхід суттєво підвищив надійність додатку та забезпечив стабільну роботу навіть у складних умовах використання.

На рисунку 4.7 показано приклад перегляду логів через Logcat під час роботи додатку. Найкориснішим використанням Logcat став перегляд інформації про запит в Routes API.

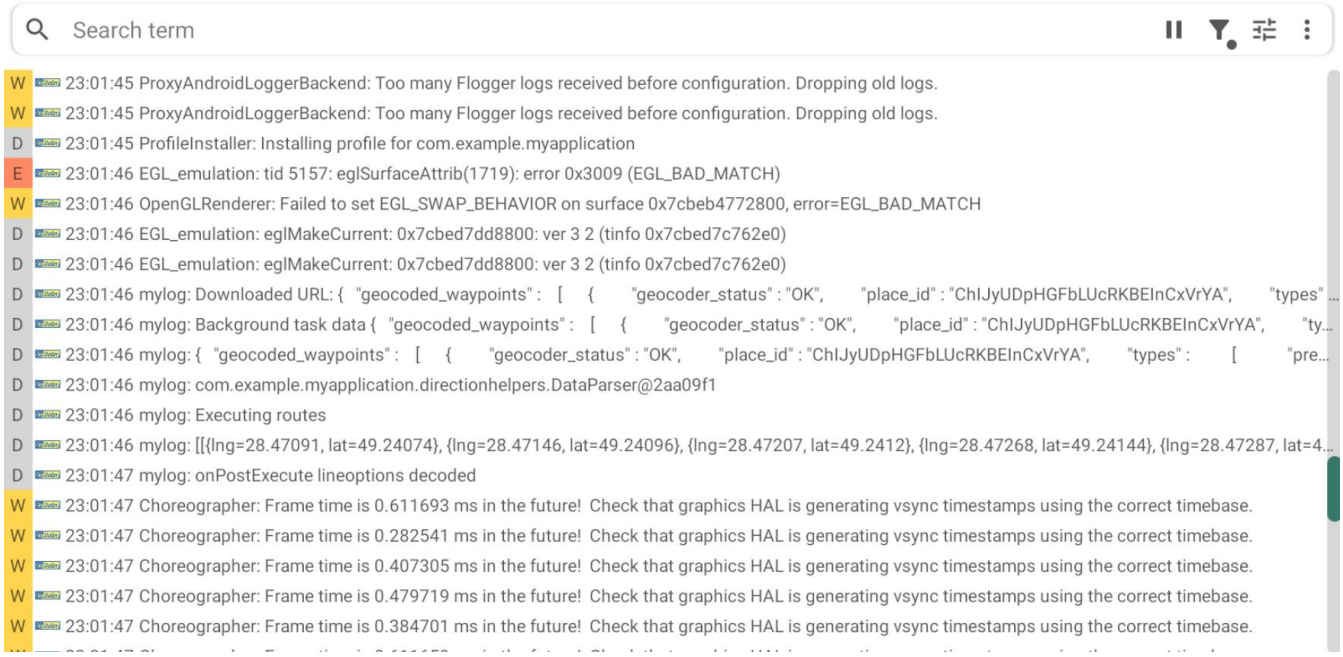

Рисунок 4.7 – Аналіз логів роботи додатку

4.2 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску програмного продукту на мобільних пристроях під керуванням операційної системи Android. Деталі щодо мінімальної та рекомендованої конфігурації наведено в таблиці 4.1.

Таблиця 4.1 – Програмно-апаратні вимоги

Варіант	Мінімальні	Рекомендовані
Локальний	Операційна система Android 10 Процесор: Qualcomm Snapdragon 665 ОЗП: 4ГБ Рівень API пристрою: 24	Операційна система Android 12 Процесор: Qualcomm Snapdragon 865 ОЗП: 6ГБ Рівень API пристрою: 30

Окрім вимог до апаратної частини, для коректної роботи програмного забезпечення необхідно передбачити наявність вільного місця у внутрішній

пам’яті пристрою. У таблиці 4.2 наведено обсяг дискового простору, який необхідний на різних етапах роботи із застосунком.

Таблиця 4.2 – Вимоги до вільного місця на жорсткому диску для програмного забезпечення

Тип системи	Вільне (не стиснуте) місце на жорсткому диску	
	До інсталяції	Під час роботи
ПЗ	400 МБ	200 МБ

Після встановлення та запуску застосунку користувач потрапляє на головний екран, де очікує завершення ініціалізації основних сервісів. Далі доступна повноцінна взаємодія з функціоналом відповідно до потреб: пошук найближчих укриттів, перегляд опису об’єктів, прокладання маршрутів за допомогою Google Maps.

Першим етапом є надання дозволу на доступ до геолокації – після його підтвердження система автоматично визначає поточну позицію користувача та виводить список доступних укриттів поблизу. Через головне меню також можна перейти до довідкової інформації, налаштувань або ознайомитися з даними про застосунок. Для побудови маршруту достатньо натиснути на обране укриття на карті та скористатися відповідною кнопкою навігації.

4.3 Висновки

У четвертому розділі було здійснено всебічне тестування створеного мобільного застосунку для визначення найближчих укриттів з використанням сервісу Google Maps. Особливу увагу приділено перевірці роботи застосунку в умовах некоректного введення, нестабільного інтернет-з’єднання, а також правильності обробки дозволів і типових помилкових сценаріїв.

Крім того, підготовлено інструкцію користувача, яка включає перелік технічних вимог до пристрою та описує основні дії, необхідні для ефективної взаємодії з функціоналом додатку.

ВИСНОВКИ

У рамках виконання дослідження було створено мобільний застосунок для визначення найближчих укриттів із використанням Google Maps API та Routes API. Розробка проводилася в середовищі Android Studio з використанням мови програмування Java, відповідно до методичних рекомендацій [31, 32].

У ході роботи було проаналізовано наявні мобільні застосунки подібного призначення, що дозволило виокремити їхні переваги та недоліки й обґрунтувати вибір функціоналу для власного рішення. На основі цього аналізу були сформульовані основні цілі та задачі проєкту. Розробка передбачала послідовне проходження всіх етапів життєвого циклу програмного забезпечення — від постановки задачі до тестування та оформлення інструкції користувача.

У результаті реалізації виконано наступне:

- сформульовано ключові функціональні та технічні вимоги до застосунку;
- розроблено архітектуру програмного забезпечення;
- реалізовано інтеграцію з Google Maps API для візуалізації карти та маркерів укриттів;
- використано Google Routes API для побудови маршрутів до обраних об’єктів;
- створено базу даних укриттів із функцією пошуку найближчих точок;
- розроблено зручний, інтуїтивно зрозумілий інтерфейс;
- проведено тестування застосунку, включно з аналізом логів через Logcat та використанням хмарного сервісу Firebase Test Lab;
- підготовлено інструкцію користувача для швидкого ознайомлення з функціоналом застосунку.

У процесі проєктування було також створено UML-діаграми основних компонентів, що допомогло структурувати логіку функціонування системи. Окрема увага приділялася зручності користувача, швидкодії інтерфейсу та надійності передачі запитів до картографічних сервісів.

Під час реалізації застосунку вдалося забезпечити високу стабільність роботи навіть у складних умовах, таких як втрата мережевого з’єднання чи відсутність доступу до служб геолокації. Проведене тестування підтвердило відповідність функціоналу заявленим вимогам, а також готовність програмного продукту до практичного використання.

Розроблений застосунок має високий потенціал для подальшого розвитку: зокрема, можна розширити функціонал системи сповіщення, додати інтеграцію з локальними базами укриттів інших міст, а також реалізувати функції офлайн-навігації. Це дозволить зробити програму ще ефективнішою у реальних надзвичайних ситуаціях.

ЛІТЕРАТУРА

1. Мисловський. КОНФЕРЕНЦІЇ ВНТУ електронні наукові видання. URL: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/24967 (дата звернення: 08.06.2025).

2. Van Dijcke D., Wright A.L., Polyak M. Public response to government alerts saves lives during Russian invasion of Ukraine. Proceedings of the National Academy of Sciences of the United States of America (PNAS). 2023; 120(18): e2220160120. DOI: 10.1073/pnas.2220160120. (Last accessed: 06.06.2025).

3. Opach T., Rød J. K., Munkvold B. E. Map functions to facilitate situational awareness during emergency events. Cartography and Geographic Information Science. 2023. Vol. 50, № 6. P. 546–561. DOI: 10.1080/15230406.2023.2264759. (Last accessed: 06.06.2025).

4. Daud M., Ugliotti F. M., Osello A. Comprehensive Analysis of the Use of Web-GIS for Natural Hazard Management: A Systematic Review. Sustainability. 2024. Vol. 16, № 10. Art. 4238. DOI: 10.3390/su16104238. (Last accessed: 06.06.2025).

5. Elhashash M., Albanwan H., Qin R. A Review of Mobile Mapping Systems: From Sensors to Applications. Sensors. 2022. Vol. 22, № 11. Art. 4262. DOI: 10.3390/s22114262. (Last accessed: 06.06.2025).

6. Kremenchuk A. Укриття кременчук – додатки в google play. Android Apps on Google Play. URL: https://play.google.com/store/apps/details?id=com.kremenchuk.android.ukryttia&hl=uk (дата звернення: 05.06.2025).

7. Deven Joshi. Building Cross-Platform Apps with Flutter and Dart . BPB Publications, Delhi 2023, 378 p.

8. Navisoft E. ДеУкриття - додатки в google play. Android Apps on Google Play. URL: https://play.google.com/store/apps/details?id=com.navisoft.e_ternopil&hl=uk&gl=US (дата звернення: 05.06.2025).

9. Kremenchuk A. Укриття вінниця – додатки в google play. Android Apps on Google Play. URL: https://play.google.com/store/apps/details?id=com.vinnytsya.android.ukryttia&hl=uk (дата звернення: 05.06.2025).

10. Bevor Sie zu Google Maps weitergehen. Google. URL: https://www.google.com/maps (дата звернення: 08.06.2025).

11. SQLite home page. SQLite Home Page. URL: https://sqlite.org/ (дата звернення: 08.06.2025).

12. Google maps platform documentation | routes API | google for developers. Google for Developers. URL: https://developers.google.com/maps/documentation/routes (дата звернення: 05.06.2025).

13. KMZ files | keyhole markup language | google for developers. Google for Developers. URL: https://developers.google.com/kml/documentation/kmzarchives (дата звернення: 08.06.2025).

14. GeeksforGeeks. Top programming languages for android app development [2025 updated] - geeksforgeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/top-programming-languages-for-android-app-development/ (дата звернення: 29.04.2025).

15. Team L. E. Java for mobile app development: a complete guide in 2025. Leanware. URL: https://www.leanware.co/insights/guide-for-java-mobile-app-development (дата звернення: 29.04.2025).

16. GeeksforGeeks. Top programming languages for android app development [2025 updated] - geeksforgeeks. GeeksforGeeks. URL: https://www.geeksforgeeks.org/top-programming-languages-for-android-app-development/ (дата звернення: 29.04.2025).

17. Maps SDK for android quickstart | google for developers. Google for Developers. URL: https://developers.google.com/maps/documentation/android-sdk/start (дата звернення: 29.04.2025).

18. Save data using SQLite | App data and files | Android Developers. Android Developers. URL: https://developer.android.com/training/data-storage/sqlite (дата звернення: 29.04.2025).

19. Flowchart maker & online diagram software. Flowchart Maker & Online Diagram Software. URL: https://app.diagrams.net/ (дата звернення: 08.06.2025).

20. Download android studio & app tools - android developers. Android Developers. URL: https://developer.android.com/studio (дата звернення: 08.06.2025).

21. Effective pycharm: learn the pycharm IDE with a hands-on approach. Independently Published, 2019. 221 с.

22. McKinney W. Python for data analysis: data wrangling with pandas, numpy, and jupyter. O'Reilly Media, 2022. 550 с.

23. Styling wizard: google maps apis. Styling Wizard: Google Maps APIs. URL: https://mapstyle.withgoogle.com/ (дата звернення: 29.04.2025).

24. Microsoft. Visual studio code - code editing. redefined. Visual Studio Code - Code Editing. Redefined. URL: https://code.visualstudio.com/ (дата звернення: 08.06.2025).

25. Укриття та приміщення для тимчасового перебування - Google My Maps. Google My Maps. URL: https://shelter.vmr.gov.ua/ (дата звернення: 08.06.2025).

26. Sangapu, S., Panyam, D., Marston, J. The Definitive Guide to Modernizing Applications on Google Cloud (1st ed.). Birmingham : Packt Publishing, 2022. 488 p.

27. Simple polylines | maps javascript API | google for developers. Google for Developers. URL: https://developers.google.com/maps/documentation/javascript/examples/polyline-simple (дата звернення: 29.04.2025).

28. BlueStacks android emulator & cloud gaming plattform: spiele auf PC & mac spielen!. Bluestacks - The Best Android Emulator on PC as Rated by You. URL: https://www.bluestacks.com/ (дата звернення: 29.04.2025).

29. Logcat command-line tool | android studio | android developers. Android Developers. URL: https://developer.android.com/tools/logcat (дата звернення: 29.04.2025).

30. Firebase test lab. Firebase. URL: https://firebase.google.com/docs/test-lab (дата звернення: 29.04.2025).

31. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 51с.

ДОДАТКИ

ДОДАТОК А
Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ

д.т.н., проф. О. Н. Романюк

"25" березня 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу
Розробка мобільного додатку для пошуку найближчих укриттів на платформі Android»
за спеціальністю 121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:

PhD, ст. викладач каф. ПЗ Стахов О.Я.

"24" березня 2025 р.

Виконав:

студент гр. 2ПІ-21б Мисловський А.В.

"24" березня 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування.

Бакалаврська кваліфікаційна робота: «Розробка мобільного додатку для пошуку найближчих укриттів на платформі Android». Галузь застосування – мобільні технології.

2. Підстава для розробки.

Підставою для виконання бакалаврської кваліфікаційної роботи (БКР) є індивідуальне завдання на БКР та наказ №97 від «20» березня 2025 р. ректора по ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки.

Метою роботи є підвищення швидкості орієнтації користувача в умовах надзвичайних ситуацій за рахунок створення мобільного застосунку, що дозволяє оперативно знаходити найближчі укриття та прокладати маршрут до них з урахуванням поточної геолокації. Призначення роботи – розробка методів і засобів пошуку та маршрутизації до сховища.

4. Вихідні дані для проведення НДР.

Перелік основних літературних джерел, на основі яких буде виконуватись БКР.

1. Van Dijcke D., Wright A.L., Polyak M. Public response to government alerts saves lives during Russian invasion of Ukraine. Proceedings of the National Academy of Sciences of the United States of America (PNAS). 2023; 120(18): e2220160120. DOI: 10.1073/pnas.2220160120. (Last accessed: 06.06.2025).

2. Deven Joshi. Building Cross-Platform Apps with Flutter and Dart . BPB Publications, Delhi 2023, 378 p.

3. Sangapu, S., Panyam, D., Marston, J. The Definitive Guide to Modernizing Applications on Google Cloud (1st ed.). Birmingham : Packt Publishing, 2022. 488 p.

5. Технічні вимоги.

Платформа – Android; мінімальні вимоги – Android 10, Snapdragon 665, 4 ГБ ОЗП, API 24; рекомендовані – Android 12, Snapdragon 865, 6 ГБ ОЗП, API 30; вхідні дані – координати укриттів та місцезнаходження користувача; вихідні – карта з маркерами, маршрут, інформаційні вікна; бібліотеки – Google Maps SDK, Firebase, SQLite.

6. Конструктивні вимоги.

Конструкція пристрою повинна відповідати естетичним та ергономічним вимогам, повинна бути зручною в обслуговуванні та керуванні. Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до БКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації.

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

Ч.ч.	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання та постановка задач дослідження	25.03.25 – 29.03.25	
2	Розробка структури та моделі застосунку	30.03.25 – 05.04.25	
3	Розробка алгоритмів взаємодії з мапою та логіки визначення маршрутів	06.04.25 – 12.04.25	
4	Вибір технологій, бібліотек і сервісів для реалізації програмного забезпечення	13.04.25 – 15.04.25	
5	Аналіз та вибір засобів для реалізації програмного продукту	16.04.25 – 05.05.25	
6	Програмна реалізація	06.05.25 – 12.05.25	
7	Тестування програмного застосунку	13.05.25 – 19.05.25	
8	Оформлення матеріалів до захисту БКР	20.04.25 – 30.05.25	

10. Порядок контролю та прийняття.

Виконання етапів бакалаврської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Захист бакалаврської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

ДОДАТОК Б
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка мобільного додатку для пошуку найближчих укриттів на платформі Android»

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра програмного забезпечення, ФІТКІ
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 8,1 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

■ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

(прізвище, ініціали, посада) (підпис)

(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник Стахов О.Я.
(підпис) (прізвище, ініціали, посада)

Здобувач Мисловський А.В.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програми

Модуль «MapsActivity»:

```
//головна activity з мапою в додатку
package com.example.myapplication;


import static com.example.myapplication.MapsActivity.modeSelected.ClosestShelterForAllStyles;
import static com.example.myapplication.MapsActivity.modeSelected.OneClosestShelter;
import android.Manifest;
import android.annotation.SuppressLint;
import android.app.AlertDialog;
import android.content.Context;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.graphics.Color;
import android.graphics.Typeface;
import android.location.Location;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ImageButton;
import android.widget.LinearLayout;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;
import androidx.core.text.HtmlCompat;
import androidx.fragment.app.FragmentActivity;

import com.example.myapplication.databinding.ActivityMapsBinding;
```

```
import com.example.myapplication.directionhelpers.FetchURL;
import com.example.myapplication.directionhelpers.TaskLoadedCallback;
import com.google.android.gms.location.FusedLocationProviderClient;
import com.google.android.gms.location.LocationServices;
import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.OnMapReadyCallback;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MapStyleOptions;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;
import com.google.android.gms.maps.model.Polyline;
import com.google.android.gms.maps.model.PolylineOptions;
import com.google.android.gms.tasks.Task;
import com.google.maps.android.data.kml.KmlLayer;

import org.xmlpull.v1.XmlPullParserException;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;

public class MapsActivity extends FragmentActivity implements OnMapReadyCallback,
AdapterView.OnItemSelectedListener, TaskLoadedCallback {
   private final int ACCESS_LOCATION_REQUEST_CODE = 10001; //код який відповідає
наданню запиту на надсилання геолокації в додатку
   static GoogleMap mMap;
   Context context;
   ImageButton settingsButton;
```

```
    public static int style = 1;
    Marker tmp;
    private Polyline currentPolyline;
    Location geolocation;
   DatabaseHelper databaseHelper;
    FusedLocationProviderClient fusedLocationProviderClient;
    // буфер для додання одного або 12ти найближчих маркерів
    //MarkerOptions[] markers= new MarkerOptions[12]; //використовувати в циклі foreach o->o.remove
    //можна замінити на mMap.clear() для витирання всієї карти враховуючи контури якщо будуть якісь додані
    KmlLayer layer;
    //    String[] styles={"Класична карта", "Silver", "Dark", "Night", "Aubergine"};
//              int[]    images={R.drawable.icon1style1,R.drawable.icon1style2,R.drawable.icon1style3,R.drawable.icon1style4,R.drawable.icon1style5};//додати потім ще два стиля і додати в масив
    ArrayList<CustomItem> styleList, modeList;

    @Override
    public void onTaskDone(Object... values) {
        if (currentPolyline != null)
            currentPolyline.remove();
        currentPolyline = mMap.addPolyline((PolylineOptions) values[0]);
    }



    private String getUrl(LatLng origin, LatLng dest) {
        // Ось цей код залишається без змін
        String str_origin = "origin=" + origin.latitude + "," + origin.longitude;
        String str_dest = "destination=" + dest.latitude + "," + dest.longitude;
        String mode = "mode=" + "driving";
        String parameters = str_origin + "&" + str_dest + "&" + mode;
        String output = "json";
        String url = "https://maps.googleapis.com/maps/api/directions/" + output + "?" +
                parameters + "&key=APIKEY";
```

```
        return url;
    }


    private boolean checkAndRequestLocationPermission() {
        if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) !=
PackageManager.PERMISSION_GRANTED &&
                ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) !=
PackageManager.PERMISSION_GRANTED) {

            if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED) {
                enableUserLocation();
            } else {
                ActivityCompat.requestPermissions(this, new
String[]{Manifest.permission.ACCESS_FINE_LOCATION},
ACCESS_LOCATION_REQUEST_CODE);
            }
            return false;
        }
        return true;
    }

    enum modeSelected {
        OneClosestShelter,
        ClosestShelterForAllStyles,
        FullMap
    }

    Object Mode = OneClosestShelter; //режим який обирається з енуму після обрання пукнту з
режимів роботи
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        com.example.myapplication.databinding.ActivityMapsBinding binding =
ActivityMapsBinding.inflate(getLayoutInflater());
        setContentView(binding.getRoot());
        fusedLocationProviderClient = LocationServices.getFusedLocationProviderClient(this);



            settingsButton = findViewById(R.id.button3);
            settingsButton.setOnClickListener(v -> {
                Intent intent = new Intent(context, SettingsMenu.class);
                startActivity(intent);
            });
        context = this;
        databaseHelper = new DatabaseHelper(context,mMap);//оголошення змінної *ЯКА Б* мала
допомагати з підключенням до бази даних та включенням інформації (полів) з неї
        SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.map);
        mapFragment.getMapAsync(this);
    }

    static List<Marker> markersList = new ArrayList<>(); // Список для маркерів

    // Метод для очищення маркерів

    @SuppressLint("PotentialBehaviorOverride")
    @Override
    public void onMapReady(@NonNull GoogleMap googleMap) {
        mMap = googleMap;
        initModeList();
        initStyleList();
        mMap.setInfoWindowAdapter(new GoogleMap.InfoWindowAdapter() {
            @Override
            public View getInfoWindow(Marker marker) {
```

```
            return null; // залишаємо стандартний фон інфо-вікна
        }

        @Override
        public View getInfoContents(Marker marker) {
            LinearLayout info = new LinearLayout(MapsActivity.this);
            info.setOrientation(LinearLayout.VERTICAL);
            info.setPadding(10, 10, 10, 10);

            TextView title = new TextView(MapsActivity.this);
            title.setText(marker.getTitle());
            title.setTypeface(null, Typeface.BOLD);
            title.setTextColor(Color.BLACK);

            TextView snippet = new TextView(MapsActivity.this);
            snippet.setText(HtmlCompat.fromHtml(marker.getSnippet(),
HtmlCompat.FROM_HTML_MODE_LEGACY));
            snippet.setTextColor(Color.DKGRAY);

            info.addView(title);
            info.addView(snippet);
            return info;
        }
    });

    //setOnMarkerClickListener
    mMap.setOnMarkerClickListener(marker -> {
        if (geolocation != null) {
            LatLng currentLatLng = new LatLng(geolocation.getLatitude(),
geolocation.getLongitude());
            LatLng destination = marker.getPosition();

            String url = getUrl(currentLatLng, destination);
            new FetchURL(MapsActivity.this).execute(url, "driving");
```

```
            Toast.makeText(this, "Прокладаю маршрут до: " + marker.getTitle(),
Toast.LENGTH_SHORT).show();
        }
        return false; // повертаємо false, щоб карта також показувала інфо-вікно
    });


    if (!checkAndRequestLocationPermission()) return;
    // Отримуємо останню локацію
    Task<Location> locationTask = fusedLocationProviderClient.getLastLocation();
    locationTask.addOnCompleteListener(this, task -> {
        if (task.isSuccessful() && task.getResult() != null) {
            geolocation = task.getResult();
            if (Mode.equals(modeSelected.OneClosestShelter)) {
                databaseHelper.findNearestMarker(geolocation, getApplicationContext());
            } else if (Mode.equals(modeSelected.ClosestShelterForAllStyles)) {
                databaseHelper.findNearestMarkerByStyleUrls(geolocation, getApplicationContext());
            } else if (Mode.equals(modeSelected.FullMap)) {// нічого
            }
        } else {
            Log.e("TAG", "Failed to get location");
            Toast.makeText(this, "Не вдалося отримати геолокацію",
Toast.LENGTH_SHORT).show();
        }
    });

    // Обробка кліку по мапі
    googleMap.setOnMapClickListener(latLng -> {
        if (tmp != null) tmp.remove();

        MarkerOptions markerOptions = new MarkerOptions().position(latLng).title(latLng.latitude
+ " : " + latLng.longitude).alpha(0);
        // Якщо геолокація доступна — будуємо маршрут
        if (geolocation != null) {
            new FetchURL(MapsActivity.this).execute(
```

```
                getUrl(new LatLng(geolocation.getLatitude(), geolocation.getLongitude()), latLng),
"driving"
            );
        } else {
            Toast.makeText(this, "Геолокація ще не визначена", Toast.LENGTH_SHORT).show();
        }

        googleMap.animateCamera(CameraUpdateFactory.newLatLng(latLng));
        tmp = googleMap.addMarker(markerOptions);
    });

    // Налаштування modeSpinner
    Spinner modeSpinner = findViewById(R.id.spinner2);
    CustomAdapter modesAdapter = new CustomAdapter(context, modeList);
    modeSpinner.setAdapter(modesAdapter);
    modeSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
        @Override
        public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {
            CustomItem clickedItem = (CustomItem) adapterView.getItemAtPosition(i);
            Toast.makeText(MapsActivity.this, clickedItem.getCustomName() + " обрано!",
Toast.LENGTH_SHORT).show();

            if (layer != null) {
                layer.removeLayerFromMap();
            }

            switch (i) {
                case 0:
                    mMap.clear();
                    Mode = OneClosestShelter;
                    databaseHelper.findNearestMarker(geolocation, context);
                    break;
                case 1:
                    if(layer!=null){
                        layer.removeLayerFromMap();
```

```
                }
                mMap.clear();
                Mode = ClosestShelterForAllStyles;
                databaseHelper.findNearestMarkerByStyleUrls(geolocation, context);
                break;
            case 2:
                if(layer!=null){
                    layer.removeLayerFromMap();
                }
                mMap.clear();
                Mode = modeSelected.FullMap;
                loadLayer();
                break;
        }
    }

    @Override
    public void onNothingSelected(AdapterView<?> adapterView) {
        if (layer != null) {
            layer.removeLayerFromMap();
        }
        Mode = modeSelected.FullMap;
    }
});

// Налаштування styleSpinner
Spinner styleSpinner = findViewById(R.id.spinner);
CustomAdapter adapter = new CustomAdapter(context, styleList);
styleSpinner.setAdapter(adapter);
styleSpinner.setOnItemSelectedListener(new AdapterView.OnItemSelectedListener() {
    @Override
    public void onItemSelected(AdapterView<?> parent, View view, int position, long id) {
        CustomItem clickedItem = (CustomItem) parent.getItemAtPosition(position);
        Toast.makeText(MapsActivity.this, clickedItem.getCustomName() + " обрано!",
Toast.LENGTH_SHORT).show();
```

```
        switch (position) {
            case 0:
                style = 1;
                changeStyle(R.raw.custom_map_style, R.raw.markersstyle1);
                break;
            case 1:
                style = 2;
                changeStyle(R.raw.custom_map_style2, R.raw.markersstyle2);
                break;
            case 2:
                style = 3;
                changeStyle(R.raw.custom_map_style3, R.raw.markersstyle3);
                break;
            case 3:
                style = 4;
                changeStyle(R.raw.custom_map_style4, R.raw.markersstyle4);
                break;
            case 4:
                style = 5;
                changeStyle(R.raw.custom_map_style5, R.raw.markersstyle5);
                break;
        }
    }

    @Override
    public void onNothingSelected(AdapterView<?> parent) {
        style = 1;
        changeStyle(R.raw.custom_map_style, R.raw.markersstyle1);
    }
});

// Фокус на Вінниці
googleMap.setMinZoomPreference(7.5f);
LatLng Vinnitsia = new LatLng(49.102275, 28.675528);
```

```
    googleMap.moveCamera(CameraUpdateFactory.newLatLng(Vinnitsia));

    if (checkAndRequestLocationPermission()) {
        googleMap.setMyLocationEnabled(true);
    }
}

private void loadLayer() {
    int resourceId = R.raw.markersstyle1;

    if (style == 2) resourceId = R.raw.markersstyle2;
    else if (style == 3) resourceId = R.raw.markersstyle3;
    else if (style == 4) resourceId = R.raw.markersstyle4;
    else if (style == 5) resourceId = R.raw.markersstyle5;

    try {
        layer = new KmlLayer(mMap, resourceId, getApplicationContext());
        layer.addLayerToMap();

    } catch (XmlPullParserException | IOException e) {
        e.printStackTrace();
    }
}
```

```
public String downloadUrl(String strUrl) throws IOException {
    String data = "";
    InputStream iStream = null;
    HttpURLConnection urlConnection = null;
```

```
        try {
            URL url = new URL(strUrl);
            urlConnection = (HttpURLConnection) url.openConnection();
            urlConnection.connect();

            iStream = urlConnection.getInputStream();
            BufferedReader br = new BufferedReader(new InputStreamReader(iStream));
            StringBuffer sb = new StringBuffer();
            String line;
            while ((line = br.readLine()) != null) {
                sb.append(line);
            }
            data = sb.toString();
            Log.d("API Response", "Response: " + data);
            br.close();
        } catch (Exception e) {
            Log.d("mylog", "Exception downloading URL: " + e.toString());
        } finally {
            iStream.close();
            urlConnection.disconnect();
        }
        return data;
    }


    private void initModeList(){
        modeList= new ArrayList<>();
        String[] modes={"Найближче сховище","Найближчі сховища кожного типу", "Повна мапа
сховищ"};
        int[] images={R.drawable.map_marker,R.drawable.map_marker_multiple,R.drawable.map};
        for (int i = 0; i < modes.length; i++) {
            modeList.add(new CustomItem(modes[i],images[i]));
        }
    }
    private void initStyleList() {
```

```
        styleList=new ArrayList<>();
        String[] styles={"Класична карта", "Silver", "Aubergine", "Night","Dark"};
        int[]       images={R.drawable.icon10style1,R.drawable.icon10style2,R.drawable.icon10style5,
R.drawable.icon10style4,R.drawable.icon10style3};//додати потім ще два стиля і додати в масив
        for (int i = 0; i < styles.length; i++) {
            styleList.add(new CustomItem(styles[i],images[i]));
        }
    }




    @Override
    protected void onDestroy() {
//        databaseHelper.close();
        super.onDestroy();
    }

    private void enableUserLocation() {
        if                                                  (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION)                                                  !=
PackageManager.PERMISSION_GRANTED) {
            mMap.setMyLocationEnabled(true);
        }

        //Перемістив nMap звідси наверх, не крашиться, але все потрібно перезапуск програми
для того щоб все працювало...
        //...тому думаю проблема не в цій штуковині, но це треба ще провірити
    }
    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions,
@NonNull int[] grantResults) {
```

```
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);
        if (requestCode == ACCESS_LOCATION_REQUEST_CODE) {
            if (grantResults.length > 0 && grantResults[0]==PackageManager.PERMISSION_GRANTED){
                enableUserLocation();
            }
            else{
                //вивести повідомлення що користувач не зможе користуватись всіма функціями додатку не надавши доступу до місцезнаходження
                new AlertDialog.Builder(this)
                        .setTitle("Permission needed")
                        .setMessage("This permisiion is needed because you won't have all option enabled")
                        .setNegativeButton("ok", (dialog, which) -> dialog.dismiss())
                        .create().show();
            }
        }
    }

    @Override
    public void onItemSelected(AdapterView<?> adapterView, View view, int i, long l) {
        //для зміни стилів
        //порядок: (порядок вкотре чомусь збився, хоча ролі особої не грає, хіба що змінити ім'я файлів)
        //1 звичайний вид мапи (контурний, обрано в додатку за замовчуванням)
        //2.Dark
        //3.Aubergine
        //4.Night
        //5.Silver

        switch (i) {
            case 0:
                style=1;
                changeStyle(R.raw.custom_map_style,R.raw.markersstyle1);
                break;
            case 1:
```

```
            style=2;
            changeStyle(R.raw.custom_map_style2,R.raw.markersstyle2);
            break;
        case 2:
            style=3;
            changeStyle(R.raw.custom_map_style3,R.raw.markersstyle3);
            break;
        case 3:
            style=4;
            changeStyle(R.raw.custom_map_style4,R.raw.markersstyle4);
            break;
        case 4:
            style=5;
            changeStyle(R.raw.custom_map_style5,R.raw.markersstyle5);
            break;
    }
}

public void changeStyle(int style, int kmz){
    if(layer!=null)
        layer.removeLayerFromMap();
    setMapStyle(MapStyleOptions.loadRawResourceStyle(context,style));
    if(Mode==modeSelected.FullMap) {
        try {
            layer = new KmlLayer(mMap, kmz, getApplicationContext());
            layer.addLayerToMap();
        } catch (XmlPullParserException | IOException e) {
            throw new RuntimeException(e);
        }
    }
}

@Override
public void onNothingSelected(AdapterView<?> adapterView) {
    style=1;
```

```
        changeStyle(R.raw.custom_map_style,R.raw.markersstyle1);
    }


    private void setMapStyle(MapStyleOptions style) {
        if (mMap != null) {
            mMap.setMapStyle(style);//застосування обаного стилю
        }
    }
}
```

Модуль «CustomAdapter»:

```
package com.example.myapplication;

import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.ImageView;
import android.widget.TextView;

import androidx.annotation.NonNull;

import java.util.ArrayList;

//це адаптер для того аби виводити крім тексту картинку або картинку і текст
//використання: позначити вибір режимів роботи та стилів картинками

public class CustomAdapter extends ArrayAdapter<CustomItem> {
    Context context;
```

```
    String[] styles;
    int[] images;


    public CustomAdapter(Context context, ArrayList<CustomItem> itemArrayList) {
        super(context,0, itemArrayList);
    }
    @NonNull
    @Override
    public View getDropDownView(int positions, View convertView, @NonNull
ViewGroup parent) {
        return initView(positions, convertView, parent);
    }

    @Override
    public View getView(int position, View convertView, ViewGroup parent) {
        return initView(position, convertView, parent);
    }

    private View initView(int position,View convertView, ViewGroup parent){
        if(convertView==null){

convertView=LayoutInflater.from(getContext()).inflate(R.layout.spinner_item,parent,
false);
        }
        ImageView imageView=convertView.findViewById(R.id.image_view);
        TextView textView=convertView.findViewById(R.id.text_view);
        CustomItem currentItem=getItem(position);
        if(currentItem!=null) {
            imageView.setImageResource(currentItem.getCustomNum());
```

```
            textView.setText(currentItem.getCustomName());
        }
        return convertView;
    }
}
```

Модуль «CustomItem»:

```
package com.example.myapplication;

public class CustomItem {
    private String customName;
    private int customNum;

    public CustomItem(String customName, int customNum) {
        this.customName = customName;
        this.customNum = customNum;
    }

    public String getCustomName() {
        return customName;
    }

    public int getCustomNum() {
        return customNum;
    }
}
```

Модуль «DatabaseHelper»:

```
package com.example.myapplication;
```

```
import static com.example.myapplication.MapsActivity.mMap;
import static com.example.myapplication.MapsActivity.markersList;

import android.content.Context;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.graphics.Bitmap;
import android.graphics.drawable.BitmapDrawable;
import android.location.Location;
import android.util.Log;
import android.widget.Toast;

import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.model.BitmapDescriptorFactory;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.Marker;
import com.google.android.gms.maps.model.MarkerOptions;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

// клас для з'єднання з базою даних (не працює належним чином чомусь, під'єднується до бази даних і до таблиці, але проходячи курсором через рядки видає що таблиця має 0 рядків)
public class DatabaseHelper extends SQLiteOpenHelper {
```

```
private static final String DATABASE_NAME = "markersdb";

private static final int DATABASE_VERSION = 1;

public static final String TABLE_NAME = "Markers";
public static final String COLUMN_NAME = "name";
public static final String COLUMN_DESCRIPTION = "description";
public static final String COLUMN_LATITUDE = "lat";
public static final String COLUMN_LONGITUDE = "lng";
public static final String COLUMN_STYLEURL = "styleUrl";



public DatabaseHelper(Context context, GoogleMap map) {
   super(context, DATABASE_NAME, null, DATABASE_VERSION);
   fillDatabase(context);
}


@Override
public void onCreate(SQLiteDatabase db) {
   String sql = "CREATE TABLE IF NOT EXISTS " + TABLE_NAME + " (" +
         COLUMN_NAME + " TEXT, " +
         COLUMN_DESCRIPTION + " TEXT, " +
         COLUMN_LATITUDE + " REAL, " +
         COLUMN_LONGITUDE + " REAL, " +
         COLUMN_STYLEURL + " TEXT);";
   db.execSQL(sql);
}
```

```
public void fillDatabase(Context context){
    // Копіюємо базу даних з ресурсів в папку додатку
    //МЕТОД ДЛЯ ДОДАННЯ БАЗИ ДАНИХ НА НОСІЙ, ДОДАТИ ПЕРЕВІРКУ ЧИ ІСНУЄ ФАЙЛ ПЕРЕД СТВОРЕННЯМ НОВОЇ,
    try {
        // Відкриваємо вхідний потік для бази даних у папці assets
        InputStream inputStream = context.getResources().openRawResource(R.raw.markersdb);
        // Визначаємо шлях для копіювання бази даних в папку додатку
        String outDir = context.getFilesDir().getPath() + "/databases/markersdb.db";
        File dir = new File(outDir);
        if (!dir.exists()) {
            dir.mkdirs(); // Створюємо папку database, якщо вона не існує
        }
        String outFileName = context.getDatabasePath(DATABASE_NAME).getPath();
        // Створюємо вихідний потік для копіювання бази даних
        OutputStream outputStream = new FileOutputStream(outFileName);

        // Копіюємо дані з вхідного потоку в вихідний потік
        byte[] buffer = new byte[1024*4];
        int length;
        while ((length = inputStream.read(buffer)) > 0) {
            outputStream.write(buffer, 0, length);
        }

        // Закриваємо потоки
        outputStream.flush();
        outputStream.close();
        inputStream.close();
```

```
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
//        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
//        onCreate(db);
    }

    public void findNearestMarker(Location lastLocation, Context context) {
        if (lastLocation == null) {
            Log.e("DatabaseHelper", "lastLocation is null, cannot find nearest marker.");
            Toast.makeText(context, "Неможливо отримати місцезнаходження. Спробуйте пізніше.", Toast.LENGTH_SHORT).show();
            return;
        }

        SQLiteDatabase sqLiteDatabase = this.getReadableDatabase();

        try (Cursor result = sqLiteDatabase.rawQuery("SELECT * FROM " + TABLE_NAME, null)) {
            if (result.getCount() != 0) {
                double minDistance = Double.MAX_VALUE;
                MarkerOptions nearestMarker = null;

                while (result.moveToNext()) {
```

```
String styleUrl = result.getString(result.getColumnIndexOrThrow(DatabaseHelper.COLUMN_STYLE URL));
double lat = result.getDouble(result.getColumnIndexOrThrow(DatabaseHelper.COLUMN_LATIT UDE));
double lng = result.getDouble(result.getColumnIndexOrThrow(DatabaseHelper.COLUMN_LONG ITUDE));
int height = 100;
int width = 100;

float[] results = new float[1];
Location.distanceBetween(lastLocation.getLatitude(), lastLocation.getLongitude(), lat, lng, results);
double distance = results[0];

if (distance < minDistance) {
   minDistance = distance;
   BitmapDrawable bitmapdraw = (BitmapDrawable) context.getResources().getDrawable(getIconForStyleUrl(styleUrl, context));
   Bitmap b = bitmapdraw.getBitmap();
   Bitmap smallMarker = Bitmap.createScaledBitmap(b, width, height, false);
   nearestMarker = new MarkerOptions()
         .position(new LatLng(lat, lng))

.title(result.getString(result.getColumnIndexOrThrow(DatabaseHelper.COLUMN_N AME)))
```

```
.snippet(result.getString(result.getColumnIndexOrThrow(DatabaseHelper.COLUMN
_DESCRIPTION)))
                        .icon(BitmapDescriptorFactory.fromBitmap(smallMarker));
                }
            }

            if (nearestMarker != null) {
                mMap.addMarker(nearestMarker);
            }
        }
    }
  }


  public void findNearestMarkerByStyleUrls(Location mLastKnownLocation,
Context context) {
    String[] styleUrls = {"1","2","3","4","5","6","7","8","9","10","11","12"};

    for (String styleUrl : styleUrls) {
      SQLiteDatabase db = this.getReadableDatabase();

      String[] projection = {
            DatabaseHelper.COLUMN_NAME,
            DatabaseHelper.COLUMN_DESCRIPTION,
            DatabaseHelper.COLUMN_LATITUDE,
            DatabaseHelper.COLUMN_LONGITUDE,
            DatabaseHelper.COLUMN_STYLEURL
      };
```

```
            String selection = DatabaseHelper.COLUMN_STYLEURL + "=?";
            String[] selectionArgs = { styleUrl };

            String sortOrder = DatabaseHelper.COLUMN_NAME + " DESC";

            Cursor cursor = db.query(
                    DatabaseHelper.TABLE_NAME,
                    projection,
                    selection,
                    selectionArgs,
                    null,
                    null,
                    sortOrder
            );

            findNearestMarkerByUrl(cursor, mLastKnownLocation, context);
        }
    }

    private Marker findNearestMarkerByUrl(Cursor cursor, Location
mLastKnownLocation, Context context) {
        double currentLat = mLastKnownLocation.getLatitude();
        double currentLng = mLastKnownLocation.getLongitude();
        double minDistance = Double.MAX_VALUE;
        Marker nearestMarker = null;


        while (cursor.moveToNext()) {
```

```
        String styleUrl = cursor.getString(cursor.getColumnIndexOrThrow(DatabaseHelper.COLUMN_STYLEURL));
        double lat = cursor.getDouble(cursor.getColumnIndexOrThrow(DatabaseHelper.COLUMN_LATITUDE));
        double lng = cursor.getDouble(cursor.getColumnIndexOrThrow(DatabaseHelper.COLUMN_LONGITUDE));
        int height = 75;
        int width = 75;
        Bitmap b = null;
        Bitmap smallMarker;
        float[] results = new float[1];
        Location.distanceBetween(currentLat, currentLng, lat, lng, results);
        double distance = results[0];
        if (distance < minDistance) {
           minDistance = distance;
           BitmapDrawable bitmapdraw = (BitmapDrawable)context.getResources().getDrawable(getIconForStyleUrl(styleUrl,context));
           b = bitmapdraw.getBitmap();
           smallMarker = Bitmap.createScaledBitmap(b, width, height, false);
           nearestMarker = mMap.addMarker(new MarkerOptions()
                 .position(new LatLng(lat, lng))

.title(cursor.getString(cursor.getColumnIndexOrThrow(DatabaseHelper.COLUMN_NAME)))
```

```
.snippet(cursor.getString(cursor.getColumnIndexOrThrow(DatabaseHelper.COLUM
N_DESCRIPTION)))
                    .icon(BitmapDescriptorFactory.fromBitmap(smallMarker)).alpha(0));



            }
        }
        nearestMarker.setAlpha(1);
        return nearestMarker;
    }

    private int getIconForStyleUrl(String styleUrl, Context context) { //провірити
        int iconNumber;
        try {
            iconNumber = Integer.parseInt(styleUrl);
        } catch (NumberFormatException e) {
            // Неможливо перетворити номер в ціле число
            return -1;
        }
        String iconName = "icon" + iconNumber+"style"+MapsActivity.style;
        return context.getResources().getIdentifier(iconName, "drawable",
context.getPackageName());
    }
}
```

Модуль «SettingsMenu»:

```
package com.example.myapplication;

import android.app.AlertDialog;
```

```
import android.app.Notification;
import android.app.NotificationChannel;
import android.app.NotificationManager;
import android.content.Context;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.widget.Button;

import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.NotificationCompat;

public class SettingsMenu extends AppCompatActivity {
    private static final String CHANNEL_ID = "MY_CHANNEL_ID"; // Unique ID for the notification channel
    private static final int NOTIFICATION_ID = 123; // Unique ID for the notification

    Button button, about, help;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        button = findViewById(R.id.button);
        button.setOnClickListener(view -> {
            Handler handler = new Handler();
            handler.postDelayed(this::showNotification, 5000); // виведення сповіщення через 5 секунд
        });
```

```
        about=findViewById(R.id.btn_about);
        about.setOnClickListener(view -> {
            new AlertDialog.Builder(this)
                    .setTitle("Про програму")
                    .setMessage("Розробник: Мисловський Антон 2ПІ-21б\n\nДодаток допомагає знаходити найближчі укриття та прокладати до них маршрут на карті. Розроблено для забезпечення безпеки користувачів.")
                    .setPositiveButton("ОК", null)
                    .show();
        });
        help=findViewById(R.id.btn_help);
        help.setOnClickListener(view -> {
            new AlertDialog.Builder(this)
                    .setTitle("Довідка")
                    .setMessage("Щоб скористатись додатком, оберіть укриття на карті або знайдіть найближче. Для побудови маршруту натисніть на маркер і оберіть 'Прокласти маршрут'.")
                    .setPositiveButton("ОК", null)
                    .show();
        });
    }
    //TODO:
    private void showNotification() {
        // створення менеджеру сповіщень
        NotificationManager notificationManager = (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

        // створення каналу для сповіщення (required for Android Oreo and later)
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
```

```
        NotificationChannel channel = new NotificationChannel(CHANNEL_ID, "My
Channel", NotificationManager.IMPORTANCE_DEFAULT);
        notificationManager.createNotificationChannel(channel);
    }

    Notification notification = new NotificationCompat.Builder(this,
CHANNEL_ID)
            .setSmallIcon(R.drawable.ic_launcher_foreground)
            .setContentTitle("\uD83D\uDEA8ПОВІТРЯНА
ТРИВОГА!\uD83D\uDEA8")
            .setContentText("Найближче сховище за адресою: ")
            .setPriority(NotificationCompat.PRIORITY_DEFAULT)
            .build();

    // вивести сповіщення
    notificationManager.notify(NOTIFICATION_ID, notification);
  }
}
```

Модуль «DataParser»:

```
package com.example.myapplication.directionhelpers;

import com.google.android.gms.maps.model.LatLng;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

import java.util.ArrayList;
import java.util.HashMap;
```

```
import java.util.List;


public class DataParser {
    public List<List<HashMap<String, String>>> parse(JSONObject jObject) {

        List<List<HashMap<String, String>>> routes = new ArrayList<>();
        JSONArray jRoutes;
        JSONArray jLegs;
        JSONArray jSteps;
        try {
            jRoutes = jObject.getJSONArray("routes");
            /** Traversing all routes */
            for (int i = 0; i < jRoutes.length(); i++) {
                jLegs = ((JSONObject) jRoutes.get(i)).getJSONArray("legs");
                List path = new ArrayList<>();
                /** Traversing all legs */
                for (int j = 0; j < jLegs.length(); j++) {
                    jSteps = ((JSONObject) jLegs.get(j)).getJSONArray("steps");

                    /** Traversing all steps */
                    for (int k = 0; k < jSteps.length(); k++) {
                        String polyline = "";
                        polyline = (String) ((JSONObject) ((JSONObject)
jSteps.get(k)).get("polyline")).get("points");
                        List<LatLng> list = decodePoly(polyline);

                        /** Traversing all points */
                        for (int l = 0; l < list.size(); l++) {
                            HashMap<String, String> hm = new HashMap<>();
```

```
                    hm.put("lat", Double.toString((list.get(l)).latitude));
                    hm.put("lng", Double.toString((list.get(l)).longitude));
                    path.add(hm);
                }
            }
            routes.add(path);
        }
    }

} catch (JSONException e) {
    e.printStackTrace();
} catch (Exception e) {
}
return routes;
}


/**
 * Method to decode polyline points
 * Courtesy : https://jeffreysambells.com/2010/05/27/decoding-polylines-from-google-maps-direction-api-with-java
 */
private List<LatLng> decodePoly(String encoded) {
    List<LatLng> poly = new ArrayList<>();
    int index = 0, len = encoded.length();
    int lat = 0, lng = 0;
    while (index < len) {
        int b, shift = 0, result = 0;
        do {
            b = encoded.charAt(index++) - 63;
```

```
            result |= (b & 0x1f) << shift;
            shift += 5;
        } while (b >= 0x20);
        int dlat = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
        lat += dlat;

        shift = 0;
        result = 0;
        do {
            b = encoded.charAt(index++) - 63;
            result |= (b & 0x1f) << shift;
            shift += 5;
        } while (b >= 0x20);
        int dlng = ((result & 1) != 0 ? ~(result >> 1) : (result >> 1));
        lng += dlng;

        LatLng p = new LatLng((((double) lat / 1E5)),
                (((double) lng / 1E5)));
        poly.add(p);
      }
      return poly;
   }
}
```

Модуль «FetchURL»:

```
package com.example.myapplication.directionhelpers;

import android.content.Context;
import android.os.AsyncTask;
import android.util.Log;
```

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;


public class FetchURL extends AsyncTask<String, Void, String> {
    Context mContext;
    String directionMode = "driving";

    public FetchURL(Context mContext) {
        this.mContext = mContext;
    }

    @Override
    protected String doInBackground(String... strings) {
        // For storing data from web service
        String data = "";
        directionMode = strings[1];
        try {
            // Fetching the data from web service
            data = downloadUrl(strings[0]);
            Log.d("mylog", "Background task data " + data.toString());
        } catch (Exception e) {
            Log.d("Background Task", e.toString());
        }
        return data;
```

```
}

@Override
protected void onPostExecute(String s) {
    super.onPostExecute(s);
    PointsParser parserTask = new PointsParser(mContext, directionMode);
    // Invokes the thread for parsing the JSON data
    parserTask.execute(s);
}

private String downloadUrl(String strUrl) throws IOException {
    String data = "";
    InputStream iStream = null;
    HttpURLConnection urlConnection = null;
    try {
        URL url = new URL(strUrl);
        // Creating an http connection to communicate with url
        urlConnection = (HttpURLConnection) url.openConnection();
        // Connecting to url
        urlConnection.connect();
        // Reading data from url
        iStream = urlConnection.getInputStream();
        BufferedReader br = new BufferedReader(new InputStreamReader(iStream));
        StringBuffer sb = new StringBuffer();
        String line = "";
        while ((line = br.readLine()) != null) {
            sb.append(line);
        }
        data = sb.toString();
        Log.d("mylog", "Downloaded URL: " + data);
```

```
            br.close();
        } catch (Exception e) {
            Log.d("mylog", "Exception downloading URL: " + e);
        } finally {
            assert iStream != null;
            iStream.close();
            urlConnection.disconnect();
        }
        return data;
    }
}
```

Модуль «PointsParser»:

```
package com.example.myapplication.directionhelpers;

import static com.example.myapplication.MapsActivity.style;

import android.content.Context;
import android.graphics.Color;
import android.os.AsyncTask;
import android.util.Log;

import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.PolylineOptions;

import org.json.JSONObject;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
```

```
public class PointsParser extends AsyncTask<String, Integer, List<List<HashMap<String, String>>>> {
    TaskLoadedCallback taskCallback;
    String directionMode = "driving";

    public PointsParser(Context mContext, String directionMode) {
        this.taskCallback = (TaskLoadedCallback) mContext;
        this.directionMode = directionMode;
    }

    // Parsing the data in non-ui thread
    @Override
    protected List<List<HashMap<String, String>>> doInBackground(String... jsonData) {

        JSONObject jObject;
        List<List<HashMap<String, String>>> routes = null;

        try {
            jObject = new JSONObject(jsonData[0]);
            Log.d("mylog", jsonData[0].toString());
            DataParser parser = new DataParser();
            Log.d("mylog", parser.toString());

            // Starts parsing data
            routes = parser.parse(jObject);
            Log.d("mylog", "Executing routes");
            Log.d("mylog", routes.toString());
```

```
    } catch (Exception e) {
        Log.d("mylog", e.toString());
        e.printStackTrace();
    }
    return routes;
}

// Executes in UI thread, after the parsing process
@Override
protected void onPostExecute(List<List<HashMap<String, String>>> result) {
    ArrayList<LatLng> points;
    PolylineOptions lineOptions = null;
    // Traversing through all the routes
    for (int i = 0; i < result.size(); i++) {
        points = new ArrayList<>();
        lineOptions = new PolylineOptions();
        // Fetching i-th route
        List<HashMap<String, String>> path = result.get(i);
        // Fetching all the points in i-th route
        for (int j = 0; j < path.size(); j++) {
            HashMap<String, String> point = path.get(j);
            double lat = Double.parseDouble(point.get("lat"));
            double lng = Double.parseDouble(point.get("lng"));
            LatLng position = new LatLng(lat, lng);
            points.add(position);
        }
        // Adding all the points in the route to LineOptions
        lineOptions.addAll(points);
        if (style==1) lineOptions.width(15).color(Color.BLUE);
```

```
            if (style==2) lineOptions.width(15).color(Color.DKGRAY);
            if (style==3) lineOptions.width(15).color(Color.CYAN);
            if (style==4) lineOptions.width(15).color(Color.YELLOW);
            if (style==5) lineOptions.width(15).color(Color.WHITE);
            Log.d("mylog", "onPostExecute lineoptions decoded");
        }

        // Drawing polyline in the Google Map for the i-th route
        if (lineOptions != null) {
            //mMap.addPolyline(lineOptions);
            taskCallback.onTaskDone(lineOptions);

        } else {
            Log.d("mylog", "without Polylines drawn");
        }
    }
}
```

Модуль «OnTaskDone»:

```
package com.example.myapplication.directionhelpers;

public interface TaskLoadedCallback {
    void onTaskDone(Object... values);
}
```

ДОДАТОК Г
Ілюстративна частина

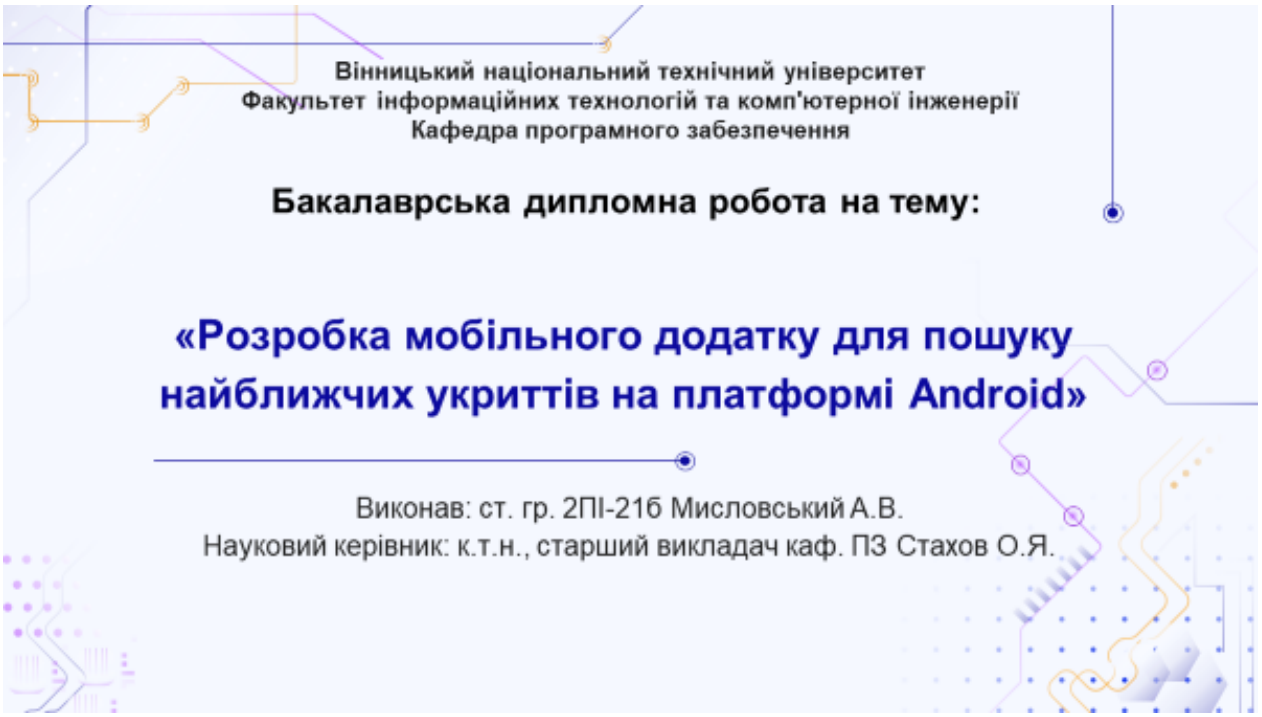

Рисунок Г.1 – Титульний слайд

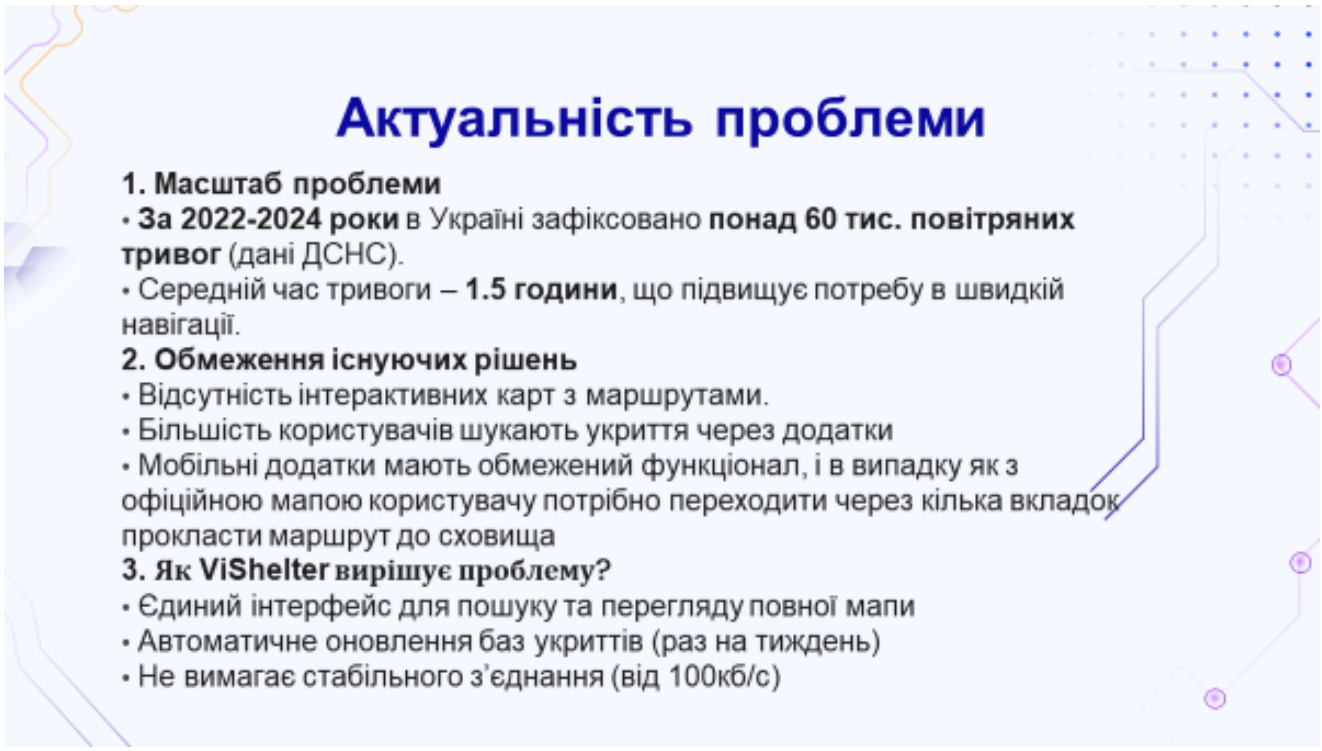

Рисунок Г.2 – Актуальність теми

Рисунок Г.3 – Мета, об’єкт та предмет розробки

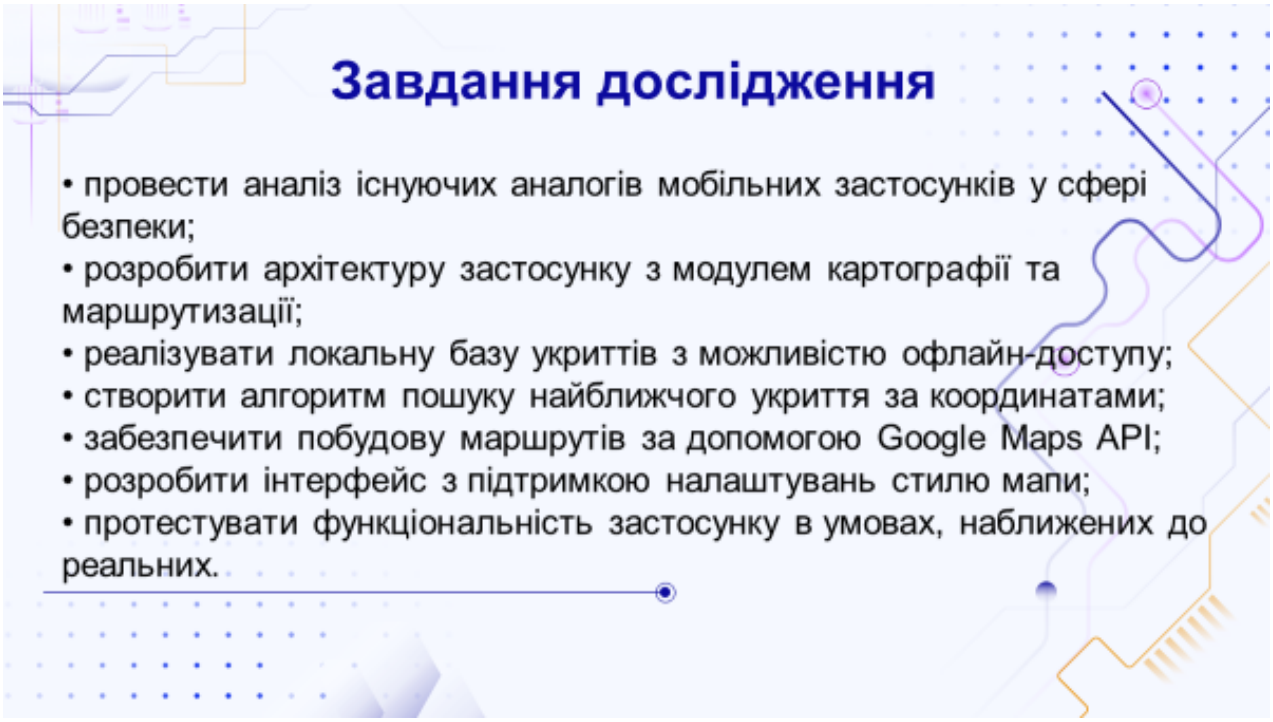

Рисунок Г.4 – Задачі дослідження

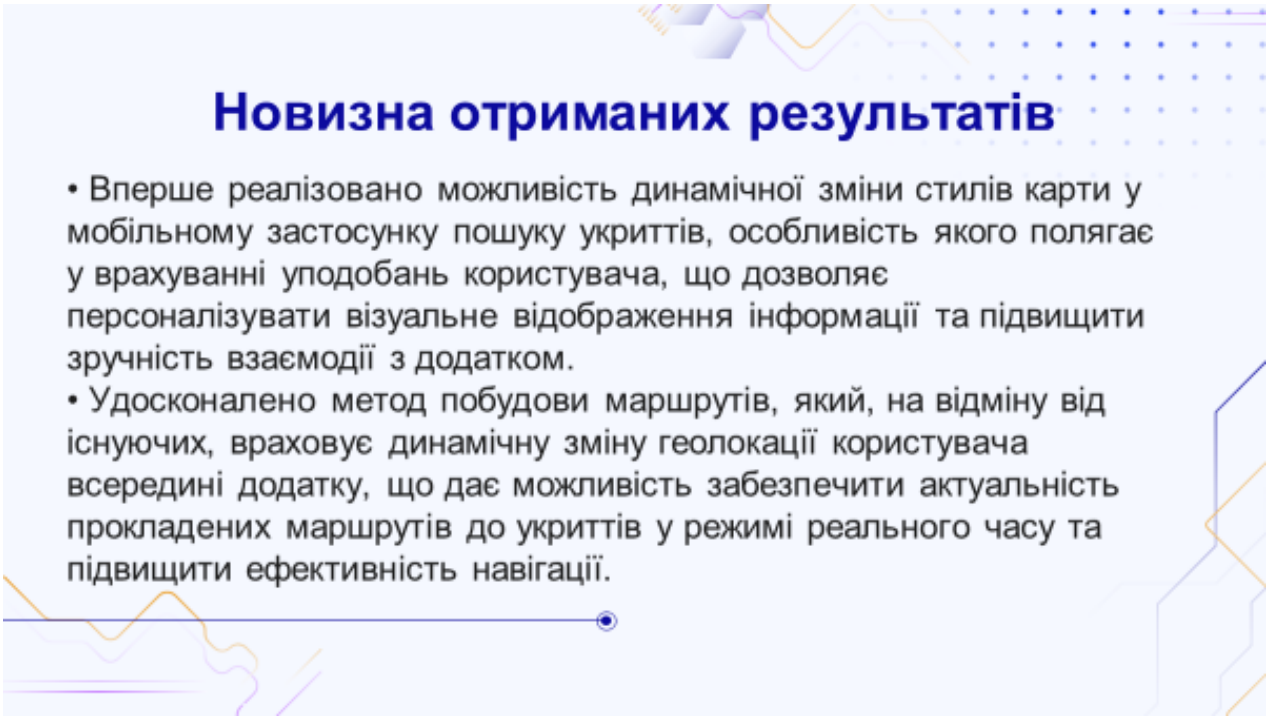

Рисунок Г.5 – Новизна отриманих результатів

Рисунок Г.6 – Практична цінність отриманих результатів

Порівняльний аналіз програм-аналогів

Критерії \ Додатки	«Укриття Кременчук»	«ДеУкриття»	«Укриття Вінниця»	«ViShelter»
Виведення місцезнаходження користувача на мапі	-	+	-	+
Прокладання маршруту	-	-	-	+
Вбудований пошук найближчого укриття	+	-	+	+
Довідка про поведінку під час екстрених ситуацій	-	+	-	-
Джерела води та пункти незламності	+	+	+	-
Вибір стилізації мапи	-	-	-	+
Сумарний коефіцієнт	2	3	2	4

Рисунок Г.7 – Порівняльний аналіз програм-аналогів

Модель системи

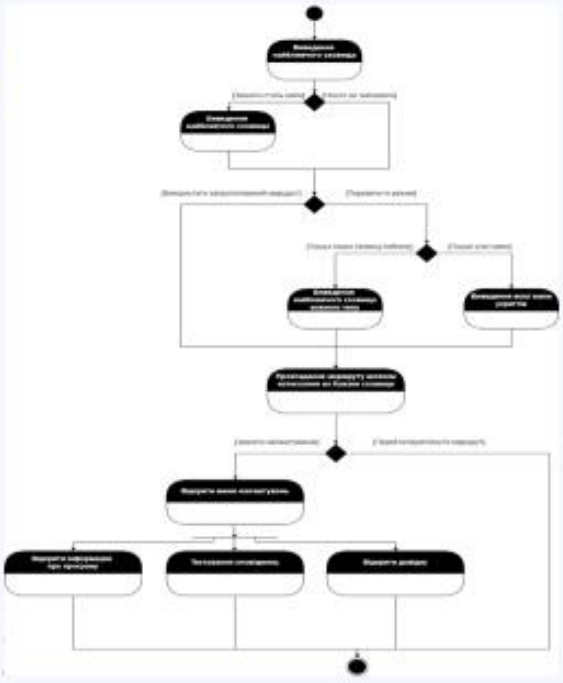

Рисунок Г.8 – Модель системи

Рисунок Г.10 – Метод і блок-схема алгоритму визначення маршруту між користувачем та маркером

Рисунок Г.11 – Метод і блок-схема алгоритму пошуку найближчого маркеру

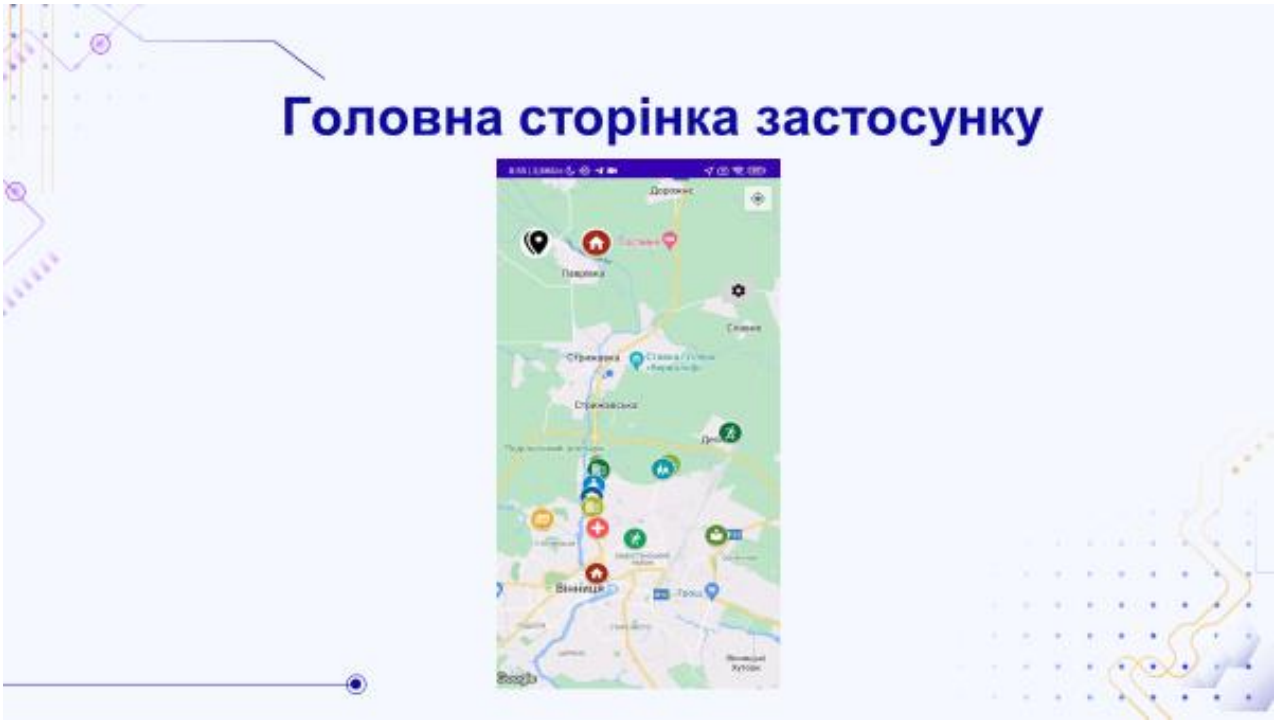

Рисунок Г.13 – Головна сторінка застосунку

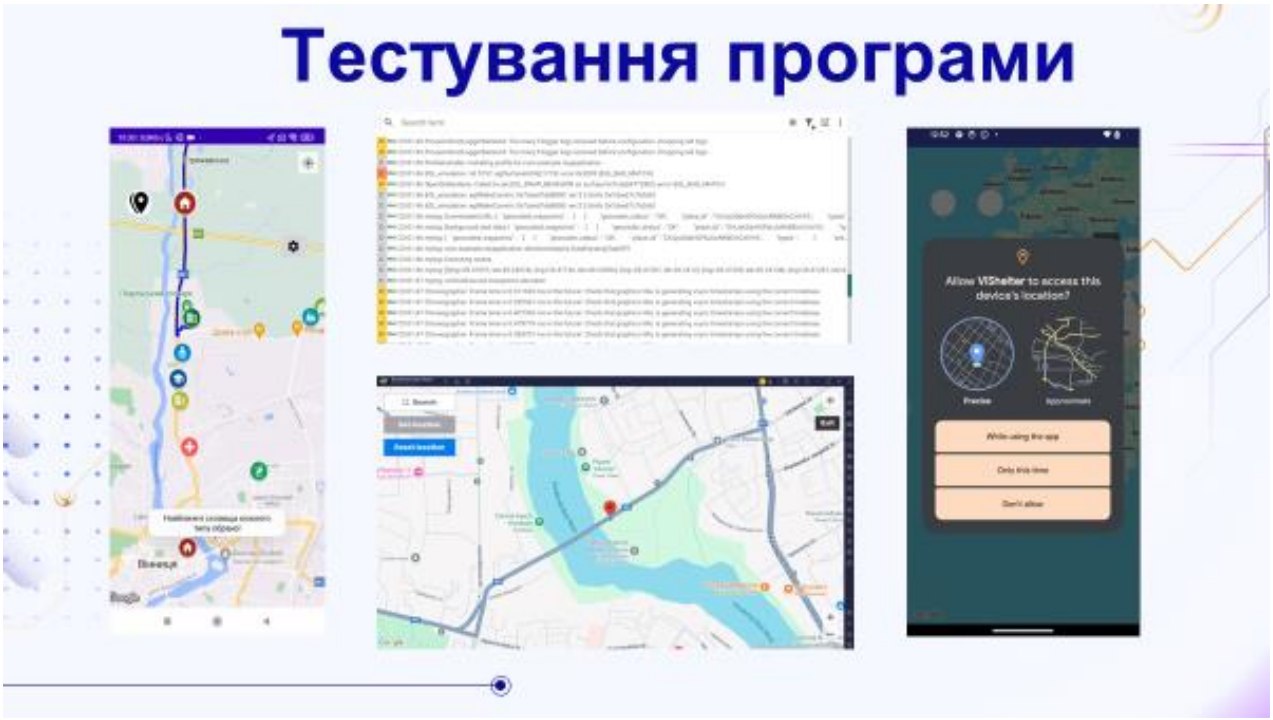

Рисунок Г.15 – Тестування програми

Рисунок Г.16 – Апробація та публікація матеріалів бакалаврської кваліфікаційної роботи

Рисунок Г.17 – Висновки

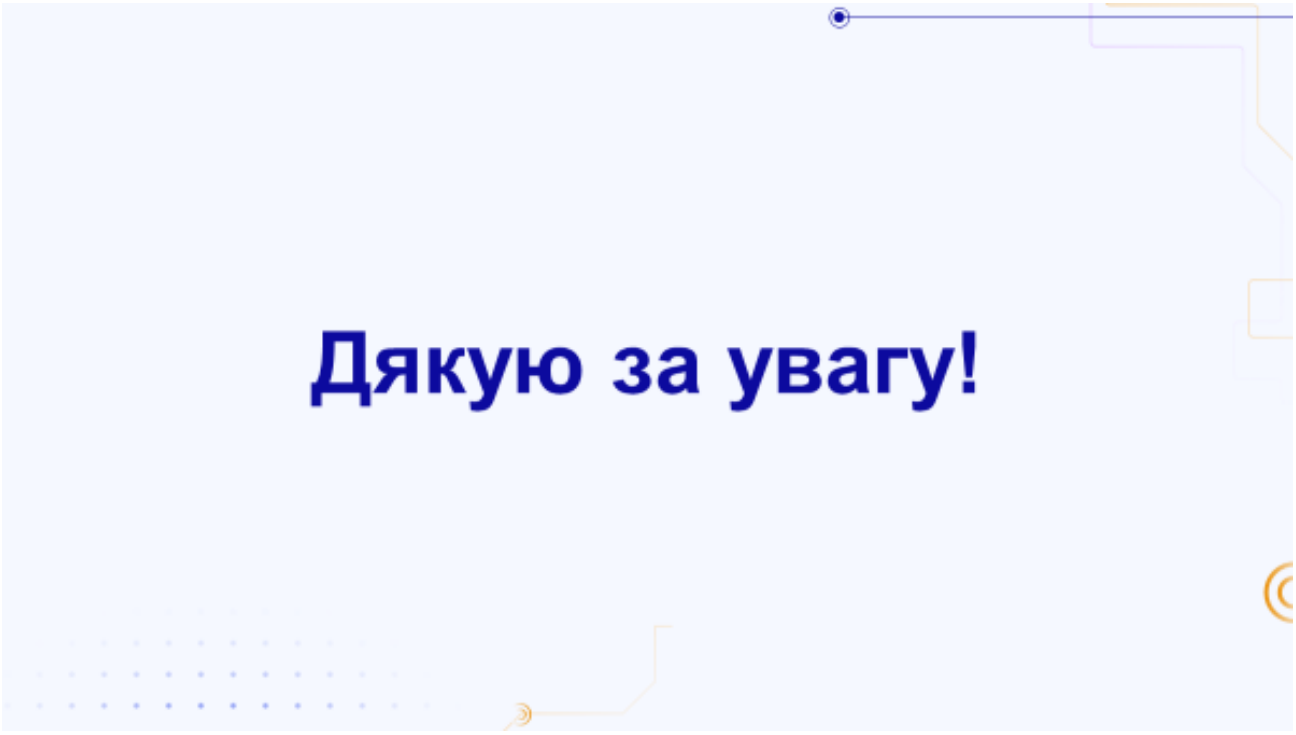

Рисунок Г.18 – Фінальний слайд