Прикарпатський національний університет імені Василя Стефаника
Факультет математики та інформатики
Кафедра комп’ютерних наук та інформаційних систем

ДИПЛОМНА РОБОТА

Тема: «Розробка мобільного застосунку-довідника Українських Карпат»

Виконав: студент IV курсу гр. ІСТ-41
ОР бакалавр
спеціальності 126 “Інформаційні системи та технології”
Хом’як Євген Васильович

Науковий керівник: к.т.н., доц.
Превисокова Наталія Володимирівна

Рецензент: доц. Горєлов В.О.

м. Івано-Франківськ – 2025

Прикарпатський національний університет імені Василя Стефаника

Інститут, факультет математика та інформатика
Кафедра комп'ютерних наук та інформаційних систем
Освітньо-кваліфікаційний рівень бакалавр
Напрям підготовки (Спеціальність)126 - інформаційні системи та технології
(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри _____
(підпис)
_____
(прізвище, ініціали)
“____” _____20___ р.

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Хомяк Євген Васильович
(прізвище, ім’я, по батькові)

1. Тема роботи Розробка мобільного застосунку-довідника Українських Карпат,

керівник роботи к.т.н., доц. Превисокова Наталія Володимирівна
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)

затверджені наказом університету від “___”_____20__ р. №_____.

2. Строк подання студентом роботи_____

3. Вихідні дані до роботи_____
_____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_____
_____
_____
_____
_____
_____

5. Перелік графічного матеріалу
_____

7. Дата видачі завдання _____

АНОТАЦІЯ

Пояснювальна записка: 82 с., 3 частини, 18 рисунків, 6 таблиць, 2 додатка, 18 джерел.

У дипломній роботі проведено аналіз предметної галузі, що стосується мобільних застосунків для туристичної підтримки в регіоні Українських Карпат. Розглянуто існуючі рішення та інструменти для реалізації мобільного застосунку. На основі аналізу спроєктовано та реалізовано мобільний застосунок-довідник з функціями авторизації, перегляду туристичних об’єктів, фільтрації, інтерактивної карти, додавання нових локацій та відправлення SOS-сигналів. Реалізацію виконано з використанням технологій React Native та локального сховища даних.

Список ключових слів: Туристичний довідник, мобільний застосунок, українські Карпати, JavaScript, React Native.

ABSTRACT

Explanatory note: 82 pp., 3 parts, 18 figures, 6 tables, 2 annex, 18 sources.

The diploma work conducted an analysis of the subject area concerning mobile applications for tourist support in the Ukrainian Carpathian region. Existing solutions and tools for mobile application are considered. On the basis of the analysis, a mobile application-reference applies with the functions of authorization, revision of tourist sites, filtration, interactive map, adding new locations and departure of SOS signals were developed and implemented. The implementation was made using React Native and Local Data Repaus Technologies.

List of keywords: Tourist guide, mobile application, Ukrainian Carpathians, JavaScript, React Nati

ЗМІСТ

ВСТУП

У сучасних умовах розвитку цифрових технологій та зростання популярності внутрішнього туризму в Україні особливого значення набуває впровадження інноваційних рішень для підтримки туристичної активності. Туристи дедалі частіше віддають перевагу мобільним інструментам, які забезпечують зручність у плануванні та здійсненні подорожей. Це сприяє підвищенню вимог до якості та функціональності інформаційних сервісів, що супроводжують туристів під час їх перебування у різних регіонах. Одним із перспективних напрямів розвитку таких сервісів є створення мобільних застосунків-довідників, які поєднують інформативність, інтерактивність та доступність. Подібні рішення особливо доречні для регіонів з високим туристичним потенціалом, таких як Українські Карпати, де поєднуються природна привабливість, культурна спадщина та потреба у сучасному цифровому супроводі мандрівника.

Особливої актуальності набуває розробка таких застосунків для регіону Українських Карпат — одного з найпривабливіших туристичних напрямків країни. Створення мобільного рішення дозволить користувачам зручно переглядати інтерактивну карту з локаціями, отримувати детальну інформацію про місця, додавати нові об'єкти, мати змогу надсилати SOS-сигнал у надзвичайних ситуаціях та взаємодіяти з довідковим контентом навіть в умовах обмеженого інтернет-покриття [1].

Це обумовило мету даної кваліфікаційної роботи, яка полягає у розробці мобільного застосунку-довідника для підтримки туристичної діяльності в Українських Карпатах. Для досягнення поставленої мети було сформульовано наступні завдання:

1. Здійснити аналіз предметної області.
2. Виконати дослідження та обґрунтування вибору технологій для створення мобільного застосунку-довідника.

3. Здійснити програмну реалізацію мобільного застосунку-довідника Українських Карпат.

Об'єкт дослідження – мобільний застосунок для туристичного супроводу в регіоні Українських Карпат.

Предмет дослідження – програмні засоби, що забезпечують створення довідникових туристичних мобільних застосунків із можливістю офлайн-доступу, інтерактивної взаємодії з картою і контентом.

Методологічною основою дослідження є загальнонаукові та спеціальні методи, які дозволили вивчити предмет і об'єкт дослідження, визначити технічні вимоги до системи, обґрунтувати вибір архітектури програмного забезпечення та обрати ефективні інструменти реалізації.

Практичне значення отриманих результатів полягає в тому, що розроблений мобільний застосунок може бути використаний туристами для планування подорожей, навігації у гірській місцевості, отримання важливої довідкової інформації, що підвищує рівень безпеки та комфорту під час перебування в Українських Карпатах.

1. ОГЛЯД ПРЕДМЕТНОЇ ГАЛУЗІ

1.1. Характеристика Українських Карпат як туристичного регіону

Українські Карпати (Рисунок 1.1) — це гірська система, яка охоплює південно-західну частину України та є частиною загальноєвропейського Карпатського хребта. Регіон займає території Львівської, Івано-Франківської, Закарпатської та Чернівецької областей. Загальна площа Українських Карпат становить понад 24 тисячі квадратних кілометрів. Цей край вирізняється не лише мальовничими пейзажами, але й значним природним, рекреаційним, історико-культурним потенціалом, що робить його одним з найпривабливіших туристичних напрямків в Україні. Географічно Карпати поділяються на декілька гірських хребтів: Ґорґани, Свидовець, Чорногора, Вододільно-Верховинський хребет та інші. Найвищою точкою є гора Говерла (2061 м), яка входить до складу масиву Чорногора [1].

Рисунок 1.1 – Природа Українських Карпат

Клімат регіону помірно-континентальний, із прохолодним літом та м’якою зимою. Завдяки цьому сезонність туристичних послуг охоплює увесь рік: взимку популярні гірськолижні курорти (Буковель, Драгобрат, Славське), а влітку — піші, велосипедні, кінні та еко-маршрути. Розвинута мережа мінеральних джерел, зокрема в Трускавці, Східниці, Поляні, дозволяє проводити лікувально-оздоровчі тури [2].

Особливої уваги заслуговує культурна самобутність Карпатського регіону. Тут мешкають представники етнічних груп — гуцули, бойки, лемки, кожна з яких має власні звичаї, діалекти, кухню, елементи одягу та архітектури. Традиції місцевого населення активно популяризуються через фестивалі, ярмарки, фольклорні свята, які щороку приваблюють тисячі відвідувачів. Одними з найвідоміших є фестиваль «Гуцульська бринза» в Рахові, «Лемківська ватра» у Львівській області, «На Івана, на Купала» в Косові. Окрему цінність мають архітектурні та історичні пам’ятки. У Карпатах збереглися численні дерев’яні церкви та каплиці, багато з яких включено до списку світової спадщини ЮНЕСКО. Замки, музеї, стародавні поселення та природні пам’ятки є важливою складовою туристичних маршрутів [3].

1.2 Проблеми та виклики туристичної інфраструктури в Українських Карпатах

Попри високий туристичний потенціал Українських Карпат, регіон стикається з рядом труднощів, які ускладнюють ефективний розвиток туризму та погіршують якість перебування мандрівників. Ці проблеми стосуються як інфраструктурного, так і інформаційного рівнів, а також організаційних аспектів туристичних послуг. Наявність таких викликів створює передумови для пошуку інноваційних підходів до їх вирішення, зокрема з використанням цифрових технологій. Розглянемо основні проблеми і виклики, характерні для Карпатського туристичного регіону.

Однією з ключових проблем туризму в Українських Карпатах є інформаційна фрагментованість — відсутність єдиного джерела достовірної та актуальної інформації про регіон. Дані про маршрути, об’єкти, розміщення, події розпорошені між сайтами, соцмережами, друкованими матеріалами або передаються усно. Туристам доводиться витрачати багато часу на пошук, аналіз і перевірку відомостей, які часто дублюються або суперечать одне одному. Частина ресурсів застаріла, що може призвести до дезорієнтації чи небезпеки під час мандрівок.

Також спостерігається нерівномірність інформаційного покриття. Популярні місця (Буковель, Говерла, Шипіт) представлені краще, тоді як менш відомі об’єкти (етномузеї, дерев’яні церкви, екомаршрути) — залишаються в тіні. Це сприяє перевантаженню окремих локацій і знижує загальну привабливість регіону. Додатково ускладнює ситуацію відсутність мовної адаптації, некоректні переклади та слабка оптимізація під мобільні пристрої — головний інструмент туриста в дорозі.

Ще один аспект — брак зворотного зв’язку. Туристи не можуть залишати відгуки, ділитися попередженнями чи рекомендаціями, що ускладнює розвиток спільноти відповідального туризму. Таким чином, подолати інформаційну фрагментованість можливо через створення єдиного мобільного застосунку-довідника, який об’єднує ключову інформацію, оновлюється регулярно та працює офлайн — відповідаючи сучасним потребам мандрівників [4].

Також, попри зростання туристичних потоків, цифрова присутність багатьох об’єктів і послуг у Карпатах залишається низькою. Багато готелів, садиб, гідів чи агентств не мають власних сайтів, мобільних версій сторінок, інтерактивних карт або зручної системи бронювання. Контакти часто застарілі, а відгуки — відсутні чи неповні, що ускладнює планування та знижує довіру до послуг. Здебільшого взаємодія з туристом обмежується соцмережами, де немає структури, категорій чи гарантії актуальності інформації. Це змушує користувача перевіряти кілька джерел, знижуючи

зручність і ефективність пошуку. Низький рівень цифровізації стримує розвиток регіону та зменшує його конкурентоспроможність у порівнянні з іншими цифрово розвиненими туристичними локаціями. Одним із рішень є мобільний застосунок, що об’єднає цифрові профілі об’єктів, забезпечить доступ до перевіреної інформації, відгуків та інструментів для планування маршрутів [5].

Через складний рельєф та віддаленість багатьох об'єктів у Карпатах мобільний зв’язок часто відсутній або нестабільний. Це особливо помітно у високогір’ї, полонинах, заповідниках і малонаселених пунктах. За таких умов онлайн-ресурси стають недоступними, а типові дії — перегляд карти чи зв’язок із гідом — вимагають з’єднання, якого може не бути. Більшість цифрових сервісів орієнтовані лише на онлайн-використання, ігноруючи потребу в офлайн-доступі. Це змушує туристів вдаватися до незручних рішень: скріншотів, роздруківок або нотаток, що знижує зручність і безпеку подорожі. Відсутність офлайн-інструментів у поєднанні з поганим зв’язком є серйозною перешкодою для розвитку комфортного туризму в регіоні [6].

Карпати — це не лише природа, а й унікальна культура, традиції та побут етнічних спільнот. Проте багато локальних ресурсів — музеї, майстерні, садиби, фестивалі — залишаються маловідомими. Причина — слабка взаємодія між туристичними структурами, інформаційними платформами та місцевими громадами. Найчастіше популяризація об’єктів обмежується особистими сторінками у соцмережах, що не забезпечує широкого охоплення [7]. Відсутність зворотного зв’язку з туристами унеможливлює поліпшення сервісів та адаптацію до змін інтересів. Без активної участі громад і постійного оновлення контенту привабливість регіону залишається нестабільною. Це веде до концентрації туристів у кількох популярних місцях, тоді як інші, не менш цікаві локації — ігноруються. Як наслідок, громади втрачають економічні можливості, а регіон — потенціал сталого розвитку [8].

1.3 Актуальність розробки мобільного довідника

На основі аналізу сучасного стану туристичної інфраструктури в Українських Карпатах можна зробити висновок про нагальну потребу у створенні зручного, функціонального та локалізованого цифрового інструменту для туристів. Виявлені в попередніх підрозділах проблеми — фрагментованість інформації, низький рівень цифровізації сервісів, обмежене покриття мобільного зв'язку та недостатня інтеграція локального контенту — вказують на відсутність єдиного, ефективного рішення, яке б повністю задовольняло потреби мандрівників у регіоні. У цьому контексті розробка мобільного застосунку-довідника Українських Карпат є надзвичайно актуальною. Такий застосунок може стати сучасною відповіддю на низку викликів, що стоять перед туристичною галуззю регіону, а також інструментом підвищення зручності та безпеки для мандрівників. Мобільний формат обрано не випадково — смартфон є основним інструментом для планування подорожей, орієнтації на місцевості, пошуку житла, закладів харчування, відгуків і рекомендацій [7].

До основних причин, що обґрунтовують актуальність створення такого застосунку, належать:

1. Попит на локалізовану туристичну інформацію - більшість існуючих ресурсів не орієнтовані на специфіку Українських Карпат, не враховують місцеву специфіку, мову, традиції та об'єкти інтересу.
2. Зростання внутрішнього туризму в Україні - особливо в умовах обмеженого виїзного туризму, все більше українців обирають Карпати як безпечний і цікавий напрямок для відпочинку.
3. Необхідність офлайн-доступу до інформації - через нестабільне покриття мобільного зв'язку туристам потрібен інструмент, що дозволяє використовувати карти, маршрути та описи без підключення до інтернету.

4. Потреба в інтерактивності та зворотному зв’язку - можливість додавати відгуки, оцінювати об’єкти, формувати власні маршрути підвищує зацікавленість користувачів та допомагає іншим мандрівникам.
5. Популяризація локальних громад і маловідомих об’єктів - застосунок може сприяти рівномірному розподілу туристичних потоків, що позитивно вплине на розвиток малих населених пунктів і підвищить економічну активність регіону.
6. Зростання цифрової культури та очікувань користувачів - туристи очікують сучасного інтерфейсу, швидкого доступу до інформації, простоти користування та естетичного оформлення — усе це можливо реалізувати у мобільному застосунку.
7. Підвищення безпеки туристів - наявність надійної карти, опису маршрутів, місць для зупинки, контактів служб допомоги та можливість орієнтації в офлайн-режимі значно покращує безпечність мандрівки.

Важливо також зазначити, що мобільний застосунок дозволяє постійно оновлювати та доповнювати інформацію, адаптуючись до змін туристичної ситуації, сезонності, подій, оновлення маршрутів або появи нових сервісів. Отже, створення мобільного довідника Українських Карпат є не лише технічно виправданим, а й стратегічно доцільним кроком, що сприятиме розвитку туризму, залученню нових відвідувачів, підтримці місцевого бізнесу та формуванню позитивного іміджу регіону як відкритого, безпечного та зручного для мандрівника [7].

1.4 Аналіз існуючих програмних продуктів

Перед розробкою мобільного застосунку-довідника доцільно проаналізувати наявні рішення, які частково або повністю охоплюють туристичний сегмент Українських Карпат. Такий огляд дозволяє виявити переваги й недоліки конкурентів, а також визначити напрями вдосконалення,

що дозволить новому застосунку ефективніше відповідати потребам користувачів.

Одним із найвідоміших українських інформаційних ресурсів, присвячених туризму в Карпатах, є Karpaty.info (Рисунок 1.2).

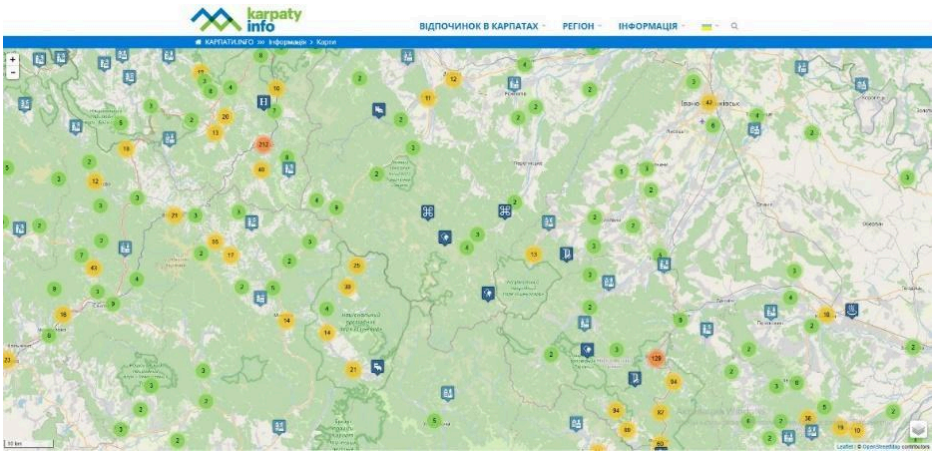

Рисунок 1.2 – Інтерфейс користувача ресурсу Karpaty.info

Платформа Karpaty.info працює з 2004 року й охоплює основні туристичні регіони Карпат, надаючи інформацію про маршрути, проживання, розваги та пам’ятки. Сайт доступний трьома мовами й орієнтований на широку аудиторію. Завдяки великій базі даних ресурс зручний для планування подорожей, однак має обмеження щодо мобільної взаємодії та офлайн-доступу. До функціоналу ресурсу Karpaty.info належать:

1. Каталог населених пунктів і курортів — структурований список міст, сіл і селищ із детальними описами, фото, особливостями географічного положення та основними локаціями.
2. База засобів розміщення — розширена інформація про готелі, приватні садиби, котеджі, включаючи ціни, умови проживання, фото, контакти.
3. Довідник розваг і сервісів — список туристичних атракцій, закладів харчування, прокатів спорядження, екскурсійних бюро тощо.
4. Опис природних та історико-культурних об'єктів — інформація про водоспади, печери, гори, включаючи локацію та фото.

5. Публікації та поради туристам — тематичні статті про традиції, звичаї, кухню, транспортну доступність, цікаві маршрути та події.

Розглянемо переваги, виявлені користувачами при використанні Karpaty.info:

- Дуже широка база даних, яка охоплює як популярні, так і менш відомі туристичні локації.
- Чітка структура і логічна організація розділів, що забезпечує легке орієнтування в інформації.

Розглянемо недоліки, виявлені користувачами при використанні Karpaty.info:

- Відсутність мобільного застосунку або повноцінної адаптації під смартфони, що ускладнює користування ресурсом у дорозі.
- Немає інтерактивної карти з маршрутами або функції побудови маршрутів між об'єктами.

Karpaty.info є потужним довідковим ресурсом, що популяризує Карпати та надає широкий обсяг інформації для планування подорожей. Однак платформа зосереджена переважно на попередньому інформуванні, без підтримки користувача під час мандрівки.

Наступним розглянемо сервіс Карпати.travel. Карпати.travel (Рисунок 1.3) — це український веб-портал, присвячений туризму в Карпатському регіоні. Ресурс надає інформацію про туристичні маршрути, природні та культурні пам'ятки, місця для відпочинку, а також пропонує послуги з організації турів. До функціоналу ресурсу "Карпати.travel" належать:

1. Опис туристичних маршрутів — детальна інформація про пішохідні, велосипедні та екскурсійні маршрути з вказівкою складності, тривалості та особливостей.
2. Каталог природних та культурних об'єктів — інформація про гори, водоспади, ліси, музеї, церкви та інші цікаві місця з фотографіями та описами.

3. Інформація про проживання та харчування — список готелів, садиб, ресторанів та кафе з короткими описами та контактами.
4. Новини та статті — актуальні новини про події в регіоні, поради для туристів та цікаві факти про Карпати.

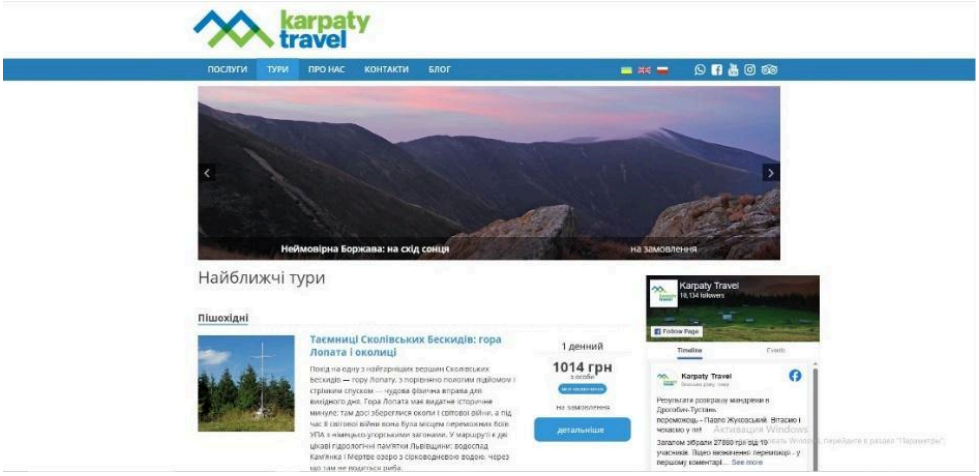

Рисунок 1.3 – Інтерфейс користувача ресурсу Карпати.travel

Розглянемо переваги, виявлені користувачами при використанні "Карпати.travel":

- Комплексний підхід — ресурс об'єднує інформацію про маршрути, об'єкти, проживання та тури в одному місці.
- Зручна навігація — структурований інтерфейс дозволяє легко знаходити потрібну інформацію.

Розглянемо недоліки, виявлені користувачами при використанні "Карпати.travel":

- Відсутність мобільного застосунку — немає спеціалізованої мобільної версії або додатку для зручного використання під час подорожей.
- Обмежена інтерактивність — відсутність можливості створювати власні маршрути або залишати відгуки безпосередньо на сайті.

Таким чином, "Карпати.travel" є корисним ресурсом для попереднього планування подорожей у Карпати, проте його функціональність обмежена для використання безпосередньо під час мандрівок.

Наступним розглянемо сервіс Karpaty.rocks. Karpaty.rocks (Рисунок 1.4) — український онлайн-ресурс, що популяризує туристичний потенціал Карпат. Він поєднує функціонал довідника з інтерактивною мапою, яка дозволяє самостійно планувати маршрути, переглядати об’єкти та прокладати шлях. Платформа орієнтована на індивідуальних мандрівників, однак функціонує лише як веб-сайт, що обмежує її зручність у реальних умовах подорожі без мобільного застосунку. До функціоналу інтерактивної карти Karpaty.rocks належать:

1. Пішохідні маршрути – треки з позначенням протяжності, орієнтовного часу проходження, набору висоти та описом маршруту.
2. Гірські вершини – точки з інформацією про висоту, можливі підйоми та рекомендовані шляхи.
3. Проживання – розміщення готелів, садиб, хостелів із контактами та вказаною вартістю (за наявності).
4. Цікаві місця – водоспади, джерела, оглядові майданчики, музеї та пам’ятки архітектури.

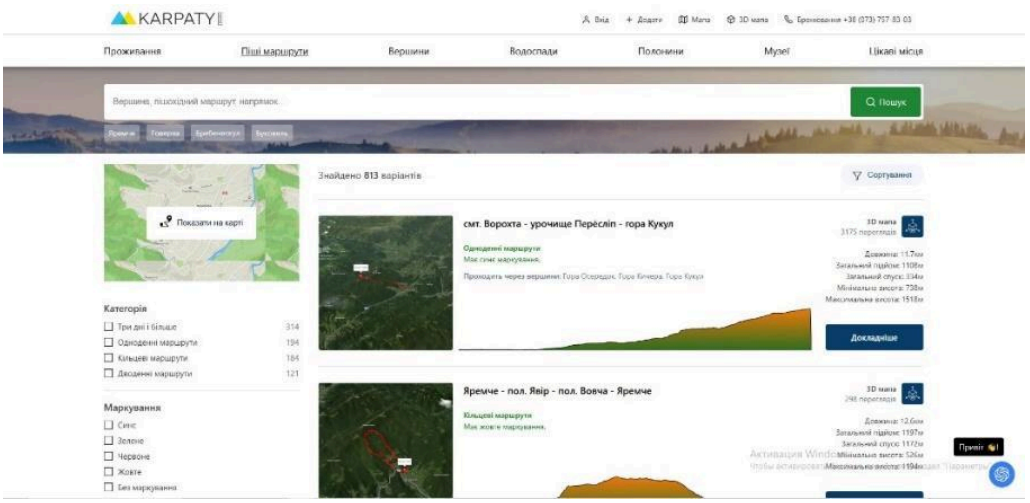

Рисунок 1.3 – Інтерфейс користувача ресурсу Karpaty.rocks

Розглянемо переваги, виявлені користувачами при використанні Karpaty.rocks:

- Інтерактивна карта з великою кількістю нанесених об’єктів дозволяє комплексно оцінити туристичний потенціал місцевості.
- Об’єднання інформації про маршрути, природні об’єкти та інфраструктуру в одному ресурсі.

Розглянемо недоліки, виявлені користувачами при використанні Karpaty.rocks:

- Неінтуїтивний та незручний інтерфейс — структура сайту перевантажена, окремі функції неочевидні, навігація по мапі ускладнена.
- Відсутність мобільного застосунку — користування з мобільного браузера обмежене, а офлайн-доступу до мапи не передбачено.

Karpaty.rocks має значний потенціал завдяки великій базі даних і картографічному підходу, що робить його корисним на етапі планування подорожі. Проте відсутність мобільної адаптації, офлайн-доступу та зручного інтерфейсу обмежує його використання під час мандрівок, що підкреслює потребу у створенні сучасного мобільного застосунку.

1.5 Визначення вимог до проектованого мобільного застосунку

На основі проведеного аналізу сучасного стану цифрових рішень у сфері туристичного супроводу Карпатського регіону, було виявлено як окремі переваги існуючих продуктів, так і значні обмеження, які стримують їх ефективне використання. До позитивних рис, що варто зберегти або розвивати у майбутньому застосунку, належать:

1. Наявність інтерактивної карти з основними об’єктами туристичної інфраструктури;
2. Орієнтація контенту виключно на український Карпатський регіон;

3. Прагнення забезпечити простоту навігації для користувачів.

Водночас недоліки більшості існуючих рішень обумовили необхідність створення нового застосунку:

1. Відсутність можливості додавання нових об'єктів безпосередньо з мобільного пристрою;
2. Недостатньо адаптований та перевантажений інтерфейс користувача;
3. Брак функцій інтерактивної взаємодії з контентом;
4. Відсутність критичних функцій безпеки, таких як швидкий SOS-виклик.

З огляду на це, було визначено такі ключові вимоги до функціональності мобільного застосунку-довідника "Українські Карпати":

- Інтерактивна мапа з додаванням нових об’єктів - забезпечення можливості переглядати актуальні туристичні місця (міста, гори, озера, водоспади) на карті та додавати нові локації безпосередньо через застосунок. Це дозволить оперативно оновлювати інформацію й розширювати базу туристичних об’єктів.
- Інтуїтивний і покращений інтерфейс користувача - інтерфейс має бути максимально зрозумілим і простим у користуванні, навіть для людей із мінімальним досвідом роботи з мобільними застосунками. Особлива увага приділяється зручній навігації та мінімізації кількості дій для доступу до основних функцій.
- Функція SOS-сповіщення - наявність спеціальної кнопки екстреної допомоги, яка дозволяє користувачеві швидко надіслати сигнал про небезпеку разом із геолокацією.
- Зручне отримання інформації про об’єкти - застосунок повинен надавати користувачам можливість легко переглядати повну інформацію про кожен об’єкт: назву, тип, опис, фотографії, координати, висоту та розташування. Особлива увага приділяється візуальній представленості об’єктів і їх структурованому поданню.

Запропоновані вимоги враховують очікування сучасних користувачів, які потребують не лише доступу до інформації, а й можливості активно взаємодіяти з нею. Застосунок має забезпечувати високу швидкодію, офлайн-доступ до карти та об’єктів, а також надавати критично важливі функції для перебування в гірських умовах. Мобільний довідник повинен поєднуватт точність, зручність, актуальність і орієнтацію на безпеку, формуючи якісно новий рівень туристичного сервісу в Карпатах.

2. АНАЛІЗ ТА ВИЗНАЧЕННЯ ІНСТРУМЕНТІВ РЕАЛІЗАЦІЇ МОБІЛЬНОГО ЗАСТОСУНКУ-ДОВІДНИКА УКРАЇНСЬКИХ КАРПАТ

2.1 Вибір програмного забезпечення для розробки мобільного застосунку-довідника Українських Карпат

У сучасному цифровому середовищі мобільні застосунки є ключовим інструментом для надання користувачам зручного доступу до туристичної інформації. У контексті Карпат актуальним є створення мобільного довідника, що містить маршрути, об’єкти, сервіси та культурні пам’ятки. Для реалізації такого застосунку необхідно обрати оптимальні технології, які забезпечать стабільну роботу, зручний інтерфейс і можливість масштабування. На початковому етапі розробки важливим є вибір мови програмування, фреймворку, середовища розробки, інструментів для роботи з геолокацією, картами та базами даних [9].

Особливої уваги потребують інструменти для кросплатформенної розробки — вони дають змогу створювати застосунок одночасно для Android і iOS, що значно розширює охоплення користувачів. Також важливо враховувати можливість роботи застосунку в офлайн-режимі, швидкість завантаження контенту та інтеграцію з зовнішніми сервісами. Зручність

користувача безпосередньо залежить від того, наскільки добре продумано інтерфейс та навігація всередині програми. Наявність зворотного зв’язку, локалізація та підтримка мультимовності — також важливі аспекти для успішного функціонування туристичного мобільного застосунку.

На рисунку 2.1 подано статистику використання мов програмування, які найчастіше застосовуються у сфері туристичних мобільних застосунків [10].

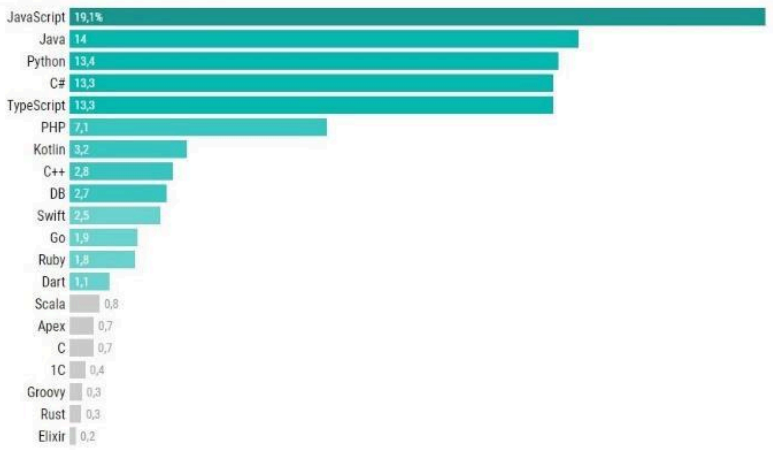

Рисунок 2.1 – Статистика використання мов програмування у розробці серверної логіки застосунків інформаційного характеру

За цими даними можна зробити висновок, що JavaScript на даний час є найпопулярнішою мовою програмування для виконання задач у предметній галузі, але Python, C#, Java тощо також є доволі затребуваними.

2.2 Порівняльний аналіз мов програмування для розробки мобільного застосунку-довідника Українських Карпат

Для аналізу характеристик найбільш актуальних для виконання поставленого завдання технологій, складемо порівняльні таблиці для проведення порівняння за такими характеристиками як: парадигми, типізація,

керування пам'яттю, об'єктно - орієнтованими можливостями та частотою використання [11]

Таблиця 2.1 - Порівняльна характеристика за парадигмами

Можливість	C#	Java	C++	PHP	Python	JS
Імперативна	+	+	+	+	+	+
Об'єктно-орієнтована	+	+	+	+	+	+
Функціональна	+/-	+/-	+/-	+/-	+	+
Рефлексивна	+/-	+/-	+	+	+	+

У Таблиця 2.1 подано порівняння мов програмування за їхніми парадигмальними характеристиками, що дозволяє визначити ступінь підтримки ключових підходів до програмування, зокрема імперативного, об'єктно-орієнтованого та функціонального. Це зіставлення є важливим для розуміння гнучкості мови під час вирішення різноманітних задач.

Таблиця 2.2 - порівняльна характеристика за типізацією

Можливість	C#	Java	C++	PHP	Python	JS
Статична типізація	+	+	-	-	-	-
Динамічна типізація	+	-	+	+	+	+
Явна типізація	+/-	+	-	+/-	+/-	-
Неявна типізація	+	-	+	+	+	+
Неявне приведення	-	+	+	+	+	+

У Таблиці 2.2 своєю чергою, проведено аналіз типізаційних особливостей, що має вирішальне значення для вибору мови залежно від вимог до надійності, зручності налагодження та швидкості розробки. Поєднання цих двох аспектів дає змогу сформувати більш цілісне уявлення про доцільність

застосування тієї чи іншої технології для створення мобільного застосунку.

Таблиця 2.3 - Порівняльна характеристика за можливостями керування пам’яттю

Можливість	C#	Java	C++	PHP	Python	JS
Об’єкти на стеку	+	-	-	-	-	-
Некеровані вказівники	+	-	-	-	-	-
Ручне керування	+	-	-	-	-	-
Збирання сміття	+	+	+	+	+	+

У Таблиці 2.3 наведено порівняння здібностей керування пам’яттю у розглянутих мовах. Спеціальну увагу приділено таким аспектам, як наявність збирача сміття, підтримка ручного управління та використання стеку. Ці характеристики мають важливе значення для ефективної роботи застосунку в умовах обмежених ресурсів мобільного середовища.

Таблиця 2.4 - Порівняльна характеристика за функціональними можливостями

Можливість	C#	Java	C++	PHP	Python	JS
Декларації функцій	-	-	-	-	-	-
First class функції	+	-	+	+	+	+
Анонімні функції	+	+	+	+	+	+
Лексичні замкнення	+	+	+	+	+	+
Часткове застосування	-	-	+	-	+	+

У Таблиці 2.4 представлено функціональні особливості мов програмування, зокрема здатність працювати з анонімними функціями, замиканнями та частковим застосуванням. Ці можливості є надзвичайно корисними для побудови гнучкої архітектури програми та забезпечення її здатності до масштабування.

Таблиця 2.5 - Порівняльна характеристика за об’єктно – орієнтованими можливостями

Можливість	C#	Java	C++	PHP	Python	JS
Інтерфейси	+	+	+	+	+	-
Мультиметоди	+/-	-	-	-	-	-
Міксини	-	+	+	+	+	+
Перейменування	-	-	-	-	-	-
Множинне спадкування	-	-	-	-	+	-

У таблиці 2.5 здійснено зіставлення об’єктно-орієнтованих функцій, що дає змогу визначити рівень реалізації концепцій ООП. Розглядаються такі важливі елементи, як інтерфейси, міксини, множинне успадкування й інші, що можуть бути ключовими при проєктуванні структурованої та модульної архітектури мобільного додатку.

Особливу увагу було приділено зручності написання і читання коду, що суттєво впливає на швидкість розробки та супровідність програмного продукту. З аналізу видно, що мови Java та C потребують значної кількості коду для реалізації базової логіки, у той час як такі мови, як JavaScript (зокрема з використанням React Native), дозволяють досягати того самого результату з меншими зусиллями завдяки декларативному підходу та наявності великої кількості готових компонентів.

При розробці мобільного довідника особливо важливо враховувати такі можливості: кросплатформенна підтримка (Android/iOS), інтеграція з картографічними сервісами (Google Maps, Mapbox), геолокація, збереження даних користувача, робота з REST API та адаптивність інтерфейсу до різних типів пристроїв. Усі ці вимоги повністю задовольняє технологічний стек JavaScript + React Native. В контексті виконання поставленого завдання наведена зв'язка має наступну низку переваг:

1. Можливість написання одного коду для двох платформ (iOS і Android);
2. Активна спільнота та постійна підтримка з боку Facebook;
3. Проста інтеграція з веб-сервісами та API;
4. Широка база знань і навчальних ресурсів.
5. Швидка розробка та оновлення завдяки гарячому перезавантаженню (hot reload);
6. Простий у засвоєнні синтаксис;
7. Велика екосистема модулів і плагінів;
8. Потужні інструменти розробника, такі як Expo;
9. Наявність офіційної документації та великої кількості прикладів реалізації подібних проєктів.

Підіб'ємо підсумки. Кожна мова програмування має свої переваги та недоліки, і вибір залежить від вимог до проєкту. У випадку розробки мобільного застосунку-довідника Українських Карпат найдоцільнішим є використання JavaScript у поєднанні з React Native, оскільки ця технологія забезпечує кросплатформенність, ефективну роботу з геолокацією та картографічними сервісами, адаптивний інтерфейс і швидку розробку з багатим набором бібліотек і підтримкою з боку спільноти [12].

2.3 Вибір середовище для кодування: технічний огляд

Для реалізації коду, потрібного для дипломного проєкту, зазвичай використовують текстовий редактор — програму або компонент, що служить для створення та корекції текстових файлів. Редактор є одним із головних інструментів розробника, дозволяючи не тільки писати код, а й відразу його перевіряти і виправляти.

Вибір редактора для роботи над проєктом — непросте завдання. Воно включає аналіз доступних варіантів, врахування власних вподобань, а також практичну перевірку, що допомагає обрати редактор, який найкраще відповідатиме конкретним вимогам завдання.

Sublime Text Editor (Рисунок 2.6) — це потужний та гнучкий текстовий редактор, популярний серед розробників завдяки своїй швидкості, широким можливостям налаштування і підтримці численних мов програмування. Він забезпечує інтуїтивний інтерфейс та функціонал, що робить процес кодування більш зручним і ефективним.

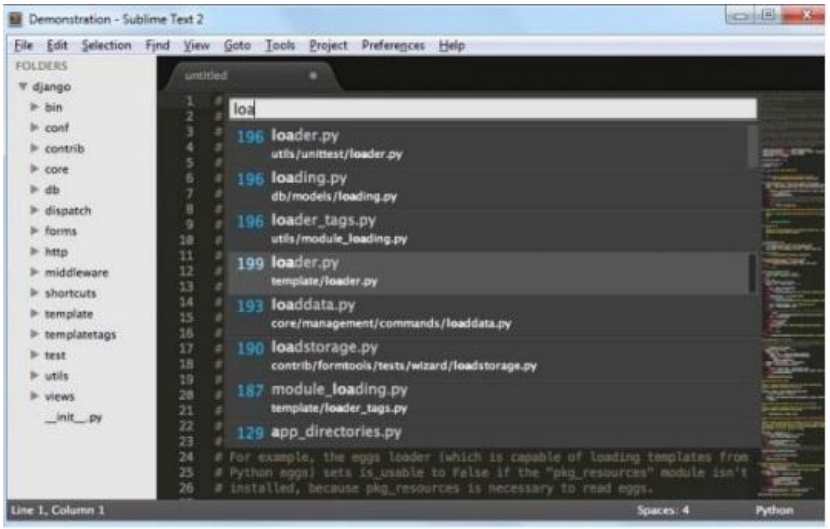

Рисунок 2.6 – Інтерфейс користувача Sublime Text editor

Розглянемо функціонал Sublime Text Editor:

- Швидкий пошук і навігація — забезпечує можливість швидко знаходити необхідні файли або рядки коду, значно прискорюючи робочий процес.
- Режим розділення екрану — дозволяє одночасно працювати з кількома файлами, що дуже зручно при багатозадачності.
- Підтримка плагінів — Sublime Text легко розширюється за допомогою плагінів, що додає нові можливості та покращує зручність роботи.
- Розширені можливості налаштування — дозволяє користувачам підлаштовувати інтерфейс та функції редактора під свої індивідуальні потреби.

Розглянемо переваги, що були виявлені користувачами при використанні Sublime Text Editor:

1. Швидкодія та мінімальне навантаження на систему — редактор працює дуже швидко навіть на старих комп’ютерах.
2. Інтуїтивний інтерфейс — простий і зрозумілий інтерфейс, який легко налаштувати під себе.

Розглянемо недоліки, що були виявлені користувачами при використанні Sublime Text Editor:

1. Відсутність безкоштовної повної версії —повноцінний доступ вимагає придбання ліцензії.
2. Обмежена підтримка проєктів з великою кількістю файлів — робота може сповільнюватися при відкритті великих проєктів.

Загалом, Sublime Text Editor — це відмінний інструмент для написання коду з багатьма корисними функціями та можливостями для налаштування. Він ідеально підходить для розробників, які шукають швидкий, гнучкий і зручний редактор, хоч деякі його недоліки можуть обмежити використання для великих командних проєктів або проєктів із масштабною файловою структурою [13].

Visual Studio Code (рисунок 2.7) – це безкоштовний і потужний текстовий редактор, створений компанією Microsoft. Він відзначається

широким функціоналом, підтримкою різних мов програмування, вбудованими інструментами для розробки та активною спільнотою користувачів. Завдяки своїм можливостям і простоті використання, VS Code здобув популярність серед розробників у всьому світі.

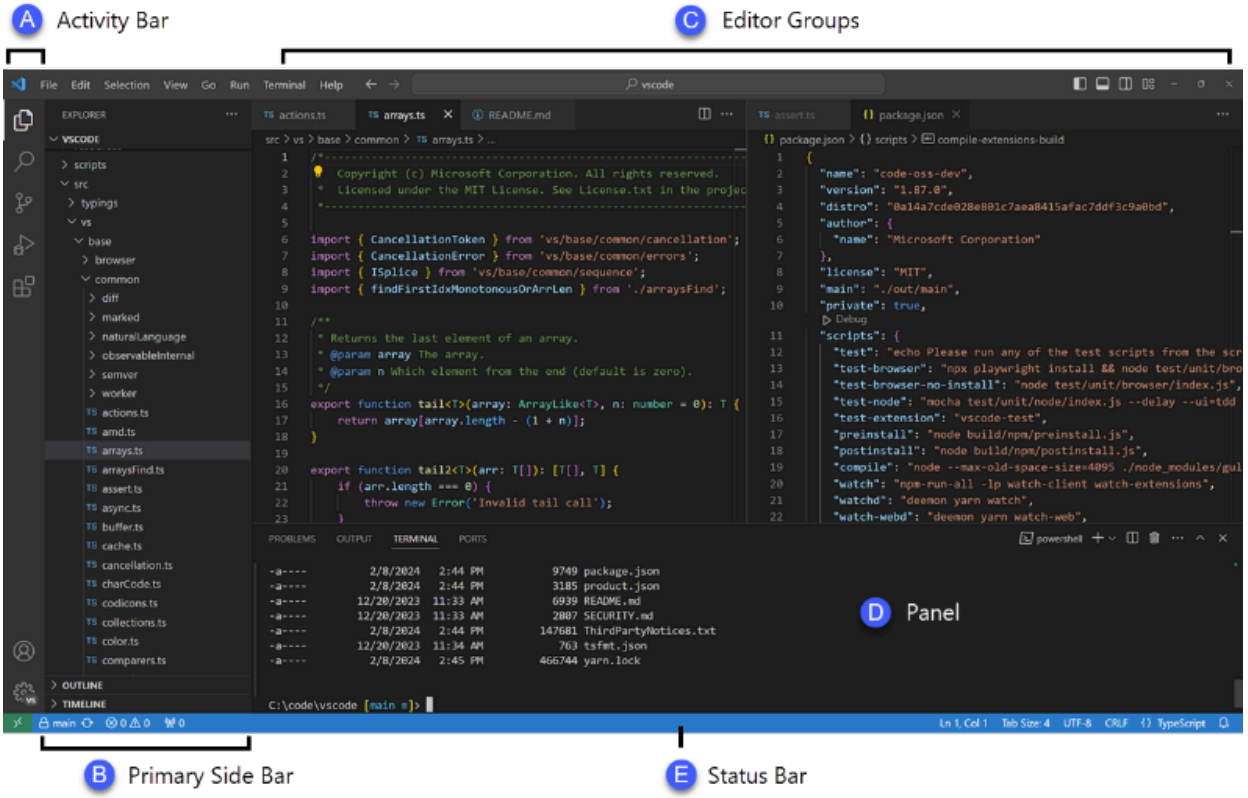

Рисунок 2.7 – Інтерфейс користувача Visual Studio Cod

Розглянемо функціонал Visual Studio Code:

- Інтегрований термінал — дозволяє виконувати командний рядок прямо в редакторі, що значно прискорює процес розробки.
- Вбудована підтримка Git — забезпечує зручну роботу з системою контролю версій, дозволяючи відслідковувати зміни та управляти репозиторіями без необхідності виходити з редактора.
- Маркетплейс розширень — надає доступ до величезної кількості плагінів, що додають функції для будь-яких потреб, від підсвічування синтаксису до інтеграції з різними сервісами.

- Підтримка дебагінгу — дозволяє налагоджувати код безпосередньо в редакторі з допомогою інтегрованих інструментів для багатьох мов програмування.

Розглянемо переваги, що були виявлені користувачами при використанні Visual Studio Code:

1. Безкоштовність та відкритий код — VS Code є повністю безкоштовним та підтримує open-source спільноту, що дозволяє його розширювати та вдосконалювати.
2. Широка підтримка розширень — маркетплейс VS Code пропонує численні плагіни, що дозволяють розширити функціонал редактора для будь-яких задач.

Розглянемо недоліки, що були виявлені користувачами при використанні Visual Studio Code:

1. Високе споживання ресурсів — VS Code може сповільнюватися на комп’ютерах з обмеженими ресурсами через свою функціональність і підтримку розширень.
2. Затримки при роботі з великими проєктами — при відкритті великих проєктів редактор може працювати повільніше, особливо якщо активовано багато розширень.

Visual Studio Code — це універсальний і багатофункціональний редактор для розробки, що підійде як новачкам, так і досвідченим розробникам. Його можливості для налаштування та розширення роблять його ефективним інструментом для широкого спектра задач, хоча високе навантаження на систему може обмежити його використання на менш потужних пристроях [13].

Notepad ++ (Рисунок 2.8) - це безкоштовний текстовий редактор для Windows, який є легким і швидким інструментом для написання і редагування коду. Завдяки простому інтерфейсу, підтримці багатьох мов програмування та мінімальному навантаженню на систему, Notepad++

популярний серед розробників, які шукають ефективний та легкий у використанні редактор.

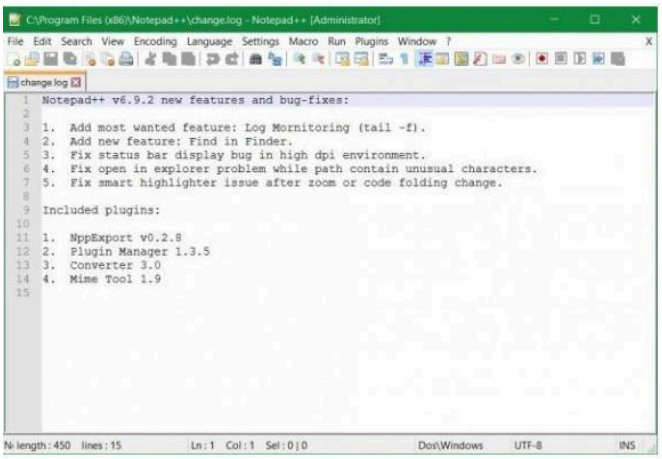

Рисунок 2.8 – Інтерфейс користувача Notepad++

Розглянемо функціонал Notepad++:

- Підсвічування синтаксису та складання коду — дозволяє легше читати код завдяки підсвічуванню ключових слів і можливості автоматичного форматування.
- Підтримка макросів — забезпечує автоматизацію повторюваних дій шляхом запису макросів, що значно прискорює роботу.
- Робота з вкладками — зручно відкривати і редагувати кілька файлів одночасно за допомогою вкладок, що полегшує багатозадачність.
- Автозбереження та відновлення сесії — дозволяє автоматично зберігати зміни та відновлювати сесію після аварійного завершення роботи редактора.

Розглянемо переваги, що були виявлені користувачами при використанні Notepad++:

1. Легкість і швидкість роботи — Notepad++ працює швидко навіть на слабких комп'ютерах завдяки низьким вимогам до ресурсів.

2. Повністю безкоштовний — доступний для завантаження і використання безкоштовно, що робить його зручним вибором для будь-якого користувача.

Розглянемо недоліки, що були виявлені користувачами при використанні Notepad++:

1. Обмежена функціональність порівняно з іншими редакторами — немає вбудованих інструментів для дебагінгу або терміналу, що робить його менш функціональним для складних проєктів.
2. Тільки для Windows — Notepad++ офіційно доступний лише для Windows, що обмежує його використання на інших операційних системах.

Notepad++ — це надійний і легкий редактор, ідеальний для швидких змін та невеликих проєктів, особливо для тих, хто цінує мінімальне навантаження на систему. Проте, через обмежену функціональність він може не відповідати потребам складніших проєктів або професійної розробки на різних платформах [13]. Проаналізувавши всі переваги та недоліки наведених вище інструментів для реалізації проектованого програмного продукту було обрано SublimeText.

2.4 Аналіз та вибір ПЗ для тестування запитів до проектованого програмного забезпечення

Для тестування взаємодії користувача з нашим проектованим програмним забезпеченням у мережі нам знадобиться ПЗО, яке надасть нам змогу ініціювати запит до розробленого нам програмного забезпечення з передачею інформації для взаємодії. API клієнти - це інструменти середнього рівня складності. Це як правило десктопні програми, які дозволяють звернутися до ендпойнтів. Також вони надають додаткові можливості використання змінних, рандомайзерів, скриптів та інших функцій [14].

Розглянемо найбільш актуальні та сучасні інструменти , які задовольняють цим вимогам:

Insomnia (Рисунок 2.9) - це популярний інструмент для тестування API, який забезпечує розробникам зручний інтерфейс для роботи з REST та GraphQL запитами. Цей інструмент дозволяє швидко налаштовувати запити, переглядати та аналізувати відповіді сервера, працювати з аутентифікацією та додавати різноманітні параметри для глибшого тестування.

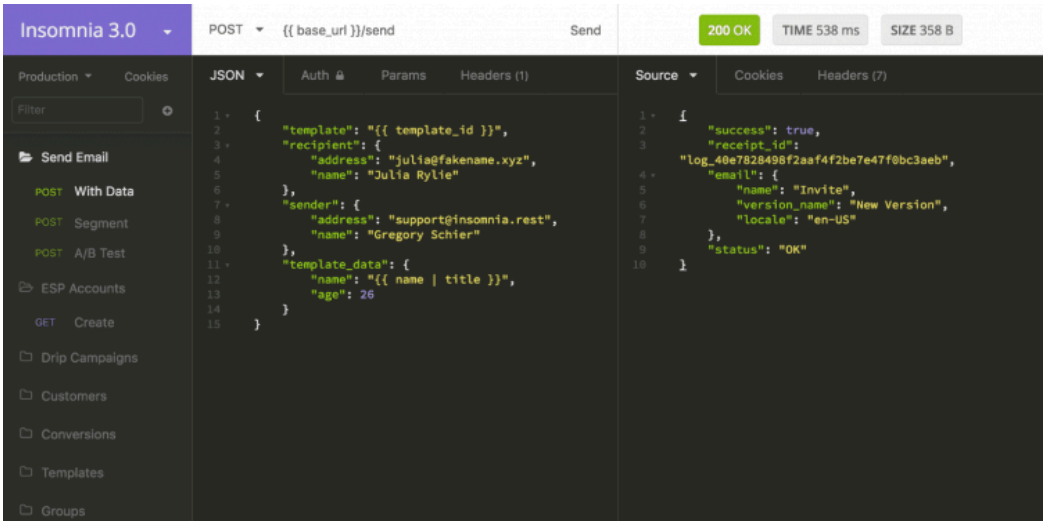

Рисунок 2.9 – Інтерфейс користувача Insomnia

Insomnia також підтримує роботу з колекціями запитів, що робить процес тестування структурованим та організованим. Він сумісний з різними форматами передачі даних, включаючи JSON, XML, та Form URL-encoded, що робить його універсальним у використанні.

Розглянемо переваги, що були виявлені користувачами при використанні Insomnia:

1. Інтуїтивний інтерфейс — зрозумілий і простий у використанні, що дозволяє швидко створювати запити навіть новачкам.
2. Зручна робота з токенами аутентифікації — Insomnia спрощує додавання токенів, що необхідні для роботи з захищеними API.

3. Налаштування середовищ (environment) — дозволяє зберігати змінні для різних середовищ (наприклад, продакшн, тестування), що дуже зручно для управління запитами.

Розглянемо недоліки, що були виявлені користувачами при використанні Insomnia:

1. Відсутність багатьох інтеграцій — Insomnia має менше інтеграцій з іншими інструментами порівняно з альтернативами, такими як Postman.
2. Обмежена кастомізація — інструмент не дозволяє вносити значні зміни в інтерфейс або функціонал, що може обмежувати професійних користувачів.
3. Високе споживання ресурсів на великих проєктах — при роботі з великими колекціями запитів споживання пам’яті може значно зрости.

Insomnia — це простий у використанні та ефективний інструмент для тестування API, що відмінно підходить для роботи з REST і GraphQL. Його можливості роблять його чудовим вибором для розробників, які шукають простоту та швидкість у створенні та перевірці API-запитів, хоча деякі обмеження можуть стати перепоною для більш складних сценаріїв тестування або інтеграцій.

Postman (Рисунок 2.10) - це потужний інструмент для розробників, призначений для тестування API. Він підтримує створення, налагодження та автоматизацію REST, SOAP та GraphQL запитів. Postman дозволяє зручно налаштовувати запити, переглядати відповіді серверів, а також забезпечує підтримку колекцій, що полегшує організацію запитів для великих проєктів. Окрім базового функціоналу, Postman пропонує можливості для налаштування середовищ, створення автоматизованих тестів, інтеграцію з CI/CD системами та підтримує спільну роботу над проєктами в команді, що робить його важливим інструментом для API-розробки.

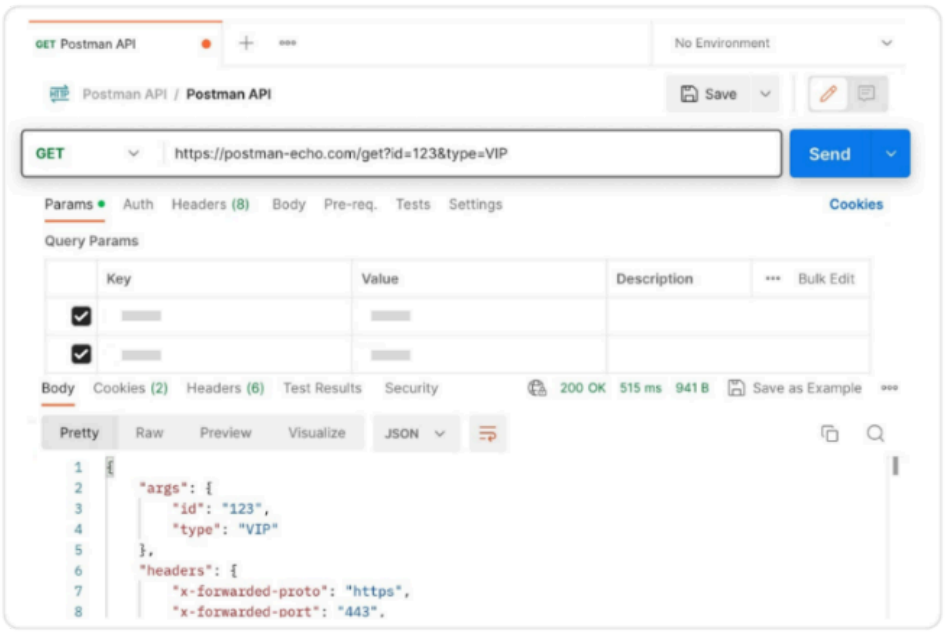

Рисунок 2.10 – Інтерфейс користувача Postman

Розглянемо переваги, що були виявлені користувачами при використанні Postman:

1. Розширені функції тестування та автоматизації — Postman підтримує написання тестів для запитів, що дозволяє легко автоматизувати перевірку API.
2. Спільна робота в команді — зручний інтерфейс для спільної роботи над запитами та проєктами, що дозволяє команді синхронізувати зміни та працювати над API в реальному часі.
3. Інтеграція з CI/CD інструментами — Postman легко інтегрується з популярними системами CI/CD, що дозволяє включати тестування API у загальний процес розробки.

Розглянемо недоліки, що були виявлені користувачами при використанні Postman:

1. Високе споживання ресурсів — на великих проєктах Postman може значно навантажувати систему, що впливає на швидкість роботи.

2. Платні функції для професійного використання — деякі важливі функції, як-от розширені можливості спільної роботи, доступні лише у платних версіях.
3. Складність інтерфейсу для новачків — через багатий функціонал новим користувачам може знадобитися час на освоєння інтерфейсу та всіх доступних функцій.

Загалом, Postman є потужним інструментом для тестування API, який підходить як для індивідуального використання, так і для роботи в команді. Завдяки широкому функціоналу, він ідеальний для розробників, яким потрібні автоматизація та організація запитів, хоча високе споживання ресурсів та складність для новачків можуть бути недоліками при використанні Postman для простих або невеликих проєктів.

SOAPUI (Рисунок 2.11) - це професійний інструмент для тестування веб-служб, спеціалізований на роботу з протоколами SOAP та REST.

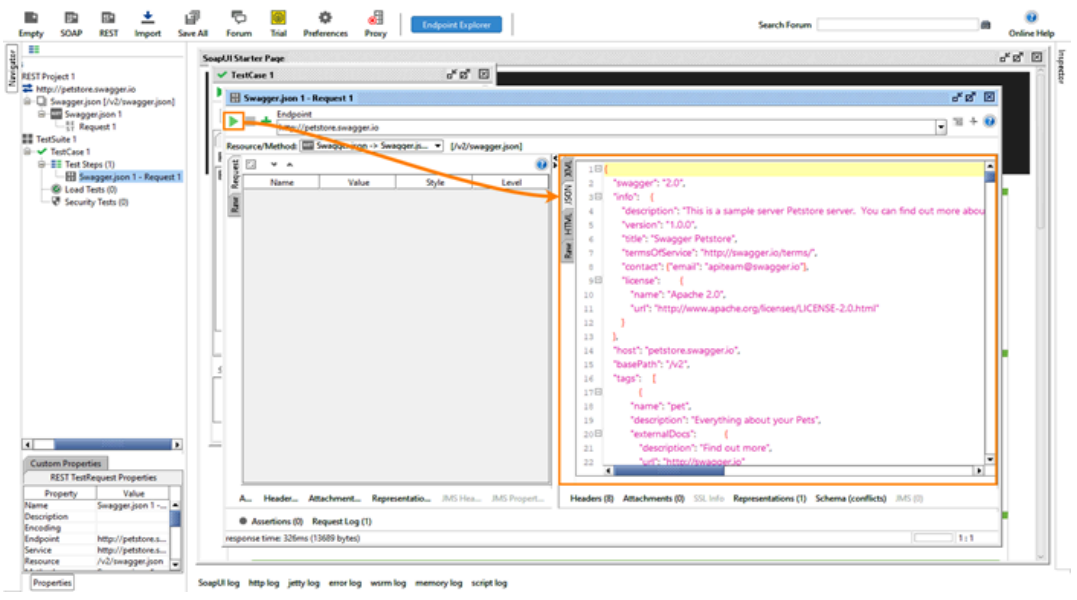

Рисунок 2.11 – Інтерфейс користувача SOAPUI

Відомий своєю гнучкістю, SoapUI дозволяє створювати та виконувати запити, автоматизовані тести, перевірки безпеки та навантажувальні тести,

що робить його особливо корисним для великих проєктів і комплексного тестування API. Цей інструмент підтримує функції побудови сценаріїв, налаштування середовищ, перевірки відповідей та параметризацію тестів, що допомагає у створенні гнучких і складних тестів для веб-служб.

Розглянемо переваги, що були виявлені користувачами при використанні SoapUI:

1. Розширені можливості для тестування SOAP — інструмент надає глибоку підтримку SOAP-запитів, що дозволяє тестувати всі аспекти веб-служб, заснованих на SOAP.
2. Автоматизація та налаштування сценаріїв — SoapUI дозволяє створювати складні сценарії для автоматизованого тестування, що полегшує роботу з великими API-проєктами.
3. Можливості навантажувального тестування — інструмент забезпечує можливість виконання навантажувальних тестів для оцінки продуктивності API під великим навантаженням.

Розглянемо недоліки, що були виявлені користувачами при використанні SoapUI:

1. Високе споживання ресурсів — під час виконання складних або навантажувальних тестів SoapUI може значно навантажувати систему.
2. Обмежені можливості інтерфейсу для REST API — хоча SoapUI підтримує REST, він не такий зручний для цього протоколу порівняно з іншими інструментами.
3. Платна версія для розширеного функціоналу — розширені можливості, такі як навантажувальні тести і детальні налаштування безпеки, доступні лише у платній версії.

У цілому, SoapUI є ефективним інструментом для тестування веб-служб, особливо тих, що побудовані на SOAP. Це чудовий вибір для проєктів, де потрібна автоматизація складних сценаріїв та оцінка продуктивності API, але його високе споживання ресурсів і обмежена

зручність для REST можуть зробити його менш привабливим для деяких команд, особливо тих, що працюють лише з REST API.

Обираючи інструмент для тестування запитів до програмного забезпечення, що реалізує мобільний застосунок-довідник Українських Карпат, було враховано низку важливих аспектів, щоб забезпечити ефективність, надійність і зручність процесу тестування. На основі проведеного аналізу було визначено, що Postman є оптимальним інструментом для наших потреб, з огляду на такі переваги:

1. Підтримка різних типів API - Postman дозволяє тестувати сервіси, які працюють з протоколами HTTP, REST, SOAP та GraphQL. Це критично важливо для нашого застосунку, який активно використовує REST API для отримання та надсилання даних про туристичні маршрути, об’єкти, розташування користувача тощо.

2. Зручний інтерфейс - інтуїтивно зрозумілий інтерфейс Postman значно спрощує процес створення, надсилання та аналізу API-запитів, що є корисним під час перевірки працездатності функціоналу застосунку, зокрема роботи з мапами, запитів до серверної бази даних і підвантаження об'єктів.

3. Можливості автоматизації - Postman дозволяє створювати колекції запитів та запускати їх у пакетному режимі, що спрощує автоматизоване тестування всього API-доступу застосунку — від отримання даних про місцевість до фільтрації за категоріями об’єктів.

4. Активна спільнота та документація - велика база знань, активна спільнота та офіційна документація допомагають швидко вирішувати будь-які питання, що виникають під час налаштування чи запуску тестів, а також підвищують загальну ефективність процесу розробки [15].

Таким чином, використання Postman забезпечує гнучке та масштабоване тестування REST API, що є важливим компонентом якісної розробки мобільного довідника з динамічними запитами до серверної частини.

2.5 Аналіз та вибір інструментарію обраної мови програмування для реалізації функціоналу

Вибір інструментів для реалізації мобільного застосунку-довідника Українських Карпат є критично важливим етапом розробки. Застосунок має забезпечувати швидке надання довідкової інформації, інтерактивну карту, функцію SOS-сповіщення та підтримку офлайн-доступу. Тому необхідно обрати технології, які дозволяють ефективно реалізувати вказаний функціонал. При виборі враховувались: сумісність із мовою програмування, продуктивність, швидкість розробки, підтримка платформ, можливість розширення та активність спільноти.

Для створення користувацького інтерфейсу було обрано React Native — кроссплатформенный фреймворк, що дозволяє будувати нативні мобільні застосунки з єдиною кодовою базою для Android і iOS. Важливими перевагами є підтримка адаптивного дизайну, жестових взаємодій, мультимедійних компонентів і зручної навігації. Використання платформи Expo спрощує тестування та пришвидшує розгортання без складного налаштування середовища. React Native має розвинену екосистему, велику спільноту, готові компоненти та приклади, що дозволяє прискорити реалізацію. Декларативний підхід до створення інтерфейсу полегшує розробку, даючи змогу зосередитися на логіці взаємодії. Таким чином, зв’язка React Native + Expo була обрана як оптимальне рішення для реалізації інтерфейсної частини мобільного застосунку [16].

API є ключовим елементом функціоналу мобільного застосунку-довідника Українських Карпат, оскільки забезпечує отримання, оновлення та відображення даних про туристичні об'єкти, зокрема з прив’язкою до географічних координат на інтерактивній карті. У межах проєкту, реалізованого з використанням JavaScript і React Native, для здійснення HTTP-запитів було обрано вбудований інструмент fetch. Він не потребує додаткових залежностей, має простий синтаксис і підтримує всі

основні методи запитів (GET, POST, PUT, DELETE), що дозволяє ефективно взаємодіяти з віддаленим сервером. Fetch забезпечує зручну обробку відповідей у форматі JSON, налаштування заголовків, авторизації та обробку помилок. Завдяки підтримці async/await його використання сприяє простішому управлінню асинхронними процесами. Крім того, fetch легко інтегрується з картографічними сервісами та іншими API, не погіршуючи продуктивність застосунку. Універсальність і стабільність роботи на різних пристроях зробили fetch оптимальним рішенням для забезпечення API-взаємодії в межах цього проєкту [17].

Однією з ключових вимог до мобільного застосунку-довідника Українських Карпат є можливість використання в офлайн-режимі. Це особливо важливо для гірських регіонів, де зв’язок нестабільний або відсутній. У таких умовах виникає потреба у локальному збереженні даних, зокрема інформації про авторизованого користувача та нові об’єкти, додані на карту. Це дозволяє забезпечити персоналізований досвід і продовжити роботу із застосунком без підключення до інтернету. Для реалізації цього функціоналу, з урахуванням використання React Native та потреби в простій інтеграції, було обрано бібліотеку @react-native-async-storage/async-storage. Вона є стандартним інструментом для зберігання даних у форматі ключ–значення в екосистемі React Native. AsyncStorage забезпечує зручний API для збереження, читання та видалення даних, стабільно працює на Android і iOS, не потребує додаткових нативних налаштувань і відзначається активною підтримкою спільноти. Простота реалізації, стабільність і ефективність у роботі з невеликими обсягами інформації роблять цю бібліотеку оптимальним рішенням для мобільного застосунку-довідника, орієнтованого на використання у віддалених регіонах Карпат [18].

3. РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ-ДОВІДНИКА УКРАЇНСЬКИХ КАРПАТ

3.1 Визначення функціональності та основних можливостей додатку

Першим кроком для реалізації поставленого завдання є визначення функціональності та основних можливостей проектованого мобільного застосунку. На основі проведеного у першому розділі аналізу предметної області було сформовано перелік вимог до функціоналу, який має бути реалізований у межах розробки мобільного застосунку-довідника Українських Карпат. Ці вимоги відповідають актуальним потребам туристів та мандрівників, які шукають надійний і зручний інструмент для планування подорожей у гірському регіоні. До основних функціональних можливостей, які має реалізовувати проектований застосунок, належать:

1. Реєстрація користувачів — необхідно забезпечити можливість створення нового облікового запису шляхом введення персональних даних (ім'я, вік, номер телефону, електронна пошта та пароль).

2. Авторизація користувачів — необхідно реалізувати вхід користувача до системи за допомогою введення електронної пошти та пароля з перевіркою правильності введених даних.

3. Головна сторінка з довідковою інформацією — необхідно реалізувати головний екран, на якому відображатимуться об'єкти Карпат із короткою інформацією: назвою, фотографією, категорією, висотою та координатами.

4. Фільтрація об'єктів за категоріями — необхідно надати користувачеві можливість фільтрувати список місць за обраною категорією (туристичні місця, міста, гори, озера, водоспади, готелі тощо) для спрощення навігації.

5. Детальний перегляд інформації про обраний об'єкт — при виборі об'єкта користувач повинен мати можливість ознайомитися з розгорнутою інформацією: повним описом, галереєю зображень, висотою над рівнем моря, місцем розташування та іншими важливими даними.

6. Інтерактивна карта — необхідно реалізувати карту з відображенням усіх об'єктів на основі їхніх координат. При натисканні на маркер місця користувач повинен мати змогу переходити до детальної сторінки об'єкта.

7. Функціонал додавання нових об'єктів — необхідно надати можливість користувачам додавати нові місця безпосередньо через застосунок, заповнюючи форму з інформацією про локацію (назва, опис, координати, зображення тощо).

8. Функція SOS-сповіщення — необхідно реалізувати можливість швидкого надсилання екстреного повідомлення разом із координатами поточного місцезнаходження користувача. Перед відправкою має бути передбачено підтвердження дії.

9. Адаптивний інтерфейс для мобільних пристроїв — інтерфейс застосунку має бути оптимізований для різних розмірів екранів смартфонів і планшетів, забезпечуючи комфортне користування, зручну навігацію та чітке відображення контенту.

Таким чином, функціональні вимоги до застосунку охоплюють усі основні аспекти інформаційного супроводу туристів у Карпатах. Реалізація зазначених можливостей дозволить користувачам отримувати актуальну довідкову інформацію, орієнтуватися в реальному часі на місцевості та діяти оперативно у випадку надзвичайних ситуацій.

Запропонований перелік функціональностей базується на сучасних підходах до архітектури мобільних застосунків, що активно використовуються у процесі розробки програмного забезпечення різного призначення. Використання React Native дозволяє реалізувати адаптивний та інтерактивний інтерфейс, що охоплює усі основні функціональні модулі, зокрема — навігацію, авторизацію, роботу з картою та введенням нових даних [1]. Така технологія забезпечує не лише зручність для кінцевого користувача, а й гнучкість у процесі проєктування та подальшого вдосконалення застосунку. Структуру функціональності було сформовано з урахуванням типових сценаріїв використання, які рекомендовано враховувати

при створенні туристичних мобільних систем [2], адже саме вони дозволяють максимально точно задовольнити потреби цільової аудиторії та забезпечити ефективну взаємодію користувача з системою.

3.2 Проектування алгоритму роботи мобільного застосунку-довідника Українських Карпат

Перш ніж перейти до практичної реалізації мобільного застосунку-довідника Українських Карпат, необхідно спроєктувати загальний алгоритм його роботи. Такий алгоритм є ключовим етапом розробки програмного забезпечення, оскільки визначає логіку взаємодії між користувачем і системою, послідовність обробки даних, способи обміну інформацією між екраном карти, довідником об'єктів та сервісними функціями застосунку. Основні кроки алгоритму роботи проектованого програмного забезпечення можна визначити так:

1. Реєстрація користувачів:

- Користувач запускає застосунок і вибирає опцію "Зареєструватися".
- Користувач вводить необхідні дані (ім'я, телефон, вік, електронну пошту та пароль).
- Система перевіряє правильність заповнення полів і створює новий обліковий запис у локальній базі даних.
- Після успішної реєстрації користувач автоматично переходить до авторизації.

2. Авторизація користувачів:

- Користувач вибирає опцію "Увійти" та вводить свої облікові дані (email і пароль).
- Система перевіряє правильність введених даних у локальній базі даних.
- У разі успішної перевірки користувач переходить на головний екран застосунку; у разі помилки виводиться повідомлення про невірні дані.

3. Перегляд головної сторінки:

- Система відображає головний екран зі списком доступних об'єктів Карпат.
- Кожен об'єкт показується у вигляді картки з назвою, фотографією, категоріями, висотою та координатами.

4. Фільтрація об'єктів:

- Користувач має можливість вибрати категорію (наприклад, туристичні місця, міста, гори, озера тощо).
- Система оновлює список об'єктів відповідно до обраного фільтра.

5. Детальний перегляд інформації про об'єкт:

- Користувач натискає на потрібний об'єкт у списку.

- Система відкриває окремий екран з розгорнутою інформацією: повний опис, галерея зображень, висота над рівнем моря, координати місця на мапі, місцезнаходження.

6. Робота з інтерактивною картою:

- Користувач переходить на вкладку "Карта".
- Система завантажує карту з маркерами всіх об'єктів із бази даних.
- При натисканні на маркер відкривається коротка інформація про місце з можливістю переходу до детального перегляду.

7. Додавання нового об'єкта:

- Користувач натискає кнопку "Додати місце" на сторінці карти.
- Відкривається форма для введення даних нового об'єкта (назва, опис, координати, категорія, фотографії).
- Після заповнення і підтвердження введені дані зберігаються у локальне сховище у вигляді окремого файлу new_locations.json.
- Система автоматично оновлює карту та список місць, додаючи новий об'єкт.

8. Використання SOS-функції:

- Користувач відкриває вкладку "SOS".

- На екрані відображається стандартна інструкція поведінки у надзвичайній ситуації.
- При натисканні на кнопку "Надіслати SOS" система запитує підтвердження дії.
- У разі підтвердження система імітує надсилання поточних координат на допомогу.

9. Вихід із облікового запису:

- Користувач переходить на вкладку "Вийти".
- Система очищає локальне збереження даних користувача і перенаправляє на екран авторизації.

Таким чином, спроєктований алгоритм роботи мобільного застосунку забезпечує комплексну реалізацію функціональних вимог до системи, спрямованих на полегшення доступу користувачів до туристичної інформації, підтримку інтерактивної взаємодії з об'єктами на карті, гарантування безпеки під час подорожей та забезпечення зручного й інтуїтивного користування застосунком навіть у складних польових умовах. Послідовне виконання цього алгоритму дозволяє створити надійний, ефективний і зручний інструмент для дослідження Карпатського регіону.

3.3 Програмна реалізація ключових функцій застосунку

Наступним кроком перейдемо до опису програмної реалізації основних функціональних модулів мобільного застосунку-довідника Українських Карпат. Кожен модуль виконує окрему роль у забезпеченні повноцінної роботи системи: від обробки взаємодії з головним екраном і перегляду детальної інформації до відображення об’єктів на карті, додавання нових локацій, використання SOS-функції та завершення сесії користувача. У подальших підрозділах послідовно представлено архітектуру, логіку роботи та призначення кожного з реалізованих елементів.

1.Реалізація механізму реєстрації користувача.

У межах реалізації клієнтської логіки мобільного застосунку-довідника Українських Карпат окрему увагу приділено механізму реєстрації нового користувача. Цей функціональний блок дозволяє створити обліковий запис, зберегти його локально та надати персоналізований доступ до подальших функцій застосунку. Збереження даних реалізовано за допомогою засобів бібліотеки AsyncStorage, що дозволяє працювати з ключ-значенням у пам’яті пристрою.

Основна логіка додавання нового користувача реалізована у функції addUser(user), яка приймає об’єкт користувача, зчитує збережені раніше облікові записи, додає новий об’єкт до масиву та зберігає оновлений список у локальному сховищі. Всі дії супроводжуються обробкою помилок, що фіксуються у консолі у разі виникнення збоїв на етапах читання або запису.

Інтерфейсна частина, що відповідає за реєстрацію, реалізована у компоненті RegisterScreen. У ньому використано хук useState для керування значеннями всіх полів форми: імені, телефону, віку, електронної пошти та пароля. При натисканні кнопки «Зареєструватися» активується функція handleRegister, яка перевіряє, чи всі поля заповнено, викликає функцію додавання користувача до сховища, зберігає електронну пошту поточного користувача та виконує перенаправлення на головний екран застосунку.

Графічний інтерфейс форми містить компоненти TextInput для кожного поля з відповідними налаштуваннями типу введення, стилізації та обробки подій зміни тексту. Завершується форма кнопкою Button, натискання якої ініціює процедуру реєстрації. Таким чином, реалізований функціонал забезпечує повний цикл створення користувача в застосунку з локальним збереженням даних і логікою перенаправлення після успішної реєстрації. Повна реалізація програмного коду наведена у Додатку А.

2.Програмна реалізація логіки авторизації користувача.

Механізм авторизації в мобільному застосунку-довіднику Українських Карпат реалізовано засобами JavaScript у середовищі React Native з використанням бібліотеки AsyncStorage для збереження та читання даних користувачів. Основна функція авторизації передбачає перевірку відповідності введеної електронної пошти та пароля раніше збереженим обліковим даним.

У логіку реалізації входить функція getUser(email, password, onSuccess, onError), яка здійснює зчитування збережених користувачів через метод AsyncStorage.getItem('users'). Дані перетворюються у формат масиву об’єктів методом JSON.parse(), після чого здійснюється пошук користувача, що відповідає критеріям автентифікації. У разі збігу викликається зворотний метод onSuccess з передачею знайденого користувача, інакше — з порожнім масивом. Якщо при обробці виникає помилка, викликається метод onError.

У компоненті LoginScreen реалізовано збереження введених облікових даних через хуки стану useState, що відповідають за поля email та password. Для зберігання токена автентифікації використовується контекст AuthContext, а метод signIn() ініціює запуск сесії користувача та навігацію до головного екрану застосунку. У разі відсутності відповідності введених даних у базі, система формує повідомлення про помилку через вбудовану функцію Alert.

Інтерфейс форми авторизації побудовано на базі компонентів TextInput та Button. Усі поля мають властивість двостороннього зв’язку зі станом (value/onChangeText) та стилізовані відповідно до загальної візуальної теми застосунку. Кнопка виклику функції авторизації забезпечує прямий перехід до головного екрану лише у випадку успішного входу, що реалізує базові принципи безпечної взаємодії з користувачем.

Повна реалізація коду клієнтської частини модуля авторизації наведена у додатках (Додаток Б).

3.Програмна реалізація головного екрана застосунку.

Головний екран мобільного застосунку-довідника Українських Карпат реалізовано як стартову точку взаємодії користувача із системою. Він поєднує в собі кілька ключових функціональних зон: доступ до рекомендацій з безпеки, інтерактивної карти, випадаючого меню для вибору категорій туристичних об’єктів та динамічного списку доступних локацій.

Основна логіка реалізована у вигляді компонента, що зберігає внутрішній стан обраного типу місця та видимості меню категорій. Для відображення об'єктів використано попередньо підключений JSON-файл, який містить структуру даних про локації з полями назви, типу, зображення та координат. Вбудована функція фільтрації динамічно оновлює список на основі вибраної категорії, дозволяючи користувачеві оперативно знаходити потрібні об'єкти.

Інтерфейс побудовано за допомогою компонувальних засобів React Native та бібліотеки react-native-paper, яка використовується для створення випадаючого меню з піктограмами типів локацій. Кожна категорія має прив’язану іконку, що покращує сприйняття інтерфейсу. Також реалізовано кнопки для переходу до окремих розділів — інструкцій з безпеки та інтерактивної карти, — що дозволяє користувачу швидко отримати доступ до супутнього функціоналу.

Список локацій формується на основі фільтрованих даних і виводиться за допомогою компонента FlatList, який забезпечує ефективне рендерення великої кількості елементів. Кожен об’єкт подається у вигляді картки, що містить зображення, назву, тип та базову інформацію. Для збереження зручності навігації реалізовано плавне оновлення списку після вибору категорії без необхідності перезавантаження сторінки.

Уся реалізована логіка головного екрана розміщена у відповідному модулі, повний код якого наведено у додатках (Додаток В).

4.Програмна реалізація логіки відображення об’єктів на інтерактивній карті.

Реалізація інтерактивної карти є ключовим функціональним компонентом мобільного застосунку-довідника Українських Карпат. Метою даного модуля є забезпечення просторового представлення туристичних об’єктів, що дозволяє користувачеві швидко знаходити місця на карті, орієнтуватися в місцевості, а також взаємодіяти з інформаційними точками інтересу.

Для побудови карти використано компонент MapView з бібліотеки react-native-maps, що забезпечує повноцінну підтримку векторної географічної візуалізації з можливістю встановлення координат, масштабування та інтеграції маркерів. Карта ініціалізується з параметром initialRegion, де задаються координати центральної точки регіону та параметри масштабування (latitudeDelta, longitudeDelta), що відповідають території Українських Карпат. Дані для відображення маркерів надходять із двох джерел: статичного файлу locations.json, що містить базовий набір об’єктів, та з локального сховища користувача, реалізованого через AsyncStorage. При завантаженні компонента карти виконується об’єднання даних з обох джерел, після чого формуються об’єкти з уніфікованою структурою. Кожен об’єкт має унікальний ідентифікатор, координати (latitude, longitude), текстові поля name, description, а також, за наявності, масив зображень images.

Виведення об’єктів на карту здійснюється за допомогою компонента Marker. Кожному маркеру передається об'єкт з полями coordinate, title та description, що формує інформаційне вікно при взаємодії. У випадку, якщо до локації прикріплено фотографію, вона додатково виводиться на маркері як мініатюра з обмеженими розмірами та округленням — це досягається через компонент Image зі стилізацією, що імітує аватарку. Ключовим елементом є синхронне завантаження даних при монтуванні компонента. Для цього використовується хук useEffect(), у якому реалізована функція loadLocations(). Вона асинхронно зчитує JSON-рядок з ключа new_locations у AsyncStorage, трансформує його у масив за допомогою JSON.parse() та об’єднує його з

основними даними. Це забезпечує постійне оновлення карти без потреби оновлення застосунку або сервера.

Особливу увагу приділено обробці виключень: у випадку помилки при зчитуванні даних реалізоване повідомлення про помилку в консоль, що допомагає відслідковувати працездатність модуля на етапі тестування.

Таким чином, інтерактивна карта виконує подвійну функцію — відображення актуальної географічної інформації про туристичні об’єкти та інтеграцію з іншими модулями (перехід до деталей, перегляд фото тощо). Така реалізація забезпечує повну автономність роботи карти, підтримку динамічного контенту та можливість масштабування у майбутньому. Повний код модуля відображення карти наведено у додатку Ґ.

5. Програмна реалізація логіки додавання нових об’єктів.

У межах цього етапу реалізовано механізм додавання нових туристичних локацій безпосередньо із мобільного застосунку, що дає змогу користувачам самостійно розширювати базу об’єктів. Функціональність реалізовано у вигляді інтерактивної форми з полями введення, можливістю прикріплення зображень та автоматичним збереженням результату у локальне сховище пристрою (AsyncStorage).

Для керування станом форми створено об’єкт newPlace, що містить набір полів: name, type, description, location, latitude, longitude, images. Усі поля ініціалізуються порожніми, з метою їх подальшого динамічного заповнення користувачем. Окремий булевий стан modalVisible відповідає за керування модальним вікном з формою.

Обробку зображень реалізовано через бібліотеку expo-image-picker. Користувач отримує доступ до медіатеки пристрою, де може вибрати одну або кілька фотографій. URI обраного зображення зберігається в масиві images, що дозволяє виводити його на інтерфейсі або зберігати для подальшої

візуалізації на карті. Вибір зображень не є обов’язковим із технічної точки зору, але реалізаційно є умовою завершення додавання нового місця.

Перед збереженням нової локації здійснюється валідація всіх обов’язкових полів. У випадку незаповнення одного з них виводиться сповіщення через модуль Alert. При успішному проходженні валідації створюється новий об’єкт newLocation, у якому обов’язково формуються:

- Унікальний id, що визначається на основі поточного розміру масиву locations.
- Поле type, що трансформується у масив значень на основі розділення за комами.
- Координати latitude та longitude, які конвертуються у числовий формат методом parseFloat.

Створений об’єкт додається до наявного масиву locations, після чого виконується оновлення стану та запис змін до AsyncStorage під ключем new_locations. При збереженні застосовується фільтрація — у сховищі фіксуються лише ті об'єкти, які було додано користувачем (id > data.length), щоб уникнути дублювання основних даних. Після завершення процесу форма скидається до початкового стану, модальне вікно закривається, і користувач отримує повідомлення про успішне додавання. Завдяки цьому реалізованому механізму користувачі можуть самостійно створювати нові записи, які негайно відображаються на карті без необхідності перезапуску застосунку.

Функціональність була реалізована з урахуванням стабільної інтеграції з раніше реалізованими модулями: виведенням на карті (через MapView), візуалізацією зображень (Image), збереженням у пам’яті пристрою (AsyncStorage). Такий підхід дозволяє гнучко масштабувати застосунок, забезпечуючи користувацький внесок у розвиток контентної бази. Повний код модуля додавання об’єктів наведено у додатку Д.

6. Програмна реалізація відображення детальної інформації про обраний об’єкт.

Наступним кроком реалізовано механізм виведення детальної інформації про туристичний об'єкт, що був вибраний користувачем. Цей екран є критично важливим для забезпечення повноцінної довідкової взаємодії у межах застосунку, оскільки містить поглиблені характеристики кожного місця — його назву, опис, координати, висоту, тип, місцезнаходження, а також фотогалерею. З технічної точки зору, компонент отримує всі необхідні дані через параметри навігації (route.params) та виводить їх в інтерфейсі з урахуванням умовного рендерингу.

На початку реалізується компонент DetailPage, який імпортує бібліотеку react-native-swiper для організації каруселі зображень, а також стандартні компоненти React Native (ScrollView, Text, Image, TouchableOpacity, View). Структура компонента передбачає використання локального стану isDescriptionExpanded для динамічного контролю за розгортанням довгого опису. Основна структура компонента будується з використанням ScrollView, що дозволяє відображати великі об’єми інформації з підтримкою вертикального прокручування. Текстові поля стилізовано відповідно до загального дизайну, а дані виводяться за умовної наявності: для кожного поля перевіряється його існування перед візуалізацією, що запобігає помилкам рендерингу.

Функція truncateDescription() динамічно обрізає текст, якщо його довжина перевищує 450 символів, і додає перемикач для розгортання повного опису. Функції renderCoordinates() і renderHeight() відповідають за формування відображення координат (широта і довгота) та висоти над рівнем моря, використовуючи структури View з текстовими блоками. Ці функції реалізовано як окремі компоненти в межах сторінки з метою покращення читабельності та повторного використання.

Фотогалерея побудована на основі компонента Swiper, який отримує масив URI з поля images. Кожне зображення відображається через компонент Image, що автоматично масштабовано до встановлених розмірів. Стилізація

елементів галереї відповідає стандартам UI/UX для мобільних застосунків і забезпечує зручність перегляду.

Наприкінці реалізовано блок із виведенням категорій (типів) об'єкта. Значення поля type подається у вигляді масиву рядків, кожен з яких відображається у стилізованому контейнері, що імітує теги. Це дозволяє користувачеві швидко зорієнтуватись у функціональному призначенні місця — чи це оглядовий майданчик, гірська вершина, озеро тощо.

Вся взаємодія між екранами організована через React Navigation, а дані передаються безпосередньо з попередньої сторінки вибору об’єкта. Уся інформація зберігається локально і не потребує підключення до мережі для її відображення, що особливо важливо в умовах туристичних маршрутів з обмеженим інтернет-доступом. Таким чином, реалізовано повноцінний інтерфейс деталізації туристичних об'єктів, який поєднує текстову, візуальну та географічну інформацію в одному екрані. Такий підхід дозволяє користувачу отримати усі необхідні відомості про локацію безпосередньо в застосунку, що покращує зручність використання і практичну цінність системи. Повний код модуля відображення детальної інформації наведено у додатку Е.

7. Програмна реалізація SOS-функції у мобільному застосунку.

Наступним етапом реалізовано функціональність SOS-функції, що призначена для швидкого інформування про надзвичайну ситуацію із зазначенням поточного місцезнаходження користувача. Дана функція є критично важливою для користувачів, які перебувають у важкодоступних гірських районах, де може виникнути потреба у виклику допомоги.

З технічної точки зору, реалізація функції передбачає доступ до геолокації пристрою в режимі реального часу, підтвердження наміру користувача активувати сигнал SOS, отримання координат через відповідний API та виведення інформації у вигляді повідомлень на екран. Для взаємодії з геолокаційними сервісами використовується модуль expo-location, який

забезпечує запит дозволу на доступ до поточних координат пристрою та отримання даних з GPS.

Основна логіка функціоналу реалізована у вигляді асинхронної функції, що викликається при натисканні на кнопку SOS. Перед виконанням основної дії система виводить діалогове вікно з підтвердженням намірів користувача. У разі підтвердження виконується запит на дозвіл до геолокації, після чого за допомогою методу getCurrentPositionAsync() отримуються точні координати (широта та довгота). Успішне виконання супроводжується виведенням повідомлення про успішне надсилання SOS-сигналу, а у випадку помилки – повідомленням про збій.

Інтерфейсний компонент даної функції реалізовано у вигляді окремого екрану, що містить текстові інструкції з правилами дій у випадку загублення, а також кнопку надсилання SOS-сигналу. Візуальна частина підкріплена графічними іконками для покращення сприйняття та акцентування уваги на критичних діях.

Таким чином, реалізований функціонал дозволяє користувачу швидко надіслати тривожний сигнал, отримати координати поточної локації та ознайомитися з базовими інструкціями щодо поведінки в екстремальних умовах. SOS-функція підвищує загальний рівень безпеки застосунку та адаптує його до реальних потреб туристів у польових умовах. Повний код реалізованого модул. наведено у додатку Ж.

8.Програмна реалізація логіки завершення сесії користувача.

Наступним кроком реалізовано функціональність завершення сесії користувача у мобільному застосунку-довіднику. Цей функціонал виконує важливу роль з точки зору безпеки, оскільки дозволяє користувачеві вийти зі свого облікового запису, очистити тимчасові дані автентифікації та забезпечити правильне переключення між різними користувачами.

З технічної точки зору, завершення сесії реалізується через механізм глобального контексту автентифікації, який відповідає за збереження та зміну

стану доступу до системи. При переході на спеціальний екран виходу, у контексті застосунку ініціюється функція, що виконує очищення збереженого токена авторизації (який містився у локальному сховищі пристрою) та скидає глобальний стан автентифікації. У результаті цього користувач автоматично перенаправляється на екран авторизації.

Процес виконується без додаткової взаємодії з боку користувача: при відкритті екрану виходу система одразу виконує процедуру завершення сесії. У візуальній частині компонента на цей час відображається індикатор виконання та повідомлення «Вихід...», що дає зрозуміти користувачеві, що процес триває.

Загальна логіка передбачає:

- Зчитування функції виходу з глобального контексту автентифікації;
- Автоматичне виконання цієї функції при відкритті екрану;
- Очищення локального сховища від авторизаційного токена;
- Скидання стану поточного користувача;
- Повернення до початкового екрану входу.

Візуальне відображення побудовано у вигляді централізованого контейнера з елементом індикації завантаження та текстовим блоком. Це дозволяє не лише забезпечити технічну правильність завершення сесії, а й надати користувачеві чіткий зворотний зв’язок про виконувану дію. Таким чином, реалізована логіка забезпечує безпечне завершення сесії та є важливою частиною загальної архітектури автентифікації в межах мобільного застосунку. Повний код реалізованого модуля наведено у додатку З.

3.4 Архітектура мобільного застосунку-довідника

Створення дієвого мобільного додатку потребує чітко окресленої його архітектури. Додаток-довідник «Українські Карпати» побудований за принципами багатошарової архітектури, що дає змогу зробити робочу логіку

на окремі складові,полегшити супровід та масштабування системи. Основою є клієнт-серверна модель, де мобільний додаток виступає клієнтом,а серверна складова забезпечує обробку запитів, зберігання даних та доступ до API.
Основні складові архітектури:

- Користувацький Інтерфейс(UI) – реалізовано на базі React Native З Використанням Компонування екранів, навігації та обробки подій.
- Логіка бізнес-процесів – включає перевірку вхідних даних, роботу з локальним сховищем, втіленняSOS-функції, обробку помилок
- Робота з даними – взаємодія з віддаленим API виконується через fetch;при відсутності інтернет-з'єднання використовується локальне сховище (AsyncStorage).
- Інтеграція з картографічними сервісами – за допомогою Google Maps або аналогічних для візуалізації геолокаційних об’єктів.
- Локальна база – зберігає історію відвідувань, профіль користувача та кешовану інформацію для роботи офлайн.

Такий модульний підхід дозволяє ефективно управляти проектом, а також забезпечує стабільну роботу додатку навіть у складних умовах,притаманні гірським регіонам.

3.5 Тестування розробленого програмного забезпечення

Наступним кроком після реалізації основних функціональних можливостей є тестування розробленого мобільного застосунку-довідника Українських Карпат. Основною метою тестування є перевірка коректності роботи ключових функцій, забезпечення їх стабільності, відповідності заданим вимогам, а також виявлення можливих помилок або недопрацювань у логіці роботи програми. Тестування буде зосереджено на перевірці таких аспектів, як реєстрація нового користувача, авторизація існуючого користувача, перегляд та фільтрація списку туристичних об'єктів, детальний

перегляд інформації про об'єкти, робота інтерактивної карти, додавання нових локацій на карту, а також функціональність надсилання SOS-сигналу.

Після тестування процесу реєстрації та авторизації буде проведено перевірку роботи головного екрана застосунку та механізму фільтрації об'єктів за категоріями. Окрему увагу буде приділено тестуванню інтерактивної карти: коректності відображення маркерів, точності координат і роботи модуля додавання нових локацій користувачами. Також буде перевірено працездатність механізму надсилання SOS-сигналу для екстрених ситуацій. Таким чином, тестування дозволить переконатися у працездатності всіх критично важливих компонентів застосунку та підготувати продукт до подальшого використання кінцевими користувачами.

Першим кроком проведемо тестування процесу реєстрації у застосунку, ініціювавши процес реєстрації з тестовими даними користувача (Рисунок 3.1).

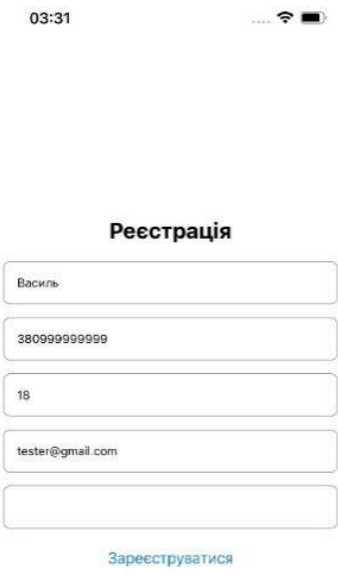

Рисунок 3.1 – Інтерфейс реєстрації користувача

Для перевірки коректності роботи процесу реєстрації, спробуємо авторизуватися у застосунку за допомогою даних, що були введені при реєстрації (Рисунок 3.2).

Рисунок 3.2 – Процес авторизації користувача у застосунку

Як можна побачити, в результаті авторизації користувача було успішно авторизовано у додатку та на його інтерфейс було відображено головну сторінку застосунку з інформацією про поточні туристичні об'єкти у базі даних (Рисунок 3.3).

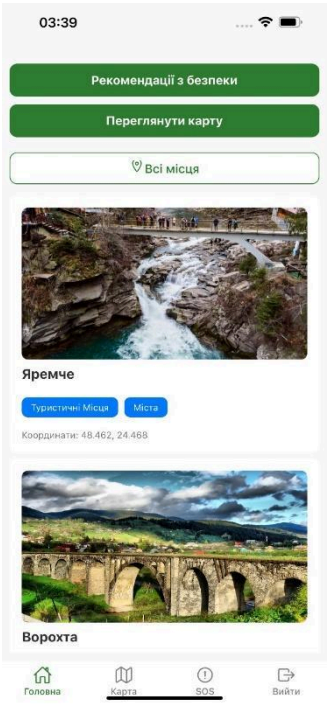

Рисунок 3.3 – Головна сторінка застосунку після авторизації

Далі проведемо тестування головної сторінки застосунку. Після авторизації користувача, на головному екрані відображається кнопка переходу до інтерактивної карти, кнопка перегляду рекомендацій з безпеки та список доступних локацій із можливістю фільтрації за категоріями.

Для перевірки роботи механізму фільтрації локацій за категоріями скористаємося випадаючим меню, вибравши одну з доступних категорій, наприклад «Гори». У результаті список на головній сторінці було автоматично оновлено та відфільтровано лише ті об'єкти, які відповідають вибраній категорії (Рисунок 3.4).

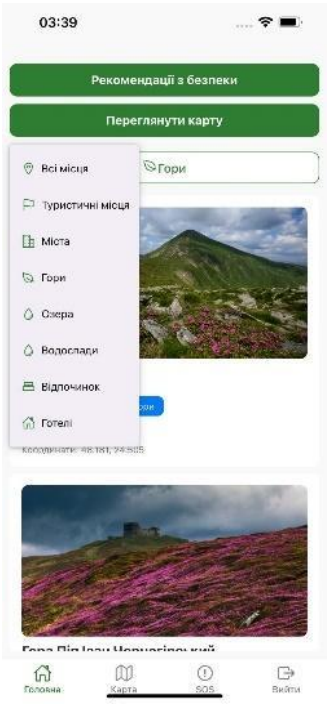

Рисунок 3.4 – Фільтрація туристичних об’єктів за категорією «Гори»

Наступним етапом проведемо тестування функціоналу детального перегляду інформації про обраний об'єкт. Для цього обираємо один із доступних елементів у списку та переходимо на сторінку деталізації. На цій сторінці успішно відображаються детальна інформація про об'єкт, координати, опис, висота (за наявності), а також реалізований механізм перегляду зображень за допомогою каруселі (Рисунок 3.5).

Рисунок 3.5 – Детальний перегляд інформації про туристичний об’єкт

Після тестування роботи основного списку об'єктів, перейдемо до тестування інтерактивної карти. Для цього натискаємо кнопку "Переглянути карту" на головному екрані. Система відкриває інтерактивну карту з маркерами, що відповідають існуючим об'єктам у базі даних (Рисунок 3.6).

Рисунок 3.6 – Відображення туристичних об’єктів на інтерактивній карті

Для перевірки функціоналу додавання нового місця на карті скористаємося кнопкою «Додати нове місце», розташованою під картою.

Система відкриває спеціальну форму, де необхідно заповнити основну інформацію про нове місце: назву, тип, опис, координати та завантажити зображення (Рисунок 3.7).

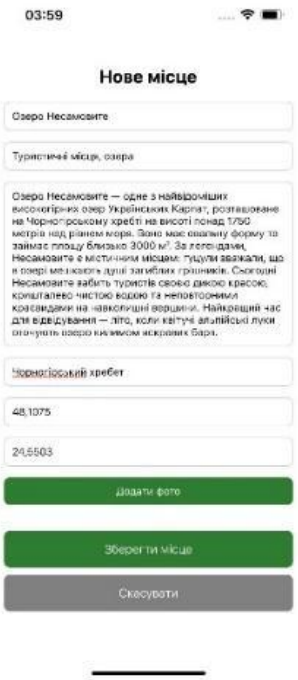

Рисунок 3.7 – Форма додавання нового туристичного об’єкта

Після заповнення всіх полів та натискання кнопки «Зберегти місце», система успішно зберігає новий об'єкт у локальному сховищі та відображає його серед існуючих маркерів на карті. Доданий об'єкт миттєво з'являється на карті без необхідності перезапуску застосунку (Рисунок 3.8).

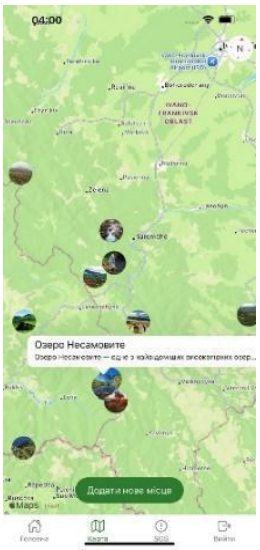

Рисунок 3.8 – Відображення нового об’єкта на карті після додавання

На завершення проведемо тестування функціоналу SOS-сигналу. Для цього переходимо на відповідну сторінку "SOS". На екрані відображено коротку інструкцію дій у разі надзвичайної ситуації, а також велику червону кнопку для виклику SOS-сигналу (Рисунок 3.9).

Рисунок 3.9 – Сторінка SOS-сигналу у мобільному застосунку

Натискання на кнопку SOS викликає діалог підтвердження (Рисунок 3.10). Після підтвердження система запитує дозвіл на доступ до геолокації користувача, зчитує поточні координати та моделює надсилання сигналу про допомогу. У результаті виводиться повідомлення про успішне надсилання SOS-сигналу.

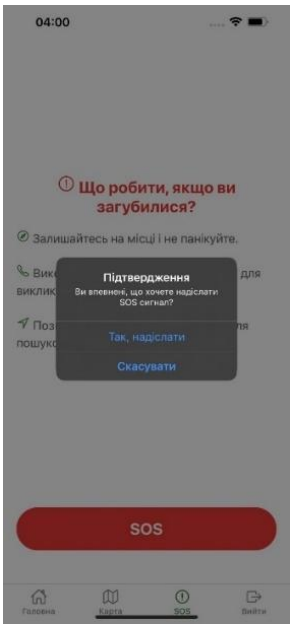

Рисунок 3.10 – Підтвердження надсилання SOS-сигналу у мобільному застосунку

Отже, створена мобільний застосунок цілковито відповідає висунутим спочатку функціональним потребам. Втілення основних функцій —від інтерактивної карти до відправлення SOS-сигналів —надає користувачам комфортний засіб для орієнтування та взаємодії з туристичними пам'ятками Карпатського краю. Досягнуті результати демонструють практичну корисність впровадження такого цифрового рішення у сфері туристичного обслуговування. з безпеки та список доступних локацій із можливістю фільтрації за категоріями.

ВИСНОВОК

У процесі виконання дипломної роботи було проведено аналіз сучасного стану цифрових рішень у сфері туристичної навігації Карпатського регіону, що виявило необхідність створення мобільного застосунку, здатного забезпечити зручний доступ до інформації про туристичні об’єкти та маршрути навіть в умовах обмеженого інтернет-з'єднання. На основі проведеного аналізу предметної області та визначення відповідного інструментарію було розроблено мобільний додаток-довідник для користувачів, які подорожують Українськими Карпатами.

У процесі розробки мобільного застосунку було реалізовано основний функціонал, зокрема: реєстрацію та авторизацію користувачів, відображення головної сторінки з інтерактивною фільтрацією об'єктів, деталізований перегляд інформації про кожний туристичний об’єкт, інтеграцію інтерактивної карти з можливістю додавання нових локацій користувачами, відправлення SOS-сигналу в екстрених ситуаціях, а також завершення сесії користувача. Реалізовані функції забезпечують користувачу легкий доступ до довідкової інформації, підвищують рівень безпеки мандрівників та сприяють розвитку спільної туристичної бази даних.

Проведене тестування розробленого програмного забезпечення підтвердило його стабільність, відповідність поставленим технічним вимогам та зручність у користуванні. За результатами перевірки встановлено, що мобільний застосунок ефективно виконує свої функції, забезпечуючи мандрівникам можливість оперативного доступу до актуальної інформації про об'єкти Карпатського регіону, навіть у складних умовах пересування гірською місцевістю.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kobylianska O. On Sunday Morning She Gathered Herbs. – Lviv: Litopys, 2008. – 240 p.

2. Olbracht I. The Sorrowful Eyes of Hannah Karajich. – Prague: Melantrich, 1937. – 312 p.

3. Matios M. Sweet Darusia. – Chernivtsi: Bukrek, 2004. – 288 p.

4. Thorpe N. Walking Europe's Last Wilderness: A Journey Through the Carpathian Mountains. – New Haven: Yale University Press, 2024. – 400 p.

5. Sparks A.E. Into the Carpathians: A Journey Through the Heart and History of Central and Eastern Europe. – Raleigh: Lulu Press, 2016. – 350 p.

6. Ivanchuk R. Fire Poles. – Kharkiv: Folio, 2019. – 832 p.

7. Samchuk U. The Mountains Speak. – Toronto: Homin Ukrainy, 1951. – 256 p.

8. Feehan C. Dark Curse. – New York: Berkley Hardcover, 2008. – 416 p.

9. Dabit N. React Native in Action. – Shelter Island: Manning Publications, 2019. – 300 p.

10. Grzesiukiewicz M. Hands-On Design Patterns with React Native. – Birmingham: Packt Publishing, 2018. – 350 p.

11. Ward D. React Native Cookbook, 2nd Edition. – Sebastopol: O'Reilly Media, 2019. – 592 p.

12. Paul A. React Native for Mobile Development. – Birmingham: Packt Publishing, 2017. – 300 p.

13. Ross W.P. Learning React Native: Building Native Mobile Apps with JavaScript. – Sebastopol: O'Reilly Media, 2015. – 300 p.

14. Paul A., Kumar N. React Native for Mobile Development. – Birmingham: Packt Publishing, 2017. – 300 p.

15. Linden G. Expo Guidelines Book. – San Francisco: Blurb, 2017. – 200 p.

16. Bacon E. Who's using Expo in 2025. – [Online Article] Available at: https://evanbacon.dev/blog/expo-2024

17. Expo Documentation. – [Online Resource] Available at: https://docs.expo.dev/ Expo Documentation

18. Idaszak D. 9 Must-Read React Native Books You Need on Your Bookshelf. – [Online Article] Available at: https://www.netguru.com/blog/react-native-books

ДОДАТКИ

ДОДАТОК А

Програмний код модуля реєстрації користувача:

```
import React, { useState } from 'react';
import { View, Text, TextInput, Button, StyleSheet, Alert } from
'react-native';
import { useNavigation } from '@react-navigation/native';
import AsyncStorage from '@react-native-async-storage/async-storage';
import { addUser } from '../../database';

const RegisterScreen = () => {
    const navigation = useNavigation();
    const [name, setName] = useState('');
    const [phone, setPhone] = useState('');
    const [age, setAge] = useState('');
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');

    const handleRegister = async () => {
        if (!name || !phone || !age || !email || !password) {
            Alert.alert('Помилка', 'Будь ласка, заповніть усі поля.');
            return;
        }

        try {
            await addUser({
                name,
                phone,
                age: parseInt(age),
                email,
                password,
            });

            await AsyncStorage.setItem('currentUser', email);

            Alert.alert('Успіх', 'Реєстрація пройшла успішно!', [
                { text: 'OK', onPress: () => navigation.navigate('Main') }
            ]);
        } catch (error) {
            console.error('Помилка реєстрації:', error);
            Alert.alert('Помилка', 'Помилка при реєстрації. Спробуйте ще
раз.');
        }
    };

    return (
        <View style={styles.container}>
            <Text style={styles.title}>Реєстрація</Text>

            <TextInput
                style={styles.input}
                placeholder="Ім'я"
                value={name}
                onChangeText={setName}
```

```
            />
            <TextInput
                style={styles.input}
                placeholder="Телефон"
                value={phone}
                onChangeText={setPhone}
                keyboardType="phone-pad"
            />
            <TextInput
                style={styles.input}
                placeholder="Вік"
                value={age}
                onChangeText={setAge}
                keyboardType="numeric"
            />
            <TextInput
                style={styles.input}
                placeholder="Email"
                value={email}
                onChangeText={setEmail}
                keyboardType="email-address"
            />
            <TextInput
                style={styles.input}
                placeholder="Пароль"
                value={password}
                onChangeText={setPassword}
                secureTextEntry
            />

            <Button title="Зареєструватися" onPress={handleRegister} />
        </View>
    );
};
```

ДОДАТОК Б

Програмний код модуля авторизації користувача:

```
import React, { useState, useContext } from 'react';
import { View, Text, TextInput, Button, StyleSheet, Alert } from
'react-native';
import { getUser } from '../../database';
import { useNavigation } from '@react-navigation/native';
import { AuthContext } from '../../AuthContext'; // !!! додали імпорт контексту
const LoginScreen = () => {
    const navigation = useNavigation();
    const { signIn } = useContext(AuthContext); // !!! отримуємо функцію signIn
з контексту
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');

    const handleLogin = () => {
        if (!email || !password) {
            Alert.alert('Помилка', 'Будь ласка, заповніть усі поля.');
            return;
        }
        getUser(
            email,
            password,
            (users) => {
                if (users.length > 0) {
                    signIn('dummy-token'); // !!! після успішної авторизації
викликаємо signIn
                } else {
                    Alert.alert('Помилка', 'Невірний email або пароль.');
                }
            },
            (error) => {
                console.log(error);
                Alert.alert('Помилка', 'Сталася помилка при спробі входу.');
            }
        );
    };
    return (
        <View style={styles.container}>
            <Text style={styles.title}>Вхід</Text>
            <TextInput
                style={styles.input}
                placeholder="Email"
                value={email}
                onChangeText={setEmail}
                keyboardType="email-address"
            />
            <TextInput
                style={styles.input}
                placeholder="Пароль"
                value={password}
                onChangeText={setPassword}
                secureTextEntry
            />
            <Button title="Увійти" onPress={handleLogin} />
            <Button
                title="Ще немає акаунту? Зареєструватися"
```

```
                onPress={() => navigation.navigate('Register')}
                color="gray"
            />
        </View>
    );
};
const styles = StyleSheet.create({
    container: {
        flex: 1,
        justifyContent: 'center',
        paddingHorizontal: 20,
        backgroundColor: '#fff',
    },
    title: {
        fontSize: 26,
        fontWeight: 'bold',
        marginBottom: 20,
        alignSelf: 'center',
    },
    input: {
        height: 50,
        borderColor: '#999',
        borderWidth: 1,
        borderRadius: 8,
        marginBottom: 15,
        paddingHorizontal: 15,
    },
});
export default LoginScreen;
```

ДОДАТОК В

Програмний код модуля головної сторінки застосунку:

```
import React, { useState } from 'react';
import { View, Text, TouchableOpacity, FlatList, StyleSheet, Image } from
'react-native';
import { useNavigation } from '@react-navigation/native';
import { Menu, Button, Provider } from 'react-native-paper';
import Ionicons from 'react-native-vector-icons/Ionicons';
import locations from '../../locations.json';
import styles from './styles';

const types = [
    'Всі місця',
    'Туристичні місця',
    'Міста',
    'Гори',
    'Озера',
    'Водоспади',
    'Відпочинок',
    'Готелі'
];

const getIconName = (type) => {
    switch (type.toLowerCase()) {
        case 'туристичні місця':
            return 'flag-outline';
        case 'міста':
            return 'business-outline';
        case 'гори':
            return 'leaf-outline';
        case 'озера':
            return 'water-outline';
        case 'водоспади':
            return 'water-outline';
        case 'відпочинок':
            return 'bed-outline';
        case 'готелі':
            return 'home-outline';
        default:
            return 'location-outline';
    }
};

const MainPage = () => {
    const navigation = useNavigation();
    const [selectedType, setSelectedType] = useState('Всі місця');
    const [menuVisible, setMenuVisible] = useState(false);

    const openMenu = () => setMenuVisible(true);
    const closeMenu = () => setMenuVisible(false);

    const filteredData = selectedType === 'Всі місця'
        ? locations
        : locations.filter((item) => item.type.some(type => type.toLowerCase()
=== selectedType.toLowerCase()));

    const handleTypeSelect = (type) => {
```

```
        setSelectedType(type);
        closeMenu();
    };
    const renderItem = ({ item }) => (
        <TouchableOpacity
            style={styles.card}
            onPress={() =>
                navigation.navigate('Details', {
                    name: item.name,
                    type: item.type,
                    description: item.description,
                    images: item.images,
                    height: item.height,
                    coordinates: item.coordinates,
                    location: item.location
                })
            }
        >
            <Image source={{ uri: item.images[0] }} style={styles.image} />
            <Text style={styles.name}>{item.name}</Text>
            <View style={styles.typeContainer}>
                {item.type.map((type) => (
                    <View key={type} style={styles.typeBox}>
                        <Text style={styles.typeText}>{type}</Text>
                    </View>
                ))}
            </View>
            <View style={styles.infoContainer}>
                {item.height && (
                    <Text style={styles.height}>Висота: {item.height} м</Text>
                )}
                {item.coordinates && (
                    <Text style={styles.coordinates}>
                        Координати: {item.coordinates.latitude.toFixed(3)},
{item.coordinates.longitude.toFixed(3)}
                    </Text>
                )}
            </View>
        </TouchableOpacity>
    );

    return (
        <Provider>
            <View style={pageStyles.container}>
                <TouchableOpacity
                    style={pageStyles.button}
                    onPress={() => navigation.navigate('Rules')}
                >
                    <Text style={pageStyles.buttonText}>Рекомендації з
безпеки</Text>
                </TouchableOpacity>

                <TouchableOpacity
                    style={pageStyles.button}
                    onPress={() => navigation.navigate('Map')}
                >
                    <Text style={pageStyles.buttonText}>Переглянути
карту</Text>
                </TouchableOpacity>

                <Menu
                    visible={menuVisible}
                    onDismiss={closeMenu}
                    anchor={
```

```
                        <Button
                            mode="outlined"
                            onPress={openMenu}
                            style={pageStyles.dropdownButton}
                            labelStyle={{ color: '#2e7d32', fontSize: 16 }}
                            contentStyle={{ flexDirection: 'row', alignItems:
'center' }}
                        >
                            <Ionicons
                                name={getIconName(selectedType)}
                                size={20}
                                color="#2e7d32"
                                style={{ marginRight: 8 }}
                            />
                            {selectedType}
                        </Button>
                    }
                >
                    {types.map((type) => (
                        <Menu.Item
                            key={type}
                            onPress={() => handleTypeSelect(type)}
                            title={
                                <View style={{ flexDirection: 'row',
alignItems: 'center' }}>
                                    <Ionicons
                                        name={getIconName(type)}
                                        size={18}
                                        color="#2e7d32"
                                        style={{ marginRight: 8 }}
                                    />
                                    <Text>{type}</Text>
                                </View>
                            }
                        />
                    ))}
                </Menu>

                <FlatList
                    data={filteredData}
                    keyExtractor={(item) => item.id.toString()}
                    renderItem={renderItem}
                    contentContainerStyle={styles.list}
                />
            </View>
        </Provider>
    );
};
```

ДОДАТОК Г

Програмний код модуля відображення карти:

```
import React, { useEffect, useState } from 'react';
import { View, Text, StyleSheet, TouchableOpacity, Modal, TextInput,
ScrollView, Alert, Image } from 'react-native';
import MapView, { Marker } from 'react-native-maps';
import * as Location from 'expo-location';
import * as ImagePicker from 'expo-image-picker';
import AsyncStorage from '@react-native-async-storage/async-storage';
import data from '../../locations.json';

const MapPage = () => {
    const [locations, setLocations] = useState([]);
    const [modalVisible, setModalVisible] = useState(false);
    const [newPlace, setNewPlace] = useState({
        name: '',
        type: '',
        description: '',
        location: '',
        latitude: '',
        longitude: '',
        images: [],
    });

    useEffect(() => {
        const loadLocations = async () => {
            try {
                const storedLocations = await
AsyncStorage.getItem('new_locations');
                const parsedLocations = storedLocations ?
JSON.parse(storedLocations) : [];
                setLocations([...data, ...parsedLocations]);
            } catch (error) {
                console.log('Помилка при завантаженні локацій:', error);
            }
        };

        loadLocations();
    }, []);

    const handleChooseImage = async () => {
        let result = await ImagePicker.launchImageLibraryAsync({
            mediaTypes: ImagePicker.MediaTypeOptions.Images,
            quality: 1,
        });

        if (!result.canceled) {
            setNewPlace((prev) => ({
                ...prev,
                images: [...prev.images, result.assets[0].uri],
            }));
        }
    };

    const handleAddPlace = async () => {
        if (
            !newPlace.name ||
            !newPlace.type ||
            !newPlace.description ||
            !newPlace.location ||
```

```
            !newPlace.latitude ||
            !newPlace.longitude ||
            newPlace.images.length === 0
        ) {
            Alert.alert('Помилка', 'Будь ласка, заповніть усі поля.');
            return;
        }

        const newLocation = {
            id: locations.length + 1,
            name: newPlace.name,
            type: newPlace.type.split(',').map((t) => t.trim()),
            description: newPlace.description,
            location: newPlace.location,
            height: null,
            images: newPlace.images,
            coordinates: {
                latitude: parseFloat(newPlace.latitude),
                longitude: parseFloat(newPlace.longitude),
            },
        };

        const updatedLocations = [...locations, newLocation];
        setLocations(updatedLocations);

        try {
            await AsyncStorage.setItem('new_locations',
JSON.stringify(updatedLocations.filter(loc => loc.id > data.length)));
        } catch (error) {
            console.log('Помилка при збереженні нової локації:', error);
        }

        setModalVisible(false);
        setNewPlace({
            name: '',
            type: '',
            description: '',
            location: '',
            latitude: '',
            longitude: '',
            images: [],
        });

        Alert.alert('Успіх', 'Нове місце додано!');
    };

    return (
        <View style={styles.container}>
            <MapView
                style={styles.map}
                initialRegion={{
                    latitude: 48.4622,
                    longitude: 24.4676,
                    latitudeDelta: 2,
                    longitudeDelta: 2,
                }}
            >
                {locations.map((location) => (
                    <Marker
                        key={location.id}
                        coordinate={{
                            latitude: location.coordinates.latitude,
                            longitude: location.coordinates.longitude,
                        }}
```

```
                        title={location.name}
                        description={location.description}
                    >
                        {location.images.length > 0 && (
                            <Image
                                source={{ uri: location.images[0] }}
                                style={{ width: 40, height: 40, borderRadius:
20 }}
                            />
                        )}
                    </Marker>
                ))}
            </MapView>

            <TouchableOpacity style={styles.addButton} onPress={() =>
setModalVisible(true)}>
                <Text style={styles.addButtonText}>Додати нове місце</Text>
            </TouchableOpacity>

            <Modal visible={modalVisible} animationType="slide">
                <ScrollView contentContainerStyle={styles.modalContent}>
                    <Text style={styles.modalTitle}>Нове місце</Text>

                    <TextInput
                        placeholder="Назва"
                        style={styles.input}
                        value={newPlace.name}
                        onChangeText={(text) => setNewPlace({ ...newPlace,
name: text })}
                    />
                    <TextInput
                        placeholder="Тип (через кому)"
                        style={styles.input}
                        value={newPlace.type}
                        onChangeText={(text) => setNewPlace({ ...newPlace,
type: text })}
                    />
                    <TextInput
                        placeholder="Опис"
                        style={styles.input}
                        multiline
                        value={newPlace.description}
                        onChangeText={(text) => setNewPlace({ ...newPlace,
description: text })}
                    />
                    <TextInput
                        placeholder="Локація"
                        style={styles.input}
                        value={newPlace.location}
                        onChangeText={(text) => setNewPlace({ ...newPlace,
location: text })}
                    />
                    <TextInput
                        placeholder="Широта (latitude)"
                        style={styles.input}
                        value={newPlace.latitude}
                        onChangeText={(text) => setNewPlace({ ...newPlace,
latitude: text })}
                        keyboardType="numeric"
                    />
                    <TextInput
                        placeholder="Довгота (longitude)"
                        style={styles.input}
                        value={newPlace.longitude}
```

```
                        onChangeText={(text) => setNewPlace({ ...newPlace,
longitude: text })}
                        keyboardType="numeric"
                    />

                    <TouchableOpacity style={styles.imageButton}
onPress={handleChooseImage}>
                        <Text style={styles.imageButtonText}>Додати фото</Text>
                    </TouchableOpacity>

                    <View style={styles.imagePreview}>
                        {newPlace.images.map((uri, index) => (
                            <Image key={index} source={{ uri }}
style={styles.previewImage} />
                        ))}
                    </View>

                    <TouchableOpacity style={styles.saveButton}
onPress={handleAddPlace}>
                        <Text style={styles.saveButtonText}>Зберегти
місце</Text>
                    </TouchableOpacity>

                    <TouchableOpacity style={styles.cancelButton} onPress={()
=> setModalVisible(false)}>
                        <Text style={styles.cancelButtonText}>Скасувати</Text>
                    </TouchableOpacity>
                </ScrollView>
            </Modal>
        </View>
    );
};
```

ДОДАТОК Д

Програмний код модуля додавання об’єктів:

```
const handleChooseImage = async () => {
    let result = await ImagePicker.launchImageLibraryAsync({
        mediaTypes: ImagePicker.MediaTypeOptions.Images,
        quality: 1,
    });

    if (!result.canceled) {
        setNewPlace((prev) => ({
            ...prev,
            images: [...prev.images, result.assets[0].uri],
        }));
    }
};

const handleAddPlace = async () => {
    if (
        !newPlace.name ||
        !newPlace.type ||
        !newPlace.description ||
        !newPlace.location ||
        !newPlace.latitude ||
        !newPlace.longitude ||
        newPlace.images.length === 0
    ) {
        Alert.alert('Помилка', 'Будь ласка, заповніть усі поля.');
        return;
    }

    const newLocation = {
        id: locations.length + 1,
        name: newPlace.name,
        type: newPlace.type.split(',').map((t) => t.trim()),
        description: newPlace.description,
        location: newPlace.location,
        height: null,
        images: newPlace.images,
        coordinates: {
            latitude: parseFloat(newPlace.latitude),
            longitude: parseFloat(newPlace.longitude),
        },
    };

    const updatedLocations = [...locations, newLocation];
    setLocations(updatedLocations);

    try {
        await AsyncStorage.setItem('new_locations',
JSON.stringify(updatedLocations.filter(loc => loc.id > data.length)));
    } catch (error) {
        console.log('Помилка при збереженні нової локації:', error);
    }
```

ДОДАТОК Е

Програмний код модуля відображення детальної інформації:

DetailPage.js:

```
import React, { useState } from 'react';
import { View, Text, Image, ScrollView, TouchableOpacity } from 'react-native';
import Swiper from 'react-native-swiper';
import styles from './styles';

const DetailPage = ({ route }) => {
  const { name, type, description, images, height, coordinates, location } =
route.params;
  const [isDescriptionExpanded, setIsDescriptionExpanded] = useState(false);

  const truncateDescription = (text) => {
    if (!isDescriptionExpanded && text.length > 450) {
      return `${text.substring(0, 450)}...`;
    }
    return text;
  };

  const renderCoordinates = (coordinates) => {
    if (coordinates) {
      return (
        <View style={styles.coordinatesContainer}>
          <Text style={styles.coordinatesText}>Координати: {`Lat:
${coordinates.latitude}, Long: ${coordinates.longitude}`}</Text>
        </View>
      );
    }
    return null;
  };

  const renderHeight = (height) => {
    if (height) {
      return (
        <View style={styles.heightContainer}>
          <Text style={styles.heightText}>Висота: {height} м</Text>
        </View>
      );
    }
    return null;
  };

  return (
    <ScrollView style={styles.container}>
      <Text style={styles.title}>{name}</Text>

      {/* Висота */}
      {renderHeight(height)}

      {/* Розташування */}
      {location && (
        <View style={styles.locationContainer}>
          <Text style={styles.locationText}>{location}</Text>
        </View>
      )}

      {/* Опис */}
      <Text style={styles.description}>
        {truncateDescription(description)}
        {description.length > 450 && (
          <TouchableOpacity onPress={() =>
```

```
setIsDescriptionExpanded(!isDescriptionExpanded)}>
            <Text style={styles.toggleText}>
              {isDescriptionExpanded ? 'Згорнути' : 'Розгорнути...'}
            </Text>
          </TouchableOpacity>
        )}
      </Text>

      {/* Координати */}
      {renderCoordinates(coordinates)}

      <View style={styles.carouselContainer}>
        <Swiper
          style={styles.wrapper}
          showsButtons={true}
          loop={true}
          dotStyle={styles.dot}
          activeDotStyle={styles.activeDot}
          paginationStyle={styles.pagination}
        >
          {images.map((image, index) => (
            <View key={index} style={styles.slide}>
              <Image source={{ uri: image }} style={styles.image} />
            </View>
          ))}
        </Swiper>
      </View>

      {/* Теги */}
      <View style={styles.typeContainer}>
        {type.map((tag, index) => (
          <Text key={index} style={styles.type}>
            {tag}
          </Text>
        ))}
      </View>
    </ScrollView>
  );
};

export default DetailPage;
```

ДОДАТОК Ж

Програмний код модуля логіки SOS - функції:

```
import React from 'react';
import { View, Text, StyleSheet, TouchableOpacity, Alert } from 'react-native';
import * as Location from 'expo-location';
import { Ionicons } from '@expo/vector-icons';

const SosPage = () => {

    const sendSosSignal = async () => {
        Alert.alert(
            'Підтвердження',
            'Ви впевнені, що хочете надіслати SOS сигнал?',
            [
                {
                    text: 'Скасувати',
                    style: 'cancel',
                },
                {
                    text: 'Так, надіслати',
                    onPress: async () => {
                        try {
                            let { status } = await
Location.requestForegroundPermissionsAsync();
                            if (status !== 'granted') {
                                Alert.alert('Помилка', 'Доступ до геолокації
заборонено.');
                                return;
                            }
                            let location = await
Location.getCurrentPositionAsync({});
                            console.log('SOS Signal Sent with location:',
location.coords);
                            Alert.alert('Успіх', 'SOS сигнал надіслано!');
                        } catch (error) {
                            console.log(error);
                            Alert.alert('Помилка', 'Не вдалося надіслати SOS
сигнал.');
                        }
                    }
                }
            ]
        );
    };

    return (
        <View style={styles.container}>
            <View style={styles.content}>
                <Text style={styles.title}>
                    <Ionicons name="alert-circle-outline" size={28}
color="#e53935" /> Що робити, якщо ви загубилися?
                </Text>
                <Text style={styles.instruction}>
                    <Ionicons name="compass-outline" size={20} color="#2e7d32"
/> Залишайтесь на місці і не панікуйте. {"\n\n"}
                    <Ionicons name="call-outline" size={20} color="#2e7d32" />
Використовуйте мобільний телефон для виклику допомоги. {"\n\n"}
                    <Ionicons name="navigate-outline" size={20} color="#2e7d32"
/> Позначте своє місцезнаходження для пошукової команди.
                </Text>
            </View>

            <TouchableOpacity style={styles.sosButton} onPress={sendSosSignal}>
```

```
                <Text style={styles.sosButtonText}>SOS</Text>
            </TouchableOpacity>
        </View>
    );
};
```

ДОДАТОК З

Програмний код модуля завершення сесії користувача:

```
import React, { useEffect, useContext } from 'react';
import { View, Text, StyleSheet, ActivityIndicator } from 'react-native';
import { AuthContext } from '../../AuthContext'; // Імпортуємо

const LogoutPage = () => {
    const { signOut } = useContext(AuthContext);

    useEffect(() => {
        signOut(); // просто викликаємо
    }, []);

    return (
        <View style={styles.container}>
            <ActivityIndicator size="large" color="#e53935" />
            <Text style={styles.text}>Вихід...</Text>
        </View>
    );
};

const styles = StyleSheet.create({
    container: {
        flex: 1,
        justifyContent: 'center',
        alignItems: 'center',
        backgroundColor: '#fff',
    },
    text: {
        marginTop: 15,
        fontSize: 18,
        color: '#555',
    },
});

export default LogoutPage;
```