Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

Мобільний туристичний AI-асистент

Виконав: студент 4 курсу, групи 1КН-21б
спеціальності 122 – Компютерні науки
_____ Рудий Р.С.
Керівник: к.т.н., доцент кафедри КН
_____ Колодний В.В.
«04» червня 2025 р.
Рецензент: к.т.н., доцент каф. АІІТ
_____ Барабан М.В.
«05» червня 2025 р.

Допущено до захисту
Завідувач кафедри КН
д.т.н., проф. Яровий А. А.
«06» червня 2025 р.

Вінниця ВНТУ – 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 – Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
завідувач кафедри КН д. т. н.,
професор Яровий А. А.
"05" грудня 2025 року

ЗАВДАННЯ
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Рудому Ростиславу Сергійовичу

1. Тема роботи: «Мобільний туристичний AI-асистент».
Керівник роботи: Колодний Володимир Володимирович, к.т.н., доцент, доцент кафедри комп'ютерних наук, затверджено наказом ВНТУ від «20» березня 2025 року № 97.

2. Строк подання студентом роботи: 30 травня 2025 року.

3. Вхідні дані до роботи:
мова програмування – об'єктно-орієнтована, має забезпечувати можливість та управління базами даних;
обмеження запитів – не більше 1 за один сеанс;
кількість міст – не більше 10;
інтерфейс користувача має бути інтуїтивно зрозумілим.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):
вступ, аналіз предметної області мобільного туристичного AI-асистента, розробка алгоритму роботи мобільного туристичного AI-асистента, розробка та тестування мобільного туристичного AI-асистента. Висновки, список використаних джерел, додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):
flowchart архітектури мобільного додатку туристичний AI-асистент, алгоритм роботи мобільного додатку туристичний AI-асистент, сторінки додатку.

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Колодний В.В., к.т.н., доцент		

6. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1.	Обгрунтування доцільності розробки програмного модуля прогнозування обсягу закупівлі квітів	05.05.25	07.05.25	
2.	Розробка алгоритму роботи мобільного туристичного AI-асистента	08.05.25	15.05.25	
3.	Розробка та тестування мобільного туристичного AI-асистента	16.05.25	26.05.25	
4.	Розробка інструкції користувача.	27.05.25	29.05.25	
5.	Висновки та аналіз отриманих результатів	30.05.25	01.06.25	
6.	Оформлення матеріалів до захисту БКР	02.06.25	04.06.25	

Студент _____ Рудий Р.С.
(підпис) (прізвище та ініціали)

Керівник проєкту (роботи) _____ Колодний В.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 104 сторінок формату А4, на яких є 13 рисунків, список використаних джерел містить 19 найменувань.

Розглянуто найпопулярніші програмні рішення у сфері подорожей, проаналізовано їх основні функціональні можливості, переваги та недоліки. Розроблено структурну схему та загальний алгоритм роботи мобільного додатку туристичного AI-асистента. Обрано мову програмування, середовище розробки, бібліотеки та фреймворк, завдяки яким було створено додаток. Виконано тестування яке довело правильність функціонування створеного додатку.

Ключові слова: подорож, мобільний туристичний AI-асистент, генерація маршрутів за допомогою ШІ, інтерактивна карта, мобільний додаток, C++, Qt, QtQuick, QML, QtCreator, Firebase.

ABSTRACT

The bachelor's thesis consists 104 of pages in A4 format, which include 13 figures; the list of references contains 19 titles.

The most popular software solutions in the travel sector were reviewed, their main functional capabilities, advantages, and disadvantages were analyzed. A structural scheme and general algorithm of the mobile tourist AI assistant application were developed. The programming language, development environment, libraries, and framework were chosen, thanks to which the application was created. Testing was performed, which proved the correctness of the created application's functioning.

Keywords: travel, mobile tourist AI assistant, route generation using AI, interactive map, mobile application, C++, Qt, QtQuick, QML, QtCreator, Firebase.

ВСТУП

Актуальність досліджень. У світі, де штучний інтелект щодня змінює правила гри, туризм також переживає цифрову трансформацію. Мандрівники шукають не просто інформацію, а миттєвий, персоналізований досвід – і саме тут на допомогу приходять мобільні AI-асистенти. Як можна помітити ШІ інтегрують скрізь – від пошукових систем, помічників-асистентів з документації, відео-ігор до невеликих веб-додатків. Тому, на даний момент, туризм теж не став винятком та теж стрімко впроваджує цифровізацію. Цей стрімкий процес цифровізації в туристичній сфері обумовлений низкою ключових чинників, серед яких:

1. Після пандемії COVID-19 туризм почав трансформуватися: туристи все більше почали використовувати цифрові технології.
2. Веб і мобільні додатки стали невід'ємною частиною планування подорожей. За статистикою більша частка туристів використовують смартфон для бронювання квитків, готелів, екскурсій тощо.
3. ШІ дозволяє зробити туризм персоналізованим, унікальним, швидким і зручним, усуваючи потребу в посередниках.

Серед ключових проблем які може вирішити AI-асистент – відсутність локального гіда, труднощі з мовою, переобтяження інформацією в Google, потреба в швидких рішеннях у незнайомій місцевості, зниження залежності від інтернету тощо.

Мобільний формат є надзвичайно зручним і практичним для реалізації туристичних AI-рішень. Насамперед, мобільні додатки завжди під рукою — користувачеві не потрібно мати при собі ноутбук або друковані карти, адже вся необхідна інформація доступна прямо зі смартфона. Крім того, мобільні пристрої забезпечують високий рівень персоналізації, оскільки мають доступ до GPS,

камери, календаря та історії взаємодії користувача, що дозволяє створювати максимально адаптований досвід подорожі.

Отже, інтеграція штучного інтелекту в мобільні туристичні додатки набуває особливого значення, оскільки саме ці технології здатні забезпечити користувачам новий рівень зручності, оперативності та індивідуального підходу. ШІ дозволяє адаптувати функціонал додатка до особистих вподобань мандрівника, пропонуючи найактуальнішу інформацію в режимі реального часу, оптимізуючи маршрути та скорочуючи час на прийняття рішень. Саме завдяки таким можливостям туристичний досвід стає більш комфортним, ефективним і приємним, що особливо важливо в умовах швидкоплинного ритму сучасного життя.

Метою дослідження є розширення функціональних можливостей додатку мобільного туристичного AI-асистента.

Об'єктом дослідження є процес інформаційного супроводу туристів за допомогою цифрових технологій.

Предметом дослідження є програмні засоби інформаційного супроводу туристів за допомогою цифрових технологій.

Задачі дослідження:

1. Здійснити аналіз сучасного ринку мобільних туристичних додатків із підтримкою штучного інтелекту. Виявити їхні функціональні можливості, переваги, недоліки, а також ступінь персоналізації та адаптивності до користувача.
2. Провести варіантний аналіз підходів до реалізацій мобільного AI-асистента.
3. Розробити архітектуру мобільного додатку.
4. Розробити алгоритми роботи штучного інтелекту в межах додатку.
5. Обґрунтувати вибір технологій, мов програмування та середовища розробки.

6. Реалізувати програмний додаток мобільного туристичного AI-асистента та провести його тестування.

Результати дослідження впроваджено в роботу ТОВ «ІТІ».

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ МОБІЛЬНОГО ТУРИСТИЧНОГО AI-АСИСТЕНТА

1.1 Аналіз предметної області

Аналіз предметної області є одним з найважливіших етапів у процесі проектування та розробки програмного забезпечення. Цей етап допомагає більш широко зрозуміти специфіку задач, які має вирішувати система, виявити основні функціональні вимоги, визначити потенційних користувачів, а також сформувати чітке уявлення про середовище, в якому буде працювати розроблений програмний продукт. У цьому підрозділі буде розглянуто основні аспекти функціонування туристичної галузі, що мають безпосереднє відношення до створення мобільного AI-асистента.

На даний момент, туристи все більше намагаються уникати появ у туристичних агенціях та безпосередньо користуватись різними цифровими ресурсами для бронювання квитків, готелів, турів та путівок тощо. Та більшість із них стикаються з труднощами під час такої практики, що призводить до перебігу різних обставин, які можуть сказатись на комфорті, безпеці та загальному враженні від подорожі. Відсутність централізованого інструменту, здатного враховувати індивідуальні потребу, персоналізувати запити, відсіювати ризиковані варіанти часто змушує туристів витрачати значну кількість часу на пошук даних або приймати спонтанні рішення, що не завжди є оптимальним.

Завдяки впровадження автоматизації, створення більш централізованого рішення можна контролювати весь процес подорожі, екскурсії чи просто невеличкої поїздки громадським транспортом. Буде можливість враховувати всі аспекти подорожі: поліпшити пошукові запити персоналізовано під конкретного користувача, максимізувати комфортабельність та впевненість в виборі,

мінімізувати ризики, які можуть виникнути під час подорожі або ж заявити про них.

У наш час, з розвитком технологій, з'являється все більше мобільних та веб-додатків, які покращують досвід туристів у сфері подорожей. До таких рішень належать: мобільні карти, функціоналом яких є можливість переглядати актуальну інформацію про різні визначні місця, заклади, пам'ятники, міста тощо; додатки для купівлі квитків та бронювання проживання такі як Booking.com, Kiwi, TripAdvisor, Airbnb та інші; додатки для пошуку цікавих локацій для фотозйомки, зокрема PinToBeIn, розробником якого є автор цією дипломної роботи. Усі ці сервіси мають спільну мету – забезпечити зручність, інформативність та персоналізованість подорожей. Проте, більшість з них вирішують лише окремі задачі, не забезпечуючи комплексного та централізованого підходу до планування подорожі.

Саме тому додаток, який буде розроблено під час праці над дипломною роботою, має велику актуальність у наш час. У ньому ми об'єднаємо деякий функціонал вищеперерахованих додатків, що напряму зв'язані з сектором туризму, із використанням можливостей штучного інтелекту. Розроблений мобільний туристичний AI-асистент зможе адаптуватися до потреб користувача, надавати персоналізовані рекомендації, прокладати маршрути, рекомендувати комплексні подорожі, попереджати про потенційні ризики під час подорожі та значно підвищувати комфорт і ефективність планування. Тому інтеграція штучного інтелекту не тільки зробить наш додаток конкурентоспроможним, а й забезпечить його відповідність сучасним технологічним трендам, що дозволить залишатися актуальним у довгостроковій перспективі.

1.2 Огляд та аналіз існуючих рішень

На сучасному ринку представлено велику різноманітність програмних засобів та рішень, що спрямовані на підвищення ефективності інформування клієнтів у сфері туризму. Ці рішення орієнтовані на полегшення пошуку, планування та організація подорожей, а також на покращення взаємодії користувачів із туристичними сервісами. Так як зараз відбувається швидкий розвиток технологій, особливе місце посідають мобільні додатки з використанням штучного інтелекту, які можуть швидко та якісно забезпечити індивідуальний підхід до користувачів та можуть зберігати ефективність на різних етапах туристичної подорожі.

Існуючі рішення можна поділити на кілька основних категорій залежно від їх фугкціонального призначення, технологічної основи та способу взаємодії х користувачем:

1.Туристичні мобільні додатки з базовими функціями пошуку та бронювання. Ці додатки орієнтовані на надання користувачам зручного UI/UX для пошуку та бронювання готелів, квитків, турів, а також перегляду відгуків та оцінок. Вони містять великі бази даних, інтегровані з різними постачальниками послуг. Розглянемо самі основні:

1) Booking.com – один із найбільших і найпопулярніших онлайн-сервісів для бронування готелів, апартаментів, хостелів і інших варіантів проживання у всьому світі. Заснований у 1996 році в Нідерландах, сервіс поступово розширився і став частиною групи Booking Holdings Inc., що також володіє такими платформами, як Priceline та Kayak.

Booking.com працює як платформа-агрегатор, що дозволяє мандрівникам легко знаходити житло, порівняти ціни, читати відгуки і бронювати проживання у більш ніж 150 країнах.

Booking.com подано на рисунку 1.1:

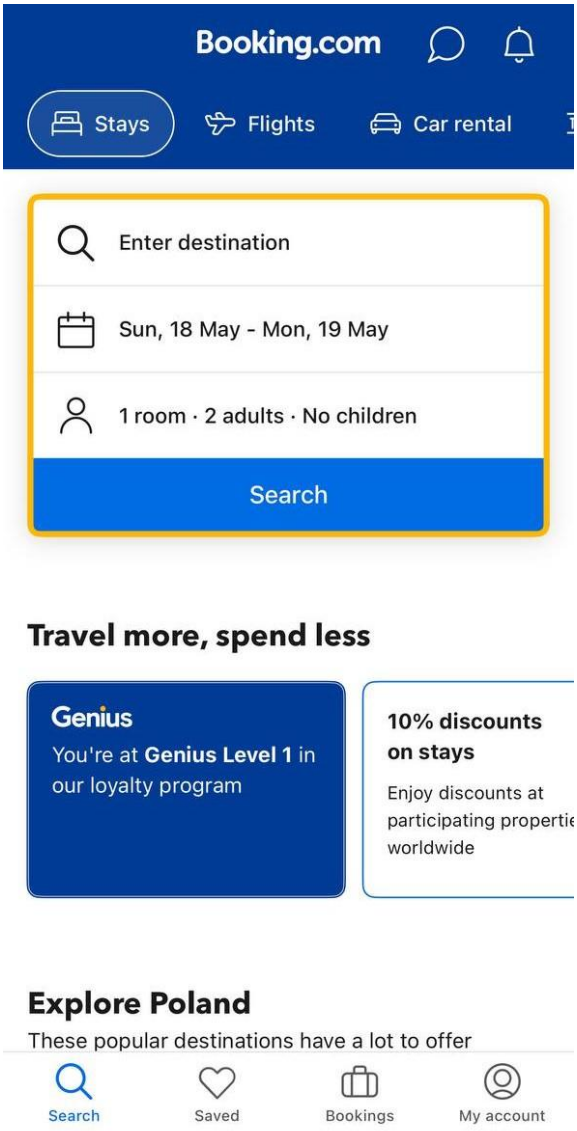

Рисунок 1.1 - Додаток «Booking.com»

1. Користувач може задавати параметри пошуку – дата, локація, кількість кімнат, кількість осіб тощо. Також є можливість налаштувати плаваючу дату, щоб пошук доступних результатів міг видати дати зі здвигом в ту кількість днів, яку введе користувач.

2. Користувач може обрати вид подорожі – лише проживання, проживання з авіаперельотом, оренду транспорту, трансфер, наприклад таксі, або розваги: атракціони, екскурсії, музеї тощо.

3. Результати пошуку відображаються у вигляді списку з фотографіями, основною інформацією, цінами і відгуками. Кожен об’єкт має детальний опис та карту розташування.

4. Ключовий елемент платформи – система оцінок і коментарів від реальних гостей чи користувасів під кожним об’єктом. Вони допомагають приймати об’ґрунтовані рішення.

5. Користувачі можуть шшвидко оформити бронювання, часто без необхідності передоплати, з можливістю скасувати без штрафів.

6. На нижній панелі навігації знаходятся основні елементи платформи: кнопка пошуку, збережених туристичних об’єктів, бронювання користувача та кнопка переходу в профіль користувача де є можливість налаштувати всю необхідну інформацію.

Booking.com надає зручний та потужний інструмент для планування подорожей, що охоплює бронювання житла, транспорту, трансферів і розваг. Завдяки гнучкому інтерфейсу, користувачі можуть швидко знаходити необхідні варіанти, орієнтуючись на широкий спектр фільтрів, включно з місцем, датою, кількістю гостей, бюджетом тощо.

Сервіс пропонує можливість бронювання з гнучкими датами, що дозволяє знайти оптимальний варіант навіть за умов обмеженої доступності. Завдяки системі відгуків від реальних користувачів, платформа допомагає приймати обґрунтовані рішення при виборі об’єктів.

Крім того, Booking.com інтегрує різні сервіси – бронювання перельотів, оренда авто, таксі, туристичні активності, - що робить його універсальним рішенням для подорожей.

Також варто виділити деякі недоліки, про які варто знати:

1. Велика кількість пропозицій може заплутати. Для деяких користувачів вибір надмірно великий, що ускладнює прийняття рішення.

2. Рекомендації базуються переважно на загальних алгоритмах, а не на глибокому аналізі інтересів користувача.

3. Без мережі сервіс стає практично недоступний для використання.

4. Одним із суттєвих недоліків платформи Booking.com є система комісій для постачальників послуг. Платформа утримує комісію, яка може сягати від 10% до 25% від суми кожного бронювання, залежно від країни, типу об’єкта та додаткових опцій просування.

5. Обмежена інтеграція з іншими туристичними сервісами, зокрема з локальними транспортними компаніями, екскурсійними агентствами, службами таксі чи оренди велосипедів. Хоча платформа пропонує базовий функціонал для бронювання житла, авіаквитків, деяких трансферів і розваг, у багатьох випадках користувачам доводиться шукати окремі сервіси вручну, оскільки повноцінна екосистема для організації подорожі «все в одному» часто відсутня або реалізована частково.

2) Tripadvisor – одна з найпопулярніших онлайн-платформ для планування подорожей, що дозволяє користувачам переглядати відгуки, рейтинги, бронювати готелі, ресторани, розваги, авіаквитки, екскурсії та інші туристичні послуги(рис 1.2).

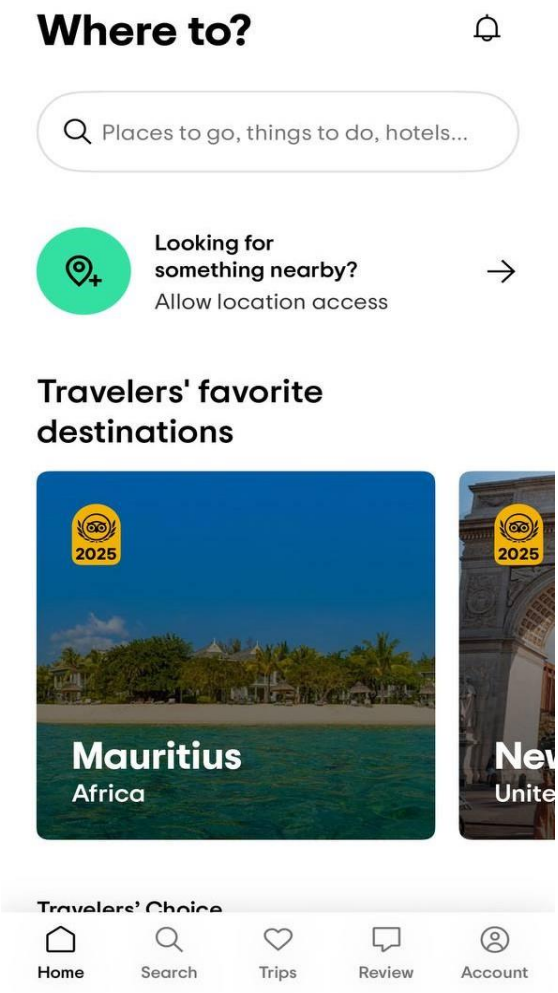

Рисунок 1.2 - Додаток «Tripadvisor»

Розглянемо основні можливості додатку Tripadvisor:

1. Користувачі можуть шукати житло, ресторани, визначні місця, екскурсії та перельоти за містом, датою, типом закладу та іншими параметрами. Доступна карта для перегляду локацій об'єктів.

2. Ключова функція сервісу — величезна база відгуків і фотографій від реальних мандрівників. Платформа дозволяє переглядати не тільки загальний рейтинг, а й детальні коментарі щодо чистоти, обслуговування, розташування та ціни.

3. Користувачі можуть створювати персоналізовані списки збережених місць, об'єднувати їх у маршрути, ділитися з друзями чи сім'єю.

4. Платна підписка, що дає доступ до знижок на готелі та ексклюзивних пропозицій.

5. Tripadvisor інтегрований з різними агрегаторами і дозволяє безпосередньо перейти до бронювання або навіть оформити замовлення прямо у застосунку.

Очевидно, що кожен додаток має свої недоліки, і програма Tripadvisor не є виключенням і ось кілька основних недоліків:

1. Надмірна кількість інформації може ускладнювати процес вибору — користувачам важко зорієнтуватися серед тисяч відгуків.

2. Не завжди актуальні відгуки — деякі об’єкти мають багато старих коментарів, які вже не відповідають дійсності.

3. Комерційний вплив — заклади, які платять за рекламу, можуть з’являтися вище в результатах пошуку, навіть якщо мають гірший рейтинг.

4. Не всі бронювання доступні напряму — для деяких послуг (особливо місцевих екскурсій) потрібно переходити на сторонні сайти.

5. Обмежена офлайн-функціональність — для перегляду карт, фото та коментарів потрібно стабільне підключення до Інтернету.

6. Залежність від сторонніх постачальників — у випадку виникнення проблем із бронюванням, відповідальність часто несе не Tripadvisor, а партнер.

Отже, Tripadvisor є без сумніву потужним інструментом для дослідження подорожей і ознайомлення з досвідом інших мандрівників. Це ідеальний варіант для тих, хто прагне приймати обґрунтовані рішення перед бронюванням житла, ресторану чи атракцій. Однак для повної автоматизації подорожі та бронювання всіх послуг в одному місці йому бракує функціональності, інтеграції й прозорості у відображені результатів пошуку.

2. Однією з важливих категорій туристичних рішень є мобільні додатки для планування маршрутів і організації подорожей. Такі додатки орієнтовані на користувачів, які бажають мати чіткий маршрут, враховувати логістику, час у

дорозі, важливі зупинки та особисті вподобання. Вони зазвичай дозволяють зберігати майбутні подорожі, синхронізувати дані між пристроями та додавати персональні нотатки. Серед цих рішень можна розглянути найпопулярніші мобільні додатки – Sygic Travel, Roadtrippers та TripIt.

1) Sygic Travel (раніше відома як Tripomatic) — це мобільний та веб-додаток, що спеціалізується на створенні персоналізованих маршрутів подорожей. Його розробником є компанія Sygic, яка також відома своїм GPS-навігатором. Основна ідея полягає у тому, щоб надати користувачеві інструмент для планування кожного дня подорожі з урахуванням реального часу на відвідування визначних місць, переміщення між ними, а також перегляду практичної інформації про локації.

Sygic Travel охоплює понад 20 мільйонів туристичних об'єктів по всьому світу, включаючи музеї, замки, історичні будівлі, ресторани, оглядові майданчики, готелі та багато іншого. Додаток має зручний, інтуїтивний інтерфейс з інтерактивною картою, яка служить основною платформою для взаємодії користувача з контентом.

Основні можливості додатку Tripomatic (Sygic Travel):

1. Планування маршрутів

- Можливість створити план подорожі на кожен день, додаючи об’єкти для відвідування
- Додаток автоматизований під різні задачі, наприклад підрахування часу перебування у різних місцях, що є важливим для складення максимально чіткого плану подорожі чи маршруту.
- Також є можливість самостійно коригувати час який буде витрачений в окремих місцях, а також видалення або додавання інших місць в маршруті у будь-який момент.

2. Інтерактивна мапа

- Також досить популярна можливість – всі об’єкти на мапі інтерактивні: натиснувши на будь-яку точку, користувач отримає короткий опис, рейтинг, фото, години роботи, опінію інших людей тощо.
- На мапі можна знайти точки інтересу.

3. Офлайн доступ

- Sygic Travel дозволяє завантажувати маршрути та карти для офлайн-використання, що є важливою особливістю додатків такої категорії.
- Офлайн-карти ідентичні з онлайн-картами.

4. Рекомендації

- На головному екрані мобільного додатку чатсо показуються підбірки різних «топ місць»
- Існує багато видів фільтрів та сортування визначних місць, пам’яток тощо

5. Інтеграція з зовнішніми сервісами

- Можливість інтеграції з календарем Google

Sygic Travel вирізняється серед аналогічних сервісів завдяки добре продуманому функціоналу, що охоплює широкий спектр потреб сучасного мандрівника. Його можливості дозволяють ефективно планувати як короткі вікенд-подорожі, так і більш складні, багатоденні маршрути, зручно поєднуючи їх з інтерактивною картою, великою базою туристичних об'єктів та підтримкою офлайн-режиму. Завдяки синхронізації між пристроями та експорту даних у різні формати, додаток добре інтегрується у цифрову екосистему користувача.

Втім, попри розвинену функціональність, Sygic Travel має низку обмежень, які можуть вплинути на досвід користування:

1. Не підтримує бронювання напряму. Користувач може виконати бронювання тільки перейшовши за зовнішніми посиланнями.

2. Sygic Travel не є актуальним у подорожах по маловідомим локаціям, містам, регіонам тощо.

3. Додаток має платні функції, які навідміну від Sygic Travel є безкоштовними у аналогів, наприклад офлайн-карти.

2) Roadtrippers - це спеціалізований сервіс, орієнтований на подорожі автомобілем, що пропонує користувачам інструменти для детального планування маршрутів з урахуванням численних точок інтересу. Додаток дозволяє легко прокладати маршрути між містами, додаючи зупинки у вигляді природних локацій, історичних пам’яток, ресторанів, готелів, кемпінгів тощо. Його головна мета — перетворити саму дорогу на частину пригоди, допомагаючи знаходити цікаві місця не лише у пункті призначення, а й уздовж шляху.

Розглянемо основні функції, переваги:

1. Користувач може легко додавати зупинки, переглядати відстані, час у дорозі та гнучко редагувати маршрут.

2. Сервіс має інтегрований функціонал інтерактивних рекомендацій в реальному часі. Додаток може підказувати цікаві місця, які варто відвідати неподалік маршруту, з урахуванням уподобань користувача.

3. Спільне планування подорожі - функція колаборації дозволяє планувати маршрут разом із друзями або родиною, спільно редагуючи зупинки та обговорюючи плани.

Roadtrippers вирізняється серед інших сервісів завдяки своїй чіткій спеціалізації на автомобільних подорожах та фокусі на тому, щоб зробити сам шлях не менш цікавим, ніж пункт призначення. Його функціональність дозволяє детально планувати маршрут із зупинками, отримувати персоналізовані рекомендації та ділитися планами з іншими мандрівниками. Проте, незважаючи на зручність та орієнтацію на користувача, сервіс має й певні обмеження, які варто враховувати:

1. Основна частина даних та POI, точок інтересу, зосереджена на США, Канаді, Австралії та Новій Зеландії. В інших регіонах (наприклад, Європа чи Азія) функціональність може бути суттєво знижена.

2. Платна підписка Roadtrippers Plus - деякі ключові функції (наприклад, побудова довших маршрутів із більш ніж 7 зупинками) доступні лише у платній версії.

3. Додаток не може похвастатись моживістю вчасної актуалізації даних, що іноді може негативно відобразитись на досвіді користувачів.

4. Також додаток не має можливості бронювання готелів, квитків напряму – тільки через сторонні сервіси.

3) TripIt — це зручний інструмент для мандрівників, який фокусується на організації всієї інформації про подорож в одному місці. Його головна перевага — автоматичне збирання даних із підтверджень бронювань, отриманих на електронну пошту. Це дозволяє створити детальний і структурований маршрут без потреби вручну вводити кожну деталь. Додаток підтримує бронювання авіаквитків, готелів, оренди авто, трансферів та інших послуг, синхронізуючи всю інформацію в одному календарі подорожі. TripIt також пропонує зручний інтерфейс, сповіщення про зміни у планах та доступ до маршруту в офлайн-режимі.

Однак, попри зручність та автоматизацію, TripIt не позбавлений певних недоліків, які можуть впливати на досвід користування:

1. Немає функцій планування маршруту по місту або пошуку POI — додаток не підходить для користувачів, які хочуть планувати маршрути всередині міста, шукати ресторани, музеї чи пам’ятки.

2. Залежність від e-mail підтверджень — ефективність роботи напряму залежить від якості розпізнавання листів; нестандартні підтвердження або PDF-файли можуть не розпізнаватися.

3. Обмежена інтерактивність — маршрут є радше статичною інформацією, без інтегрованої карти з прокладанням шляху чи рекомендацій.

4. Деякі важливі функції доступні лише у преміум-версії — наприклад, сповіщення про зміну рейсу в реальному часі або відстеження місця в літаку.

3. Тематичні туристичні додатки. Останім часом набирають популярності додатки з неабияко-різною тематикаю з акцентом на туризм. Прикладом таких рішень можуть бути:

- Travello - соціальна мережа для мандрівників.
- Tandem – мовний обмін з носіями мови в інших країнах.
- Komoot – маршрути на природі з GPS-навігацією
- PinToBeIn – великий сервіс точок інтересу для фотосесій з реальними прикладами.

Розглянем для прикладу PinToBeIn:

PinToBeIn – ультимативний додаток для туристів, які мають захоплення в фотосесіях. Це велика платформа, в розробці якої я брав немалу участь, яка містить «піни» - фотографії прикріплені до відповідних місць на мапі. Додаток має гнучкий інтерфейс, можливість додавати чи видаляти свої «піни», щоб мандрівники зі всього світу могли дізнатись найкращі місця для фотосесій.

Однак, попри очевидну привабливість і функціональне розмаїття, додаток має і свої недоліки, які варто враховувати перед використанням:

- Обмежений доступ до «пінів» без підписки — чимало найцікавіших локацій приховані за преміум-доступом, що може зменшити користь для безкоштовних користувачів.
- Висока вартість підписки — повний доступ до функціональності додатку потребує платної підписки, яка може здатися занадто дорогою для пересічного користувача.

- Не всі регіони однаково наповнені контентом — в деяких країнах або малопопулярних містах кількість підів може бути обмеженою.
- Потреба в інтернеті для повноцінного користування — офлайн-режим доступний частково, що не завжди зручно під час подорожей.

Проведений аналіз існуючих туристичних рішень дозволяє зробити кілька ключових висновків, які мають безпосереднє значення для проєктування і реалізації власного додатку.

По-перше, сучасні туристичні сервіси можна умовно розділити на функціональні категорії: від класичних пошуково-бронювальних систем (Booking, TripAdvisor), до інтерактивних гідів, планувальників подорожей, соціальних платформ та нішевих рішень. Кожна з них має свої сильні сторони, але рідко коли об’єднує кілька ключових функцій в одному додатку.

По-друге, більшість популярних рішень фокусуються лише на одному аспекті подорожі — або бронювання, або маршрут, або соціальна взаємодія. Це створює розрив у досвіді користувача, адже йому доводиться перемикатися між кількома додатками.

По-третє, ряд сервісів демонструє успішне впровадження інновацій — таких як GPS-аудіогіди, доповнена реальність чи автоматичне зчитування бронювань із пошти. Ці інструменти значно покращують користувацький досвід і можуть бути адаптовані для реалізації у власному додатку.

Таким чином, найбільш перспективним підходом для реалізації дипломного проєкту є створення мультифункціонального туристичного додатку, який поєднує ключові функції сучасних сервісів: персоналізоване планування маршруту, інтерактивну карту, актуальну погоду, AI-асистента та можливість взаємодії з іншими мандрівниками. Завдяки інтеграції з Firebase забезпечується зберігання користувацьких даних, авторизація та розширення функціоналу в майбутньому.

1.3 Постановка задач дослідження

Основною метою проекту є розробка додатку з розширеним функціоналом для користувачів, який має забезпечити максимально зручний інтерфейс, індивідуальний підхід до кожного користувача, ефективність в користуванні та актуальність на ринку. У додатку буде реалізовано стандартний інтерфейс з можливістю переглядати погоду, в залежності від геолокації, інтегрований AI-асистент як для спілкування в окремому чаті так і для втілення максимально унікального підходу до користувача (поради, допомога тощо). Також в додаток буде інтегрована карта та планувальних маршрутів, зберігання маршрутів та місць інтересів.

Мобільний додаток буде надавати чимало переваг над аналогами завдяки гнучкому підходу до користувача індивідуально, клієнт матиме можливість зберігати свої дані, маршрути, інтереси, а функціонал з ШІ-асистентом дозволить зробити це якнайретельніше. Такий підхід дозволить створити по-справжньому персоналізований досвід для кожного користувача. Замість типових шаблонних маршрутів, система буде враховувати уподобання мандрівника — улюблені види активностей, типи локацій (природа, музеї, гастрономія), бажаний темп подорожі, бюджет, тривалість та інші фактори. Це дає змогу не просто спланувати поїздку, а створити маршрут, який максимально відповідає очікуванням користувача.

ШІ-асистент у додатку відіграватиме роль особистого консультанта. Він зможе відповідати на запитання в реальному часі, давати рекомендації щодо найкращого часу для відвідування локацій, пропонувати альтернативи у разі зміни погодних умов чи обставин, а також надавати культурні або історичні довідки щодо визначних місць. Асистент також зможе автоматично оптимізувати маршрут, підказуючи, як краще побудувати логістику для економії часу та зменшення витрат.

Крім того, зберігання даних у хмарі через інтеграцію з Firebase дозволить користувачеві мати постійний доступ до своїх маршрутів, заміток, фото, улюблених локацій та навіть спілкування з іншими мандрівниками. Це особливо зручно при зміні пристрою або для тих, хто подорожує у команді — можна легко поділитися маршрутом з друзями чи родиною.

1.4 Висновок до розділу 1

У результаті проведеного аналізу було виявлено, що сучасний ринок туристичних мобільних додатків представлений широким спектром рішень, які охоплюють як базові функції (пошук, бронювання, відгуки), так і спеціалізовані сервіси з додатковим функціоналом. Виділено п’ять ключових категорій: класичні сервіси бронювання (наприклад, Booking, TripAdvisor), інтерактивні аудіогіди, додатки з елементами доповненої реальності, планувальники подорожей, платформи соціальної взаємодії мандрівників, а також нішеві додатки, орієнтовані на певні інтереси (походи, культура, природа тощо).

Кожна з цих категорій має свої сильні сторони, але водночас і обмеження. Жоден із розглянутих сервісів не поєднує повністю можливості персонального планування, інтерактивної взаємодії, збереження інтересів користувача, розумної аналітики на основі ШІ, а також доступності з єдиною інтегрованою логікою. Це дозволяє зробити висновок про наявність значного потенціалу для створення нового інноваційного рішення, яке не лише охоплює найкращі практики наявних сервісів, але й поєднує їх у зручному, персоналізованому мобільному додатку.

Таким чином, мобільний додаток не лише конкурує з аналогами, а й створює нову якість туристичного досвіду — персоналізованого, розумного, гнучкого та зручного в користуванні. Тому ідея створення майбутнього мобільного додатку – туристичного AI-асистента – є актуальною, цікавою та обґрунтованою як з точки зору сучасних технологічних тенденцій, так і з огляду

на потреби користувачів. Такий додаток дозволить не лише автоматизувати процеси планування подорожі, а й забезпечити глибший рівень взаємодії за допомогою штучного інтелекту, який адаптуватиметься до інтересів і звичок конкретного користувача.

Інтеграція таких функцій, як персональний маршрут, чат-бот-помічник, карта з рекомендаціями, погодні сервіси, а також збереження особистих уподобань, формує абсолютно новий підхід до взаємодії туриста з цифровим середовищем. Це відкриває нові можливості для розвитку індустрії подорожей і створення зручного цифрового супутника для кожного мандрівника.

2 РОЗРОБКА АЛГОРИТМУ РОБОТИ МОБІЛЬНОГО ДОДАТКУ ТУРИСТИЧНИЙ AI-АСИСТЕНТ

2.1 Варіантний аналіз шляхів поставленої задачі

Розробка мобільного застосунку для туристичного AI-асистента потребує ретельного аналізу можливих підходів до реалізації його основних функціональних компонентів. Такий варіантний аналіз дозволяє визначити найбільш ефективні рішення з урахуванням цільової аудиторії, вимог до продуктивності, зручності використання, масштабованості та інтеграції з зовнішніми сервісами.

У цьому розділі буде розглянуто кілька можливих шляхів реалізації ключових функціональних частин системи, таких як:

1. Релізація функціонального компонента авторизації і автентифікації

Ключовою особливістю додатку є автентифікація. Реалізація буде необхідна для збереження важливою інформації користувача, що допоможе максимізувати підхід до кожного користувача окремо, та незалежно від об’єктивних чинників. Для автентифікації буде обрано сервіс Firebase Authentication. Аутентифікація Firebase спрямована на те, щоб спростити створення безпечних систем автентифікації, а також покращити вхід і адаптацію для користувачів програми. Також цей підхід передбачає використання Firebase Firestore Database для зберігання досвіду користувача. Варто зазначити, що автентифікація та авторизація — це різні процеси: перша відповідає за підтвердження особи користувача, тоді як друга визначає його рівень доступу до функціоналу системи. Після успішної автентифікації, програма виконує авторизацію – перевірку доступу до певного функціоналу додатку, зокрема можливості збереження маршрутів, генераціхї рекомендацій або перегляду попередньої історії запитів до AI.

Переваги:

- Firebase надає готові SDK для різних платформ (Web, iOS, Android і C++ в нашому випадку, тощо).
- Простота інтеграції найпопулярніших методів автентифікації: звичайна форма логін чи пошта і пароль, соціальні мережі, телефон тощо.
- Firebase реалізує складні механізми захисту та підтримує сучасні стандарти безпеки такі як OAuth 2.0, OpenID Connect.
- Інтеграція з іншими сервісами Firebase: взаємодія на одному рівні з Firestore, Realtime Database, Cloud/Scheduled Functions, Analytics за допомогою SDK або ж використання REST-запитів
- Ключевою особливістю є швидкість розробки, завдяки вже реалізованим готовим рішеням.

Недоліки:

- Залежність від стороньогу сервісу, втрата контролю над інфраструктурою автентифікації.
- Обмежена гнучкість у дизайні інтерфейсу вбудованих UI-форм, хоча можна робити власні, якщо дозволяє середовище та архітектура проекту.
- Безкоштовний тариф має ліміти на кількість користувачів, тому високий трафік може призвести до значних витрат.
- Потрібне інтернет-з’єднання для автентифікації, що іноді є частою проблемою офлайн-додатків.

Можливі альтернативи реалізації:

- Традиційна форма (email + пароль): простота та повний контроль над UI, але базова безпека без двофакторної аутентифікації;
- Вхід через соціальні мережі (Google, Facebook тощо): швидкість та довіра користувачів, однак доволі складна інтеграція з кожним провайдером;

1. Головна сторінка (як окремий незалежний компонент)

Головна сторінка є ключовим елементом інтерфейсу додатку та виконує функцію стартового екрана, з якого користувач починає взаємодію із системою. Вона несе як естетичне, так і функціональне навантаження, створюючи перше враження про додаток і спрямовуючи користувача до основних сценаріїв використання.

Інтерфейс головної сторінки оформлено в дружньому, сучасному стилі, що поєднує простоту, легкість сприйняття та акценти на ключових елементах. Візуальна структура компонента логічно поділена на декілька зон:

- Верхня панель містить віджет актуальної погоди (реалізований як окремий компонент, див. пункт 3), а також кнопку профілю, яка дозволяє швидко перейти до перегляду особистої інформації або налаштувань. Погодні умови визначаються динамічно, відповідно до місцеположення користувача, і можуть впливати на рекомендації маршруту. Варто зазначити, що всі принципи конфіденційності приватної інформації, тобто в данному випадку актуальної геолокації, враховані, та при відкриті застосунку користувач повинен погодитись на збір інформації про геолокацію.
- Центральна частина — це візуальний і функціональний акцент сторінки. Тут розміщено головне повідомлення, яке можна класифікувати як Hero CTA (Call to Action) — короткий надихаючий текст, що стимулює до дії. Наприклад, фраза на кшталт: “Готові вирушити у подорож?” або “Плануйте свій маршрут уже зараз”. Це не лише декоративний елемент — CTA спрямовує користувача на ключову функцію.
- Під CTA розташовується кнопка “Plan a Trip”, яка викликає інтерфейс планування маршруту. Вона має високий пріоритет в ієрархії сторінки й оформлена так, щоб привертати увагу, але не перевантажувати загальну композицію.
- Нижче за основну кнопку передбачено дві стрічки — історія чату та історія маршрутів. Перша дозволяє переглянути попередні запити до AI-асистента, що

зручно для користувачів, які хочуть повернутися до раніше отриманої інформації. Друга – надає доступ до збережених або завершених маршрутів, що відкриває можливість швидко повторити поїздку або подивитися статистику.

- Нижня навігаційна панель або ж footer — це окрема область з трьома основними кнопками: Чат, Маршрути, Карта. Вона завжди доступна на екрані й дозволяє перемикатися між основними розділами додатку без необхідності повертатися на головну.

Переваги:

- Структура побудована таким чином, що окремі частини можна змінювати або доповнювати без впливу на роботу всієї сторінки.
- Користувачеві легко орієнтуватися завдяки логічному розміщенню основних елементів — верхня панель, центральний заклик до дії, нижня навігація.
- Екран адаптується до різних пристроїв без втрати функціональності або зручності.
- Інформація оновлюється автоматично, тому користувач завжди бачить актуальні дані, як-от погоду або недавню історію.
- Загальний стиль і плавність інтерфейсу створюють комфортне середовище, привабливе для щоденного використання.
- Інтеграція всіх ключових сервісів (чат, карта, історія маршрутів) на одному екрані скорочує кількість зайвих переходів

Недоліки:

- При нестабільному інтернеті або збоях зовнішніх сервісів окремі блоки можуть не завантажуватися.
- Велика кількість динамічних елементів може уповільнити перший рендер на старих пристроях.
- Відсутність ретельного кешування може призводити до затримок або неконсистентності даних.

- Щільна сітка інформації підвищує ризик перевантаження інтерфейсу, особливо на екранах із малою площею.

При розробці ми будемо прагнути зробити все якнайкраще, щоб мінімізувати появу недоліків.

Можливі альтернативи реалізації:

- Мінімалістичний варіант із самим Hero-CTA і навігацією: найшвидша реалізація, низьке навантаження на клієнт, але відсутність контексту (історії, рекомендацій).
- Розширений варіант із додатковими блоками: CTA, коротка стрічка історії, рекомендації та швидкі дії; більш інформативний UX, але складніша логіка та повільніший рендер.
- Модульний підхід: кожну зони (CTA, історія чату, історія маршрутів) винести в окремі підкомпоненти; дозволяє поетапну розробку та тестування, але потребує чітко прописаних API та додаткового часу на узгодження інтерфейсів.

Обрано: модульний підхід із поступовим додаванням блоків — спочатку реалізуємо мінімальний варіант (Hero-CTA + навігація), а потім за допомогою окремих підкомпонентів крок за кроком додамо стрічки історії та рекомендацій. Такий шлях дозволить швидко отримати робочий прототип та плавно нарощувати функціональність.

2. Верхня панель навігації (header компонент як незалежний компонент системи)

Панель навігації реалізована як самостійний модуль, що містить два ключові елементи: віджет актуальної погоди та кнопку переходу до профілю. Погодні дані підтягуються з зовнішнього API в реальному часі й відображають температуру, умови (сонце, хмари, дощ тощо) та назву локації. Такий компактний формат дозволяє користувачу одразу оцінити, чи варто планувати маршрут на відкритому повітрі, не переключаючись на інші розділи. При натисканні на іконку профілю відкривається меню з опціями — перегляд особистих даних,

налаштування та вихід із системи. У перспективі цей компонент можна розширити окремою сторінкою “Детальна погода”, де буде представлена інформація про вологість, швидкість вітру, а також прогноз на кілька днів уперед із картами атмосферних явищ. Така функціональність стане цінним доповненням, але на старті розвитку проекту можна залишити її на етапі планування й зосередитися на базовому відображенні погоди в верхній панелі навігації.

Переваги:

- Повна автономність модуля: хедер можна змінювати чи замінювати без впливу на інші компоненти.
- Миттєве відображення погоди дозволяє приймати рішення про вихід без зайвих переходів.
- Іконка профілю завжди під рукою — доступ до налаштувань і виходу здійснюється в один клік.
- Використання зовнішнього API дає гнучкість: легко додати нові параметри (вітер, вологість) або змінити провайдера даних.

Недоліки:

- При нестабільному інтернеті або відмові API користувач може бачити порожній або застарілий віджет.
- Додатковий запит до сервера трохи збільшує час першого рендеру хедера.
- Без механізму кешування відбувається “миготіння” даних при кожному оновленні сторінки.

Можливі альтернативи реалізації:

- Спрощений варіант: лише іконка профілю, без віджету погоди — швидке завантаження та мінімум залежностей, але втрачається корисна інформація про умови для поїздок.

- Інтеграція з системною панеллю: замість окремого хедера — використання системної навігації пристрою (наприклад, на iOS/Android), що спрощує дизайн, але знижує контроль над стилем і можливістю розширення.
- Розширений варіант з випадаючим блоком погоди: при натисканні на віджет відкривається деталізація (вологость, вітер, прогноз), що підвищує інформативність, але ускладнює реалізацію та UX на малих екранах.

Базовий варіант з компактним віджетом погоди та іконкою профілю, які відображаються постійно у верхній панелі. У майбутньому можливе розширення функціоналу через впровадження кнопки для відкриття детальної інформації про погоду. Такий підхід забезпечує баланс між швидкістю, інформативністю та масштабованістю.

3. Чат з AI-помічником

Сторінка чату з AI-асистентом реалізована максимально просто: при натисненні на кнопку «Чат» відкривається повноекранний інтерфейс із полем вводу запиту і кнопкою «Відправити». Вся логіка роботи відбувається «за лаштунками» — користувачу не потрібно вручну обирати режими або налаштовувати параметри: він вводить питання, отримує відповідь від бота, і чат зберігає історію діалогу у фоновій базі Firestore.

Варто зазначити, що на екрані чату буде також присутня інтерактивна картка привітання, яка одразу звертатиметься до користувача і запропонує декілька коротких питань на кшталт “Які типи маршрутів вам найцікавіші?”, “Яка тривалість прогулянки вам підходить?” чи “Чи є у вас уподобання щодо ландшафтів?”. На основі відповідей AI досить швидко сформує детальний профіль смаків і досвіду кожного користувача, зберігатиме їх у Firestore і надалі автоматично кастомізуватиме рекомендації. Це дозволить не лише пропонувати найбільш релевантні маршрути з урахуванням особистих уподобань, а й адаптуватися до зміни настрою чи погодних умов, забезпечуючи дійсно персоналізований досвід.

Переваги:

- Проста інтеграція, мінімалістичний інтерфейс не відволікає від основної задачі.
- Автоматичне збереження історії дозволяє повертатися до попередніх запитів.
- Використання одного поля вводу спрощує UX, особливо для новачків.
- Розробка підключення до Firebase Authentication і Firestore буде готова до реалізації цього компоненту, тож інтеграція “чату” не вимагає додаткової серверної логіки.

Недоліки:

- Якщо з’єднання з інтернетом уповільнене, можлива затримка відповіді.
- Одне поле вводу не дає можливості форматувати запити або вкладати файли.
- У разі великих об’ємів історії пошук по попередніх запитах може працювати повільніше.

Можливі альтернативи реалізації:

- Багатовкладковий інтерфейс чату: користувач може створювати окремі "теми" або "сесії", наприклад: «Маршрути в Карпатах», «Ідеї для сімейного відпочинку». Це дозволяє структурувати діалоги, але ускладнює навігацію і вимагає додаткової логіки керування станом.
- Голосовий інтерфейс: користувач озвучує запити, а бот відповідає голосом або текстом. Це зручно в дорозі, проте потребує складної реалізації та врахування питань конфіденційності (обробка аудіо).
- Форматований інтерфейс із кнопками-відповідями: бот пропонує вибір готових варіантів замість вільного вводу. Такий підхід зручний для швидких сценаріїв, але обмежує гнучкість користувача.

Обрано мінімалістичний одновіконний інтерфейс з текстовим полем, вітальною карткою та збереженням історії в Firestore.

Це дозволяє забезпечити простоту взаємодії, персоналізацію досвіду та масштабованість у майбутньому. Розширення функціоналу можливе через додавання голосового вводу або підтримки вкладених сесій, однак на старті реалізація фокусується на стабільній текстовій взаємодії.

4. Сторінка збережених маршрутів

Сторінка маршрутів відкривається по натисненню кнопки «Маршрути» в нижній навігації та займає весь екран. На ній представлений лінійний перелік усіх збережених користувачем поїздок: для кожного запису одразу видно назву маршруту, дату створення та короткий опис — із вказанням пункту відправлення і пункту призначення.

При торканні будь-якого запису відкривається детальний перегляд маршруту: на інтерактивній карті з’являються піни, які позначають ключові точки шляху (наприклад, А, Б, С тощо), а під картою — текстовий опис послідовності зупинок і приблизний час у дорозі. Такий підхід дає змогу швидко зорієнтуватися, куди саме пролягає шлях, побачити проміжні точки і при потребі видалити або повернутися до подорожі пізніше.

Переваги:

- Чіткий перелік усіх поїздок із найважливішими даними (назва, дата, від–до) робить навігацію простою.
- Лінійний список швидко завантажується й не навантажує інтерфейс зайвими елементами.
- Можливість одним тапом відкрити маршрут на карті із пінами допомагає одразу оцінити його маршрут і проміжні точки.
- Короткий опис поруч із картою дозволяє швидко пригадати деталі поїздки без зайвих переходів.

Недоліки:

- Відсутність фільтрів і сортування ускладнює пошук потрібної поїздки при великому обсязі маршрутів.
- Якщо збережено багато поїздок, список може стати надто довгим і важким для скролу.
- Немає попереднього перегляду карти у списку — користувач бачить піни лише після відкриття детального перегляду.
- Без офлайн-кешу користувач не зможе переглянути список чи карту без інтернет-з’єднання.
- Відсутність можливості редагувати або перейменовувати маршрут прямо зі списку обмежує гнучкість використання.

Можливі альтернативи реалізації:

- Сітка з картками (grid view): кожен маршрут подається у вигляді плитки з міні-картою, заголовком і кнопкою дій. Це забезпечує візуальну привабливість і дає уявлення про географію кожного маршруту ще до відкриття, однак значно навантажує інтерфейс і уповільнює завантаження при великій кількості маршрутів.
- Групування маршрутів за категоріями або тегами: наприклад, «Гори», «Міські прогулянки», «На вихідні». Це допомагає структурувати список, але вимагає впровадження додаткової логіки класифікації маршрутів.
- Інтеграція функції “вибране”: дозволяє виділити важливі або улюблені маршрути. Це покращує навігацію, проте потребує додаткової кнопки та збереження стану.

Обрано лінійний список з базовою інформацією та можливістю відкриття детального перегляду маршруту з картою.

5. Сторінка інтерактивної карти

Сторінка карти доступна з нижньої навігації або зі списку маршрутів. При першому вході користувачу пропонується обрати маршрут із наявних або перейти в режим AI-асистента для генерації нового шляху за вподобаннями.

Інтерфейс містить звичайну карту з масштабуванням і прокруткою, піни позначають обрані точки шляху.

Переваги:

- Інтерактивність: можна наближати карту, прокручувати її та переглядати деталі маршруту в реальному часі.
- Гнучкість вибору: одразу після відкриття можна перейти до існуючого маршруту або згенерувати новий зі штатного AI-асистента.
- Мінімалізм: нічого зайвого — лише карта та маркери, що сприяє фокусуванню на головній задачі.
- Автоматичне центрування по обраному маршруту допомагає користувачу відразу побачити всі ключові точки.
- Легка інтеграція з іншими модулями: карта використовує ті ж координати, що й список маршрутів і AI-асистент.

Недоліки:

- Без інтернету карта не завантажується
- Відсутність додаткових інструментів

Можливі альтернативи реалізації:

- Повноекранний режим карти без жодних панелей чи відволікаючих елементів — для максимального фокусу на навігації.
- Шари інформації (атмосферні умови, рельєф, визначні місця) — підходить для досвідчених користувачів, але ускладнює UI.
- Підтримка офлайн-карт — критично важливо для туристичних походів, однак вимагає попереднього кешування й збільшує обсяг додатку.

Наразі обрано реалізацію з базовою функціональністю онлайн-карти з пінами маршрутів і підтримкою масштабування та автопозиціювання. Цей мінімалістичний підхід дозволяє швидко запустити стабільну версію функції з

можливістю її поступового розширення — від додавання шарів погоди до режиму побудови власного маршруту вручну або завантаження офлайн-секцій карти.

6. Сторінка профілю

На неї можна буде потрапити через кнопку профілю в панелі навігації сторінка займає весь екран і містить: контактну інформацію, бащові налаштування користувача, кнопку виходу з акаунту. Також можна буде при бажанні розробити функціонал зміни паролю за допомогою внутрішніх сервісів Firebase – Firebase Auth .див пункт 1.

Переваги:

- Простий і зрозумілий інтерфейс: мінімум відволікаючих елементів.
- Можливість керування основними налаштуваннями в одному місці.
- Адаптованість для майбутнього розширення
- Кнопка виходу доступна завжди — що важливо з точки зору безпеки на загальнодоступних пристроях.

Недоліки:

- Обмежена функціональність на старті — лише базова інформація без розширених налаштувань.
- Немає можливості змінити аватар чи інші особисті дані (потребує додаткової логіки).
- Зміна email або прив’язка соцмереж (Google, Apple, Facebook) поки не підтримується.

Можливі альтернативи реалізації:

- Профіль у вигляді бокової панелі (слайдера) — відкривається поверх поточної сторінки без повного переходу. Зменшує кількість переходів, але обмежує простір для налаштувань.

- Профіль як таби або секції всередині одного екрану: «Основна інформація», «Безпека», «Підписки» — зручно для масштабування функціоналу, але може ускладнити інтерфейс.
- Інтеграція з Google / Apple / Facebook профілем — дозволяє автоматично підтягувати ім’я, фото та пошту, але потребує додаткового налаштування OAuth.

Було обрано стратегію реалізації базового повноекранного профілю з основними елементами — ім’ям, email, кнопкою виходу та (опційно) зміною паролю. Такий підхід дозволяє швидко впровадити ключову функцію керування обліковим записом з можливістю подальшого розширення.

2.2 Розробка структури програмного додатку

Розробка структури програмного додатку є одним із ключових етапів створення цілісного та надійного цифрового продукту. Вона визначає, як окремі компоненти системи взаємодіють між собою, які дані передаються між модулями, та як забезпечується гнучкість і масштабованість рішення. У цьому розділі детально описано логіку побудови основних екранів, модулів і взаємозв'язків між ними. Особливу увагу приділено інтеграції з хмарними сервісами, таким як Firebase, реалізації персоналізації за допомогою AI-модулів та збереженню стабільної роботи інтерфейсу незалежно від зовнішніх умов.

Функціонал впроваджений в наш додаток:

- Автентифікація
- Зберігання данних у базу даних
- Робота з базою даних для покращення користування додатком
- Можливість переглядати актуальну погоду
- Зміна приватних налаштувань
- Чат з AI-асистентом

- Маршрути згенеровані на основі запитів користувача
- Інтерактивні карти з пінами

Так як більш детально ми розібрали обрані компонети в минулому підрозділі, у цьому підрозділі ми підсумуємо ці компоненти та створимо flowchart, який буде відображати архітектуру нашої системи в цілому та логіку зв'язків між компонентами, що допоможе нам більш чітко зрозуміти архітектуру нашого додатку.

Загальна архітектура мобільного додатку виглядає наступним чином:

- Головна сторінка (HomePage): Виступає як навігаційний хаб. Виводить ключову інформацію (привітання, погоду, поточне місце, AI-підказки) та надає доступ до інших розділів.
- Карта (Map): Інтерактивна карта з підтримкою геолокації, відображенням маршрутів та маркерів. Отримує координати користувача в реальному часі та дозволяє зберігати маршрути.
- Історія маршрутів чи Маршрути (Itinerary): Відображає список збережених маршрутів, отриманих з Firebase Realtime Database. Доступна можливість перегляду деталей маршруту та його повторного використання.
- AI-асистент (Chat): Інтерфейс спілкування з AI-асистентом. Отримує контекст користувача з інших частин додатку для генерації більш точних відповідей (наприклад, рекомендації маршрутів, опис місць, погода).
- Модуль погоди (WeatherWidget): Вбудований віджет, що підтягує погодні дані з відкритого API на основі поточних координат користувача. Використаймо OpenWeatherMap API як постачальних інформації.
- Профіль користувача (Profile): Екран з інформацією про акаунт користувача (ім'я, email тощо). Дає змогу редагувати персональні дані або вийти з облікового запису.

Також для загального розуміння розділимо серверну логіку на різні “сервіси”. Вони допоможуть нам краще зрозуміти серверну (backend) логіку нашого застосунку:

- LocationService – отримує координати користувача в реальному часі; доступний для: HomePage, Map, WeatherWidget, Chat
- FirebaseAuthService – аутентифікація користувача та подальша авторизація;
- FirebaseDBService - зберігання та отримання маршрутів (Itinerary); збереження персональних даних профілю (Profile). Для кращого розуміння серверної сторони було прийнято рішення розділити Firebase Service на дві залежні частини
- WeatherService – інтеграція з OpenWeatherMap API; повертає погодні дані на основі координат; використовується в: WeatherWidget, HomePage, Chat
- ChatService – логіка обробки запитів до AI; отримує контекст (локація, погода, маршрут); повертає згенеровані відповіді для користувача

Взаємозв’язки між компонентами:

1. HomePage
   - Отримує координати користувача через LocationService
   - Отримує погоду через WeatherService
   - Отримує AI-підказки через ChatService
   - Виконує роль агрегатора персоналізованої інформації
2. Map
   - Отримує поточне місцезнаходження через LocationService
   - Зберігає маршрути через FirebaseDBService
   - Відображає геодані в реальному часі
3. Itinerary (Маршрути чи Історія маршрутів)
   - Завантажує список маршрутів через FirebaseDBService

- Відкриває обраний маршрут на карті через Map

4. Chat

- Отримує відповіді від ChatService
- Отримує координати користувача через LocationService
- Включає дані про погоду через WeatherService
- Надає персоналізовані поради з урахуванням контексту

5. WeatherWidget

- Отримує погодні дані через WeatherService
- Отримує координати користувача через LocationService
- Відображає актуальну погоду

6. Profile

- Отримує та редагує персональні дані через FirebaseAuthService
- Відповідає за управління обліковим записом

7. Панель навігації

- Забезпечує навігацію між усіма основними екранами (HomePage, Map, Itinerary, Chat, Profile)

Підсумовуючи, мобільний додаток складається з кількох ключових компонентів, які взаємодіють між собою через спеціалізовані сервіси для забезпечення коректної та зручної роботи користувача. Головна сторінка виконує роль централізованого хабу, збираючи дані про місцезнаходження, погоду, AI-підказки та персональну інформацію. Карта і історія маршрутів відповідають за візуалізацію та збереження геоданих, а чат із AI-асистентом забезпечує інтелектуальну підтримку користувача. Модуль погоди та профіль користувача доповнюють функціональність додатку, надаючи актуальні дані та можливість керувати власними налаштуваннями. Навігаційна панель забезпечує простий доступ до всіх розділів.

Для більш наочного розуміння архітектури та взаємозв'язків між компонентами наступним кроком ми побудуємо flowchart (рисунок 2.1), який відобразить логіку роботи додатку та зв'язки між його складовими. Це допоможе краще уявити загальну структуру системи та полегшить подальший процес розробки.

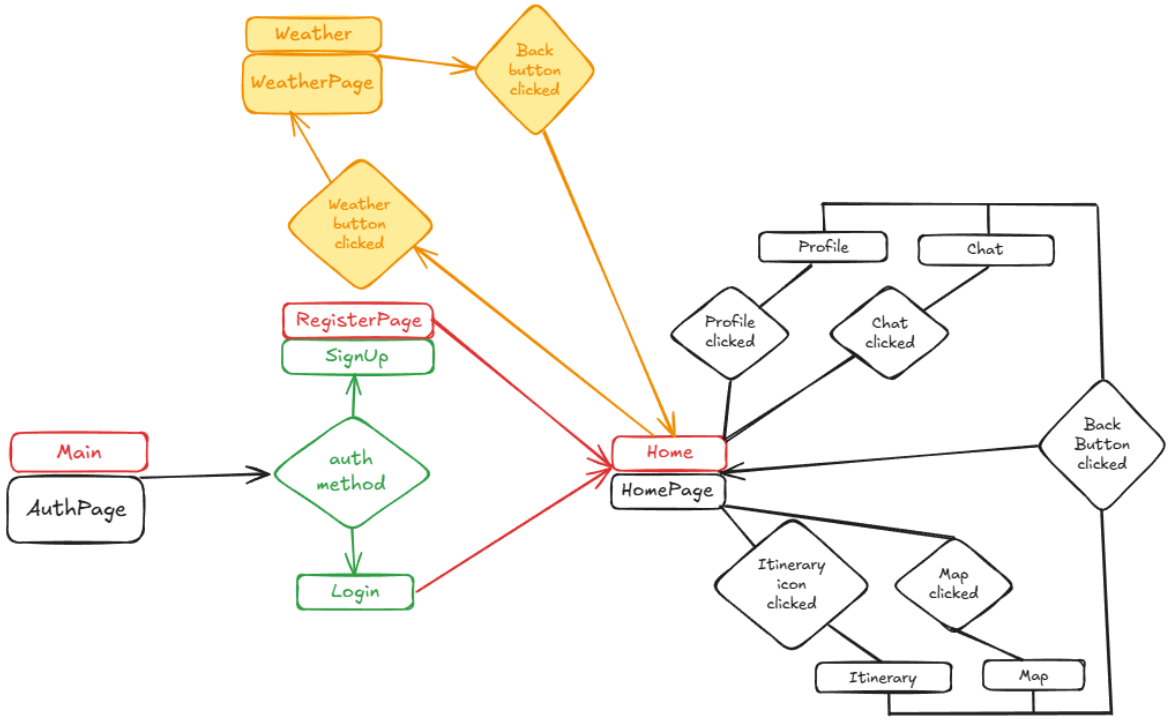

Рисунок 2.1 – Flowchart архітектури мобільного додатку туристичний AI-асистент

Легенда:

Зелений колір – позначено логіку обробки автентифікації, основні CTA елементи на етапі входження в додаток.

Червоний колір – позначено основні сторінки на етапі автентифікації/авторизації.

Чорний колір – позначено назви сторінок основного функціоналу додатку. Також кнопки навігації між цими сторінками та компонетом Home, сторінкою HomePage

Жовтий колір – позначає сторінку, компонент системи, яка не є обов'язковою (опціональною) частиною дипломного проєкту, а слугує як додатковий функціонал. Вона виділена окремо, щоб підкреслити її факультативний характер і можливість подальшого розширення системи.

2.3 Розробка загального алгоритму роботи мобільного додатку туристичний AI-асистент

Розробка алгоритму для мобільного додатку AI-туристичного асистента є важливим етапом, що забезпечує зручність та ефективність взаємодії користувачів із системою. Алгоритм визначає послідовність кроків, які формують функціонал додатку: від запуску до надання персоналізованої допомоги в подорожах.

Для кращого уявлення функціонування додатку було створено відповідну схему (рис. 2.2).

Після запуску додатку користувач спочатку проходить процедуру автентифікації — може увійти в свій акаунт або зареєструвати новий. Лише після успішного входу користувач потрапляє на головний екран, де його вітають, відображається актуальний прогноз погоди та показується його місцезнаходження на інтерактивній мапі. Тут також доступні ключові функції додатку, а персоналізовані дані та налаштування користувача стають доступними.

На наступному етапі користувач може обрати серед ключових функцій: перегляд історії своїх маршрутів, створення нових шляхів за допомогою звернення до AI-асистента через чат. Також серед вітальних елементів є кнопка

“Запланувати маршрут/подорож”, яка перекидає користувача на сторінку чату з попередньо підготовленими питаннями. Це приклад альтернативного входу до модуля чату: користувач потрапляє в той самий інтерфейс, що й при звичайному відкритті чату, але з іншим початковим контекстом. Такий підхід дозволяє підвищити зручність та інтуїтивність користування додатком, надаючи кілька шляхів доступу до однієї функції — як через головне меню, так і через тематичні кнопки, що підказують можливості на головному екрані. Під час побудови маршруту додаток в реальному часі збирає геолокаційні дані, що дозволяє зберігати вибрані локації та маршрути в базі даних для подальшого використання.

AI-асистент працює з контекстною інформацією, отриманою від інших модулів, включаючи поточну локацію, погодні умови та дані про маршрути. На основі цього він генерує персоналізовані поради щодо місць для відвідування, відповідає на запити користувачів та надає корисні рекомендації. Всі сервіси додатку функціонують асинхронно, що забезпечує швидку та плавну роботу системи.

Крім того, користувач має доступ до свого профілю, де можна редагувати особисті дані, налаштування або виконати вихід із облікового запису.

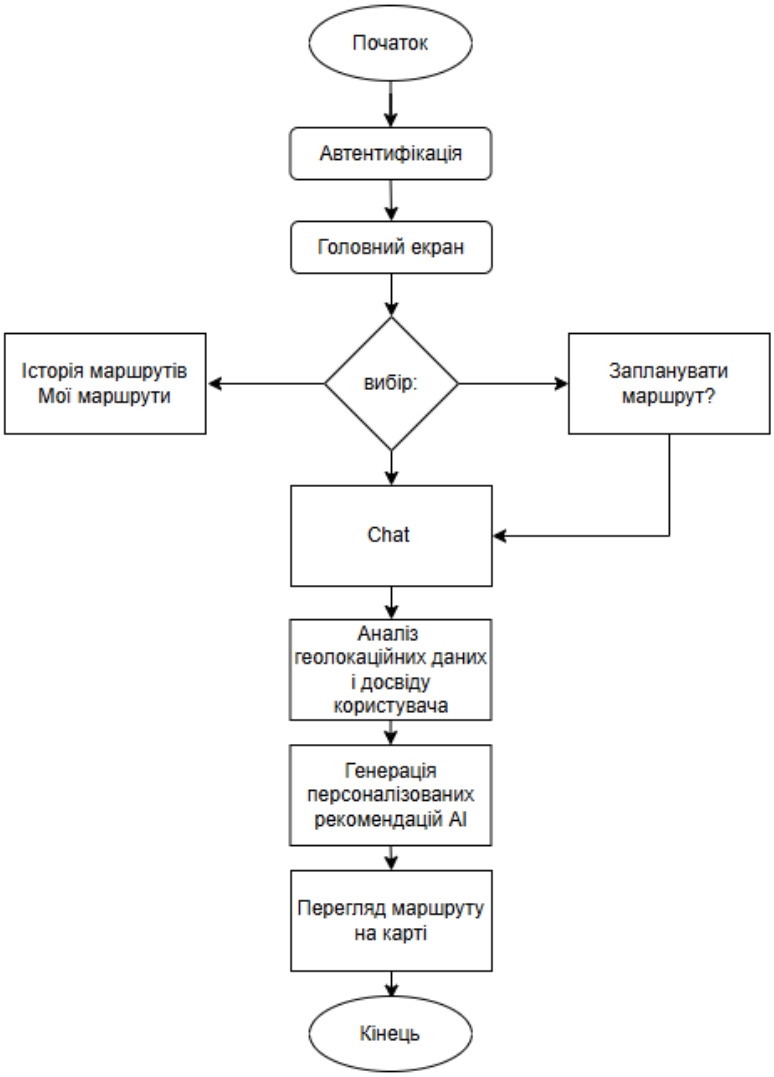

Рисунок 2.2 – алгоритм роботи мобільного додатку туристичний AI-асистент

Алгоритм, незважаючи на його відносну простоту, гарантує комфортний і логічно впорядкований користувацький досвід. Чітка структура розподілу функцій між модулями дозволяє легко розширювати можливості додатку та додавати нові функції без кардинальних змін в існуючій архітектурі.

Завдяки централізованому управлінню даними та асинхронній роботі компонентів, додаток демонструє високу надійність і продуктивність. Це сприяє підвищенню зручності для користувачів, швидкому обробленню запитів та оптимізації процесу планування подорожей.

2.5 Висновок до розділу 2

У другому розділі проведено детальний аналіз варіантів і обґрунтовано вибір ключових компонентів мобільного застосунку туристичного AI-асистента. Розглянуто кілька підходів до реалізації аутентифікації із використанням Firebase Auth і Firestore, а також проаналізовано інтерфейсні рішення для головної сторінки, хедера з віджетом погоди, чат-модуля, сторінок маршрутів, інтерактивної карти та профілю користувача. Для кожного компонента визначено переваги, недоліки та можливі альтернативи, що дозволило обрати оптимальний модульний підхід, який передбачає поступове розширення функціональних можливостей.

Описано загальну структуру програми із виокремленням фронтенд-екранів (HomePage, Map, Itinerary, Chat, WeatherWidget, Profile) і серверних служб (LocationService, FirebaseAuthService, FirebaseDBService, WeatherService, ChatService). На основі цієї архітектури створено діаграму потоків (рис. 2.1) та алгоритм роботи системи (рис. 2.2), які ілюструють логіку взаємодії між компонентами, асинхронний обмін даними й збереження користувацького контексту.

Отримана архітектура гарантує простоту розробки та масштабування: чітке розподілення відповідальностей між модулями полегшує тестування й впровадження нових функцій без значних змін у вже наявних елементах системи. Завдяки використанню хмарних сервісів Firebase та зовнішніх API у поєднанні з асинхронною обробкою забезпечено високу продуктивність, стабільність і персоналізований досвід для користувачів. Ці розробки формують надійну основу для впровадження та подальшого вдосконалення AI-туристичного асистента.

3 РОЗРОБКА ТА ТЕСТУВАННЯ МОБІЛЬНОГО ДОДАТКУ ТУРИСТИЧНИЙ AI-АСИСТЕНТ

3.1 Обґрунтування вибору мови та середовища розробки

Мова програмування, що використовується для мобільного додатку туристичного AI-асистента, - це C++ з використанням Android NDK, також модулів написаних на мові Java та TypeScript файлів, які в подальшій розробці будуть використані як носії перекладів UI.

Було прийнято рішення орієнтуватися на платформу Android як основну для створення додатку. Це зумовлено наступними причинами:

- Android є найбільш поширеною мобільною операційною системою у світі
- Платформа Android надає відкриті інструменти та SDK для розробки, тестування без потреби у використанні сторонніх ресурсів чи обладнання
- До прикладу, доступ до розробки на операційній системі IOS був обмежений, оскільки розробка під IOS вимагає використання операційної системи macOS та пристроїв компанії Apple, що є суттєвим бар’єром в студентському середовищі.

У всякому разі, інструмент який буде використовуватися в розробці дає змогу розробляти додаток кросплатформово, без необхідності, використання інших мов, середовищ розробки тощо.

C++ — мова програмування загального призначення з підтримкою кількох парадигм програмування: об'єктно-орієнтованої, узагальненої, процедурної та ін. Б'ярн Страуструп (англ. Bjarne Stroustrup) почав створювати C++ в AT&T Bell Laboratories (Мюррей-Хілл, Нью-Джерсі) у 1979 році. На етапі зародження мова мала назву «C з класами». Згодом Страуструп перейменував мову на C++ у 1984

р. Вперше описана міжнародним стандартом ISO/IEC 14882:1998 (C++98), найбільш актуальним же є стандарт ISO/IEC 14882:2020 (C++20).

C++ — одна з найбільш впливових і потужних мов програмування, яка вже багато десятиліть залишається ключовим інструментом у сфері розробки програмного забезпечення, де особливо цінуються ефективність, продуктивність і детальний контроль над ресурсами. Синтаксис мови об'єднує принципи об'єктно-орієнтованого програмування з можливістю прямого управління пам'яттю, що дає розробникам унікальну гнучкість при оптимізації під конкретне апаратне забезпечення або підвищення продуктивності.

C++ постійно розвивається: сучасні стандарти, зокрема C++11, C++17, C++20, систематично збагачують мову новими засобами, такими як підтримка багатопотоковості, сучасне метапрограмування, робота з шаблонами та використання бібліотеки STL (Standard Template Library). Завдяки цьому C++ зберігає зворотну сумісність, але водночас розширює функціональні можливості, реагуючи на сучасні виклики програмної інженерії.

Універсальність C++ відображається в його застосуванні: від системного програмування й вбудованих рішень до створення складних прикладних систем, ігор, програм для обробки графіки, наукових досліджень і штучного інтелекту. Багаторічна історія використання, велика кількість готових бібліотек і активна спільнота фахівців роблять C++ надійним вибором для розробки стабільних, масштабованих і високопродуктивних програмних продуктів.

Незважаючи на те, що основна логіка AI реалізована на сервері та доступна через REST API, C++ все ще має вагомі переваги з наступних причин:

- максимальна продуктивність. C++ забезпечує високу швидкість виконання, а також ефективне використання ресурсів пристрою, що є критично важливим у випадку обробки великих обсягів даних, таких як кешування результатів API, мультимедійний контент та GPS-навігація;

- гнучкість при роботі з мережею. Швидкі та оптимізовані HTTP-запити до зовнішніх API-сервісів (таких як Open API, картографічні сервіси, сервіси погоди) можуть виконуватися з використанням C++, що гарантує стабільну та контрольовану комунікацію з сервером;
- можливість низькорівневого доступу до функцій пристрою. Робота із сенсорами, камерою, GPS, локальним зберіганням даних тощо вимагає високого рівня контролю, який надається мовою C++;
- модульність та масштабованість. Використання C++ сприяє розробці модульної архітектури, де логіка зв'язку з сервером, обробка даних та UI можуть бути чітко розділені для легшого обслуговування та розширюваності в майбутньому.

У ході розробки мобільного додатку туристичного AI-асистента я обрав середовище розробки Qt Creator. Це офіційна інтегрована середа від розробників фреймворку Qt, яка забезпечує повну інтеграцію з усіма необхідними інструментами. Qt Creator спеціально розроблений для роботи з C++ та QML і є ефективним інструментом для створення кросплатформених графічних додатків, зокрема й для Android. Обґрунтування вибору фреймворку та бібліотек буде описано в наступному підрозділі.

Розглянемо основні переваги Qt Creater як нативного середовища розробки:

- Qt Creator забезпечує глибоку інтеграцію з Android SDK і фреймворком Qt: усі основні функції, необхідні для збирання, компіляції та розгортання додатків на Android-пристроях, доступні без необхідності застосування стороннього програмного забезпечення. Це дозволяє розробникам ефективно організовувати робочий процес, використовуючи лише базові інструменти середовища.

- Щодо інструментів розробки UI, у середовищі передбачено інтегрований Qt Designer, який дає змогу оперативно створювати адаптивні інтерфейси. Підтримується як робота з QML, так і з класичними віджетами, що забезпечує гнучкість у підходах до побудови інтерфейсу та значно спрощує процес його розробки.
- Qt Creator забезпечує глибоку інтеграцію з CMake і qmake: середовище ідентифікує структуру проєкту, автоматично створює параметри збірки, а також пропонує зручний механізм керування залежностями.
- Щодо налагодження та профілювання, IDE оснащене розвинутими інструментами: доступна інтеграція з GDB і LLDB, перегляд стеку викликів, відслідковування змінних, керування потоками — усе це сприяє ефективному аналізу коду.
- Процес налаштування мінімальний: після встановлення достатньо лише зазначити шляхи до Android SDK та NDK, після чого середовище готове до роботи.

Також в наш час є багато аналогів, одними із найпопулярніших є Clion від компанії JetBrains і VS Code від компанії Microsoft.

Розглянемо ці аналоги, їх основні недоліки порівняно з Qt Creator IDE.

Visual Studio Code позиціонується як легкий, кросплатформений редактор із багатим вибором розширень. Проте, при розробці на C++ та Qt, його можливості мають серйозні обмеження:

- Підтримка Qt реалізована не на нативному рівні. Для повноцінної роботи необхідно вручну встановлювати відповідні розширення, налаштовувати компілятор, інтегрувати qmake або CMake, а також окремо підключати налагоджувач. Це ускладнює початкову конфігурацію середовища.

- Робота з Android у VS Code також викликає труднощі: хоча інтеграція Android SDK/NDK можлива, вона вимагає значних зусиль із налаштування та подальшого обслуговування.

- Ще одним суттєвим недоліком є відсутність візуального редактора інтерфейсу. Розробка UI на базі QML чи віджетів здійснюється виключно у вигляді коду, без можливості попереднього перегляду або зручного візуального редагування.

CLion є дійсно потужним середовищем розробки для C/C++ від компанії JetBrains: сучасний інтерфейс, якісна інтеграція з CMake — усе це сприяє продуктивній роботі. Проте існують певні обмеження, які варто враховувати.

- Перш за все, відсутня офіційна підтримка Qt. Хоча інтеграцію через CMake можна реалізувати вручну, повноцінної підтримки QML та Qt Designer тут немає. Щодо розробки під Android, процес налаштування є досить складним: потрібно встановлювати Android SDK окремо, налаштовувати компілятори, профілі збірки та середовище виконання.

- Ще один важливий аспект — системні вимоги. CLion споживає значно більше ресурсів у порівнянні з Qt Creator, що може негативно впливати на продуктивність, особливо на менш потужних комп’ютерах.

Враховуючи поставлену мету проєкту — розробку кросплатформенного мобільного додатку для Android — вибір середовища розробки здійснювався з урахуванням кількох ключових критеріїв. Було важливо забезпечити легку інтеграцію з Android SDK, наявність нативної підтримки Qt та QML, а також мінімізувати час, необхідний для первинної конфігурації. Додатково значення

мала наявність візуального конструктора інтерфейсу й стійка підтримка мови C++.

Qt Creator повністю відповідає цим вимогам: середовище забезпечує стабільну роботу, зручний інтерфейс та функціональність, необхідну для реалізації проєктів різної складності. Його використання дозволяє зосередитися переважно на розробці логіки застосунку, організації взаємодії з API та впровадженні функціональних можливостей без зайвих часових витрат на налаштування інфраструктури.

3.2 Обґрунтування вибору бібліотек та фреймворків

Для створення додатку як згадувалось в попередньому додатку було використано фреймворк Qt. Qt— крос-платформовий інструментарій розробки програмного забезпечення (ПЗ) мовою програмування C++. Дозволяє запускати написане за його допомогою ПЗ на більшості сучасних операційних систем (ОС), просто компілюючи текст програми для кожної операційної системи без зміни початкового коду. Містить всі основні класи, які можуть бути потрібні для розробки прикладного програмного забезпечення, починаючи з елементів графічного інтерфейсу й закінчуючи класами для роботи з мережею, базами даних, OpenGL, SVG і XML. Бібліотека дозволяє керувати потоками, працювати з мережею та забезпечує крос-платформовий доступ до файлів.

Основна перевага Qt – це створення кросплатформових додатків без потреби зміни початкового коду. Таку універсальність дає мова програмування C++ яка є фундаментальною, різні мови, інтрументарії тощо, розробленні для створення додатків на різних платформав, наслідують від цієї мови програмування.

Qt містить великий набір готових рішень, що значно спрощує розробку, особливо архітектурно-вимогливих програмних рішень.

Насправді, якщо розглядати Qt як основу для розробки мобільного туристичного AI-асистента, це доволі виправданий вибір з низки причин.

- По-перше, Qt забезпечує доступ до ключових системних ресурсів пристрою: геолокація, камера, акселерометр, гіроскоп, компас — усе це інтегрується через стандартні API Qt. Для мобільного застосунку, орієнтованого на подорожі, така функціональність є базовою: GPS-трекінг, робота з камерою чи AR-модулями реалізується без надмірних ускладнень.
- Ще однією суттєвою перевагою є Qt Quick (QML) — декларативний підхід до створення UI, що дозволяє швидко розробляти сучасні, динамічні й інтуїтивно зрозумілі інтерфейси. Анімації, інтерактивні карти, інформаційні панелі, маршрути чи підказки від AI — усе це легко реалізується в рамках екосистеми Qt.
- Кросплатформеність також не викликає сумнівів: код, написаний з використанням Qt, компілюється під Android, iOS, Windows, macOS та Linux майже без змін. Навіть якщо поточна розробка фокусується на Android, у майбутньому порт на інші платформи здійснюється без суттєвих труднощів, що позитивно впливає на життєвий цикл продукту.
- Щодо порогу входу, Qt має розвинену документацію, активну спільноту, значний обсяг навчальних матеріалів та готових компонентів, що сприяє прискоренню розробки, масштабуванню й інтеграції нових функцій (наприклад, через REST API).
- Додатково, поєднання Qt з C++ дозволяє частину AI-логіки реалізувати безпосередньо на клієнтському пристрої — наприклад, для кешування моделей чи попередньої обробки запитів. Це забезпечує високу продуктивність, контроль над пам’яттю та передбачуваність роботи.
- Інтеграція з REST API у Qt реалізована на високому рівні: робота з HTTP-запитами, обробка JSON та XML дозволяють підключати зовнішні сервіси,

такі як OpenAI, Google Maps чи Weather API, і оперативно надавати користувачу актуальні дані.

- Безпека та стабільність також є сильними сторонами Qt: підтримуються сучасні стандарти захисту, зокрема шифрування та безпечне зберігання локальних даних, що важливо при роботі з персональною інформацією.
- Для роботи з мультимедійним контентом (аудіо-, відео-, інтерактивні карти) Qt пропонує повний стек відповідних технологій, зокрема Qt Multimedia та Qt Location, що дозволяє інтегрувати мапи, маршрути й пошук об’єктів.
- Зручність розробки забезпечується IDE Qt Creator, яка містить візуальний редактор QML, автодоповнення, інтеграцію з CMake, засоби профілювання й налагодження, що підвищує ефективність навіть у складних UI-проєктах.
- Нарешті, модульна архітектура на базі C++ дає змогу розширювати та повторно використовувати код, інтегрувати нові API або створювати бібліотеки для інших мов через C API, що сприяє довготривалій підтримці та масштабуванню проєкту.

У процесі вибору технологій для створення мобільного туристичного AI-асистента було розглянуто декілька варіантів, серед яких Flutter, React Native та Android Studio з Kotlin. Після детального порівняльного аналізу переваг і недоліків кожного з них остаточний вибір зупинили на Qt. Основними причинами стали висока продуктивність Qt, його кросплатформені можливості, надання розробнику глибокого низькорівневого контролю, а також гнучкість архітектурних рішень, що дозволяє оптимізувати розробку під специфічні потреби проєкту.

3.3 Розробка додатку

Імплементація вищезгаданих алгоритмів, та додатку в цілому вимагає побудови, великої архітектури і якісний менеджменту самого проекту. Якісний менеджмент структури папок, файлів та ресурсві дозволить налагодити процес взаємодії між ними, уникнути можливих проблем під час розроблення програмного додатку.

Цю структуру можна поділити на декілька папок та, очевидно, включаючи кореневі файли. Також ми не будем враховувати файли фреймворка:

- pages (папка для сторінок в root directory)
- pages/auth (папка для сторінок, що відповідають за автентифікацію)
- pages/icons (папка для медіа-ресурсів, іконок мобільного додатку)
- src (папка з C++ класами)
- main.cpp (головний файл, точка входу в додаток)
- Main.qml (головний QML файл)

Запропонована структура дійсно сприятиме підвищенню ефективності розробки програмного забезпечення, а також значно полегшить його підтримку та подальший розвиток. Вона забезпечить зрозумілість і зручність як для початківців, так і для досвідчених розробників, що працюють над проектом.

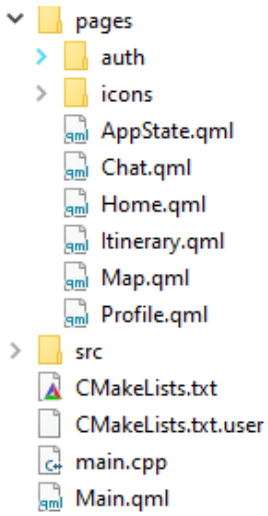

Рисунок 3.1 – структура проекту

У процесі розробки проекту була використана бібліотека Qt Quick, який забезпечує можливість створення інтерфейсів користувача за допомогою мови розмітки QML. Використання Qt Quick дозволяє значно прискорити розробку інтерактивних та адаптивних додатків завдяки декларативному опису інтерфейсу та підтримці сучасних анімацій і ефектів.

Хоча в Qt існує також традиційна система віджетів — Qt Widgets, яка більше орієнтована на класичні десктопні застосунки, застосування Qt Quick дає перевагу у розробці сучасних мобільних та кросплатформених додатків із більш гнучким та візуально привабливим інтерфейсом.

Таким чином, вибір Qt Quick і QML для розробки UI сприяв підвищенню продуктивності, зменшенню часу створення інтерфейсних компонентів та покращенню загальної якості користувацького досвіду.

Таким чином, можна перейти до переглядку всіх важливих частин проекту. Розділимо проект на логічно-зв’язані частини:

- Автентифікація
- Головне меню
- Чат

- Маршрути
- Карта
- База данних

Розгляне кожну вищезазначену частину більш детально, проаналізуємо зв’язки з іншими частинами, знайдемо переваги і недоліки кожної з частин.

1. Автентифікація

Як було зазначено ще у 2 розділі, для процесу автентифікації та подальної авторизації було використано Firebase Authentication Service.

Після запуску додатку користувач відразу потрапляє на екран входу. Даний екран реалізовано з використанням QML, що дало змогу створити сучасний та інтуїтивно зрозумілий інтерфейс із анімованим фоном, стилізованими полями для введення та інтерактивними кнопками. Окрім базової авторизації за допомогою email та пароля, на цьому екрані також передбачена можливість переходу до сторінки реєстрації через інтерактивне посилання “Don’t have an account? Sign up”. Це забезпечує зручність доступу до функцій додатку як для нових, так і для існуючих користувачів.

Для реалізації автентифікації використано сервіс Firebase Authentication. Весь процес – від надсилання запиту до обробки відповіді – організовано за допомогою класу FirebaseAuthManager, який є C++-обгорткою для взаємодії з Firebase REST API. Основні функції цього класу – signIn() та signUp(), що здійснюють HTTP POST-запити до відповідних сервісів Firebase, обробляють отримані відповіді та повідомляють про успіх або помилку операції.

У разі успішної авторизації система отримує idToken та унікальний ідентифікатор користувача (localId). Вказані дані мають ключове значення, оскільки використовуються для подальшої авторизації запитів у FirestoreManager та інших частинах застосунку. Фактично, механізм автентифікації тісно інтегрований з менеджером чату (ChatManager), менеджером зображень

(ImageManager), а також із FirestoreManager, де зберігаються й обробляються персоналізовані дані користувачів. Без отримання idToken і localId коректне функціонування цих компонентів неможливе.

У QML-логіці сторінки входу реалізовано зручний механізм обробки станів: під час автентифікації кнопка стає неактивною й відображає напис "LOADING...". Після успішної автентифікації користувач автоматично перенаправляється на головну сторінку (Home.qml), що підвищує ефективність взаємодії.

Крім того, при кожному вході idToken не лише зберігається, а й використовується для забезпечення автентифікованого доступу до всіх ресурсів користувача. Таким чином, механізм автентифікації виступає ключовою ланкою інтеграції між користувацьким інтерфейсом та бекенд-логікою застосунку.

2. Головна сторінка

Головний екран цього додатку, реалізований із використанням QML, виконує функцію центральної панелі для користувача після запуску програми. Він надає індивідуалізоване привітання із відображенням імені й аватара користувача, а також оперативно демонструє актуальні погодні умови згідно з геолокацією.

Візуальна структура головної сторінки базується на вертикальному контейнері з центруванням основних елементів інтерфейсу. У верхній частині розміщено аватар, доступний для зміни користувачем, та динамічне текстове привітання, які отримуються з локального профілю чи бази даних.

Нижче знаходиться блок із погодною інформацією: місто та температура підтягуються після успішного отримання координат через клас LocationProvider (реалізований на C++), що ініціює модуль геолокації для визначення широти й довготи. Відповідні координати передаються через сигнали до WeatherManager, який здійснює HTTP-запит до зовнішнього погодного API (наприклад,

OpenWeatherMap), здійснює парсинг відповіді та передає релевантні дані у QML через сигнально-слотову систему Qt.

Оновлення інтерфейсу відбувається автоматично: зміни у властивостях C++-класів миттєво відображаються у відповідних QML-об’єктах без необхідності ручної синхронізації. Такий підхід забезпечує динамічну й безперебійну взаємодію користувача з додатком.

У нижній частині екрану розташовані навігаційні кнопки з інтерактивними іконками, що ведуть до різних функціональних розділів додатку (карта, чат, маршрути тощо). Переміщення між розділами реалізоване через компоненти StackView.

Отже, головна сторінка додатку виконує роль інтерактивної панелі, яка надає користувачеві персоналізовану інформацію й забезпечує швидкий доступ до ключових можливостей додатку, підтримуючи сучасні стандарти зручності та динамічності взаємодії.

3. Чат

Модуль чату реалізовано у вигляді окремої QML-сторінки під назвою `ChatPage.qml`, основна функція якої — забезпечити взаємодію користувача з вбудованим AI-асистентом для формування маршрутів. При завантаженні компонента, за умови ввімкненого режиму планування (`isPlanningMode`), автоматично викликається функція `sendPlanningQuestion(0)`. Вона ініціює перше питання з масиву `planningQuestions` і додає його до моделі повідомлень (`messageModel`). Зазначена модель реалізована на основі `ListModel` і виступає єдиним сховищем для всіх повідомлень у чаті — як тих, що надходять від користувача (`fromMe: true`), так і згенерованих системою (`fromMe: false`).

Функція `sendPlanningQuestion(index)` безпосередньо взаємодіє з `messageModel`, додаючи новий запис із відповідним текстом запитання та фіксуючи поточний індекс питання у змінній `currentQuestionIndex`. Коли користувач вводить відповідь у текстове поле і натискає кнопку відправлення,

обробник події `onClicked` додає це повідомлення до `messageModel`, очищує поле введення та, якщо активовано режим планування, зберігає відповідь у масиві `planningAnswers`. Далі, якщо існує наступне питання, викликається `sendPlanningQuestion(currentQuestionIndex + 1)`. Якщо ж питання закінчилися, формується фінальний prompt за допомогою методу `buildPlanningPrompt(questions, answers)`.

Метод `buildPlanningPrompt` функціонує як генератор: він компілює масив питань та відповідей у структурований текстовий запит для штучного інтелекту, включаючи JSON-шаблон із полями на кшталт `tripTitle`, `durationDays`, `budgetPerDayEUR`, `city`, `locations`, `days`. Цей запит передається до функції `sendPromptToAI(promptText)`, яка ініціює звернення до моделі з інструкцією щодо генерації маршруту, а далі використовує C++-метод `ChatManager::getItinerary(const QString &promptInfo)`.

Клас `ChatManager`, зареєстрований у QML-контексті як `chatManager`, здійснює HTTP POST-запит до Google Gemini API. У разі успішної відповіді результат обробляється через `QNetworkAccessManager`: витягується текст маршруту з JSON-структури, після чого емиться сигнал `promptSuccess(planText)`. У QML цей сигнал перехоплюється через блок `Connections`, і після отримання результату у `messageModel` додається повідомлення про успішне створення маршруту із закликом «click to view more». У разі виникнення помилки передається сигнал `promptFailed(error)`, який також обробляється у QML для логування чи інформування користувача.

Важливою складовою архітектури є передача об’єкта `FirestoreManager` у `ChatManager` через метод `setFirestoreManager(...)`. Після успішного отримання маршруту `ChatManager` викликає `firestoreManager.saveItinerary(planText)`, забезпечуючи збереження згенерованого маршруту у Firestore. Це дозволяє отримувати доступ до маршруту в інших частинах застосунку — наприклад, у розділах «Маршрути» або «Історія».

Узагальнюючи, модуль чату інтегрує QML-інтерфейс, сигнал-слотову взаємодію та C++-реалізацію мережевих операцій. Послідовність викликів між QML і C++ забезпечує послідовний обмін даними, автоматичне формування запиту до AI і збереження результатів у хмарній базі даних. Такий підхід дозволяє реалізувати інтелектуального асистента з мінімальною затримкою та високою гнучкістю налаштувань логіки взаємодії з користувачем.

4. Карта

Модуль карти, реалізований у MapPage.qml, ініціалізується через глобальний синґлтон AppState, який надає актуальні значення latitude та longitude. У разі, якщо масив coordinates порожній, центр карти встановлюється за цими координатами; якщо ж масив містить дані, фокус автоматично зміщується на першу точку маршруту.

У межах компонента використовується карта з плагіном OpenStreetMap. Обробники WheelHandler та DragHandler відповідають за масштабування й панорамування, забезпечуючи користувачу базову інтерактивність. Одразу після завантаження, у Component.onCompleted, для кожного елемента масиву coordinates динамічно створюється MapQuickItem (маркер із підписом), який додається до карти методом map.addMapItem(pin). На основі зібраних QtPositioning.coordinate(lat, lng) формується об’єкт MapPolyline, що графічно з’єднує всі маркери у єдиний маршрут. Функціонал, пов’язаний з цим працює лише при переході з сторінки маршрутів, де ми можемо продивится кожен маршрут на даній карті, пов’язаний цими пінами.

Для користувача передбачено дві кнопки ToolButton (“+” та “–”), розташовані у правому нижньому куті, які дозволяють змінювати значення map.zoomLevel у межах map.minimumZoomLevel та map.maximumZoomLevel. Такий підхід забезпечує базовий функціонал: автоматичне розміщення маркерів, побудову маршруту між ними та інтуїтивне управління масштабом карти.

5. Маршрути

Модуль «Маршрути» функціонує на основі QML-компонента ItineraryPage.qml, який активно взаємодіє з класами FirestoreManager та ImageManager. Після ініціалізації сторінки ініціюється виклик firestoreManager.getItineraries(), що надсилає відповідний запит до Firestore і повертає збережені маршрути користувача у форматі JSON.

Обробка отриманих даних здійснюється у функції onItinerariesLoaded(json). Через JSON.parse відбувається розбір вхідного рядка, після чого для кожного документа Firestore формується запис у моделі даних itinerariesModel із зазначенням ключових полів: tripTitle, city, durationDays, budgetPerDayEUR, createdAt, а також додається сирий масив daysRaw для зберігання структурованої інформації про дні маршруту. Паралельно викликається imageManager.fetchImageForCity(city), що синхронно отримує зображення міста та повертає відповідний URL для подальшого відображення.

Список маршрутів представлений у компонентах ListView, де кожен елемент (cardRect) відображає основну інформацію: назву маршруту, місто, тривалість та бюджет на день. При взаємодії користувача із карткою через MouseArea відбувається конвертація масиву daysRaw у формат, зручний для JavaScript, з використанням функції parseDaysArray(rawValues). Ця функція проходить по структурі даних, отриманій з Firestore, витягує номер дня, заголовок, список активностей із координатами кожної з них, а результат зберігається у властивість daysList.

Після парсингу відображається detailPopup, куди передаються tripTitle, city та daysList. Popup, використовуючи Repeater, демонструє дні маршруту та перелік активностей із зазначенням часу, назви та вартості. Кнопка «View Itinerary» для кожного дня закриває діалогове вікно та ініціює навігацію на сторінку карти:

navView.push("./Map.qml", { coordinates: modelData.coordinates })

де modelData.coordinates — масив об’єктів { name, lat, lng }, сформований під час попереднього парсингу даних.

Клас FirestoreManager, реалізований на C++, відповідає за формування правильних HTTP-запитів до Firestore, обробляє відповіді у методі onGetItinerariesReply() та емулює сигнал itinerariesLoaded.

Таким чином, модуль «Маршрути» забезпечує повноцінний цикл роботи з маршрутами: завантаження з хмарної бази, відображення у списку, детальний перегляд активностей та передачу координат до модуля карти для візуалізації. Така архітектура сприяє чіткій взаємодії між QML-інтерфейсом та бекенд-логікою, що, у свою чергу, дозволяє легко розширювати функціонал або інтегрувати додаткові сервіси.

6. База даних

Підсистема роботи з базою даних реалізована за допомогою класу FirestoreManager, який виконує функції збереження створених маршрутів та завантаження вже наявних із сервісу Firestore. Після проходження автентифікації кожен маршрут трансформується з внутрішньої структури JSON у формат Firestore Fields та зберігається у колекції users/{userId}/itineraries із унікальним ідентифікатором документа. Аналогічно, при зверненні до розділу «Маршрути» здійснюється GET-запит до відповідного шляху, а отриманий у відповідь сирий JSON-рядок передається у QML для подальшої обробки та відображення. Такий підхід забезпечує централізоване зберігання даних у хмарному середовищі, автоматизовану синхронізацію між сесіями користувача та надає можливість масштабування або міграції логіки збереження без необхідності зміни інтерфейсу.

3.4 Розробка інтерфейсу

У нашому проєкті інтерфейс розроблено на основі Qt Quick та QML, що неодноразово підкреслювалося в попередніх розділах. Дизайн витримано у мінімалістичному стилі: кожен екран має градієнтне тло, стримані кольорові

акценти та лаконічну типографіку. Всі сторінки, зокрема Login.qml та ItineraryPage.qml, структуровані за єдиною схемою, що сприяє однорідності вигляду й спрощує навігацію.

Для макетування використано систему anchors, а також компоненти RowLayout та ColumnLayout, що забезпечує коректне відображення інтерфейсу на різних розмірах екранів. Навігація між екранами реалізована через StackView.push(), тоді як обробка подій взаємодії з кнопками й полями вводу здійснюється безпосередньо у QML. Всі елементи інтерфейсу пов’язані з бекендом за допомогою об’єктів authManager, geo, weather, chatManager і firestoreManager, які зареєстровано у контексті QML. Це дозволяє оперативно оновлювати відображення інтерфейсу при зміні даних. Такий підхід поєднує високу швидкість прототипування, гнучкість інтерфейсу та спрощує подальшу підтримку проєкту.

3.5 Тестування роботи розробленого додатку

Для тестування додатку використовували Android-емулятор з архітектурою x86_64, через відсутність фізичного пристрою. Емулятор на x86 працює значно швидше завдяки апаратному прискоренню (наприклад, Intel HAXM, KVM) і не потребує емуляції ARM-інструкцій, що знижує навантаження на систему. Це дає змогу ефективно перевіряти інтерфейс, логіку роботи додатку та локаційні сервіси. Водночас варто зазначити, що фінальне тестування бажано проводити безпосередньо на ARM-пристроях, аби впевнитися у сумісності з реальним обладнанням.

Після відкриття додатку користувачеві одразу пропонується вибрати спосіб автентифікації. За замовчуванням з’являється форма для входу до системи, проте за потреби можна перейти на вкладку реєстрації для створення нового облікового запису. Користувач заповнює відповідні поля - електронну

пошту та пароль. Після цього система здійснює перевірку введених даних або, якщо це новий користувач, створює новий профіль.

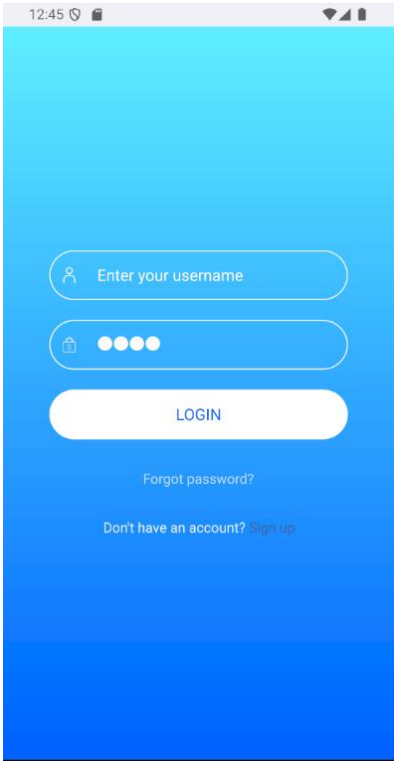

Рисунок 3.2 – форма автентифікації в додаток

Далі, після успішної автентифікації, переходимо на головну сторінку де одразу на екрані з’являється Popup-вікно з запитом на дозвіл до інформації про геолокацію на данному пристрої. Ми повинні надати дозвіл для покращення користувацького досвіду у ході тестування.

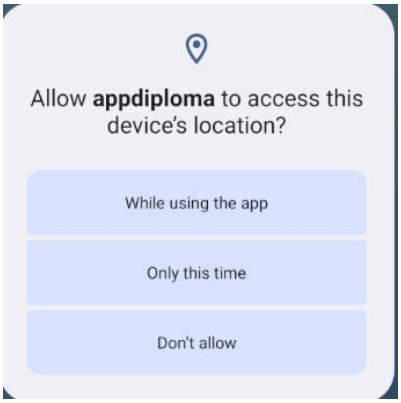

Рисунок 3.3 – запит на дозвіл поширення геолокації

Після підтвердження запиту переходимо на головну сторінку, де ми можемо побачити актуальну погоду в верхньому елементі та кнопку переходу в профіль користувача, CTA-кнопку для генерації маршруту за допомогою ШІ в головному компоненті з Hero-текстом, який вітаж користувача і призиває до дії. Також знизу екрана можна побачити footer-елемент з 3 кнопками: перехід до інтерактивної карти, чат, або ж перехід до сторінки маршрутів користувача.

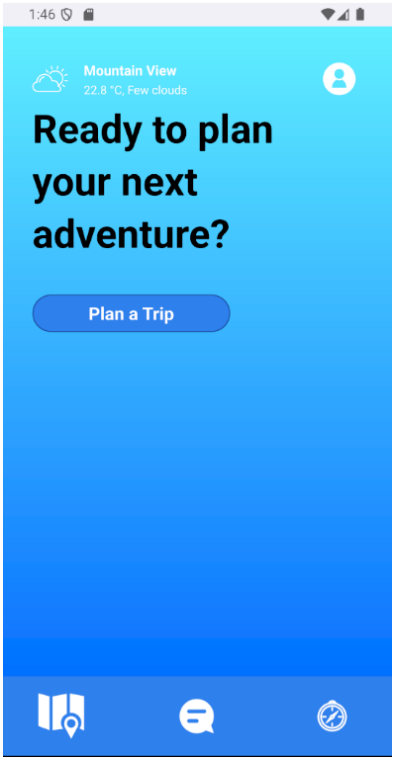

Рисунок 3.4 – головна сторінка

При натиснені на CTA-кнопку в головному компоненті, переходимо у чат з AI-асистентом в режимі заготовлених питань.

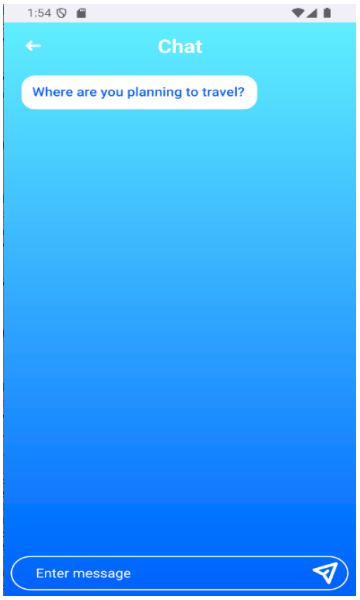

Рисунок 3.5 – чат с AI-асистентом

Відповідаємо на заготовлені питання для генерації маршруту, для тесту наша ціль – це подорож до міста Atlanta на 3 дні, там нас будуть цікавити музеї і наш денний бюджет – сто євро.

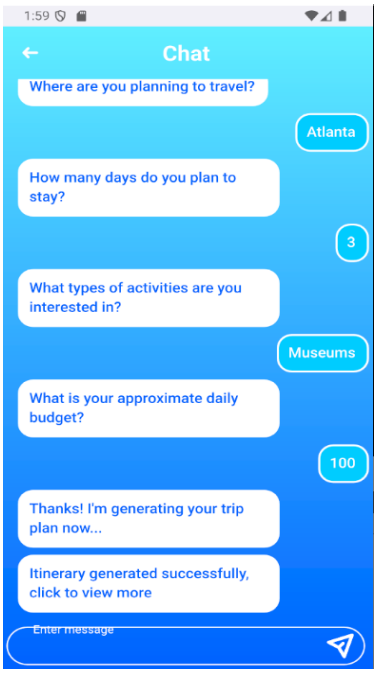

Рисунок 3.6 – листування з AI-асистентом

Після невеликої затримки, наш маршрут успішно згенеровано про що свідчить останнє повідомлення, тепер ми можемо або ж натиснути на останнє повідомлення, щоб перейти до нашого згенерованого маршруту та переглянути його або ж ми можемо повернутись на головну сторінку і натиснути кнопку маршрути.

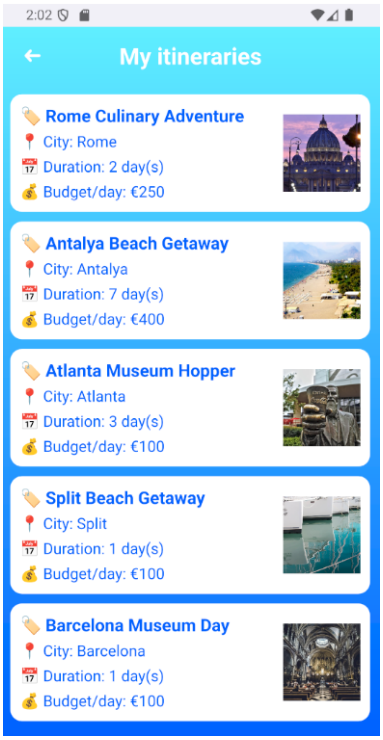

Рисунок 3.7 – маршрути користувача

Тепер ми бачимо наш маршрут в списку (інші маршрути були сгенеровані іншими тестами). Кожен елемент списку має привабливу назву, вказане місто, денний бюджет, тривалість подорожі та зображення вказаного міста. Ми можемо натиснути на необхідний маршрут, щоб дізнатись детальну інформацію про нього. Ця дія призведе до відкриття Popup-вікна поверх нашого екрану.

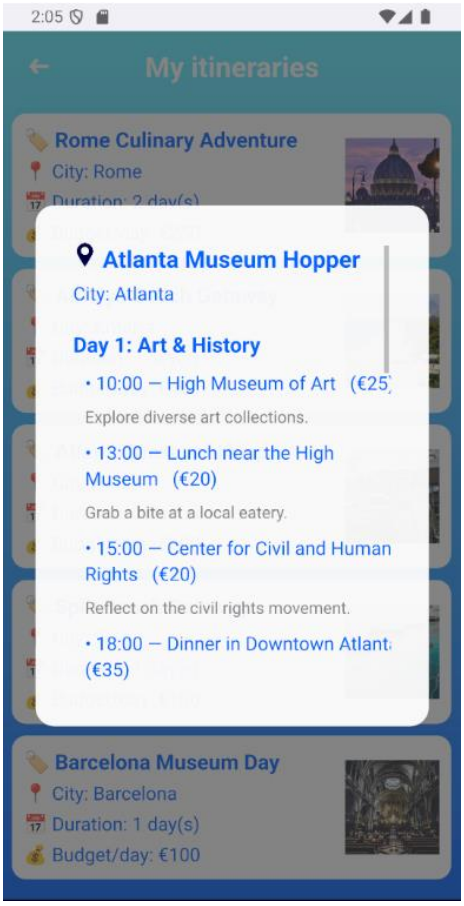

Рисунок 3.8 – детальна інформація маршруту

У цьому вікні за допомогою скролу ми можемо переглянути ввесь маршрут по дням. Детальний опис кожного дня з різними активностями погодинно та з вказаними цінами на витрати в кожній точці активності. Також можна відобразити цілий маршрут певного дня на інтерактивній карті, де будуть намальовані так званні “піни” з'єднанні лініями. Щоб перейти до карти натиснемо кнопку “View Itinerary”. Варто зазначити, що на перший погляд AI-ассистент може згенерувати активності з однаковими даними широти і довготи, це відбувається у випадку однакового місця для 2 активностей, наприклад, ресторан поряд з музеєм. Для того щоб на “піни” та їх підписи не злипались в кучу, було прийнято рішення фільтрувати однакові пари координат. Такий підхід дасть чітке

відображення маршруту і не зіпсує досвід користування додатком нашого мандрівника.

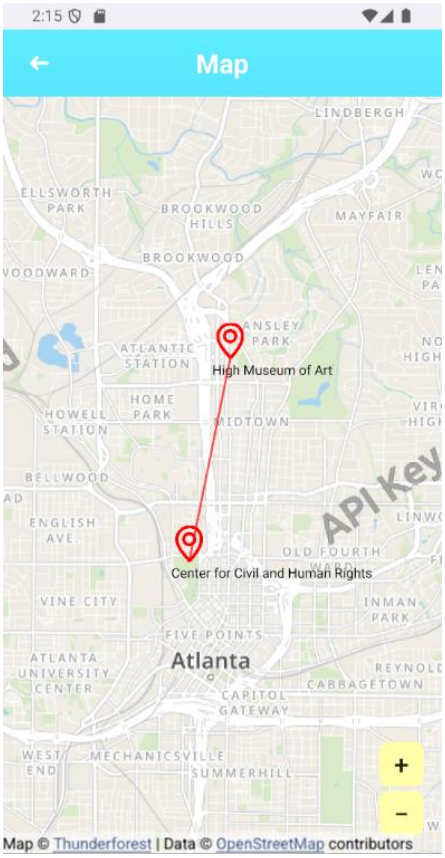

Рисунок 3.9 – інтерактивна карта з “пінами”

Підсумовуючи, розроблений додаток повністю відповідає описаним функціональним можливостям, необхідним для налагодженного користування мандрівника, а саме: автентифікацію, чат з AI-асистентом, генерацію маршрутів, інтерактивну карту для взаємодії та відображення згенерованих маршрутів.

Проведемо порівняння функціональних можливостей мобільних додатків туристичної сфери(табл. 3.1)

Таблиця 3.1 – Аналіз функціональних можливостей додатків туристичної сфери

	Автентифікація та авторизація	Інтерактивна мапа	Зручний користувацький інтерфейс	Впровадження ШІ	Індивідуальний підхід до кожного користувача
Sygic Travel	+	+	+	-	-
Roadtrippers	+	+	+	-	-
Tripadvisor	+	+	+	-	-
Розроблене програмне забезпечення	+	+	+	+	+

З-поміж наявних сервісів цей додаток демонструє помітну перевагу завдяки інтеграції штучного інтелекту, що забезпечує персоналізовану генерацію маршрутів і дає змогу здійснювати інтерактивне спілкування з AI-асистентом у чаті. На відміну від аналогічних рішень, тут користувач отримує можливість глибшої взаємодії: рекомендації формуються оперативно, з урахуванням індивідуальних вподобань і потреб мандрівника, що суттєво підвищує якість користувацького досвіду.

3.6 Висновок до розділу 3

У третьому розділі детально обґрунтовано вибір технологічного стеку для розробки мобільного туристичного AI-асистента, а також визначено основні етапи створення системи. Клієнтську частину реалізовано на C++ із використанням Android NDK, що забезпечує високу продуктивність, стабільну роботу з апаратними сенсорами й ефективну мережеву взаємодію через REST API. Для локалізації інтерфейсу залучаються Java-модулі та TypeScript, що підвищує гнучкість і масштабованість рішення.

Вибір платформи зумовлений великою аудиторією користувачів Android, а також гнучкістю SDK/NDK, що дозволяє уникати обмежень інших операційних

систем, зокрема iOS/macOS, і фокусуватися на масовому ринку. Розробка здійснюється у середовищі Qt Creator з використанням Qt Quick/QML, що забезпечує кросплатформенний інтерфейс і зручні інструменти для проєктування й налагодження, а також якісну інтеграцію з Android SDK/NDK.

Архітектура системи є модульною: окремо реалізовані модулі автентифікації, чату, маршрутів, картографії та бази даних. Взаємодія між модулями здійснюється за допомогою сигналів і слотів Qt, а також обгорток на C++ для HTTP-запитів і кешування, що сприяє підвищенню масштабованості й спрощує підтримку проєкту.

У підсумку створено систему з високою продуктивністю, потужними AI-сервісами й зручним інтерфейсом, готову до розширення на інші платформи і ефективної роботи з мультимедіа, геолокацією та зовнішніми API.

ВИСНОВКИ

Усі поставлені завдання для цієї бакалаврської кваліфікаційної роботи виконані в повному обсязі. Проведено детальний аналіз сучасного ринку мобільних застосунків для туристів, що використовують функції штучного інтелекту, автоматичну генерацію маршрутів та інтерактивну навігацію. Особливу увагу приділено основним функціональним можливостям, а також перевагам і недолікам популярних платформ, таких як Booking, TripAdvisor, Sygic Travel та TripIt. У ході дослідження виявлені ключові проблеми користувацького досвіду: обмежена персоналізація маршрутів, недостатня інтеграція чат-асистентів і складність взаємодії з картографічними сервісами.

Виконано порівняльний аналіз потенційних технологічних рішень для розробки мобільного AI-асистента. Розглянуто декілька архітектурних підходів і стеків технологій. На основі цього аналізу обрано оптимальне рішення, що включає використання C++ з Android NDK, Qt Quick/QML для розробки користувацького інтерфейсу та REST API для інтеграції із серверними AI-модулями. Такий підхід забезпечує високу продуктивність, стабільність роботи застосунку, а також дає можливість масштабування та кросплатформенності.

Розробка мобільного застосунку передбачає чітку структуризацію його основних компонентів: модуль автентифікації, чат-бот, генератор маршрутів, інтерактивна карта та база даних. Взаємодія між цими модулями здійснюється за допомогою сигнально-слотової системи Qt та REST-запитів, що забезпечує гнучкість архітектури, модульність і спрощує подальший розвиток програмного продукту.

Для кожної з ключових функцій застосунку розроблено відповідні алгоритми. Зокрема, впроваджено логіку генерації персоналізованих маршрутів на основі запитів користувача, а також механізми інтерактивної роботи з картою.

Окремо реалізовано чат з AI-асистентом, який підтримує природний діалог та сприяє ефективному плануванню подорожей у реальному часі.

У підсумку було реалізовано повнофункціональний прототип мобільного застосунку з інтуїтивно зрозумілим інтерфейсом. Користувач має змогу швидко пройти авторизацію, вести діалог із AI-асистентом, отримувати персоналізовані маршрути та взаємодіяти з інтерактивною картою.

Здійснено комплексне тестування застосунку, що підтвердило коректність реалізації всіх модулів, стабільність роботи, високу швидкодію та зручність користування. Результати тестування свідчать про перевагу застосунку за функціональністю та юзабіліті у порівнянні з низкою популярних аналогів, зокрема завдяки гнучкій персоналізації та інтерактивним можливостям додано використання AI-асистента, а також реалізований індивідуальний підхід до кожного користувача.

Отже, усі задачі бакалаврського дослідження виконано, мету роботи досягнуто.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Використання сучасних мобільних додатків у туристичній галузі[Електронний ресурс] – Режим доступу: https://social-science.uu.edu.ua/article/1419 (дата звернення: 20.05.2025).
2. Мобільні додатки як сучасний інструмент інноваційного розвитку в сфері туризму[Електронний ресурс] – Режим доступу: https://economyandsociety.in.ua/index.php/journal/article/download/4955/4898 (дата звернення: 20.05.2025).
3. Топ-10 обов’язкових туристичних додатків для завзятих мандрівників[Електронний ресурс] – Режим доступу: https://blog.lebara.co.uk/uk/топ-10-обов’язкових-туристичних-додатків/ (дата звернення 20.05.2025).
4. Gretzel, U. et al. (2015). Smart tourism: Foundations and developments[Електронний ресурс] – Режим доступу: https://www.researchgate.net/publication/280719315_Smart_tourism_foundations_and_developments (дата звернення 21.05.2025).
5. Three ways artificial intelligence is improving travel[Електронний ресурс] – Режим доступу: https://www.dailytelegraph.com.au/lifestyle/three-ways-artificial-intelligence-is-improving-travel/news-story (дата звернення 21.05.2025).
6. AI Travel Apps: The Tools Redefining How We Plan Our Trips[Електронний ресурс] – Режим доступу: https://www.lifewire.com/ai-travel-apps-11679719 (дата звернення 21.06.2025).
7. Аутентифікація і авторизація: що це і в чому відмінність[Електронний ресурс] – Режим доступу: https://qagroup.com.ua/publications (дата звернення 22.05.2025).

8. Firebase як бекенд для будь-яких застосунків, та як використовувати Firebase-сервіси[Електронний ресурс] – Режим доступу: https://dou.ua/forums/topic/44058/ (дата звернення 22.05.2025).
9. Основні принципи візуального дизайну в UX[Електронний ресурс] – Режим доступу: https://blog.ithillel.ua/articles/basic-principles-of-visual-design-in-ux (дата звернення 22.05.2025).
10. Мікросервісна архітектура[Електронний ресурс] – Режим доступу: https://medium.com/ (дата звернення 22.05.2025).
11. Блок-схема (Flowchart)[Електронний ресурс] – Режим доступу: https://www.maxzosim.com/blok-skhema/ (дата звернення 22.05.2025).
12. Мова програмування C++[Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/ (дата звернення 23.05.2025).
13. Qt[Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Qt (дата звернення 23.05.2025).
14. Qt for Android[Електронний ресурс] – Режим доступу: https://felgo.com/doc/qt/android/ (дата звернення 23.05.2025).
15. QML[Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/QML (дата звернення 23.05.2025).
16. Qt Creator[Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Qt_Creator (дата звернення 23.05.2025).
17. Using Visual Studio Code for Writing Qt Applications[Електронний ресурс] – Режим доступу: https://www.kdab.com/using-visual-studio-code-for-writing-qt-applications/ (дата звернення 23.05.2025).
18. QML vs Qt Widgets – detailed comparison[Електронний ресурс] – Режим доступу: https://scythe-studio.com/en/blog/qml-vs-qt-widgets-detailed-comparison (дата звернення 23.05.2025).
19. Qt documentation[Електронний ресурс] – Режим доступу: https://doc.qt.io/ (дата звернення 23.05.2025).

ДОДАТОК А (обов'язковий)
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: Мобільний туристичний AI-асистент

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

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 1,48 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

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

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

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

Експертна комісія:

Яровий А.А., зав. каф. КН _____ _____
(прізвище, ініціали, посада) (підпис)

Колесницький О.К., проф. каф КН _____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Озеранський В.С.
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Колодний В.В.
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Рудий Р.С.
(підпис) (прізвище, ініціали)

ДОДАТОК Б – ІНСТРУКЦІЯ КОРИСТУВАЧА МОБІЛЬНОГО ДОДАТКУ ТУРИСТИЧНИЙ AI-АСИСТЕНТ

Запустити мобільний додаток. Пройти автентифікацію або ж зареєструватись (відповідний UX для цього приведенний). Для реєстрації потрібно нажати на відповідну кнопку «Sign up», що знаходиться внизу основних елементів.

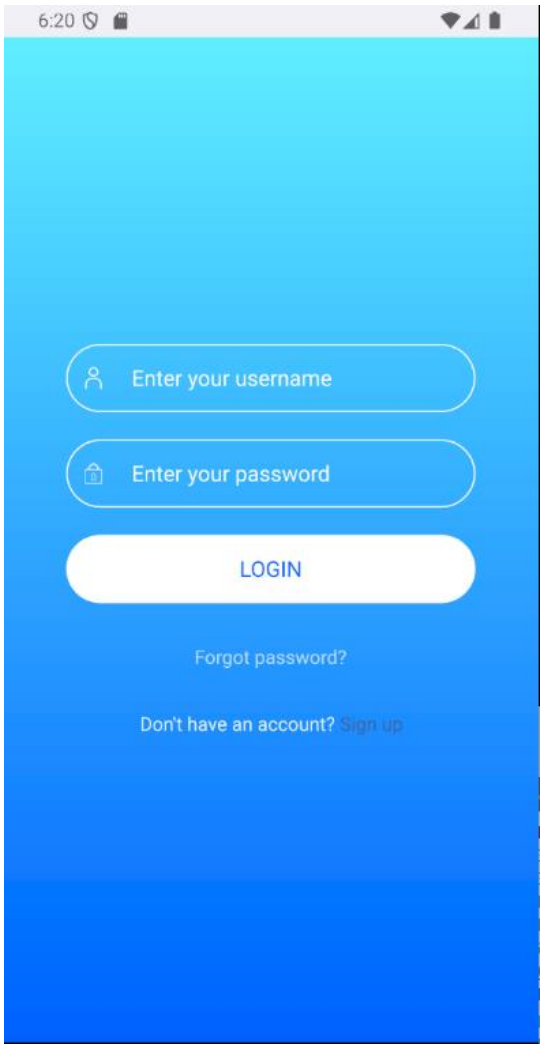

Рисунок Б.1 – Сторінка автентифікації

Далі, після успішної автентифікації, переходимо на головну сторінку де одразу на екрані з’являється Popup-вікно з запитом на дозвіл до інформації про

геолокацію на данному пристрої. Ми повинні надати дозвіл для покращення користувацького досвіду у ході тестування.

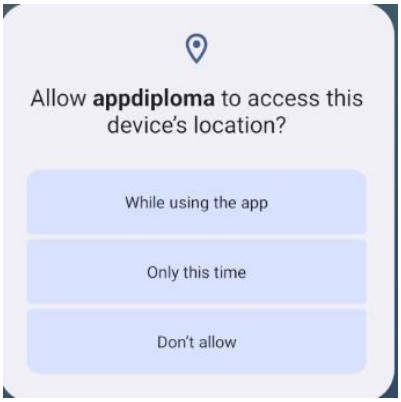

Рисунок Б.2 – Запит на дозвіл поширення геолокації

Щоб згенерувати маршрут потрібно перейти на сторінку чата з головного меню, відповісти на питання і перейти на сторінку маршрутів і переглянути його.

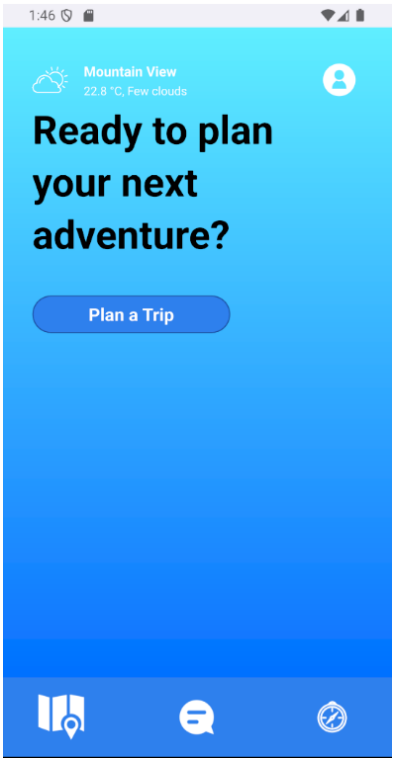

Рисунок Б.3 – Головна сторінка

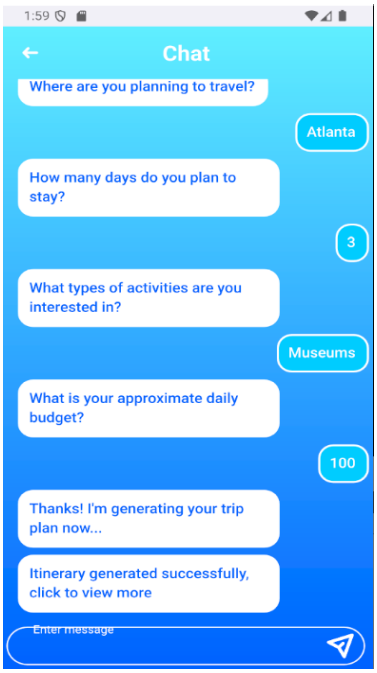

Рисунок Б.4 – листування з AI-асистентом

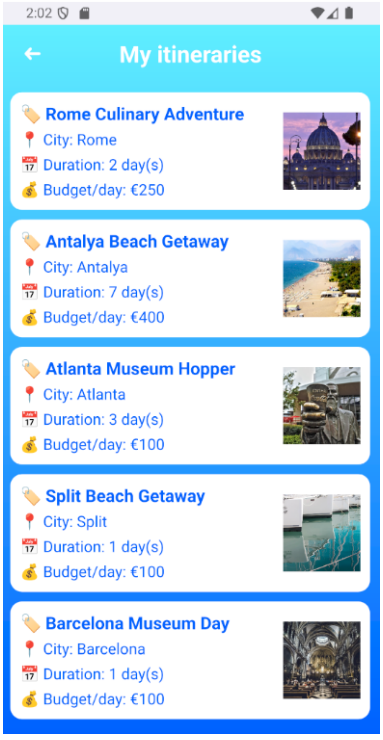

Рисунок Б.5 – маршрути користувача

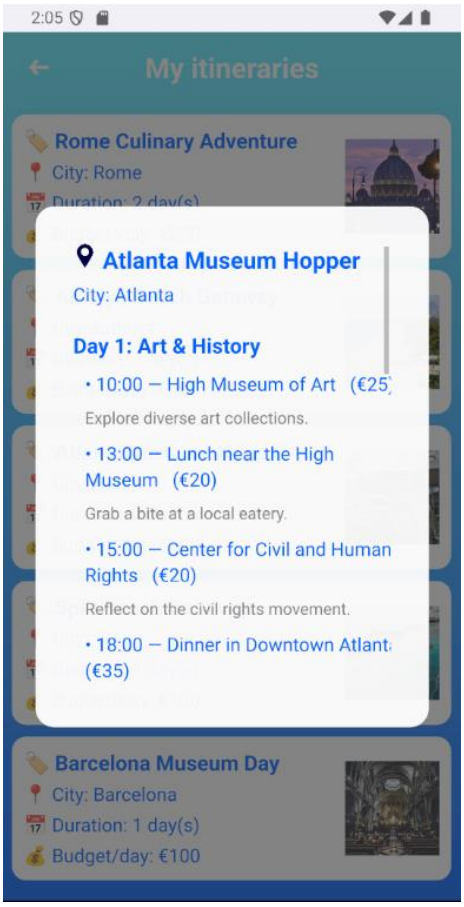

Рисунок Б.6 – детальна інформація маршруту

Також після перегляду новозгенерованного маршруту і ознайомлення з детальною інформацією про нього, можна переглянути маршрут кожного дня на інтерактивній карті, де кожна активність позначена як іконка маркера на карті, кожен маркер підписанний відповідною активністю та маркери з'єднанні лініями у правильній послідовності. Іноді, геолокація, тобто координати активностей еквівалентні, що не дозволяє додати 2 або більше маркера на карті в одну точку, що суперечить правилам UI/UX та унеможливлює коректне відображення. Через таку проблема, було прийнято рішення фільтрувати ці активності на мапі, та показувати тільки недублікати-маркери. Також AI-асистент може давати однакові координати через реальну збіжність між координатами активностей (приклад з рестораном та музеєм в одному місці).

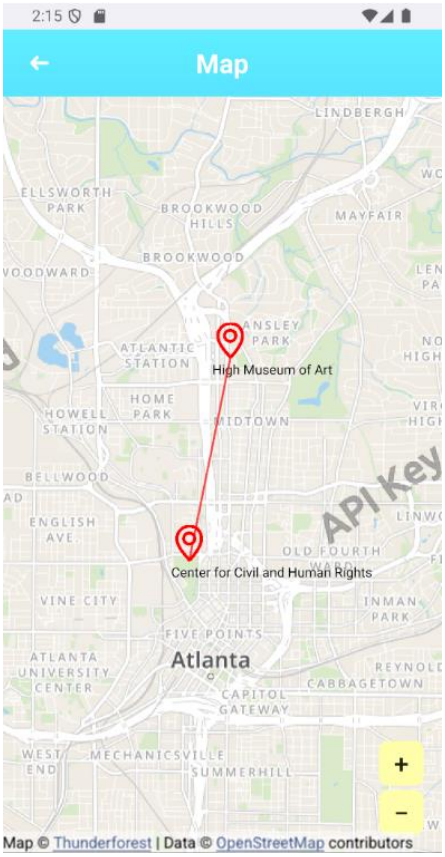

Рисунок Б.7 – інтерактивна карта з “пінами”

ДОДАТОК В – ЛІСТИНГ ПРОГРАМИ

main.cpp

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlContext>
#include "src/firebaseauthmanager.h"
#include "src/locationprovider.h"
#include "src/weathermanager.h"
#include <QQuickWindow>
#include "src/chatmanager.h"
#include "src/firestoremanager.h"
#include "src/imagemanager.h"
#include <QGeoServiceProvider>
#include <QDebug>
int main(int argc, char *argv[])
{
    QGuiApplication app(argc, argv);
    QQmlApplicationEngine engine;
    auto geoProvider = new LocationProvider(&app);
    auto weather = new WeatherManager(&app);
    qmlRegisterSingletonType(QUrl("qrc:/qt/qml/diploma/pages/AppState.qml"),
"diploma", 1, 0, "AppState");
    FirebaseAuthManager authManager;
    ChatManager chatManager;
    FirestoreManager firestoreManager;
    ImageManager imageManager;
    engine.rootContext()->setContextProperty("weather", weather);
    engine.rootContext()->setContextProperty("geo", geoProvider);
    engine.rootContext()->setContextProperty("authManager", &authManager);
    engine.rootContext()->setContextProperty("chatManager", &chatManager);
    engine.rootContext()->setContextProperty("firestoreManager",
&firestoreManager);
    engine.rootContext()->setContextProperty("imageManager", &imageManager);
    QObject::connect(&authManager, &FirebaseAuthManager::authSuccess,
                     [&](const QString &idToken) {
                         QString userId = authManager.currentUserId();
                         firestoreManager.setUserId(userId);
                         firestoreManager.setIdToken(idToken);
                         chatManager.setFirestoreManager(&firestoreManager);
                     });
    QObject::connect(
        &engine,
        &QQmlApplicationEngine::objectCreationFailed,
        &app,
        []() { QCoreApplication::exit(-1); },
```

```
        Qt::QueuedConnection);
    engine.loadFromModule("diploma", "Main");
    QObject *root = engine.rootObjects().first();
    QQuickWindow *window = qobject_cast<QQuickWindow *>(root);
    if (!window) {
        qWarning() << "Root object is not a QQuickWindow";
    } else {
        window->setFlags(window->flags() & ~Qt::WindowFullscreenButtonHint &
~Qt::FramelessWindowHint);
        if (window->visibility() == QWindow::FullScreen) {
            window->setVisibility(QWindow::Windowed);
        }
    }
    return app.exec();
}
```

chatmanager.cpp

```
#include "chatmanager.h"
#include <QNetworkRequest>
#include <QJsonDocument>
#include <QJsonObject>
#include <QJsonArray>
#include <QNetworkAccessManager>
#include <QNetworkReply>

ChatManager::ChatManager(QObject *parent) : QObject{parent} {}

void ChatManager::setFirestoreManager(FirestoreManager* firestoreManager)
{
    m_firestoreManager = firestoreManager;
}

void ChatManager::getItinerary(const QString &promptInfo) {
    if (promptInfo.isEmpty()) {
        emit promptFailed("Prompt body is empty");
        return;
    }

    QString url =
QString("https://generativelanguage.googleapis.com/v1beta/models/gemini-2.0-
flash:generateContent?key=AIzaSyCTPoR-piEOafebzAex6rJT9VQWrCJ5L3s");
    QNetworkRequest request((QUrl(url)));
    request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");

    QJsonObject textObj;
    textObj["text"] = promptInfo;
```

```
    QJsonObject partObj;
    partObj["parts"] = QJsonArray{ textObj };

    QJsonObject json;
    json["contents"] = QJsonArray{ partObj };

    QNetworkAccessManager* manager = new QNetworkAccessManager(this);
    QNetworkReply* reply = manager->post(request, QJsonDocument(json).toJson());

    connect(reply, &QNetworkReply::finished, this, [this, reply]() {
        if (reply->error() != QNetworkReply::NoError) {
            emit promptFailed(reply->errorString());
            reply->deleteLater();
            return;
        }

        QByteArray responseData = reply->readAll();
        QJsonDocument doc = QJsonDocument::fromJson(responseData);
        if (!doc.isObject()) {
            emit promptFailed("Invalid JSON response");
            reply->deleteLater();
            return;
        }

        QJsonObject root = doc.object();
        QString planText;

        if (root.contains("candidates")) {
            QJsonArray candidates = root["candidates"].toArray();
            if (!candidates.isEmpty()) {
                QJsonObject contentObj =
candidates.first().toObject()["content"].toObject();
                QJsonArray parts = contentObj["parts"].toArray();
                if (!parts.isEmpty()) {
                    planText = parts.first().toObject()["text"].toString();
                }
            }
        }

        if (planText.isEmpty()) {
            emit promptFailed("Empty plan text");
        } else {
            emit promptSuccess(planText);

            QJsonObject itinerary;
            itinerary["text"] = planText;
```

```
            if (this->m_firestoreManager) {
                m_firestoreManager->saveItinerary(planText);
            }
        }

        reply->deleteLater();
    });
}
```

chatmanager.h

```
#ifndef CHATMANAGER_H
#define CHATMANAGER_H
#include "src/firestoremanager.h"
#include <QObject>

class ChatManager : public QObject
{
    Q_OBJECT
public:
    explicit ChatManager(QObject *parent = nullptr);
    void setFirestoreManager(FirestoreManager* firestoreManager);
    Q_INVOKABLE void getItinerary(const QString &promptInfo);

signals:
    void promptSuccess(const QString &message);
    void promptFailed(const QString &errorMessage);

private:
    FirestoreManager* m_firestoreManager = nullptr;
};


#endif // CHATMANAGER_H
```

firebaseauthmanager.cpp

```
#include "firebaseauthmanager.h"
#include <QJsonDocument>
#include <QJsonObject>
#include <QNetworkRequest>

FirebaseAuthManager::FirebaseAuthManager(QObject *parent) : QObject(parent) {}

QString FirebaseAuthManager::currentUserId() const {
```

```
    return m_userId;
}

void FirebaseAuthManager::signUp(const QString &email, const QString &password) {
    if (email.isEmpty() || password.isEmpty()) {
        emit authFailed("Email and password must not be empty");
        return;
    }
    QString url =
QString("https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=%1").arg(m_a
piKey);

    QNetworkRequest request((QUrl(url)));
    request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");
    QJsonObject json;
    json["email"] = email;
    json["password"] = password;
    json["returnSecureToken"] = true;
    QNetworkReply *reply = m_networkManager.post(request,
QJsonDocument(json).toJson());
    connect(reply, &QNetworkReply::finished, [this, reply]() { handleReply(reply);
});
}

void FirebaseAuthManager::signIn(const QString &email, const QString &password) {
    if (email.isEmpty() || password.isEmpty()) {
        emit authFailed("Email and password must not be empty");
        return;
    }
    QString url =
QString("https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=
%1").arg(m_apiKey);

    QNetworkRequest request((QUrl(url)));
    request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");

    QJsonObject json;
    json["email"] = email;
    json["password"] = password;
    json["returnSecureToken"] = true;

    QNetworkReply *reply = m_networkManager.post(request,
QJsonDocument(json).toJson());
    connect(reply, &QNetworkReply::finished, [this, reply]() { handleReply(reply);
});
}
```

```
void FirebaseAuthManager::handleReply(QNetworkReply *reply) {
    QByteArray data = reply->readAll();
    qDebug() << "Reply data:" << data;

    QJsonDocument jsonDoc = QJsonDocument::fromJson(data);
    QJsonObject obj = jsonDoc.object();

    if (reply->error() == QNetworkReply::NoError) {
        QString idToken = obj.value("idToken").toString();
        m_userId = obj.value("localId").toString();
        emit authSuccess(idToken);
    } else {
        QString error = obj.value("error").toObject().value("message").toString();
        if (error.isEmpty())
            error = "Unknown error from server";
        emit authFailed(error);
    }

    reply->deleteLater();
}
```

firebaseauthmanager.h

```
#ifndef FIREBASEAUTHMANAGER_H
#define FIREBASEAUTHMANAGER_H

#include <QObject>
#include <QNetworkAccessManager>
#include <QNetworkReply>

class FirebaseAuthManager : public QObject
{
    Q_OBJECT
public:
    explicit FirebaseAuthManager(QObject *parent = nullptr);

    Q_INVOKABLE void signUp(const QString &email, const QString &password);
    Q_INVOKABLE void signIn(const QString &email, const QString &password);
    QString currentUserId() const;

signals:
    void authSuccess(const QString &idtoken);
    void authFailed(const QString &errorMessage);

private:
    QNetworkAccessManager m_networkManager;
```

```
    const QString m_apiKey = "AIzaSyBCm6kQ32m0oCa6G9jtct9jysrbq5E2OD4";
    QString m_userId;
    void handleReply(QNetworkReply *reply);
};


#endif // FIREBASEAUTHMANAGER_H
```

firestoremanager.cpp

```
#include "firestoremanager.h"
#include <QNetworkRequest>
#include <QNetworkReply>
#include <QJsonDocument>
#include <QJsonObject>
#include <QJsonArray>
#include <QDateTime>
#include <QUuid>
#include <QDebug>
#include <QEventLoop>

FirestoreManager::FirestoreManager(QObject *parent) : QObject(parent)
{
}

void FirestoreManager::getItineraries() {
    if (m_userId.isEmpty() || m_idToken.isEmpty()) {
        qWarning() << "getItineraries: User ID or ID token is empty";
        emit itinerariesLoaded("Erororororor");
        return;
    }

    QString urlStr = QString(
                         "https://firestore.googleapis.com/v1/projects/%1/databases
/(default)/documents/users/%2/itineraries")
                         .arg("mobileaiassist")
                         .arg(m_userId);

    QNetworkRequest request((QUrl(urlStr)));
    request.setRawHeader("Authorization", QString("Bearer
%1").arg(m_idToken).toUtf8());
    request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");

    QNetworkReply *reply = m_networkManager.get(request);
    connect(reply, &QNetworkReply::finished, this, [this, reply]() {
```

```
        onGetItinerariesReply(reply);
    });
}

void FirestoreManager::setUserId(const QString &userId)
{
    m_userId = userId;
}

void FirestoreManager::setIdToken(const QString &token)
{
    m_idToken = token;
}

void FirestoreManager::saveItinerary(const QString &replyPayload)
{
    if (m_userId.isEmpty() || m_idToken.isEmpty()) {
        emit saveFailed("User ID or ID token is empty");
        return;
    }

    QString rawText = replyPayload;
    const QString codeFenceStart = "```json\n";
    const QString codeFenceEnd   = "\n```";

    if (rawText.startsWith(codeFenceStart)) {
        rawText.remove(0, codeFenceStart.length());
    }
    if (rawText.endsWith(codeFenceEnd)) {
        rawText.chop(codeFenceEnd.length());
    }
    rawText = rawText.trimmed();

    if (rawText.startsWith("```")) {
        rawText.remove(0, 3);
    }
    if (rawText.endsWith("```")) {
        rawText.chop(3);
    }
    rawText = rawText.trimmed();

    qDebug() << "Cleaned JSON payload (first 200 chars):" << rawText.left(200) <<
"...";

    QJsonParseError itineraryError;
    QJsonDocument itineraryDoc = QJsonDocument::fromJson(rawText.toUtf8(),
&itineraryError);
```

```
    if (itineraryError.error != QJsonParseError::NoError) {
        qWarning() << "Failed to parse cleaned itinerary JSON:" <<
itineraryError.errorString();
        emit saveFailed(QString("Failed to parse itinerary JSON:
%1").arg(itineraryError.errorString()));
        return;
    }
    QJsonObject itineraryObj = itineraryDoc.object();

    QString title = itineraryObj.value("tripTitle").toString("Untitled Trip");

    QString itineraryId = QUuid::createUuid().toString(QUuid::WithoutBraces);

    std::function<QJsonObject(const QJsonValue&)> toFirestoreField = [&](const
QJsonValue &value) -> QJsonObject {
        if (value.isString()) {
            return QJsonObject{{"stringValue", value.toString()}};
        }
        if (value.isDouble()) {
            double num = value.toDouble();
            if (std::floor(num) == num) {
                // integerValue
                return QJsonObject{{"integerValue",
QString::number(static_cast<qint64>(num))}};
            } else {
                // doubleValue
                return QJsonObject{{"doubleValue", QString::number(num, 'g', 10)}};
            }
        }
        if (value.isBool()) {
            return QJsonObject{{"booleanValue", value.toBool()}};
        }
        if (value.isArray()) {
            QJsonArray arr = value.toArray();

            QJsonArray firestoreValues;
            for (const QJsonValue &item : arr) {
                firestoreValues.append(toFirestoreField(item));
            }
            return QJsonObject{{"arrayValue", QJsonObject{{"values",
firestoreValues}}}};
        }
        if (value.isObject()) {
            QJsonObject obj = value.toObject();
            QJsonObject mapFields;
            for (auto it = obj.begin(); it != obj.end(); ++it) {
                mapFields[it.key()] = toFirestoreField(it.value());
```

```
            }
            return QJsonObject{{"mapValue", QJsonObject{{"fields", mapFields}}}};
        }
        if (value.isNull()) {
            return QJsonObject{{"nullValue", QJsonValue::Null}};
        }

        return QJsonObject{{"mapValue", QJsonObject{{"fields", QJsonObject{}}}}};
    };

    QJsonObject firestoreFields = toFirestoreField(itineraryObj)
                                      .value("mapValue")
                                      .toObject()
                                      .value("fields")
                                      .toObject();

    firestoreFields["createdAt"] = QJsonObject{
        {"timestampValue", QDateTime::currentDateTimeUtc().toString(Qt::ISODate)}
    };

    QJsonObject root;
    root["fields"] = firestoreFields;

    QJsonDocument doc(root);
    QByteArray data = doc.toJson();

    QString urlStr = QString(
                         "https://firestore.googleapis.com/v1/projects/%1/databases
/(default)/documents/users/%2/itineraries?documentId=%3")
                         .arg("mobileaiassist")
                         .arg(m_userId)
                         .arg(itineraryId);

    QNetworkRequest request((QUrl(urlStr)));
    request.setRawHeader("Authorization", QString("Bearer
%1").arg(m_idToken).toUtf8());
    request.setHeader(QNetworkRequest::ContentTypeHeader, "application/json");

    QNetworkReply *reply = m_networkManager.post(request, data);
    connect(reply, &QNetworkReply::finished, this, [this, reply]() {
        onNetworkReply(reply);
    });
}

void FirestoreManager::onNetworkReply(QNetworkReply *reply)
{
    QByteArray responseBody = reply->readAll();
```

```
    if (reply->error() == QNetworkReply::NoError) {
        qDebug() << "Firestore save succeeded, response body:" << responseBody;
        emit saveSuccess();
    } else {
        QString errString = reply->errorString();
        qWarning() << "Firestore save error:" << errString;
        qWarning() << "Response body from Firestore:" << responseBody;
        emit saveFailed(errString);
    }
    reply->deleteLater();
}

void FirestoreManager::onGetItinerariesReply(QNetworkReply *reply)
{
    QByteArray responseBytes = reply->readAll();
    reply->deleteLater();

    if (reply->error() != QNetworkReply::NoError) {
        emit itinerariesLoaded("");
        return;
    }

    QString jsonStr = QString::fromUtf8(responseBytes);
    emit itinerariesLoaded(jsonStr);
}
```

firestoremanager.h

```
#ifndef FIRESTOREMANAGER_H
#define FIRESTOREMANAGER_H

#include <QObject>
#include <QNetworkAccessManager>
#include <QJsonArray>
class FirestoreManager : public QObject
{
    Q_OBJECT
public:
    explicit FirestoreManager(QObject *parent = nullptr);

    void setUserId(const QString &userId);
    void setIdToken(const QString &token);

    void saveItinerary(const QString &jsonItinerary);
    Q_INVOKABLE void getItineraries();
```

```
    void onGetItinerariesReply(QNetworkReply *reply);

signals:
    void saveSuccess();
    void saveFailed(const QString &error);
    void itinerariesLoaded(const QString &json);

private slots:
    void onNetworkReply(QNetworkReply *reply);

private:
    QNetworkAccessManager m_networkManager;
    QString m_userId;
    QString m_idToken;
};

#endif // FIRESTOREMANAGER_H
```

imagemanager.cpp

```
#include "imagemanager.h"
#include <QUrl>
#include <QUrlQuery>
#include <QJsonDocument>
#include <QJsonObject>
#include <QJsonArray>
#include <QDebug>
#include <QEventLoop>
#include <QNetworkAccessManager>
#include <QNetworkRequest>
#include <QNetworkReply>
ImageManager::ImageManager(QObject *parent)
    : QObject(parent)
{
}

QString ImageManager::fetchImageForCity(const QString &city)
{
    QString apiKey = "50667198-e18ddf536530283bde2510604";
    QUrl url("https://pixabay.com/api/");
    QUrlQuery query;
    query.addQueryItem("key", apiKey);
    query.addQueryItem("q", city);
    query.addQueryItem("image_type", "photo");
    query.addQueryItem("per_page", "3");
    url.setQuery(query);
```

```
    QNetworkRequest request(url);

    QNetworkReply *reply = m_networkManager.get(request);

    QEventLoop loop;
    connect(reply, &QNetworkReply::finished, &loop, &QEventLoop::quit);
    loop.exec();

    if (reply->error() != QNetworkReply::NoError) {
        qWarning() << "Error fetching image:" << reply->errorString();
        reply->deleteLater();
        return QString();
    }

    QByteArray response = reply->readAll();
    reply->deleteLater();

    QJsonDocument doc = QJsonDocument::fromJson(response);
    QJsonObject obj = doc.object();
    QJsonArray hits = obj["hits"].toArray();

    if (!hits.isEmpty()) {
        return hits[0].toObject()["webformatURL"].toString();
    }

    return QString();
}
```

imagemanager.h

```
#ifndef IMAGEMANAGER_H
#define IMAGEMANAGER_H

#include <QObject>
#include <QNetworkAccessManager>
#include <QNetworkReply>

class ImageManager : public QObject
{
    Q_OBJECT
public:
    explicit ImageManager(QObject *parent = nullptr);

    Q_INVOKABLE QString fetchImageForCity(const QString &city);
```

```
private:
    QNetworkAccessManager m_networkManager;
};

#endif // IMAGEMANAGER_H
```

locationprovider.cpp

```
#include "locationprovider.h"
#include <QGeoPositionInfo>
#include <QDebug>

LocationProvider::LocationProvider(QObject* parent)
    : QObject(parent)
    , m_source(nullptr)
{
    m_source = QGeoPositionInfoSource::createDefaultSource(this);
    if (!m_source) {
        emit errorOccurred("No position source available");
        return;
    }
    connect(m_source, &QGeoPositionInfoSource::positionUpdated,
            this, &LocationProvider::onPositionInfo);
}

void LocationProvider::start(int updateIntervalMs)
{
    if (!m_source) {
        emit errorOccurred("Position source is null");
        return;
    }
    if (updateIntervalMs > 0)
        m_source->setUpdateInterval(updateIntervalMs);

    m_source->requestUpdate();
}

void LocationProvider::onPositionInfo(const QGeoPositionInfo &info)
{
    if (!info.isValid()) {
        emit errorOccurred("Received invalid position");
        return;
    }
    const auto coord = info.coordinate();
    emit positionUpdated(coord.latitude(), coord.longitude());
}
```

ДОДАТОК Г (обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

«МОБІЛЬНИЙ ТУРИСТИЧНИЙ AI-АСИСТЕНТ»

Виконав: студент 4 курсу, групи 1КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Рудий Р.С.
(прізвище та ініціали)

Керівник: к.т.н., доцент кафедри КН
Колодний В.В.
(прізвище та ініціали)

Вінниця ВНТУ - 2025 рік

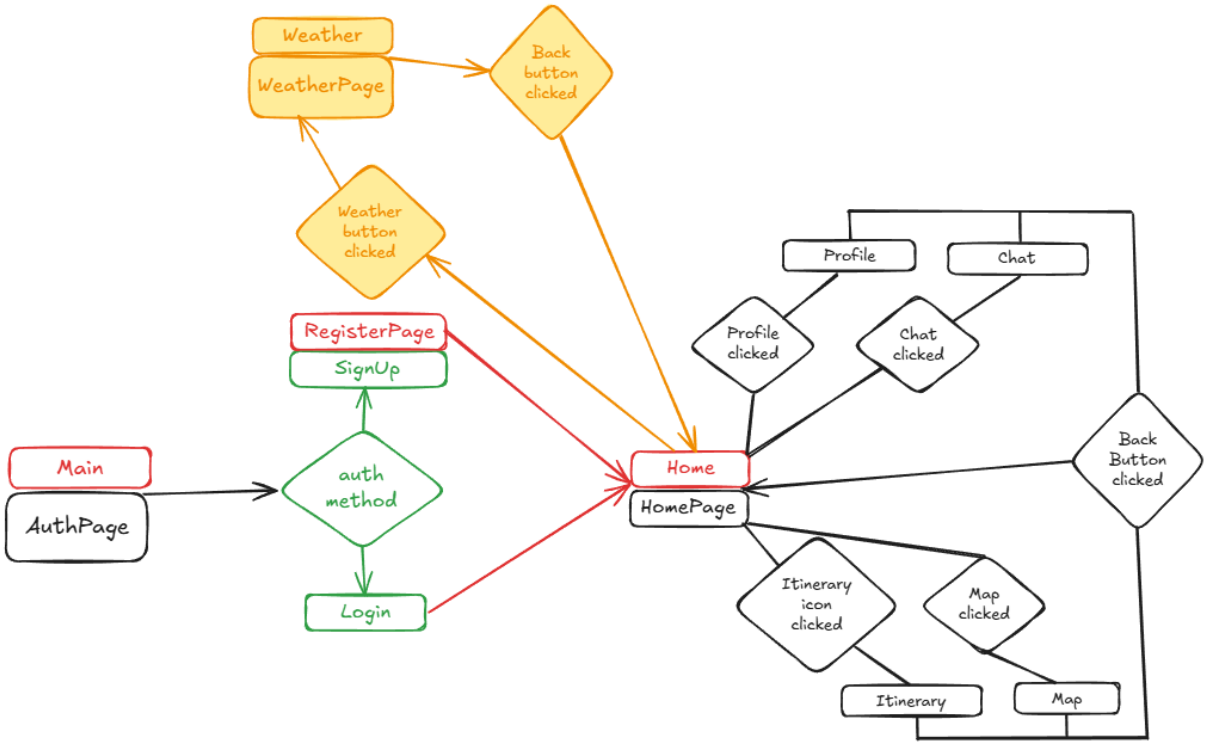

Рисунок Г.1 – Flowchart архітектури мобільного додатку туристичний AI-асистент

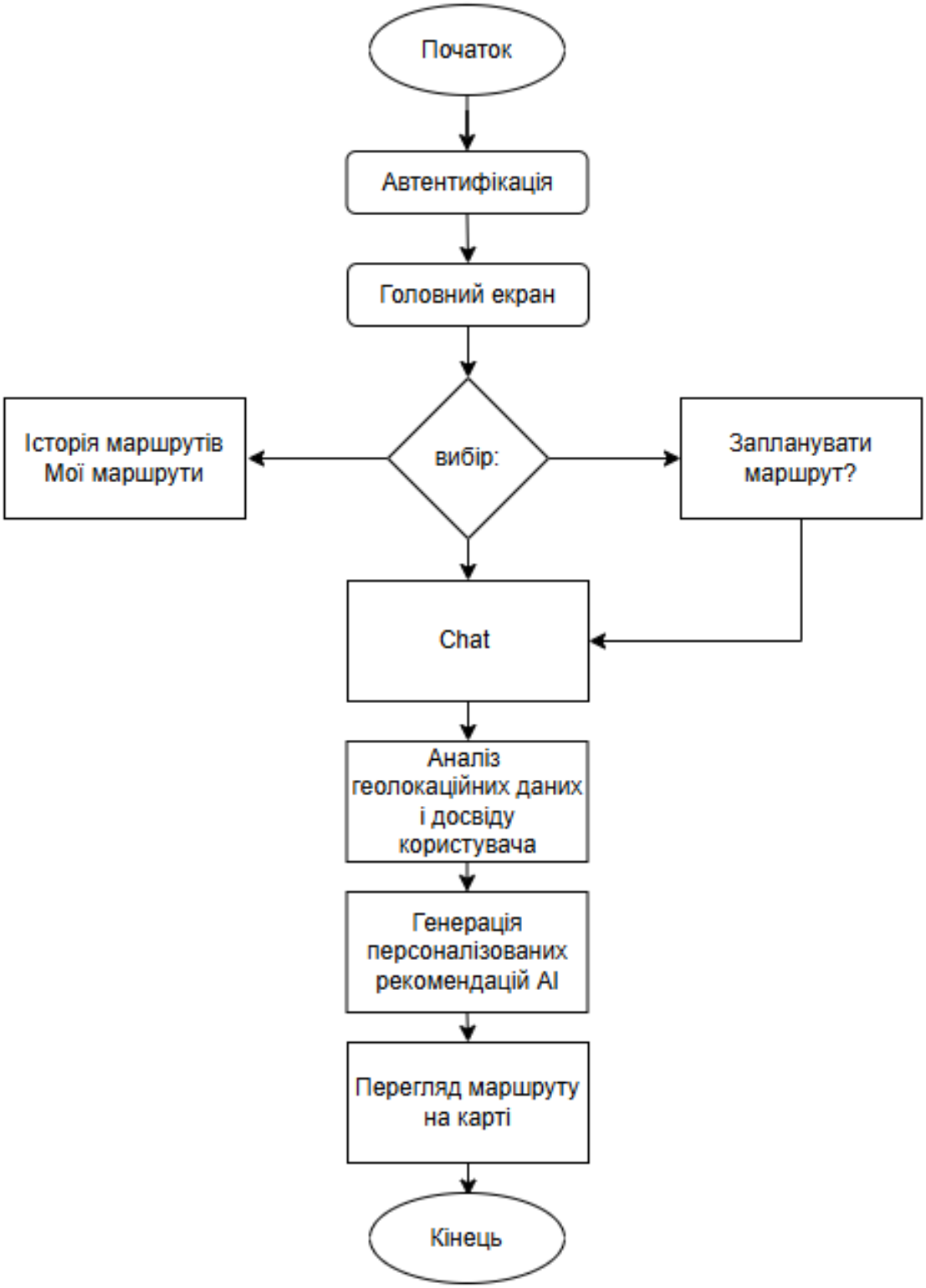

Рисунок Г.2 – алгоритм роботи мобільного додатку туристичний AI-асистент

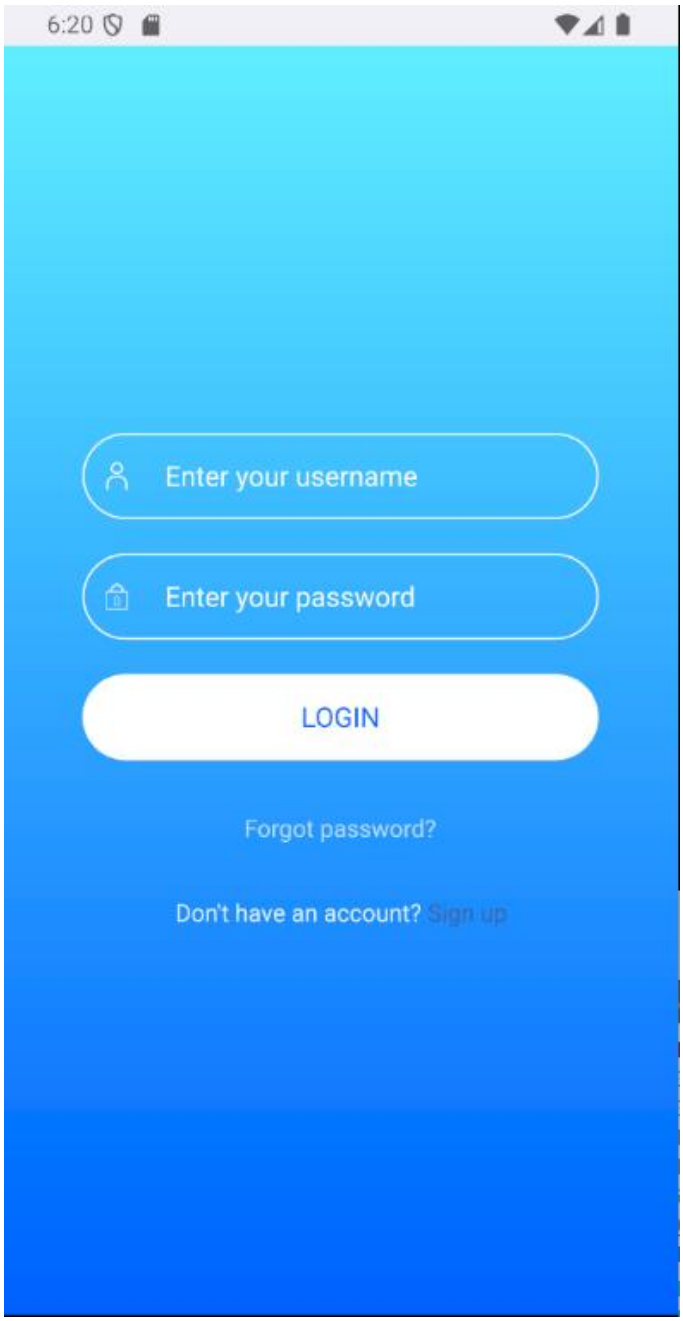

Рисунок Г.3 – Сторінка автентифікації

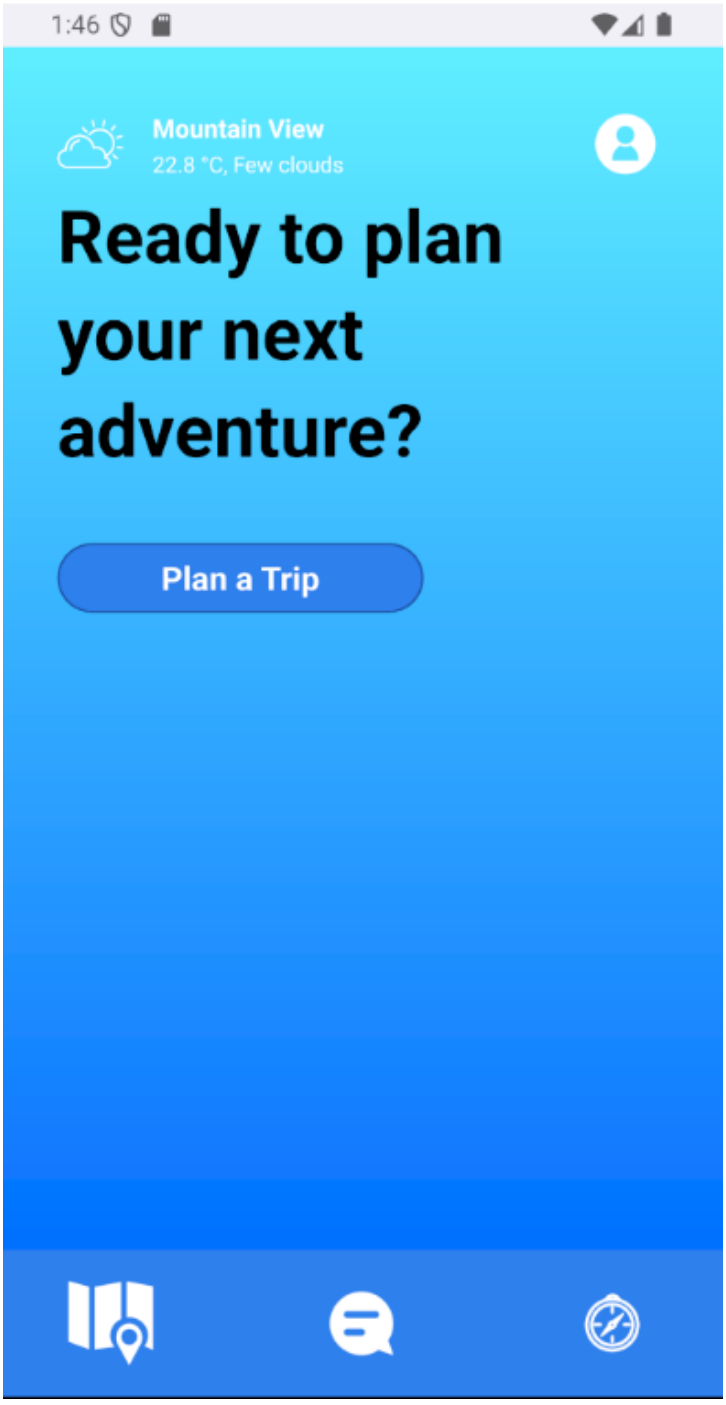

Рисунок Г.4 – Головна сторінка

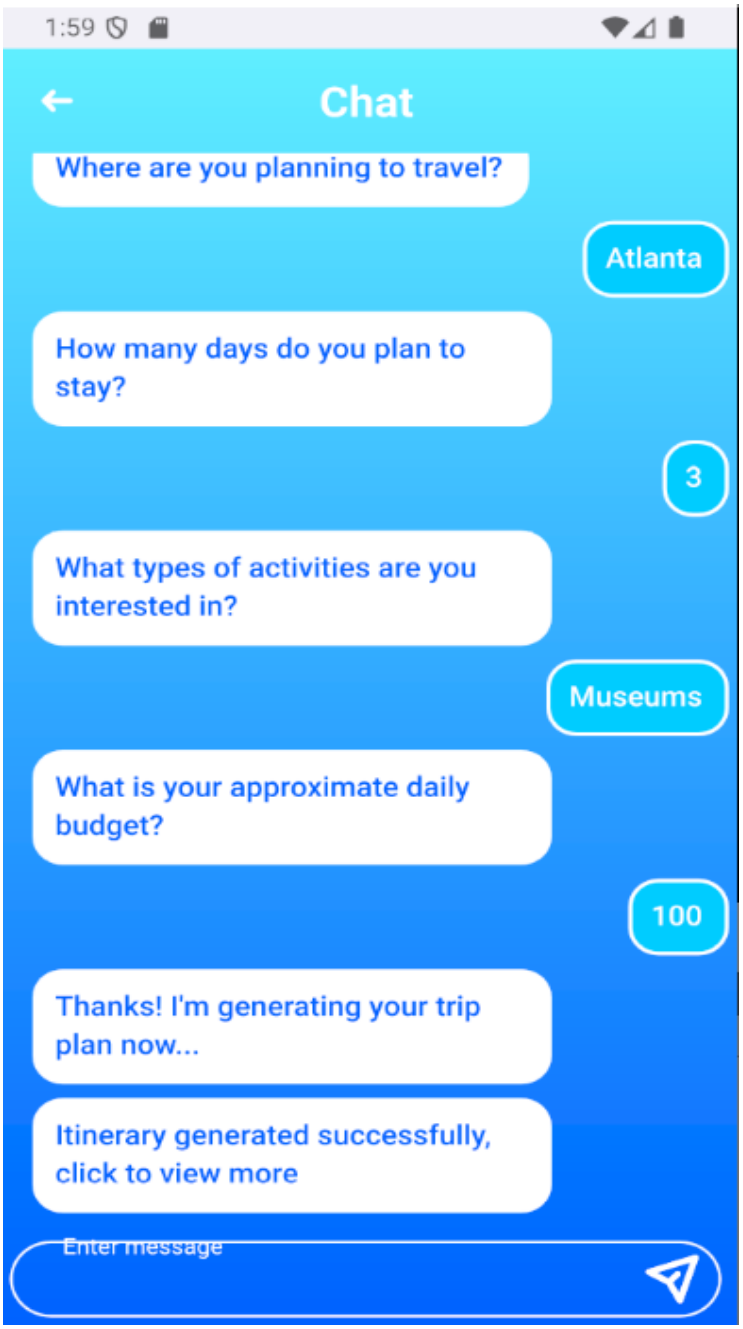

Рисунок Г.5 – листування з AI-асистентом

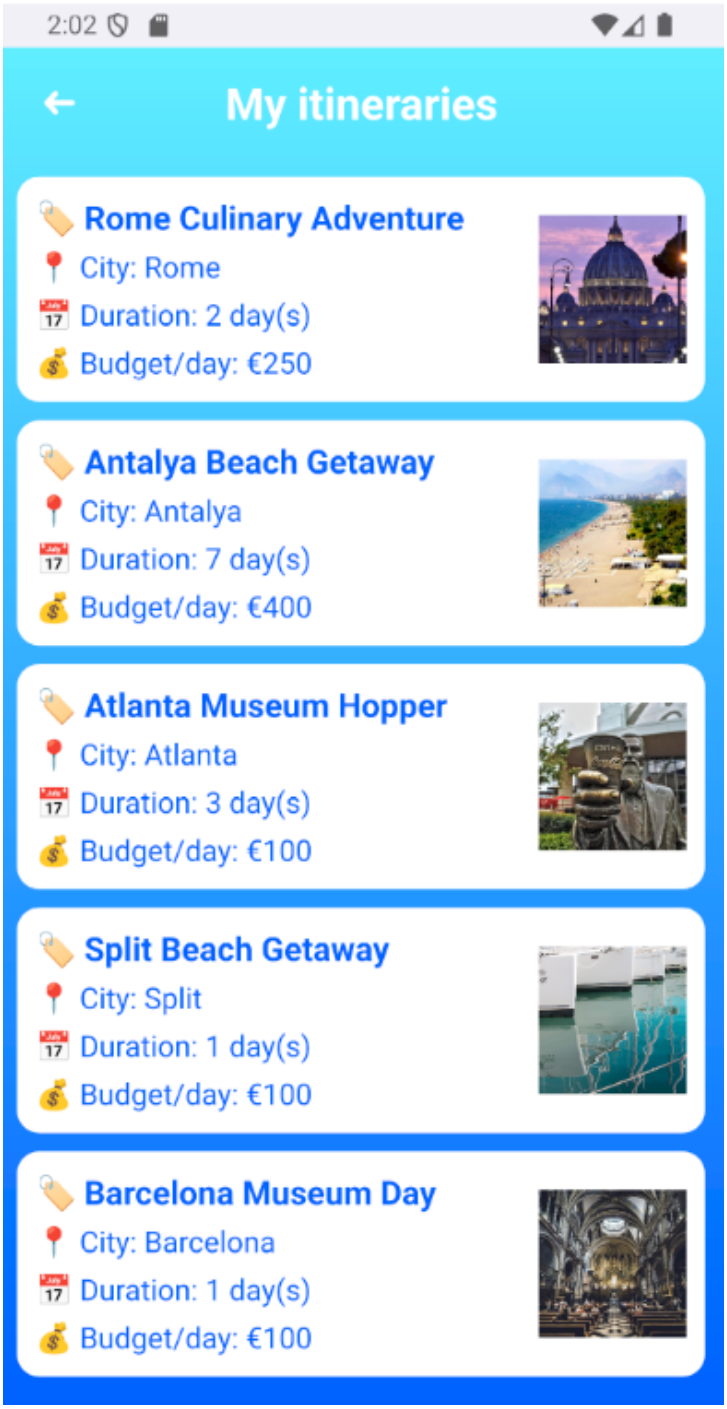

Рисунок Г.6 – маршрути користувача

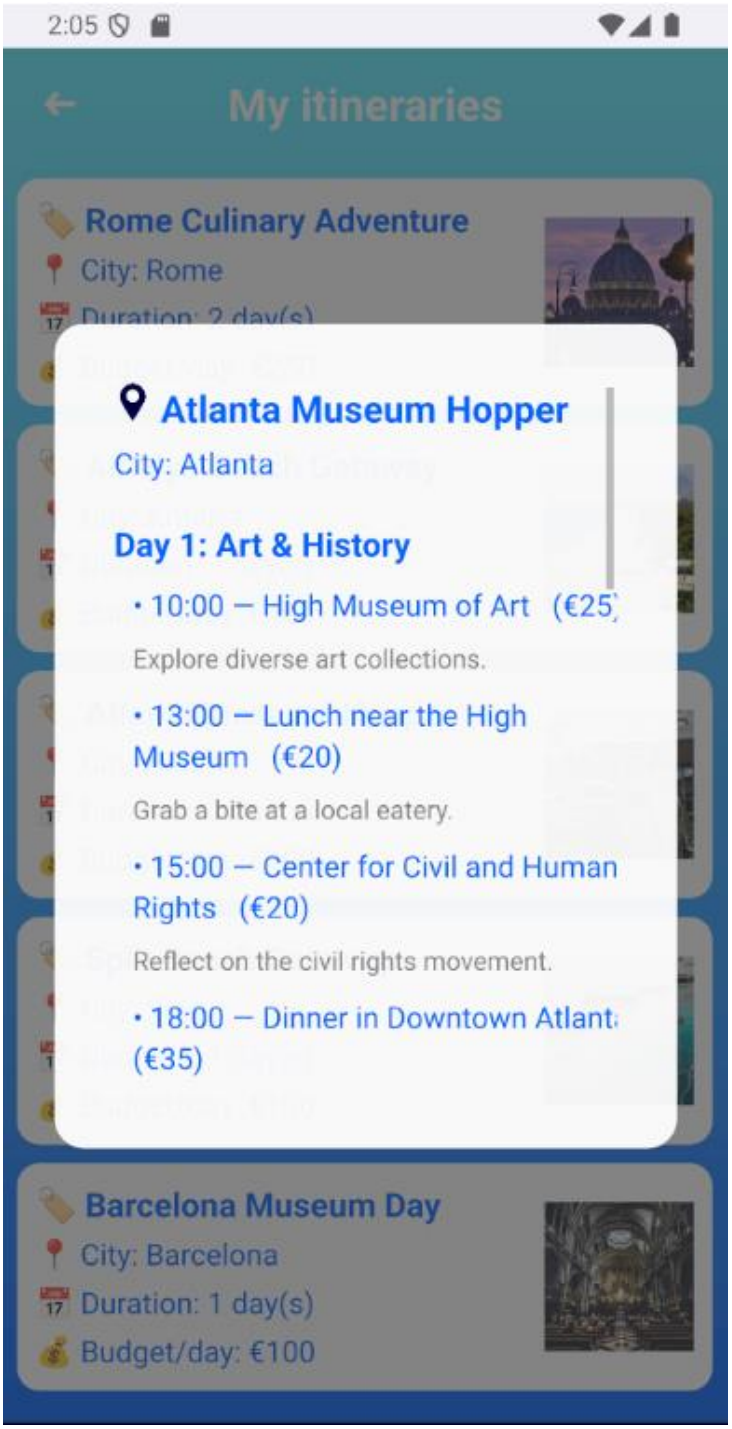

Рисунок Г.7 – детальна інформація маршруту

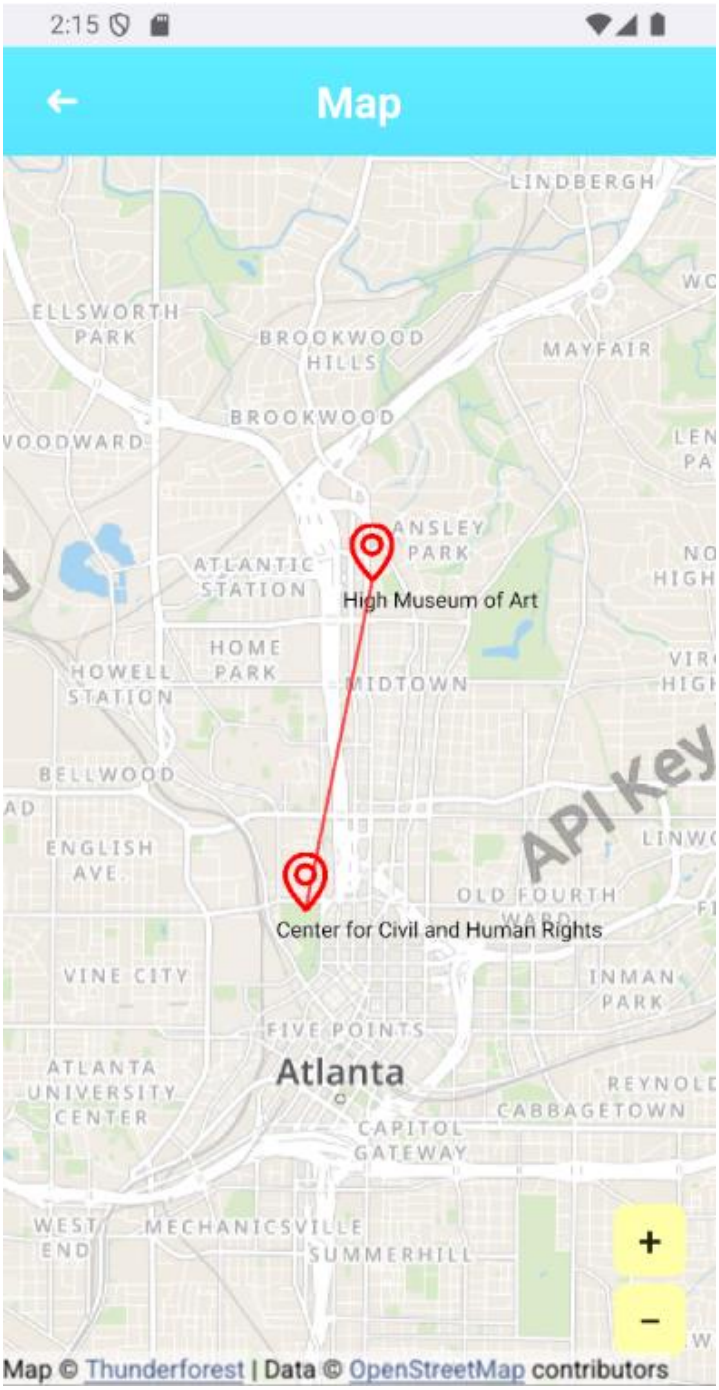

Рисунок Г.8 – інтерактивна карта з “пінами”

ДОДАТОК Д (довідниковий)

Довідка про впровадження

МАЛЕ НАУКОВО-ВИРОБНИЧЕ ПІДПРИЄМСТВО "ТОВ "ІТІ"
Україна, 21021, м.Вінниця, вул. Келецька, 56

№ 48 від "6" 06 2025 р.

ДОВІДКА

Дана Рудому Ростиславу Сергійовичу в тому, що результати, одержані нею в процесі виконання бакалаврської кваліфікаційної роботи, а саме алгоритм та мобільний туристичний AI-асистент, планується використати в розробках ТОВ «ІТІ».

Зам. директора ТОВ «ІТІ» Бодяк В.М.

ТОВ "ІТІ"