ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ СИСТЕМ ТА ТЕХНОЛОГІЙ

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«**Мобільний застосунок для пошуку та рекомндаїй локацій з інтеграцією мовних моделей штучного інтелекту**»

на здобуття освітнього ступеня магістр
за спеціальності 126 Інформаційні системи та технології
(код,найменування спеціальності)
освітньо-професійної програми Інформаційні системи та технології
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Данило БУДЖАК
(підпис) *(ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти
група ІСДМ-61 Данило БУДЖАК
(ім'я, ПРІЗВИЩЕ)

Керівник Валентина ДАНИЛЬЧЕНКО
к.т.н. *(ім'я, ПРІЗВИЩЕ)*

Рецензент: _____
(ім'я, ПРІЗВИЩЕ)

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут Інформаційних технологій

Кафедра Інформаційних систем та технологій
Ступінь вищої освіти магістр
Спеціальність 126 Інформаційні системи та технології
Освітньо-професійна прорама Інформаційні системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедрою ІСТ
_____ Каміла СТОРЧАК

“____” _____2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Буджаку Данилу Віталійовичу
(прізвище, ім’я, по батькові здобувача)

1.Тема кваліфікаційної роботи: Мобільний застосунок для пошуку та рекомндаїй локацій з інтеграцією мовних моделей штучного інтелекту.
керівник кваліфікаційної роботи: Валентина ДАНИЛЬЧЕНКО доцент
(ім’я, ПРІЗВИЩЕ, науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від “_30__ ” жовтня 2025 р. № 467____

2. Строк подання кваліфікаційної роботи «26» грудня 2025 р.

3. Вихідні дані кваліфікаційної роботи:
1. Мобільні технології та кросплатформна розробка (React Native, Expo).
2. Карти та геолокаційні сервіси (Geoapify, Google Places API, OpenStreetMap).
3. Методи обробки природної мови та інтелектуального аналізу запитів (Google Gemini).
4. Архітектура клієнт–серверних застосунків та методи інтеграції з зовнішніми API.
5. Науково-технічна література, інтернет-ресурси та офіційна документація API-сервісів.

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

1. Дослідження сучасних тенденцій розвитку мобільних геолокаційних сервісів.
2. Огляд методів аналізу природномовних запитів та технологій обробки мови.
3. Розроблення архітектури мобільного застосунку та серверної частини для інтеграції з геолокаційними та AI-сервісами.
4. Аналіз результатів тестування та оцінка ефективності роботи розробленого застосунку.
5.Перелік ілюстраційного матеріалу: *презентація* , код розробки
6. Дата видачі завдання «30» жовтня 2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Підбір технічної літератури та аналіз літератури	Вересень 2025	
2.	Аналіз методів NLP для інтерпретації природномовних запитів та вибір моделі (Google Gemini)	Вересень – жовтень 2025	
3.	Проєктування архітектури застосунку: клієнтська частина (React Native)	Жовтень 2025	
4.	Інтеграція Google Places API для пошуку та обробки геоданих	Жовтень 2025	
5.	Реалізація модуля ШІ для класифікації та фільтрації	Листопад 2025	
6.	Тестування системи, оптимізація взаємодії карти з AI-пошуком, аналіз результатів роботи	Листопад 2025	
7.	Підготовка висновків, оформлення демонстраційних матеріалів та презентації	Листопад 2025	
8.	формлення магістерської роботи	Грудень 2025	

Здобувач вищої освіти _____ Данило БУДЖАК
(підпис) *(ім'я, ПРІЗВИЩЕ)*
Керівник кваліфікаційної роботи _____ Валентина ДАНИЛЬЧЕНКО
(підпис) *(ім'я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня магістра: 83 стор., 43 рис., 30 джерел.

Мета роботи - Метою магістерської роботи є розробка мобільного застосунку, що поєднує можливості картографічних сервісів та мовних моделей штучного інтелекту для інтелектуального пошуку локацій поблизу, здатного інтерпретувати природномовні запити користувача та повертати персоналізовані результати.

Об'єкт дослідження -Процес розробки мобільних застосунків, що інтегрують мовні моделі ШІ та картографічні API.

Предмет дослідження - Методи та інструменти інтерпретації природної мови та обробки геоданих у мобільному застосунку з використанням Google Maps API, Gemini API та React Native.

Короткий зміст роботи:

У магістерській роботі здійснено комплексне дослідження підходів до поєднання картографічних сервісів з технологіями штучного інтелекту для створення інтелектуальних мобільних застосунків. Розглянуто сучасні геолокаційні сервіси, способи їх взаємодії з API, а також можливості мовних моделей для обробки текстових запитів природною мовою.

Виконано аналіз існуючих аналогів, визначено їх сильні та слабкі сторони, а також сформовано проблемне поле, що вказує на необхідність створення системи, здатної розуміти контекст, наміри та характеристики запиту користувача, а не вимагати від нього ручного налаштування фільтрів.

У роботі розроблено повноцінний мобільний застосунок із використанням React Native, Expo, Google Maps та Google Places API. Клієнтська частина реалізує інтерфейс пошуку, відображення карти, маркерів та детальних карток закладів.

Серверна частина на Node.js забезпечує безпечне використання API-ключів, виконує складні операції з геоданими та реалізує взаємодію з мовною моделлю Gemini. ШІ виконує аналіз текстових запитів, виділяє категорії та ключові характеристики, ранжує результати Google Places та створює персоналізовані описи для обраних локацій.

КЛЮЧОВІ СЛОВА:

МОБІЛЬНИЙ ЗАСТОСУНОК, ГЕОЛОКАЦІЯ, GOOGLE MAPS API, GEMINI API, ШТУЧНИЙ ІНТЕЛЕКТ, LLM, ПОШУК ЛОКАЦІЙ, РАНЖУВАННЯ, REACT NATIVE, EXPO, UX/UI, КАРТОГРАФІЧНІ СЕРВІСИ, РОЗПІЗНАВАННЯ ЗАПИТІВ, NODE.JS, ІНТЕГРАЦІЯ API.

ABSTRACT

The text part of the master's qualification thesis: 83 pages, 43 figures, 30 sources.

Purpose of the work - The purpose of the master's thesis is to develop a mobile application that integrates cartographic services and large language models of artificial intelligence to enable intelligent nearby location search capable of interpreting the user’s natural-language queries and returning personalized results.

Object of study -The process of developing mobile applications that integrate AI language models and mapping APIs.

Subject of study -Methods and tools for natural-language interpretation and geodata processing in a mobile application using Google Maps API, Gemini API, and React Native.

Summary of the work :

The master's thesis presents a comprehensive study of approaches to combining cartographic services with artificial intelligence technologies to create intelligent mobile applications. Modern geolocation services, principles of their interaction with APIs, and the capabilities of language models for processing natural-language queries were examined.

An analysis of existing analogs was performed, identifying their strengths and weaknesses, and defining the problem space that highlights the need for a system capable of understanding the context, intent, and characteristics of a user query rather than requiring manual filter adjustment.

The work includes the development of a fully functional mobile application using React Native, Expo, Google Maps, and Google Places API. The client-side implements the search interface, map rendering, markers, and detailed place cards.

The server side, implemented in Node.js, provides secure handling of API keys, performs complex geodata operations, and manages interaction with the Gemini language model. Artificial intelligence analyzes text queries, extracts categories and key characteristics, ranks Google Places results, and generates personalized descriptions for selected locations.

KEYWORDS:
MOBILE APPLICATION, GEOLOCATION, GOOGLE MAPS API, GEMINI API, ARTIFICIAL INTELLIGENCE, LLM, LOCATION SEARCH, RANKING, REACT NATIVE, EXPO, UX/UI, MAPPING SERVICES, QUERY INTERPRETATION, NODE.JS, API INTEGRATION.

ЗМІСТ

ПЕРЕЛІК ВИКОРИСТАНИХ СКОРОЧЕНЬ

AI – Artificial Intelligence (штучний інтелект)

API – Application Programming Interface (програмний інтерфейс додатків)

CSS – Cascading Style Sheets (каскадні таблиці стилів)

EXPO – Expo Application Framework (фреймворк для створення мобільних застосунків на React Native)

GPS – Global Positioning System (система глобального позиціонування)

HTML – HyperText Markup Language (мова гіпертекстової розмітки)

HTTP – HyperText Transfer Protocol (протокол передавання гіпертексту)

JSON – JavaScript Object Notation (текстовий формат обміну даними)

JS – JavaScript (мова програмування JavaScript)

LLM – Large Language Model (велика мовна модель)

UI – User Interface (інтерфейс користувача)

UX – User Experience (досвід користувача)

API Key – ключ доступу до API

SDK – Software Development Kit (комплект засобів розробника)

URL – Uniform Resource Locator (уніфікований локатор ресурсу)

ВСТУП

Розвиток інформаційних технологій значно вплинув на те, як люди взаємодіють із цифровими сервісами. На сьогодні мобільні пристрої стали невід'ємною частиною повсякденного життя, а користувачі дедалі частіше надають перевагу мобільним застосункам перед вебсайтами для стаціонарних пристроїв, оскільки вони є доступними, адаптивними та зручними у використанні [1]. Застосунки, що працюють з геолокацією, картографічними сервісами, рекомендаційними системами та інтелектуальними підказками, користуються особливою популярністю, адже вони дозволяють отримувати релевантну інформацію відповідно до місцезнаходження користувача [2].

Сучасні тенденції свідчать про зростання попиту на сервіси, які допомагають швидко знаходити інформацію про заклади поблизу — ресторани, спортивні зали, аптеки, парки тощо. Водночас користувачі очікують не лише перелік об'єктів, а персоналізовану, контекстну та адаптовану відповідь. Саме тому важливим напрямом розвитку є поєднання картографічних сервісів із технологіями штучного інтелекту, які здатні обробляти запити природною мовою та правильно інтерпретувати наміри користувача [3].

Розвиток API-сервісів додатково спрощує створення таких систем. Зокрема, API Google надає можливість отримувати детальні дані про локальні об'єкти — описи, відгуки, фото тощо, а сучасні мовні моделі, такі як OpenAI або Google Gemini, дозволяють аналізувати запити у звичній людині формі та формувати точні рекомендації. Це створює передумови для формування нового покоління мобільних застосунків, що поєднують геодані, машинне навчання та інтуїтивний інтерфейс.

Актуальність теми - Актуальність теми зумовлена зростанням потреби в мобільних інтелектуальних сервісах, які забезпечують персоналізований пошук і навігацію. В умовах цифрової трансформації бізнесу та урбанізації користувачі очікують швидкого доступу до локальної інформації, адаптованої до їхніх

вподобань та контексту. Для України це особливо важливо з огляду на активний розвиток сфери послуг, внутрішнього туризму та технологічних рішень у бізнес-середовищі. Поєднання картографічних сервісів і штучного інтелекту відкриває можливість створення сучасних мобільних застосунків, здатних обробляти запити природною мовою та надавати більш точні рекомендації, ніж традиційні фільтри.

Мета і завдання дослідження - Мета роботи полягає у розробці мобільного застосунку, який інтегрує картографічні API та системи штучного інтелекту для інтерпретації запитів природною мовою та формування персоналізованих рекомендацій для користувача.

Завдання дослідження

У процесі дослідження вирішувалися такі завдання:

- провести аналіз сучасних геолокаційних сервісів, AI-систем та мобільних фреймворків;
- визначити найбільш ефективні інструменти для реалізації проєкту;
- розробити UI/UX дизайн застосунку з урахуванням вимог до простоти та зручності використання;
- інтегрувати API для отримання картографічних даних і пошуку місць;
- реалізувати інтерпретацію природної мови за допомогою LLM API;
- створити бекенд-частину або інтегрувати хмарні сервіси для обробки запитів;
- протестувати роботу додатку на реальних користувацьких сценаріях.

Об’єкт дослідження - Об’єктом дослідження є процес розробки мобільних застосунків, що використовують зовнішні API та системи штучного інтелекту для персоналізованої взаємодії з користувачами.

Предмет дослідження - Предметом дослідження є методи та інструменти інтеграції мовних моделей і геолокаційних сервісів у мобільний застосунок.

Методи дослідження - У роботі використано такі методи:

- аналіз наукових публікацій і технічної документації;
- порівняльний аналіз інструментів розробки;
- об’єктно-орієнтований підхід до програмування;
- експериментальне тестування функціональності застосунку;
- аналіз практичних прикладів використання геолокаційних сервісів і мовних моделей.

Наукова новизна та практична значущість отриманих результатів - Наукова новизна полягає у поєднанні картографічних сервісів із мовними моделями для створення системи, здатної інтерпретувати запити природною мовою та формувати персоналізовані рекомендації. У роботі обґрунтовано архітектурний підхід, що дозволяє оптимізувати взаємодію з зовнішніми API та забезпечити високу продуктивність застосунку.

Практична значущість полягає у можливості застосування розробленого рішення в туристичних сервісах, системах навігації, локальних бізнес-процесах, а також у мобільних продуктах, що працюють з геоданими та рекомендаціями. Отримані результати можуть бути використані як основа для подальших розробок у сфері інтеграції ШІ з картографічними сервісами.

Апробація результатів та публікації

III всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу», 18 листопада 2025 року, ДУКТ – «Обмеження штучного інтелекту, використаного через API, у мобільних застосунках» , «Інтеграція чат-ботів у мобільні застосунки для покращення взаємодії користувача з системою»

VIII Всеукраїнська науково- технічна конференція «Комп'ютерні технології: інновації, проблеми, рішення», 02-03 жовтня 2025 року, Житомирська політехніка - «Інтелектуальні методи фільтрації даних у сучасних інформаційних системах»

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОГЛЯД ТЕХНОЛОГІЙ

1.1. Актуальність теми

У сучасному світі мобільні технології стали основним засобом, за допомогою якого користувачі можуть взаємодіяти з різними послугами, організаціями та інформацією. Мобільні застосунки стають все більш поширеними, оскільки вони завжди доступні, швидкі та дозволяють вирішувати повсякденні задачі за лічені кліки(Рис. 1.1). Зокрема, дуже популярні є програми, які допомагають знаходити заклади поблизу, переглядати відгуки чи отримувати поради щодо кафе, магазинів або інших місць[4].

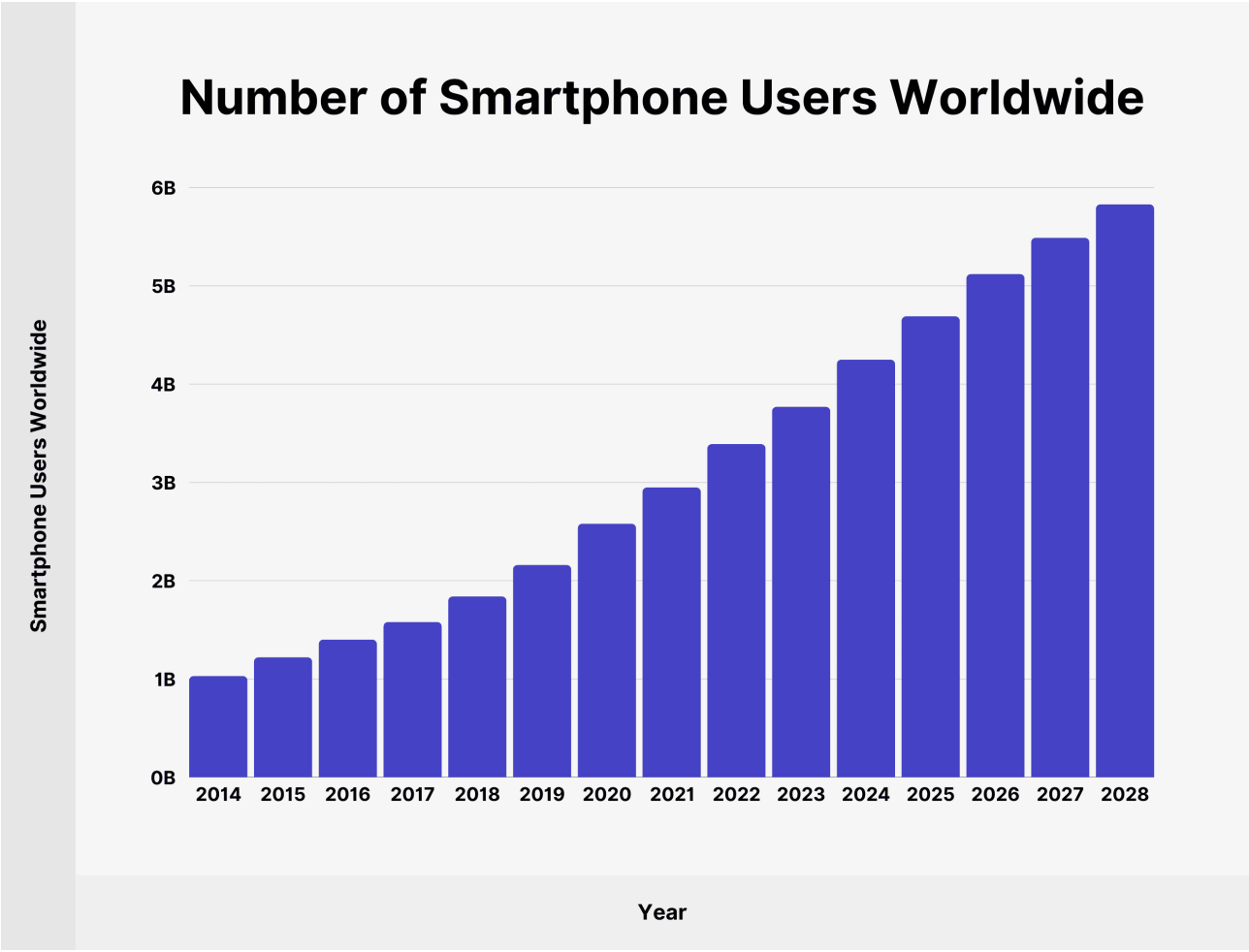


Рис. 1.1 Зростання використання сматфонів з роками та передбачення на майбутнє

Більшість гео-додатків використовують класичний принцип пошуку через вільтри та ключові слова.Через ці обмеження користовачу приходить витрачати час на уточнення запиту вручну , ообирати категорії та налаштовувати фільтри. Таким чином результат не достатньо персоналізований.З появою штучного інтелекту цей процес став більш простим. Запит можна сформувати таким чином як ми спілкуємося в побуті на приклад: "Покажи тихе кафе з вайфаєм поруч". Мовну модель можна легко використовувати, хоча цей запит є надто «живим» для класичного пошуку[5].

Також одна з ключових проблем полягає в тому, що дані про заклади розкидані по різних сервісах. Одним сервісом можна побачити карту фото та відгуки, а іншими ти можеш прочитати опис та іншу потрібну інформацію про заклади.Це не дуже зручно і користувач повинен переходити між різними джерелами, що ускладнює процес вибору. Цю проблему можна вирішити шляхом тощо щоб зібрати дані з всіх цих сервісів та об'єднати в один додаток, що в свою чергу й дозволяє зробити API.

Мовні моделі такі як Open AI та Google Gemini, можуть інтерпретувати запити користувачів природним чином, незалежно від їх складності. Це значно покращує зручність використання застосунку та робить його більш «людяним»(Рис. 1.2)[5]. Створення абсолютно нового інтерфейсу пошуку місць можна досягти за допомогою поєднання цієї стратегії з картографічними ресурсами, такими як Google Maps, Open Street Maps та Geoapify, а також інформаційними платформами, такими як Foursquare, Yelp, Tripadvisor і Google Maps.

Рис. 1.2 випадки використання Мовних моделей

Ця тема особливо важлива в Україні, де підприємства часто змінюють свої графіки роботи, місцезнаходження або умови надання послуг. Вкрай важливо, щоб користувачі мали можливість отримати актуальну інформацію без необхідності перевіряти її вручну на різних ресурсах[6]. Люди можуть мати надійний досвід і надійні дані завдяки API, які регулярно оновлюються.

Підбиваючи підсумки, ця тема дуже важлива, оскільки вирішує основні з цих проблем, адже поєднує можливості карт, сучасні AI технології та простий інтерфейс. Таким чином можна значно покращити взаємодію користувачів з сервісом , забезпечити швидкий доступ до інформації та надати адаптовану рекомендацію згідно потребам користувачів. Цей сектор має багато перспектив для розвитку, і він відповідає сучасним тенденціям розвитку цифрових мобільних продуктів.

1.2. Аналіз існуючих рішень

Сьогодні в Інтернеті є безліч сервісів і мобільних додатків, які надають картографічну інформацію про заклади, місця, маршрути та інші об’єкти. Кожен має джерела даних, функції та особливості. Сьогодні найвідомішими

глобальними платформами, які активно використовуються, є Google Maps, Apple Maps, Mapbox, Here Maps, OpenStreetMap, Foursquare, Yelp, Tripadvisor та багато інших[7]. Ці сервіси виконують схожі завдання, такі як пошук місць, навігація, перегляд фотографій, відгуки та рекомендації, але вони працюють по-різному, що впливає на зручність і точність роботи(Рис. 1.3).

Рис. 1.3 Зображення картографічного сервісу

Google Maps продовжує бути лідером серед картографічних платформ у багатьох аспектах. Він пропонує найбільш повну інформацію про місця в більшості країн світу завдяки величезній базі локацій, постійним оновленням даних і підтримці відгуків мільйонів користувачів. Тим не менш, Google має деякі недоліки, включаючи складну систему тарифів і високі витрати на використання API. Ці витрати не завжди зручні для незалежних розробників і проектів невеликого розміру. Незважаючи на те, що Apple Maps є альтернативою екосистеми iOS, його дані в багатьох регіонах менш повні, а відгуки здебільшого походять з Yelp, що робить інформацію залежною від іншої платформи.

Mapbox, на відміну від Google Maps, надає користувачам більше варіантів. Застосунки, що потребують унікального стилю карти або високої швидкості рендерингу, часто використовують цей сервіс[8]. Проте, з іншого боку Mapbox часто використовують разом з іншими API окскільки його власна база POI не достатня для повноцінного використання. Here Maps є ефективним рішенням для транспортних і навігаційних систем і широко використовується в логістичних застосунках, але вони менш підходять для завдань, пов’язаних із рекомендаціями закладів.

Сервісні даних про місця, такі як Foursquare, Yelp і Tripadvisor, також важливі. На Foursquare можна знайти інформацію про заклади, фотографії, категорії, детальні описи та теги[9]. Його перевага — якісна, ручно структурована база даних. Якщо Yelp є популярним серед американців, кількість користувачів в Європі та Україні значно нижча. Tripadvisor корисний для тематичних подорожей, оскільки він більше орієнтований на туристичні місця, ресторани та готелі.

Також одним із найбільших сервісів є OpenStreetMap[10], який є повністю відкритим джерелом для картографічних проектів, де користувачі можуть редагувати свої дані. У нього є багато переваг, такі як безкоштовність і гнучкість у використанні, але він немає фографій, описв що ж очевидним мінусом і робить його обмеженим для надання рекомендацій.

Незважаючи на те що всі ці сервіси пропонують потужні інструменти да дозволяють вирішувати велику кількість завдання всі вони мають як переваги так і недоліки.

Переваги:

- велика кількість даних про локації (Google Maps, Foursquare, Tripadvisor);
- наявність фото та відгуків від реальних користувачів;
- зручні карти та навігація;
- стабільність і глобальна доступність сервісів;
- наявність API для інтеграції у власні застосунки.

Недоліки:

- дані розкидані між різними платформами;
- API з фото або детальними тегами часто платні або обмежені;
- відсутність повноцінної інтерпретації запитів природною мовою;
- різний рівень наповненості у різних країнах;

- неможливість отримати всі потрібні дані в межах однієї системи;
- високі тарифи на запити у популярних сервісів (особливо Google).

Аналіз існуючих рішень показує, що незважаючи на те, що на ринку доступні численні потужні продукти, жоден із них не забезпечує одночасно всіх потреб сучасного користувача, таких як зручний AI-пошук, актуальні дані про місцезнаходження, фотографії, рейтинги, часова доступність та інтерфейс, адаптований до мобільних пристроїв. Через те, що дані розташовані на різних платформах, користувачам потрібно використовувати кілька додатків одразу або вручну порівнювати результати.

Відповідно, потрібен новий підхід — мобільний застосунок, який поєднує переваги різних сервісів, компенсує їхні недоліки та інтегрує штучний інтелект для правильного розуміння запитів природною мовою. Таке рішення значно спростить процес пошуку місць; він буде швидшим, точнішим і набагато зрозумілішим для користувачів.

1.3. Визначення проблемного поля застосунку

Підчас аналізу існуючих рішень стає зрозуміло, що незнаваючи на величезну кількість подібних сервісів вони не можуть забезпечити комплексний та зручний підхід. Всі ці платформи мають як сильні, але вона також має обмеження, що суттєво впливають на те як користувач взаємодіє з системою. Фактично сучасні застосунки часто вимагають від користувачів адаптувати свої запити під систему , тоді я повинно бути навпаки, система повинна адаптуватися під користувача.

Однією з найбільших проблем є те, що користувачам доводиться витрачати багато часу на уточнення запиту(Рис. 1.4). До прикладу, людині потрібне затишне кафе з Wi-Fi для роботи не подалік від дому , то їй доводиться виставляти

фільтрацію закладів , та візувально переглядати відгуки , години роботи на кожному з них щоб підібрати найкращий для себе[5].

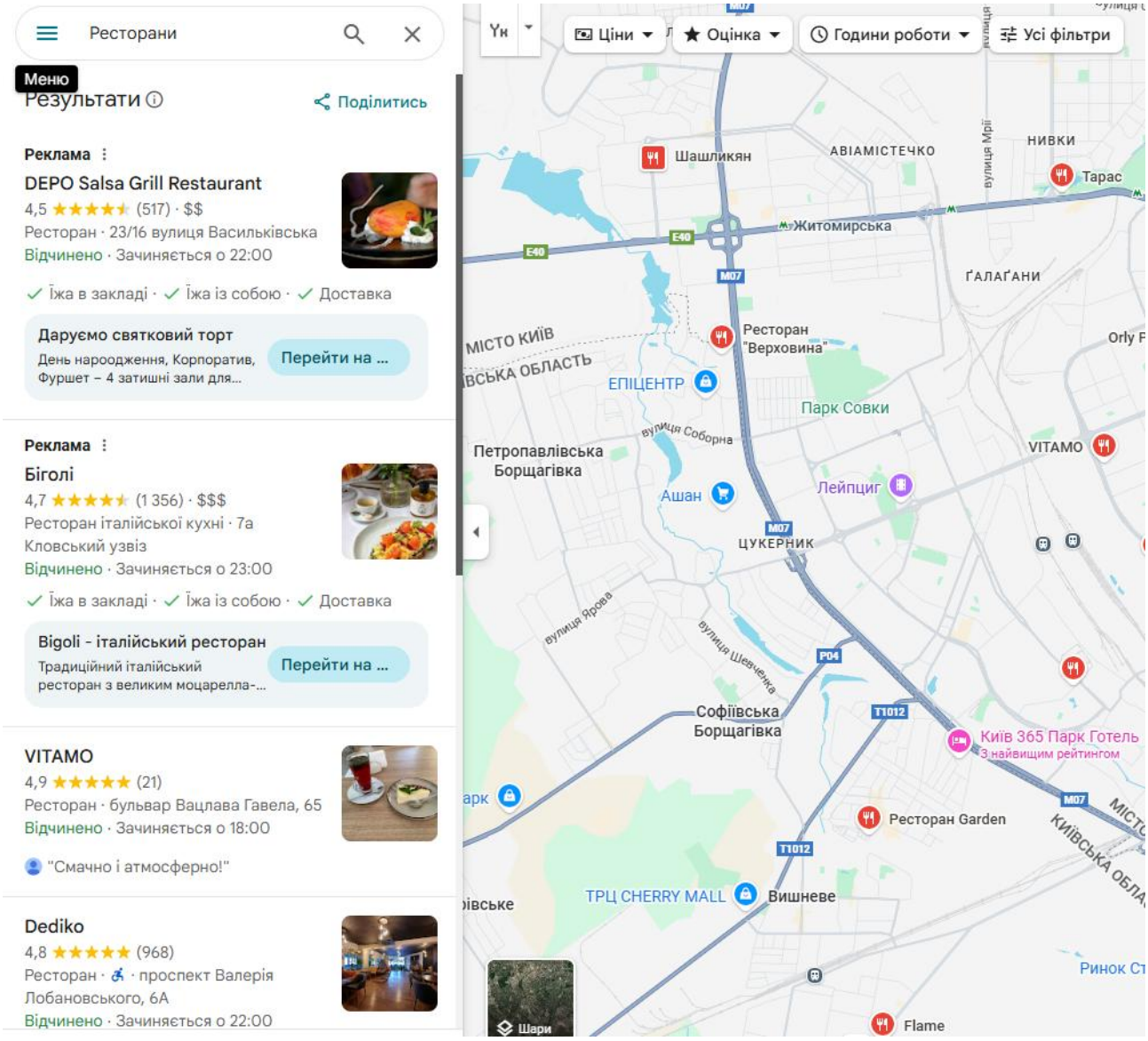


Рис. 1.4 Google Maps та система фільтрів

Несистемність або неповнота даних у різних сервісах є ще однією важливою проблемою. Незважаючи на те, що деякі програми містять детальний опис, вони не містять фотографій. Інший показує багато фотографій, але не дає інформації про часові рамки роботи. Хоча третій має рейтинги, він не містить корисних тегів, таких як "є Wi-Fi", "працює зараз" або "підходить для роботи за ноутбуком". У результаті такої фрагментації користувачі не можуть отримати

повну картину закладу, тому вони змушені переглядати кілька джерел одночасно[9].

Також важливим є відсутність адаптивності до певних обставин. Більшість програм не розглядають контекст, поведінку людини чи минулі запити. До прикладу, система повинна автоматично рекомендувати заклади або місця , якщо щось потібне і так постійно шукає користувач, щоб розвантажити ручну роботу до мінімуму.

Окремо слід підкреслити ще одну очевидну проблему, яка стосується оновлення інформації. У багатьох країнах, у тому числі в Україні, підприємства часто змінюють формат обслуговування, графік роботи або навіть місцезнаходження. Якщо сервіс не встигає оновлювати ці дані, користувач отримує неправильну інформацію, що погано впливає на досвід використання[6].

Під час виконання дипломної роботи, був розроблений застосунок , що усуває більшість з цих проблем.Використання ШІ дозволяє спростити процес пошуку та фільтрації інформації що потрібна користувачу , яку він вводить «природньою мовою». Також приєднання до цього застосунку API Google дозволяє отримати доступ до актуальних даних, фото, відгуків і тд. Це все в поєднанні дозволяє отримувати повну інформацію без перемикання між різними сервісами (Рис. 1.5).

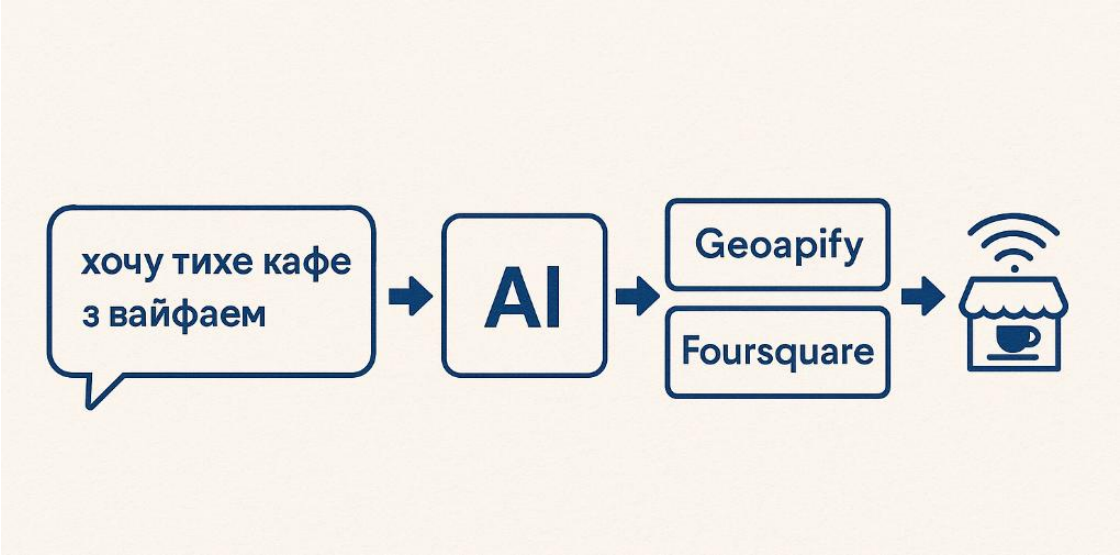


Рис. 1.5 Схема обробки запитів

Таким чином, додаток не тільки допомагає користувачам швидше знаходити потрібні місця, але й підвищує точність результатів, зменшує кількість зайвих дій, автоматично адаптується до контексту запиту та об’єднує кілька джерел інформації в одну систему. Це робить його значно простішим, ніж більшість традиційних рішень, і дозволяє вирішити проблеми, які залишаються актуальними для всіх популярних сервісів, доступних на ринку.

1.4. Постановка цілей і завдань розробки

Після аналізу уже існуючих рішень і визначення основних проблем в даній сфері виникає необхідність створити мобільний застосунок що буде виокристовувати штучний інтелект та картографічні сервіси, щоб вирішити всі існуючі проблеми. Основна мета полягає в тому, щоб зробити взаємодію з системою простою та зрозумілою для користувача. Замість того, щоб витрачати багато часу на налаштування фільтрів і ручне пошук, користувач може сформулювати свій запит природною мовою, а всі технічні деталі перетворення запиту будуть застосовані. Такий метод значно скорочує час, витрачений на пошук, робить результати точнішими та дозволяє користувачам уникнути типових помилок, коли вони не знають, який фільтр обрати.

Метою розробки є стоворення застосунку , що дозволяє використомувати мовну модель ШІ і інетграції картографічних сервісів для інтелектуального пошуку об1єктів поблизу інтерпретую текстові запити користувача.Додаток має розуміти потреби користувача та підбирати реальні міця використовуючи надані йому характеристики.

Для реалізації поставленої мети потрібно виконати низку завдань , що допоможуть в розробці. Перш за все потрібно проаналізувати технологічний стек та обрати технології що будуть найбільше пасувати для реалізації розробки, далі буде етап проектування UI/UX інетерфейсу.Іншим важливим завдання є ітеграція

мовних моделей ШІ та картографічних сервісів які дозволяють отримувати дані про місця та їх характеристики, а ШІ в свою чергу буде відповідати за фільтрацію отриманих даних та за інтерпретацію даних отриманих від користувача на мову зрозумілу для картографічних сервісів.

Після реалізації основного функціоналу , буде проведено перевірку роботоспроможності додатку, щоб переконатися що додаток працює стабільно та надає коректну інформацію.

2. АНАЛІЗ ТЕХНОЛОГІЙ ТА ОБҐРУНТУВАННЯ ВИБОРУ СТЕКУ

2.1. Огляд технологій для створення мобільних застосунків

Сучасна розробка мобільних пристроїв переживає швидку та різноманітну трансформацію, яка охоплює як технології, так і загальний підхід до створення цифрових продуктів[11]. Якщо на початку ери смартфонів розробники мали обмежений доступ до технологій, а саме нативних інструментів Android та iOS (Рис. 2.1), то ринок поступово почав змінюватися в результаті вимог користувачів, зростаючої конкуренції та появи нових способів взаємодії з мобільними пристроями. Сьогодні від застосування вже не очікують лише базових функцій. Користувачі потребують швидкості, стабільності, красивого інтерфейсу, інтеграції з різними онлайн-сервісами, роботи з геоданими, мережевих API та адаптивності до різних сценаріїв використання. Очікування бізнесу також зростають: компаніям важливо не лише створити застосунок, але й зробити це швидко, економічно та з можливістю постійного оновлення.

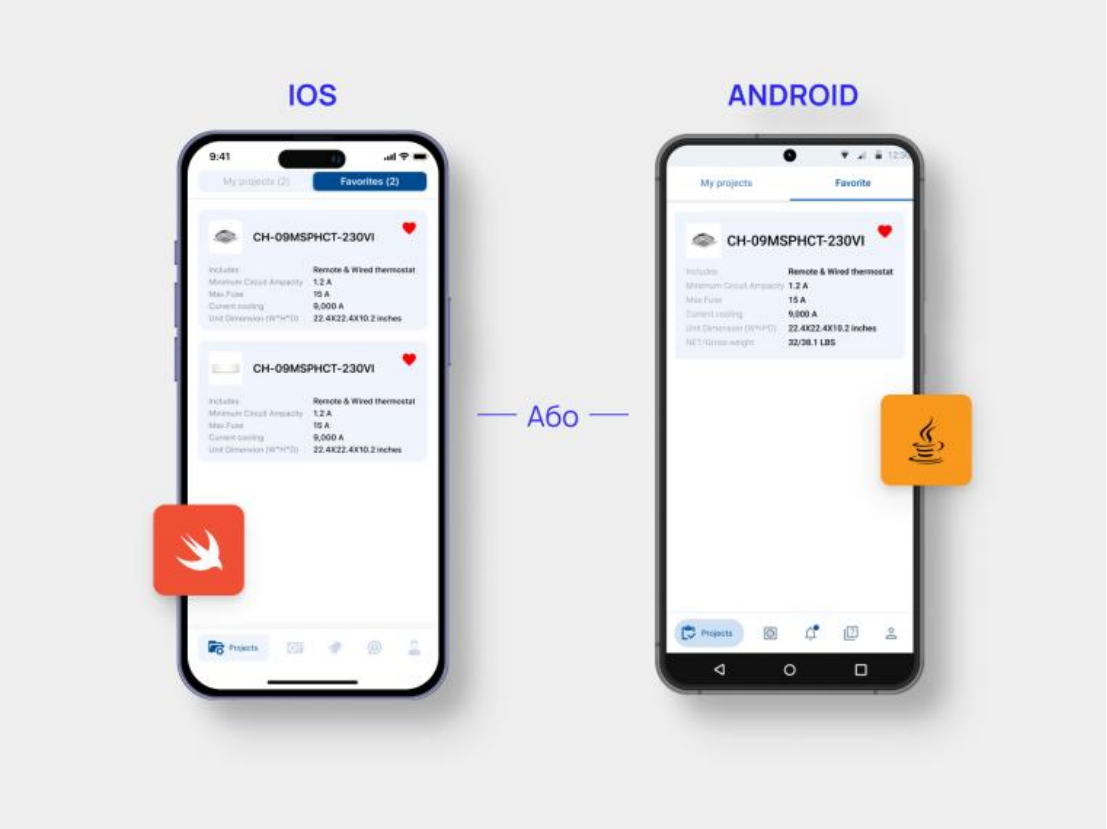


Рис. 2.1 Нативні технології створення мобільних додатків

У такому середовищі вибір технології перестає бути формальним технічним рішенням. Це стратегічний крок, який впливає на подальшу долю проекту, включаючи його масштабованість, підтримку, вартість розробки та можливість інтеграції нових функцій. Якщо технологія вибрана неправильно, продукт може втратити конкурентоспроможність ще на етапі тестування або запуску. Таким чином, сьогодні проблеми, пов’язані з розробкою мобільних застосунків, виходять за рамки програмування та включають питання, пов’язані з фінансами, організацією та навіть дизайном. З цієї причини для більшості проектів складно поєднати продуктивність, універсальність, простоту розробки та можливість швидко внести зміни.

Крім того, важливо знати, що мобільний ринок продовжує розвиватися. Враховуючи те, що кількість смартфонів у всьому світі перевищує кілька мільярдів[12], мобільні застосунки стали основним засобом, за допомогою якого люди взаємодіють із компаніями, послугами, картографічними платформами та загалом інтернетом. Користувачі дедалі частіше очікують, що застосунки майже

миттєво реагуватимуть на їхні дії, підтримуватимуть інтеграцію з онлайн-сервісами, працювати з урозміченими даними, ШІ та геолокаційними сервісами. Це підвищує тиск на технології, які повинні бути функціональними та надійними в складних ситуаціях.

Таким чином ми бачимо що аналіз існуючих технологій для розробки мобільних додатків є вкрай важливим етапом будь-якого проекту. Ми можемо оцінити наскільки підходять ті чи інші фреймворки для нашої розробки так вплинути на кінцеву якість продукту завдяки цьому. Цей аналіз дає нам основу для вибору стеку , який відповідатиме всім сучасним потребам.

З появою веб-технологій нового покоління та зростанням популярності JavaScript почалася епоха гібридних мобільних застосунків, які пропонували зовсім інший спосіб розробки. Гібридні рішення були відповіддю на потребу спростити процес створення мобільних додатків і зробити його доступнішим для ширшого кола розробників, коли нативні технології були спрямовані на повний контроль над платформою. Одним із перших значущих інструментів стала платформа PhoneGap, яка пізніше трансформувалася в Apache Cordova(Рис. 2.2)[13][14]. Основна ідея була досить простою, але революційною для свого часу: розробник створює інтерфейс за допомогою HTML, CSS і JavaScript, замість того, щоб писати код окремо для Android і iOS. Цей інтерфейс розгортається в WebView мобільного пристрою, а система плагінів дозволяє йому взаємодіяти з функціями пристрою.

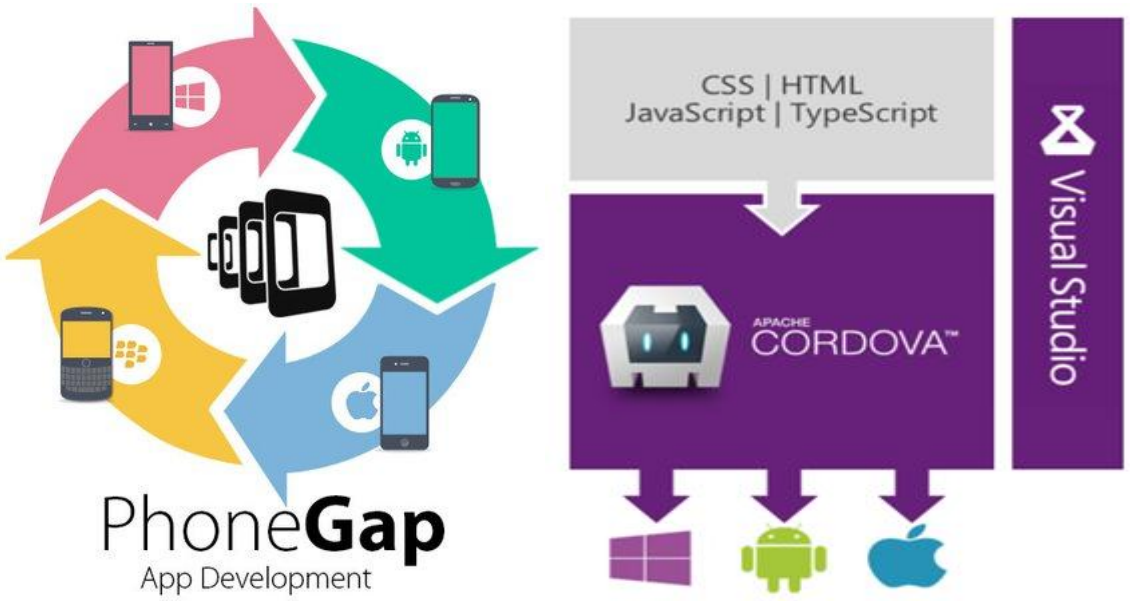


Рис. 2.2 – PhoneGap та Apache Cordova

Цей метод дозволив основній масі коду залишатися незмінною, він дозволяв значно швидше створювати застосунки , що працюють на різних платформах. Це дійсно стало проривом для багатьох компаній що вже мали досвід веб-розробки. Тож зважаючи на те , наскільки простим було створення гібридних додатків за допомогою PhoneGap і Cordova , не дивно що вони набули видкої популярності , адже вони мали нижчий поріг входу ніж нативна розробка, окрім того , веб-розробники змогли створювати мобільні додатки без необхідності вивчення Swift або Java.

Проте недоліки були теж очевидними. Основною проблемою тут була продуктивність , адже інтерфейс WebView не був розроблений для мобільних додатків із вимогами до високої швидкості. Це призводило до того що такі інтерфейси іноді могли працювати повільно.

Незважаючи на недоліки, гібридні рішення відіграли важливу роль у розвитку мобільної розробки. Вони продемонстрували, що кросплатформність є актуальною, і що веб-технології можна використовувати для створення веб-сайтів і мобільних пристроїв. Наступні покоління технологій, створених Cordova та подібними інструментами, продовжили концепцію кросплатформності, але усунули більшість недоліків веб-підходу. Багато сучасних фреймворків все ще

частково базуються на ідеях, які вперше з’явилися в гібридних рішеннях, наприклад, єдиний інтерфейс розробки, спільний код для кількох платформ і взаємодія з API за допомогою уніфікованого інструментарію.

Після появи сучасних кросплатформних платформ, таких як React Native та Flutter, розвиток мобільної розробки визначив новий стандарт створення застосунків, які працюють на Android і iOS. Ці технології з’явилися як відповідь на потребу забезпечити високу швидкість розробки, при цьому зберігаючи продуктивність і зручність використання. Flutter і React Native продовжили концепцію кросплатформності, започатковану гібридними технологіями, і значно вдосконалили її, вирішивши велику кількість проблем, пов’язаних із продуктивністю, взаємодією з користувачем і доступом до апаратних функцій пристрою.

Створений Facebook React Native привернув увагу тим, що дозволив використовувати JavaScript, одну з найпоширеніших мов програмування, у мобільній розробці[15]. Використання нативних елементів інтерфейсу є його основною характеристикою. На відміну від гібридних варіантів React Native не відтворює інтерфейс користувача у WebView. Натомість компоненти платформи відтворюються у віртуальній машині, а JavaScript-код зв’язується з нативною частиною через спеціальний міст. Таким чином було досягнуто значно кращої продуктивності порівняно з Cordova. Завдяки цьому інтерфейс виглядає органічно, відгукується швидко, а більшість візуальних елементів працюють так само, як у нативних застосунках.

Основною перевагою React Native є те що він базується на підході React – компоненти та управління станами. Це робить код легким для розуміння і простим для підтримки. За багато років розвитку технології React ми отримали безліч бібліотек для анімацій , HTTP – запитів , кешування , внтеграці API, та інше. Тож React є дуже хорошим інструментом , якщо проект потребує швидного розгортання , для роботи в команді , окрім того він значно полегшує роботу з

зовнішніми сервісами, це дуже важливо для проектів, що включають багато API-запитів чи картографічні платформи.

Flutter, розроблений Google, пропонує новий підхід до мобільної розробки. Він використовує власний високопродуктивний рендеринг-движок Skia та мову Dart[16]. Flutter не покладається на наявні компоненти інтерфейсу користувача, на відміну від React Native. Сам фреймворк створює інтерфейсні елементи, включаючи списки та кнопки, навігацію та системні контролі. Це дозволяє повністю уніфікувати зовнішній вигляд на всіх пристроях, а також створювати інтерфейси будь-якої складності, не обмежуючись можливостями нативних компонентів. Flutter став популярним у проектах, де висока візуальна складність, стабільність анімацій і загальна плавність роботи є важливими.

Використання архітектурного підходу до оновлення інтерфейсу, який базується на швидкому перерисовуванні компонентів, є ще однією перевагою Flutter. Це дозволяє легко підтримувати великі проєкти з динамічними елементами, уникнути складних взаємодій між логікою та користувачем і забезпечити правильну реакцію на зміни стану. Крім того, Flutter входить до числа найактивніших екосистем мобільних технологій завдяки постійним оновленням, плагінам, створеним спільнотою, і великій кількості навчальних матеріалів, які роблять framework доступним для нових розробників.

У підсумку можна сказати, що React Native та Flutter стали ключовими інструментами сучасної мобільної розробки, оскільки пропонують кросплатформність без суттєвих компромісів (Рис. 2.3). Вони дали змогу значно пришвидшити процес створення мобільних застосунків, зменшити витрати та зробити мобільну розробку доступнішою для ширшого кола фахівців. І хоча кожен із фреймворків має свої переваги та недоліки, їхнє поєднання продуктивності, зручності та універсальності робить їх одними з найперспективніших технологій сьогодення.

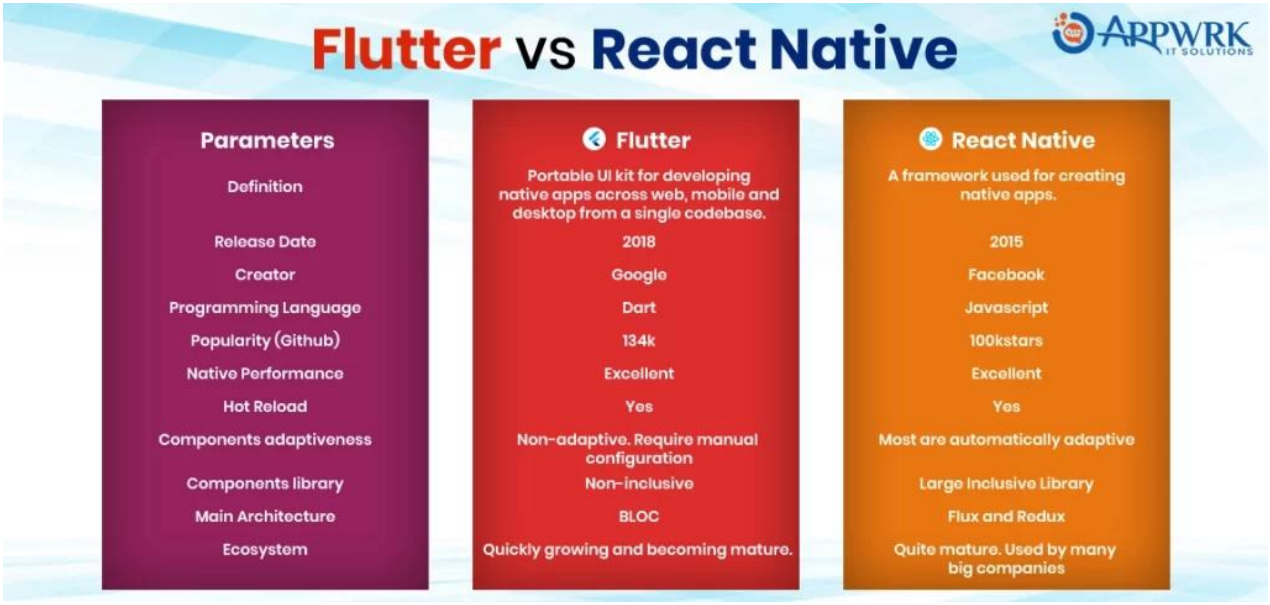


Рис. 2.3 Порівняння React Native та Flutter

Тож можу сказати що немає абсолютно кращої технології, оскільки кожна з них використовується для вирішення різноманітних завдань. Тож потрібно обирати в залежності від потреб проекту. В моєму випадку я обрав React Native, так як в мене є досвід розробки на React, а тут принцип той самий, також мені дуже сподобалось кількість різноманітних бібліотек що можна використовувати для розробки.

2.2 Огляд картографічних API

API картографічних серрвісів є важливою частиноу сучасних мобільних застосунків, оскільки вони забезпечують базу для візуалізації реального простору в цифрових середовищах, а також для роботи з геолакацією пошуком місць та побудовою маршрутів. На сьогодення карти перестали бути допоміжним інструментом, а перетворилися на сервіс який поєднує фото, рейтинги та відгуки. Раніше карти дозволяли лише базове масштабування та побудову маршрутів. Однак сучасні сервіси стали високо технологічними, які інтегрують різні джерела даних наприклад супутникові знімки. Очікування користувачів завдяки розвитку мобільних пристроїв, мокращенню GPS та мобільного інтернету підвищились,

тож сьогодні будь-який застосунок повинен відображати не тільки карту , а й інтелектуальний пошук , зручну навізацію та детальну інформацію про тещо шукає користува.

Сьогодні Google Maps є провідною картографічною платформою, хоча Mapbox, Here Maps, OpenStreetMap і Google Maps є іншими альтернативами[17][18]. Зображення карт, визначення локації, геокодування, маршрутизація, відображення даних про бізнес, аналіз трафіку в реальному часі, взаємодія з панорамами Street View, пошук місць, фотографії, рейтинги, категоризація та навіть контекстні підказки є частиною його інфраструктури. Мобільні програми можуть досягати рівня продуктивності, який практично неможливо досягти за допомогою інших платформ завдяки глибині інтеграції цих сервісів. Найбільшу у світі картографічну базу, створену Google Maps Platform[19], складається з мільярдів об’єктів і поповнюється різними джерелами, включаючи користувацькі дані, офіційні бізнес-профілі, партнерські дані, геопросторові фотоматеріали, соціальні сигнали та аналітичні алгоритми, які оцінюють популярність місця на основі поведінки користувачів. Таким чином, Google Maps є найточнішою та найповнішою картографічною платформою, надаючи розробникам постійний доступ до актуальних, інтегрованих даних про локації.

Для розробників Google Maps має найкращий набір інструменів , високу деталзацію фото, характеристики, відгуки, години роботи закладів POI. Таким чином такі застосунки як цей можуть бути створені повністю на основі екосистеми Google. Інші платформи, такі як Mapbox, Here Maps та OpenStreetMap, залишаються конкурентоспроможними, але не можуть повністю замінити Google у сферах, які вимагають великої бази POI, актуальних фотографій і точних алгоритмів пошуку. Наприклад, Mapbox має слабшу базу закладів, але пропонує широкий спектр можливостей кастомізації стилів; Here Maps має надзвичайно точну навігацію, але фокусується переважно на транспортних застосунках; OpenStreetMap пропонує відкриті дані, але не має фотографій, відгуків і комерційної підтримки. Тому при створенні системи,

орієнтованої на інтелектуальний пошук та AI-інтерпретацію запитів, вибір Google Maps Platform стає найбільш логічним і технічно виправданим.

Google Maps забезпечує чудовий рівень атрибутивних даних. Кожен POI на платформі містить сотні полів, включаючи назву, категорію, тип місця, адресу, координати, години роботи, популярні години, платіжні системи, контактні дані, атрибути доступності, тип кухні, діапазон цін, рівень шуму, фотографії, середній рейтинг, динаміку змін оцінок, оцінки Google Local Guides та інші метрики. І Mapbox, і Here, і OpenStreetMap не мають такої деталізації, оскільки вони або не містять фотографій, мають погану категорію, або не збирають користувацькі рейтинги. Оскільки модель штучного інтелекту може не лише інтерпретувати первинний запит, але й аналізувати повернуті атрибути, порівнювати їх із намірами користувача, повторно ранжувати результати та створювати розширений опис місця, розвиток структури POI є важливим для застосунків, які поєднують штучний інтелект і картографію(Рис. 2.4).

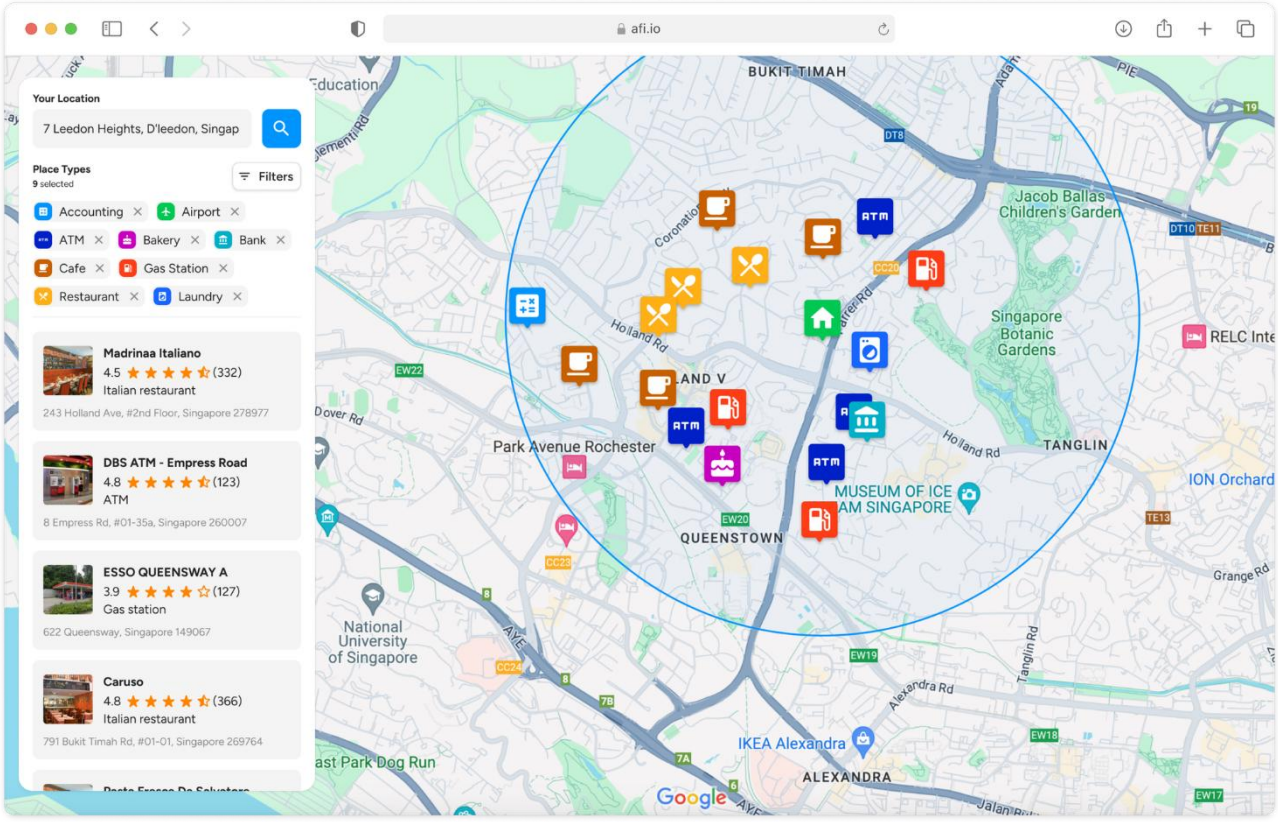


Рис. 2.4 Google Nearby API

З технічної точки зору Google Maps SDK взаємодіє з мобільним застосунком як нативний елемент, що дає змогу швидко рендерити карти і працювати безперебійно навіть на пристроях з обмеженими ресурсами. Бібліотека React Native Maps використовує нативні модулі Google Maps для Android та iOS для виконання цього. Такий метод гарантує стабільність, високу точність визначення локації та мінімальну затримку при взаємодії з картою.Додаток може використовувати Place Photos API для доступу безпосередньо до вотографій що будуть інтегруватись у великій якості. Це дозволить створити візуально привабливий інтерфейс.

Якщо порівнювати Google Maps з іншими подібними платформами стає зрозуміло чому саме його я обрав його для свого проекту. Хоч Mapbox пропонує гнучкість у візуалізації дизайну карт, проте він не має достатньо потужної бази POI. Оскільки Mapbox покладається на базу OpenStreetMap замість власної інфраструктури. Here Maps має чудову навігаційну модель, але він працює погано з закладами та немає достатньої кількості фото. Незважаючи на те, що OpenStreetMap є відкритим, безкоштовним і універсальним, він не містить рейтингів, відгуків, фотографій, актуальних графіків роботи або подробиць бізнес-атрибутів, а його дані значною мірою залежать від активності локальної спільноти. Саме деталізовані, актуальні та структуровані дані Google Places є найкращим варіантом для реального застосування, яке використовує AI для інтерпретації текстових запитів. Це пов’язано з тим, що AI може ефективно зіставляти наміри користувача з десятками реальних характеристик кожного закладу.

Google Maps має масштабну архітектуру що робить її найкращим виборо для використання у якості API, адже він може забезпечити не тільки візуальну карту , ай якісні фото , структуровану інформацію про заклади. Завдяки своїй архітектурі Google Maps не просто надає доступ до геоданих; він також створює багатогранну інформаційну модель світу, яку можна інтегрувати з AI для створення нових систем, орієнтованих на точність, сенс і персоналізацію.

З технічної точки зору взаємодія між AI та Google Maps вимагає точних запитів. AI-модель повинна «перекласти» природномовний текст, наприклад «хочу місце біля води для ранкового бігу», у різноманітні параметри, такі як place_type (парк, стежка), keywords (ріка, берег), radius (залежно від контексту), location bias (поточне місцезнаходження користувача) і іноді навіть додаткові фільтри (міська зона, зелена зона). Це перетворює AI на інтелектуальний шар перекладу та інтерпретації запитів користувача в зрозумілу мову для картографічних сервісів, а API на виконавчий модуль із визначеними параметрами. Цей механізм повинен пам’ятати про обмеження Google Places API, оскільки він має фіксований перелік типів місць, тому запити, які є надмірно складними або емоційними, не можуть бути виконані без семантичного розкладу(Рис. 2.5). Наприклад, штучний інтелект може зіставити запит «романтичне місце для вечері» з типом ресторану, рейтингом і фотографіями, а потім передати певні параметри в Google Places, хоча запит не є категорією в API. Цей процес показує важливу особливість сучасних інтелектуальних систем: API більше не визначають логіку; вони є лише частиною великої багаторівневої структури, яка дозволяє AI організовувати смисл запиту.

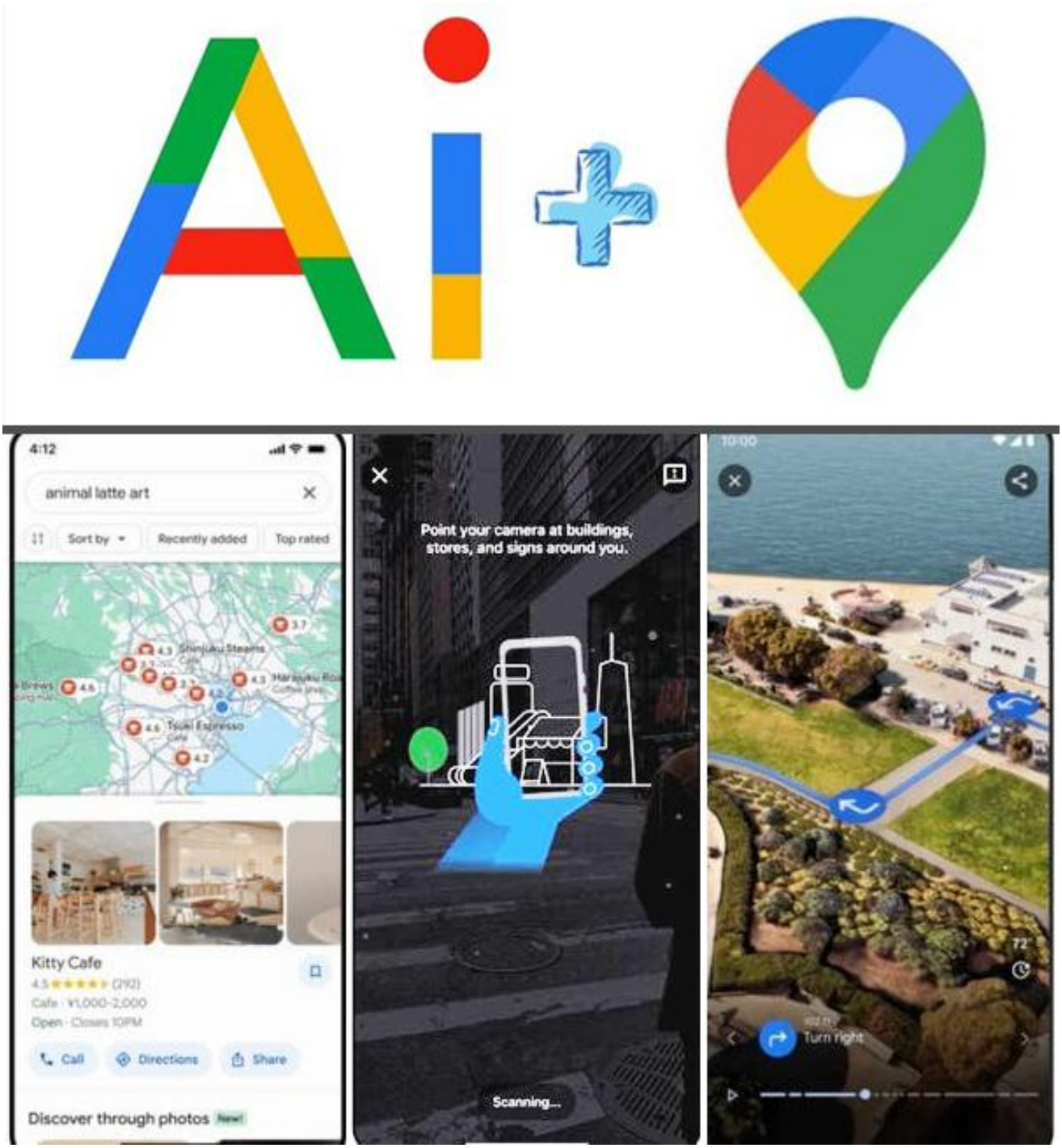


Рис. 2.5 Приклад комбінації Google Maps та AI

Підбиваючи підсумки картографічні сервіси давно перестали буди просто інструментом для навігаці. На тепер це платформи, які мають співпрацювати зі штучним інтелектом і створювати основу нових поколінь сервісів, що спрямовані на взаємодію та персоналізацію запитів користувачів. Google Maps Platform ідеально підходить для застосунків, у яких AI-інтерпретація запитів і рекомендована логіка є важливими, завдяки своїй точності, складній структурі

POI, багатій екосистемі API та стабільному SDK. Інтелектуальний пошук, динамічне ранжування та гнучкий інтерфейс, адаптований до потреб користувача, які пропонує платформа, стали основною технологією для архітектури цього проекту.

2.3. Огляд AI-моделей та технологій штучного інтелекту

Розвиток AI був природним результатом збільшення обчислювальної потужності, обсягів даних і потреби автоматизувати складні процеси, для яких раніше потрібна була безпосередньої участі людини. Дослідження в галузі штучного інтелекту спочатку зосереджувалися на символічних системах, логічних правилах і експертних моделях. Однак з часом обробка даних, статистичні методи та машинне навчання стали більш важливими. Це дозволило створювати моделі, які не тільки виконують заздалегідь запрограмовані дії, але й навчаються на прикладах і знаходять приховані закономірності.

Створення штучних нейронних мереж, які стали основою сучасних алгоритмів машинного навчання, стало справжнім проривом у розвитку штучного інтелекту. Вони дозволили ефективно працювати з великими кількостями даних і імітувати деякі частини роботи людського мозку. З іншого боку, такі моделі тривалий час були обмежені як апаратними можливостями, так і методами навчання. Поява графічних процесорів (GPU), здатних обробляти великі обчислення паралельно, сприяла їхньому стрімкому розвитку. Це дозволило тренувати складні моделі на даних масштабів, які раніше були недоступні.

Розвиток ШІ призвів до появи великої кількості мовних моделей: GPT, PaLM, LLaMA, Claude, Gemini та інших[20]. Усу ці моделі навчаються на неймовірно великих датасетах, які складаються з книжок, різноманітних статей, наукових матеріалів , реальних діалогів, тощо. Штучний інтелект вийшов за межі

класифікації та став інструментом, здатним доповнювати людську діяльність у аналітичних, творчих та інженерних процесах з появою LLM.

Моделі, які працюють з зображеннями, відео, аудіо та текстом, були створені завдяки розвитку штучного інтелекту. Це ще більше розширює можливості мобільних застосунків, включаючи аналіз фотографій, автоматичну класифікацію контенту, розпізнавання об’єктів, інтерпретацію сцен і розпізнавання тексту на зображеннях. Це дозволяє сучасним мобільним системам досягти рівня продуктивності, який раніше був обмежений високопродуктивними серверними системами.

Тож сучасний ШІ – це масштабна система, щозмінює підхід до розробки мобільних застосунків. Він дає змогу створювати нові інтелектуальні функції, так покращити розуміння користувача , Таким чином , смартфони стали не лише інструментами для обробки даних, але також й системами, що дають змогу людям приймати рішення та знаходити потрібну інформіцю з меншою кількістю зусиль.

GPT — серія моделей OpenAI — була однією з перших мовних моделей, яка продемонструвала потенціал трансформерів у масштабах індустрії. Найвідоміша GPT-3 мала 175 мільярдів параметрів і навчалася на великій кількості текстів. Вона може створювати програмний код, виконувати логічні завдання, генерувати текст, перекладати та імітувати розмову на рівні, близькому до людського. Наступні покоління GPT продемонстрували значні покращення завдяки ретельному налаштуванню, RLHF (навчанню з підкріпленням від людського зворотного зв’язку) і покращеній структурізації даних. Це дозволило моделям бути не лише потужними з точки зору параметрів, але й більш узгодженими, безпечними та функціональними (Рис. 2.6).

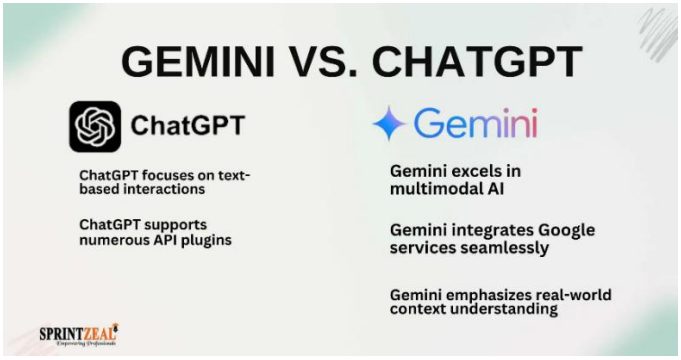


Рис. 2.6 Порівняння ChatGPT та Google Gemini

Серія моделей PaLM від Google стала альтернативою GPT. Мета PaLM 2 і наступних ітерацій полягала в тому, щоб поєднати багатомовність, логічні здібності та здатність до аналізу коду в одній моделі. Google активно працює над мультимодальною здатністю моделі, яка може працювати з зображеннями, аудіо та іншими типами даних, окрім тексту. У результаті цього прогресу була розроблена модель Gemini, яка була наступним кроком у розвитку Google AI. Мультимодальність, висока точність і здатність виконувати завдання різних класів – від математичного аналізу до генерації коду, аналізу фотографій і інтерпретації відео – є унікальними характеристиками Gemini. Цей метод робить моделі Google універсальними інструментами для аналітиків високого класу.

Спосіб навчання мовних моделей також є важливим. LLM навчаються на величезній базі даних, яка включає книги, статті, тексти, документацію, форуми та багато іншого. Після основного навчання моделі проходять інші етапи налаштування, такі як наглядова точність, RLHF, інструкційне навчання, мультимодальне налаштування та оптимізація для вузьких завдань. Сучасні моделі можуть не тільки відтворювати текст, але й правильно виконувати складні інструкції, розуміти контекст і враховувати наміри користувача завдяки цьому.

LLM відкрили можливості, які раніше здавалися неможливими. Застосунок може розуміти природну мову, аналізувати довгі запити, знайти ключові параметри, створювати індивідуальні рекомендації та створювати інтелектуальні відповіді. Завдяки появі легких inference-моделей, оптимізованих для мобільних процесорів, тепер можна виконувати певні AI-функції на місці, не

підключення до серверів. Це особливо важливо, коли йдеться про затримку, приватність або роботу офлайн.

Великі мовні моделі стали універсальним інструментом, що дає змогу реалізовувати високорівневі інтелектуальні функції у мобільних застосунках[20]. Вони дають змогу інетерпретувати живу мову в діалог що дозволяє користувачам вирішувати задачі що перед ними стоять а також надавати подальші рекомендації. Сучасний розвиток LLM стверджує, що ці моделі стануть основою для подальшої еволюції мобільних платформ і відкриють нові можливості для створення застосунків, які відповідають потребам користувачів і є інтелектуальними асистентами, які справді працюють.

Одним із основних підрозділів штучного інтелекту є природномовна обробка (NLP), яка забезпечує взаємодію на основі мови між цифровими системами та людьми. Від простих статистичних моделей і словникових алгоритмів до сучасних трансформерних архітектур лінгвістична програма може аналізувати, інтерпретувати та генерувати текст на рівні, що наближається до людської комунікативної здатності. Цей прогрес відбувся протягом багатьох років. Процес обробки мови (NLP) мав вирішальне значення для створення систем, які здатні не просто виконувати певні інструкції, але й розуміти смислову структуру висловлювань, інтерпретувати наміри користувача та адаптуватися до конкретного контексту.

Справжнім проривом стало використання нейронних мереж у NLP. Спочатку домінували рекурентні мережі — LSTM і GRU, які могли працювати з послідовностями тексту та зберігати контекст у короткочасній пам’яті. Проте їхня здатність утримувати контекст обмежувалася технічними особливостями архітектури: вони «забували» попередні частини тексту та не могли ефективно працювати з довгими абзацами. Поява трансформерів докорінно змінила підхід — моделі отримали здатність аналізувати весь текст одночасно, враховуючи глобальні контекстні зв’язки. Механізм attention став ключем до того, щоб моделі

могли розуміти не лише окремі слова, а й семантичні зв’язки між ними, смислові структури та логіку висловлювань.

Семантичний пошук (semantic search)(Рис. 2.7) — один із найважливіших продуктів цього ШІ. На відміну від класичного пошуку, який працює за збігом ключових слів, семантичний пошук порівнює смислові значення запитів із текстами у базі даних[20]. Це означає, що система може знайти релевантний результат навіть тоді, коли слова у запиті не збігаються з тими, що містяться у даних. Наприклад, запит «місце для спокійної роботи в центрі міста» може бути зіставлений із POI-закладами, де є Wi-Fi, тиха атмосфера, розетки та комфортні сидіння, навіть якщо в описах цих закладів немає слова «робота». Саме так функціонують сучасні рекомендаційні системи та AI-помічники.

Рис. 2.7 Семантичний пошук

Ще одним важливим напрямом NLP є класифікація намірів (intent classification). Модель здатна визначити, що саме хоче користувач: знайти ресторан, побудувати маршрут, отримати інформацію про години роботи, знайти спокійне місце або попросити рекомендацію. Інтерпретація намірів стала ключовою складовою мобільних застосунків, де користувач часто формулює запити природно, без структурованих команд. Завдяки NLP мобільний застосунок може аналізувати такі запити та перетворювати їх у набір параметрів:

тип закладу, атмосферу, фільтри, радіус, час доби, рівень активності або навіть настрій користувача.

На тепер NLP виконує додатково ще одну функцію – узагальнення (summarization) текстів. Це дає змогу мобільним додаткам перетворювати до прикладу вели списки відгуків чи характеристик закладів. Можель може написати короткий опис певного закладу : «прекрасна тераса», «Затишне місце»,замість 100 відгуків. Це має значно покращити користувацький досвід , та дозволяє оптимізувати інтерфейс.

Мобільні пристрої є основним засобом доступу до інформації та взаємодії людини з цифровим світом, тому інтеграція штучного інтелекту в мобільні застосунки стала ключовим напрямом розвитку сучасних технологій. З появою великих мовних моделей і мультимодальних систем мобільні застосунки отримали можливості, які ще кілька років тому здавалися недосяжними. Ці можливості включають інтелектуальну інтерпретацію запитів, контекстний пошук, планування на основі семантичних ознак, динамічний аналіз поведінки користувача, автоматичну класифікацію даних і персоналізовані рекомендації. Це призвело до суттєвої зміни парадигми розробки мобільних пристроїв, оскільки тепер існують системи, які можуть швидко визначати бажання користувачів і реагувати на них.

Інеграція ШІ в мобільні застосунки починається з побудови архітектури , що визначає як клієнт взаємодіє з ним. Най більш поширена модель це використання сервісних API які дозволяють застосункам надсилати запити і отримувати віповідь від моделі. Такий тип підключення дуже поширений , адже LLM вимагає значних ресурвів комп’ютера: потужна графічна карта, багато оперативної па’яті і тд. Це все підходить не для всіх розробників , тож завдяки архітектурі API можна уникнути цих проблем.

Такі системи штучного інтелекту потребують оптимізації мережевих запитів , щоб покращити швидкість відповіді та уникнути великих затримок.

Якість досвіду користувача залежить від того як побудована вза’ємодія між клієнтом та серверов.

У можму проекті ШІ відграє дуже важливу роль , оскільки він обробляє запит користувача що ввели людською мовою таструктурує дані для подальношії обробки картографічними API. AI працює як «інтелектуальний фільтр», розуміючи бажання людини та змінюючи карту відповідно до її потреб. У результаті ваш застосунок не просто шукає місця, а «спілкується» з користувачем і розуміє, що він хоче знайти.

Підбиваючи підсумки імплементація AI в мобільний застосунок є стрімкозростаючим трендом у сучасній розробці. ШІ робить мобільні сервіси гнучкими та дуже адаптовані для користивача , і робить його впевненим в тому, що практично кожен його запит буде інтерпретовано правильно.

РОЗДІЛ 3. РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

3.1. UI/UX ПРОЄКТУВАННЯ

Розробка користувацького інтерфейсу є також одним з ключових майлстоунів у розробці доадатку , адже саме від нього залежить те, як користувач буде сприймати систему, на скільки зруччно йому буде нею користуватись, та як швидко він зрозуміє як користуватись і взаємодіяти з додатком. Користувацьки досвід у цій роботі поляггає в тому , що потрібно визначити і сформувати метеоди, як зменшити навантаження на користувача під час користування додатком. Тож дуже важливо створити простий і функціональний інтерфейс що буде ще й візуально привабливим і дозволяти виконувати всі потрібні функції.

Моя основна ідея полягає в тому, щоб створити простий , проте інформативний інтерфейс,що буде дозволяти користовачу зосереджеватись на досягненні своєї мети , а не на розборі того , що і як працює. У додатку користувач буде використовувати мову , якою й спілкується для взаємодії з штучним інелектом просто в полі пошуку. Це значить що інтерфейс повинен бути достатньо адаптивним щоб відповідати всім потребам.

1. Кафе поруч
2. Спокійне місце для роботи
3. Ресторан з терасою
4. Парк біля води для ранкових пробіжок

Таким чином інтерфейс буде фокусуватися на пошуковій області, а також карти на якій будуть відображені маркери з місцями. У той час як ця структура надає відчуття простоти, вона також дозволяє отримати доступ до додаткових функцій, коли це потрібно.

Також наступним кроком стало розроблення карток з результатами пошуку, адже це й дає перше враження користувачеві після завершення стадії пошуку.

Картки містять назву закладу та його рейтиг. Це дозволяє користувачеві визначити, чи відповідає місце його первинним очікуванням.

Створення головного екрану почало беззупинну роботу над проектом – це місце де користувач взаємодіє з пошуком, виконується перехід на карту та взаємодія з ШІ. Тож таким чином головний екран був розроблений згідно з цих всіх вимог, щоб максимально приблизитися до бажаного результату.

Головний екран додатку складається з таких компонентів :

- Текстове поле для введення закладу
- Рекомендації для швидкого пошуку
- Меню додатку (буде знаходитись на кожному екрані)

Така структура здається дуже простою , як на менетак і має бути , щоб максимально сфокусувати увагу користувача на основному. Пошуковий рядок працює як інтерфейс між користувачем і AI, аналізуючи запити, визначаючи важливі наміри, структуруючи категорії та визначаючи найкращий спосіб взаємодії з Google Places API. Це робить його незвичайним інструментом (Рис. 3.1).

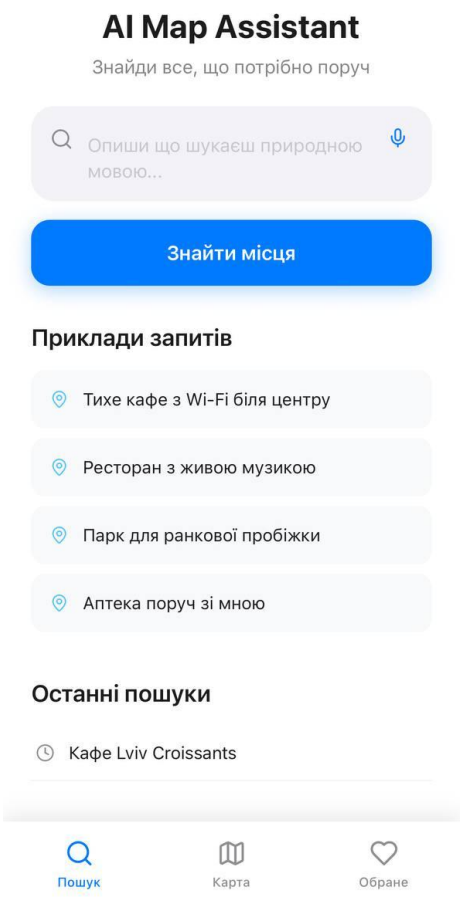


Рис. 3.1 Головний екран з фокусом на поле пошуку

Другий екран буде показувати інтерактивну карту , з маркерами які відображають результати пошуку , а також блоком де буде відображатися картки POI. Карта не просто показує координати , а служить динамічною візуалізацією оточення користувача[23]. Картки POI в свою чергу показують які саме локації обрав ШІ та виводять базову інформацію таку як :

1. Фото
2. Назва
3. Рейтинг

Це також дуже важливий елемент адже саме завдяки ньому користувач буде розуміти що за маркери перед ним , а також отримати первинну потрібну інформацію(Рис. 3.2).

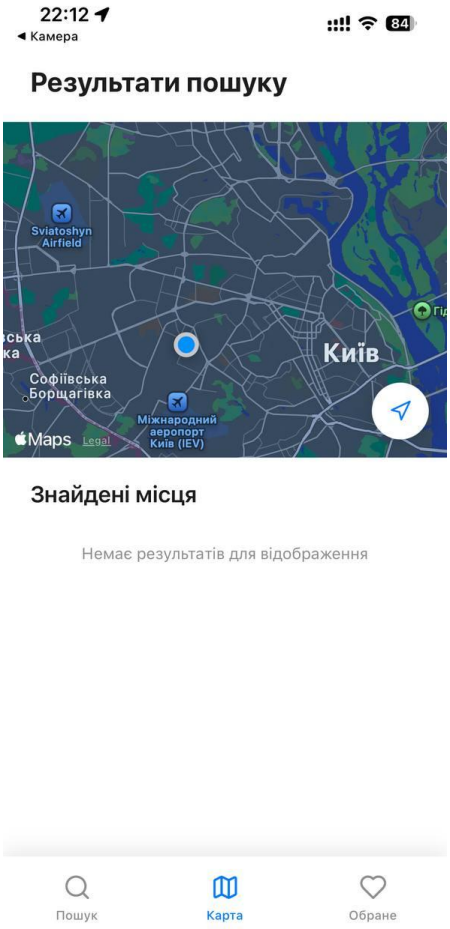


Рис. 3.2 Інтерфейс екрану з картою

Додавання маркерів на карту є також дуже важливим елементом UX , адже саме завдяки ним користувач розуміє , де саме знаходиться шукана локація. Міркери повинні бути помітними , проте не надто великими щоб не перенавантажувати карту. Система групує результати в кластери , коли їх багато , а при збільшені карти починає деталізовувати ці маркери та додавати нові точки на карті. Це допомагає користувачу швидше визначити, де розташовані більшість релевантних місць, одночасно запобігаючи хаотичному нагромадженню(Рис. 3.3).

Рис. 3.3 Відображення маркерів на карті

Кожен елемент UI побудований так щоб сконцентрувати увагу користувача на основному , а саме кожен елемент забезбечує тільки ту інформацію що потрібна користувачу наданому етапі.

Трохи детальніше про картки результатів, вони повинні бути також візуально приваблими, щоб це привертало увагу користувача та повинні надавати всю потрібну інформацію , потрібно враховувати те що екран телефону не дуже великий тож , надто велика кількість інформації буде перенавантажувати додаток.

Анімації, які значною мірою підвищують враження від карток, є важливим компонентом UX. Усі кнопки, включаючи відкриття картки, підняття, свайп, розгортання та плавне переміщення до повної сторінки, розроблені таким чином, щоб зменшити навантаження на мозок і зробити взаємодію природною. Анімації не є декоративними; натомість вони служать підказкою, допомагаючи користувачеві зрозуміти, що відбувається в системі у відповідь на його дії. Критично важливо, щоб всі ці компоненти працювали швидко, навіть на середніх пристроях. З цієї причини структура картки була оптимізована під React Native[21], а фотографії підвантажувалися поетапно, а не одразу.

Тож таким чином картки результатв ж не просто інформаційним блоком а також важливим UX принципом, щодозволяє взаємодіяти між даними, штучним інтелектом і намірами користувачів, також це дозволяє користувачам приймати рішення швидше та природніше завдяки поєднанню візуальної та аналітичної інформації.

Сторінка детального перегляду закладу стала ключовим компонентом копистувацького досвіду, який визначає ступінь взаємодії з застосунком. У цьому місці інформація про місце є повністю розкритою, що дозволяє користувачу перейти з етапу попереднього ознайомлення до етапу прийняття рішення та отримати повне розуміння особливостей місця. Сторінка деталізації відрізняється від карток результатів, які є невеликими інформаційними модулями. Натомість вона створює повноцінне середовище аналітики, де кожен елемент інтерфейсу продуманий таким чином, щоб користувач міг швидко та без проблем отримати максимум даних. Оскільки AI та Google Places API забезпечують релевантність, точність і глибину даних, які користувач отримує після натискання на картку або маркер, UX цього екрану тісно пов’язаний з ними.

Візуальний дизайн сторінки формувався за принципом «інформаційної ієрархії», коли фотографічний блок є першим враженням. Зображення є основним носієм атмосфери місця, тому в цьому розділі буде представлено галерею або широкоформатне зображення, які містять найкращі зображення, доступні з Google Places. Важливо, щоб галереї підтримували жестові взаємодії та змінювали свою поведінку відповідно до типу зображень. Наприклад, у ресторанах пріоритет отримують фотографії інтер’єру та страв, у коворкінгах — світлі місця та робочі зони, а в парках — краєвиди та пішохідні маршрути. Це додає глибокий візуальний шар UX, що дозволяє користувачеві відразу розуміти атмосферу місця, не читаючи текст. На додаток до цього галерея має інтегровані індикатори AI, які можуть підсвічувати важливі фотографічні елементи, такі як робочі столи, панорамні вікна, бігові доріжки та зони відпочинку(Рис. 3.4).

Рис. 3.4 Інтерфейс сторінки з детальною інформацією

Під блоком де знаходиться візуальна інформація, знаходиться блок де описується вся потрібна користувачеві інформація про заклад. Сюди входить аналіз запиту корисувача за допомого AI де ШІ в свою чергу комплексно досліджує запит , та описує саме ту інформацію що користувачу потрібна , наприклад «ресторан з терасою» після фільтрації всіх елементів , нейронна мережа напише опис від себе та дасть характеристику закладу згідно з потребами користувача.

На додаток до основної сторінки є інтеграція навігації Google Maps. UX дозволяє користувачеві негайно розпочати подорож, натиснувши лише одну кнопку, коли він ознайомився з місцем. Для завершення сценарію — від рішення до дії — цей компонент є життєво важливим. Важливо, щоб UX враховувала різні ситуації, такі як копіювання координат, відкриття маршруту в окремому застосунку Google Maps і швидке створення пішохідного чи автомобільного маршруту. Усе це побудовано таким чином, щоб це не перевантажувало користувача технічними подробицями.

Таким чином, сторінка детального перегляду є більше, ніж просто інформаційний екран; це інтерактивне середовище, кероване штучним інтелектом, яке допомагає користувачу швидко та впевнено приймати рішення. Використовуючи поєднання візуальних, структурних, семантичних і поведінкових елементів, вона створює цілісний UX, який завершує ланцюг взаємодії, відомий як «запит → AI → карта → карта → рішення → дія». Завдяки цьому екрану застосунок сприймається як сучасний, розумний і продуманий продукт, що покращує весь проект.

3.2 Розробка клієнтської частини (Frontend)

Почнемо з початкового екрану мобільного додатку де реалізовано функціонал пошуку. Спочатку почнемо з усіх необхідних імпортів необхідних бібліотек та функцій для написання коду (Рис. 3.5)

```
import React, { useState } from 'react';
import {
  View,
  Text,
  TextInput,
  TouchableOpacity,
  ScrollView,
  StyleSheet,
  Dimensions,
  StatusBar,
} from 'react-native';
import { SafeAreaView } from 'react-native-safe-area-context';
import { Search, Mic, MapPin, Clock, Star } from 'lucide-react-native';
import { router } from 'expo-router';
```

Рис. 3.5 Імпорти для початкового екрану

Основу стартової логіки застосунку реалізує стартовий фрагмент коду, який також визначає механіку початкової взаємодії користувача з інтерфейсом. Двома основними станами контролює компонент Index: isSearching — індикатор стану активного пошуку, а searchQuery — текст запиту, який вводить користувач. Обидва стани є основою UX-поведінки, оскільки вони регулюють реакцію

інтерфейсу на введення тексту, активацію кнопки пошуку, зміну стилю елементів і відображення процесу «обробки запиту».

Функція handleSearch є гоолвним елементом цього екрану , вана проводить валідацію тексту тому пошук не буде тригеритись , якщо пошукова строка порожня. Потім починається стан isSearching, який пов’язаний зі зміною стану кнопки та показує що система готова для подальшої обробки текстового запиту.

Після того як пошукова логіка завершила свою роботу проводиться перенаправлення користувача до екрану з картою за допомогою router.push. Дуже важливо що запит користувача передається як параметр. Це дозволяє на наступному екрані отримати запит користувача і використати його для аналізу AI. Таким чином ми бачимо що компонент Index.ts виконує роль головного(стартового) компоненту що допомагає користувачу сформувати свій запит та перенаправити його у наступне вікно(Рис. 3.6).

```
export default function SearchScreen() {
  const [searchQuery, setSearchQuery] = useState('');
  const [isSearching, setIsSearching] = useState(false);

  const handleSearch = () => {
    if (searchQuery.trim()) {
      setIsSearching(true);
      // Simulate search delay
      setTimeout(() => {
        setIsSearching(false);
        router.push({
          pathname: '/map',
          params: { query: searchQuery }
        });
      }, 1000);
    }
  };

  const handleVoiceSearch = () => {
    // Voice search implementation would go here
    console.log('Voice search activated');
  };
```

Рис. 3.6 Код логіки пошуку

Далі видно реалізацію самого відображення всієї логіки головного екрану.компонент TextInput відповідає за введення тексту користувачем , саме

звідси бере початок ланцюг роботи всього доадтку та взаємодії з модулем ШІ. Як говорили в попередньому підрозділі, поле має мінімалістичний дизай, що дозволяє зосередити увагу на основному.

Кнопка пошуку в свою чергу змінює свій стан в залежностів від стану IsSearching, вона стає затемненою та відображає текст «Пошук». Це дає важливу користувачеві інформацію що система побачила текст що він ввів та виконує уже певну обробку інформації. (Рис. 3.7).

```
{/* Header */}
<View style={styles.header}>
  <Text style={styles.title}>AI Map Assistant</Text>
  <Text style={styles.subtitle}>Знайди все, що потрібно поруч</Text>
</View>

{/* Search Section */}
<View style={styles.searchSection}>
  <View style={styles.searchContainer}>
    <Search size={20} color="#8E8E93" style={styles.searchIcon} />
    <TextInput
      style={styles.searchInput}
      placeholder="Опиши що шукаєш природною мовою..."
      placeholderTextColor="#C7C7CC"
      value={searchQuery}
      onChangeText={setSearchQuery}
      onSubmitEditing={handleSearch}
      multiline
      maxLength={200}
    />
    <TouchableOpacity
      style={styles.voiceButton}
      onPress={handleVoiceSearch}
      activeOpacity={0.7}
    >
```

Рис. 3.7 Рендеринг інтерфейсу пошукового поля

Наступний блок відповідає за те що є можливість додавання логіки голосового вводу. У поточній версії ця функція не реалізована , проте сам факт наявності такої кнопки дає стиму для подальшої реалізації більш складного процесу за допомогою інтеграції Speech-to-Text технологій.

З точки зору логіки інтерфейсу користувача кнопка «мікрофон» надає користувачеві альтернативний спосіб взаємодії з системою. Це особливо важливо

для мобільних додатків, де введення тексту може бути менш зручним у русі. Дуже важливо, що структура обробника вже закладена в компоненти інтерфейсу; це підкреслює майбутню гнучкість і масштабованість архітектури. (Рис. 3.8).

```
const handleVoiceSearch = () => {
  // Voice search implementation would go here
  console.log('Voice search activated');
};
```

Рис. 3.8 Голосовий пошук (Voice Search)

Блок «Приклади запитів» виконує дві важливі функції:

1. Навчальну: демонструє, як саме AI рекомендується формулювати запити.
2. UX-функціональну: дозволяє запустити пошук у два торкання (Тар → Знайти).

Для таких застосунків, що засновані на природному мовленні, важливо одразу показати користувачу грамотні приклади для натхнення: запити повинні бути не ключовими словами, а описами потреб(Рис. 3.9).

```
<ScrollView style={styles.content} showsVerticalScrollIndicator={false}>
  {/* Example Queries */}
  <View style={styles.section}>
    <Text style={styles.sectionTitle}>Приклади запитів</Text>
    {exampleQueries.map((query, index) => (
      <TouchableOpacity
        key={index}
        style={styles.exampleItem}
        onPress={() => setSearchQuery(query)}
        activeOpacity={0.6}
      >
        <MapPin size={16} color="#5AC8FA" />
        <Text style={styles.exampleText}>{query}</Text>
      </TouchableOpacity>
    ))}
  </View>
```

Рис. 3.9 Приклади запитів

Наступний файл клієнтської частини коду створює логіку опису та розширених даних про отримані дані з Карти.

Перший скриншот коду з файлу place-details відповідає за отримання даних про заклади що були отримані з карти. Цей блок коду створює базу для всього екрану датальної інформації :

- Він визначає чи у системи є достатньо даних для побудови інтерфейсу;
- Описує початкову логіку роботи компонента;

Це є вкрай важливим, оскільки коже застосунок що працює з API повинен правильно обробляти помилки , відсутність параметрів тощо.

Компонент PlaceDetailScreen використовує useLocalSearchParams, що є частиного навігації Expo Routers[22]. Завдяки цій навігації екран отримує параметри, що були передані з карти підчас натискання на картку закладу.Переданий параметр передається у форматі JSON. Першим етапом є перетворення JSON-рядка назад у JavaScript-об’єкт за допомогою JSON.parse().

Після того як ми перетворили дані – виконується перевірка та наявність передаваного об’єкта. Якщо немає валідних параметрів - застосунок не повинен завершувати роботу аварією. Щоб запобігти аваріям ми створили рендеринг fallback з текстом «Місце не знайдено». Такий підхід забезпечує стабільну роботу застосунку, навіть якщо дані що передавались не валідні.

Стан isFavorite також є важливим. Змінна визначає реакцію інтерфейсу на натискання кнопки «В обране». Це правда, що на даний момент збереження в локальній базі ще не реалізовано. Це закладає основу для подальших завдань, таких як синхронізація з локальним сховищем, бекенд і індивідуальні рекомендації.

Саме цей блок коду формує “точку входу” логіки для всього екрану. Він обробляє вхідні дані, захищає екран від непередбачуваних ситуацій, створює початкові стани та готує інтерфейс до роботи. Без цієї частини компонент не міг би функціонувати стабільно, а дані, отримані з карти, не могли б бути коректно відображені на сторінці детального перегляду(Рис. 3.10).

```
export default function PlaceDetailScreen() {
  const params = useLocalSearchParams();

  // Отримуємо place з карти (прийшов як JSON-рядок з MapScreen)
  const place = params.place ? JSON.parse(params.place as string) : null;

  const [isFavorite, setIsFavorite] = useState(false);

  if (!place) {
    return (
      <SafeAreaView style={styles.container}>
        <Text>Місце не знайдено</Text>
      </SafeAreaView>
    );
  }
```

Рис. 3.10 Отримання даних місця та ініціалізація станів

Header цього екрану , виконує важливі функції такі як : навігацйну, інформаційну та інтерактивну. Він є сталим елементом інтерфейсу , що розташовується над головним зображенням й залишається доступним не залежно від того , наскільки далеко користувач прокрутив сторінку.

Кнопка «Вперед» є першим і найважливішим компонентом. Він інтегрується за допомогою функції router.back(), яка дозволяє надати навігацію назад до карти або до попереднього екрану на нативній основі. У сучасних мобільних інтерфейсах користувача традиційним елементом є розміщення кнопки «Поверх» у вигляді напівпрозорої кнопки з округлими краями. Це дозволяє зберігати функціональність, не обмежуючи розмір основної фотографії. Напівпрозорий фон підвищує контрастність і робить елемент читабельним у будь-яких умовах, незалежно від того, світлі чи темні зображення чи колірні.(Рис. 3.11).

```
{/* ------- HEADER ------- */}
<View style={styles.header}>
  <TouchableOpacity
    style={styles.headerButton}
    onPress={() => router.back()}
  >
    <ArrowLeft size={24} color="#fff" />
  </TouchableOpacity>

  <View style={styles.headerActions}>
    <TouchableOpacity style={styles.headerButton} onPress={handleShare}>
      <Share size={20} color="#fff" />
    </TouchableOpacity>

    <TouchableOpacity
      style={styles.headerButton}
      onPress={() => setIsFavorite(!isFavorite)}
    >
      <Heart
        size={22}
        color={isFavorite ? "#FF3B30" : "#fff"}
        fill={isFavorite ? "#FF3B30" : "transparent"}
      />
    </TouchableOpacity>
  </View>
</View>
```

Рис. 3.11 Хедер та кнопки взаємодії

Після компоненту Header знаходиться фотографія закладу. Це один з основних елементів сторінки адже він ідентифікує місце обране користувачем. Логіка відображення фотографії передбачає, що фотографія займає приблизно 35% висоти екрану пристрою, якщо вона знаходиться в структурі місця. Така пропорція була обрана не випадково; вона дозволяє одночасно підкреслити важливість візуального контенту та запобігти затримкам текстової інформації.

Завдяки використанню Dimensions.get("window") головне зображення автоматично підлаштовується під роздільну здатність екрана. Це дозволяє рівномірно відобразити фото на пристроях з різною роздільною здатністю.

Під фото знаходиться інформаційний блок. У цьому блоці відображається основна інформація про місце:

У центрі блоку знаходиться назва (placeName), написана великим жирним шрифтом. Розмір був обраний таким чином, щоб він був якомога більш помітним, але не перевантажував інтерфейс.

Категорія (place.category) має менший шрифт і акцентний колір (#007AFF). Це привертає увагу, але не конкурує з основною назвою. Таке рішення дозволяє користувачеві швидко зрозуміти тип закладу, наприклад, коворкінг, парк, ресторан тощо.

Рейтинг із зіркою — це традиційний, добре відомий UX-патерн. Логічно оцінку округлюють до одного знаку після коми (toFixed(1)). Якщо рейтинг відсутній, показується символ «—», що сигналізує про нестачу даних, але не псує загального вигляду інтерфейсу.

Основне в цьому блоці – це секція «Опис» у якій відображений текст про заклад що сформований штучним інтелекто , та надає ключові переваги закладу згідно з потреб користувача.

Завдяки тому, що опис генерується AI:

- він адаптується до конкретного наміру користувача (робота, відпочинок, романтика, спорт);
- він об’єднує дані про фото, категорію, робочі години та атмосферу;
- він значно підвищує інформативність сторінки та покращує користувацький досвід.

Уся ця логіка в комплексі формує перше враження користувача про місце та забезпечує естетичний, структурований та добре впорядкований інтерфейс (Рис. 3.12).

```
{/* ------- MAIN PHOTO ------- */}
{!!place.image && (
  <Image source={{ uri: place.image }} style={styles.mainImage} />
)}

{/* ------- INFO BLOCK ------- */}
<View style={styles.infoBlock}>
  <Text style={styles.placeName}>{place.name}</Text>

  <Text style={styles.category}>{place.category}</Text>

  <View style={styles.ratingRow}>
    <Star size={18} fill="#FF9500" color="#FF9500" />
    <Text style={styles.ratingText}>
      {place.rating ? place.rating.toFixed(1) : "–"}
    </Text>
  </View>

  {/* ОПИС від AI */}
  <View style={styles.section}>
    <Text style={styles.sectionTitle}>Опис</Text>
    <Text style={styles.description}>
      {place.description || "Опис відсутній"}
    </Text>
  </View>

  {/* CONTACTS */}
  <View style={styles.section}>
    <Text style={styles.sectionTitle}>Контакти</Text>

    <View style={styles.contactItem}>
      <MapPin size={18} color="#8E8E93" />
      <Text style={styles.contactText}>{place.address}</Text>
    </View>

    {place.phone && (
      <TouchableOpacity
        style={styles.contactItem}
        onPress={handleCall}
      >
        <Phone size={18} color="#007AFF" />
        <Text style={[styles.contactText, { color: "#007AFF" }]}>
          {place.phone}
        </Text>
      </TouchableOpacity>
    )}
```

Рис. 3.12 Фрагмент коду — головне фото + базовий інфо-блок

Секція контактної інформації є одним із ключових блоків, реалізованих у цьому файлі. У ньому містяться адреса, номер телефону та актуальний статус закладу («Відкрито» або «Зачинено»). Важливо, що ці дані походять із інформації карти або Google Places API. Крім того, інтерфейс адаптовано, щоб відображати лише елементи, які дійсно доступні. Наприклад, коли телефону немає, відповідний блок автоматично не відображається, що зменшує візуальний шум і робить інтерфейс логічно чистим.

Кожен елемент тут грає велику роль. Адреса супроводжується іконкою що покращує UX, а номер інтерактивний , тож користувач може зробити натиск та система одразу відкриє його автоматично.

Статус роботи закладу, також підсвічується кольорами що покращує UX, адже це зменшує сенсове навантаження на користувача та робить інтерфейс більш простим. Також це підвищує функціональність, адже користувач одназу знає відкритий чи закритий заклад.

Ще одним важливим елементом є панель дій у нижній частині екрана, яка закріплена поверх інтерфейсу й завжди залишається доступною. Вона містить дві основні дії:

– «Подзвонити», якщо доступний номер телефону,
– «Маршрут» — побудова навігації до місця через Google Maps.

Кнопка проведення маршруту дозволяє одразу відкрити застосунок Google Maps та прокласти маршрут прямо в ньому .

Цей екран також містить механізм поширення місця, який працює за допомогою кнопки Share. Хоча її функціонал наразі обмежується логуванням, структура компонента передбачає підключення нативного Share API, що дає можливість пересилати локацію через месенджери, соціальні мережі або пошту. Це суттєво підвищує зручність застосунку у сценаріях групового планування зустрічей, поїздок або пошуку місця для відпочинку.

Окремої уваги заслуговує візуальна ієрархія екрану. Компонент побудований у стилі сучасних мобільних патернів:

– верхнє зображення займає значну частину екрана та задає емоційний фон,
– інформаційний блок піднімається поверх фотографії, створюючи глибину,
– усі секції розділені логічними відступами,
– розмір шрифтів підібраний відповідно до важливості інформації,
– використовуються сучасні акцентні кольори (#007AFF, #34C759, #FF3B30),
– кнопки мають чіткі акценти (primary / secondary).

Тож файл Place-details виконує роль інформатору, показує найважливіше – саме те що користувач бажає побачити. AI- згенерований текст є персоналізованим та підкреслює настрій кожного запити, що допомагає користувачеві обрати заклад швидше.

Файл AI.ts є одним із найважливіших модулів фронтенд-частини застосунку, оскільки він реалізує взаємодію з backend-сервісом, що відповідає за роботу штучного інтелекту (модель Gemini). Саме цей файл з’єднує інтерфейс користувача з AI-механізмами, забезпечуючи семантичний аналіз запиту та ранжування знайдених місць відповідно до потреб користувача(Рис. 3.13).

Файл складається з двох основних функцій:

1. analyzeQuery(query) — інтерпретує природномовний запит користувача.
2. rankPlacesByRelevance(places, keywords) — ранжує знайдені Google Places результати через AI, базуючись на намірі користувача.

Ці функції формують цикл пошуку об’єктів , спочатку форматування людського запиту в зрозумілий для API , а потім уже структурування результатів.

Функція analyzeQuery виконує основну роботу – вона показує ШІ як правильно розуміти запит корстувача , що надсилається на backend , де запускається модель ШІ й повертає два ключові параметри:

- category — базовий тип місця (cafe, restaurant, park тощо),
- keywords — набір семантичних підказок, які AI витягує з тексту запиту (наприклад: "тихий", "з wi-fi", "романтичний", "для роботи", "пробіжка").

Keywords є центральним елементом логіки структурування місць згідно користувалькому запити. Система формує структуру, щоб у будь-якому випадку мати передбачуваний вихід: якщо AI щось не повернув — система гарантує fallback-значення.

Таке рішення забезпечує стабільність застосунку навіть у випадку неідеальних відповідей backend або непередбачуваних формулювань запитів.

rankPlacesByRelevance — AI розбиває результати отримані від ШІ та API картографічних сервісів. Після того як система отримала первинні дані від API Google Places ми визначаємо які найблільше підходять під запит користувача.

Вона надсилає список знайдених місць та keywords до бекенду, де AI аналізує:

- відповідність описів місць ключовим словам,
- категорію об’єкта,
- відгуки (якщо вони передані),
- фото,
- текстові описи,
- семантичний збіг між наміром та POI.

Після цього бекенд повертає відсортований список місць, який фронтенд відображає як результати.

Таким чином, файл AI.ts не тільки виконує функції API клієнта, але й є частиною інтелектуальної логіки застосування, яка служить мостом між інтерфейсом користувача та аналізом штучного інтелекту.

```
export async function analyzeQuery(query: string) {
  try {
    const response = await fetch(`${BASE_URL}/gemini/analyze`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ query }),
    });

    const data = await response.json();
    return {
      category: data.category || null,
      keywords: Array.isArray(data.keywords) ? data.keywords : [],
    };
  } catch (e) {
    console.error("❌ Помилка виклику backend /gemini/analyze:", e);
    return { category: null, keywords: [] }; // fallback для Google Places
  }
}


export async function rankPlacesByRelevance(places: any[], keywords: string[]) {
  try {
    const response = await fetch(`${BASE_URL}/gemini/rank`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ places, keywords }),
    });

    const data = await response.json();

    if (Array.isArray(data)) {
      return data;
    }

    console.warn("⚠️ Некоректний формат ранжування:", data);
    return places;
  } catch (e) {
    console.error("❌ Помилка виклику backend /gemini/rank:", e);
    return places; // fallback
  }
}
```

Рис. 3.13 Фрагмент коду модуля AI.ts

Наступний й найважливіший файл Frontend частини – це map.ts де ми реалізуємо відображення самої карти та карток закладів.

Після відкриття екрану з картою , додаток запитує у користувача дозвіл на пикористання геолокації Якщо користувач дозволяє — застосунок зчитує точні координати, використовуючи GPS, Wi-Fi та мобільну мережу. Після цього Google Places буде шукати найближчі точки інтересу навколо записаних координат у локальному стані(Рис. 3.14)[24].

Потрібно звернути увагу , що застосунок не починає запускати наступні процеси , доки користувач не надасть дозвіл на використання свої даних, таким чином ми не перенавантажуємо систему зайвими процесами, які можливо й не будуть виконуватися.

Крім того, оскільки API Google Places стосується локації, дані, які були отримані, будуть використовуватися і в наступних запитах до Google Places. Це означає, що точне визначення геопозиції впливає на точність результатів. Користувач може отримати або застарілі дані, або результати з іншої частини міста, якщо система помилково визначить його розташування. Таким чином, логіка, представлена в цьому розділі, є основою всієї роботи застосунку.

Цей компонент також використовує реф на карту (mapRef) для анімації центрування в майбутньому, наприклад, коли користувач натискає кнопку «центрувати на мені». Це покращує досвід користування та запобігає тому, що карта залишається в іншій частині міста після переміщення користувача.

Ця логіка запускається лише раз і після того як користувачач надав дозвіл навикористання свої даних наступні процеси починають запускатись , а саме : обробка запиту користувача , отримання відповіді від Google Maps API, та ранжування отриманих відповідей.

```
// ── 1. Отримуємо поточну локацію користувача ──────────────────
useEffect(() => {
  (async () => {
    const { status } = await Location.requestForegroundPermissionsAsync();
    if (status !== 'granted') return;

    const loc = await Location.getCurrentPositionAsync({});
    setUserLocation({ lat: loc.coords.latitude, lon: loc.coords.longitude });
  })();
}, []);
```

Рис. 3.14 Фрагмент коду – отримання доступу до геолокації

Після надання дозволу на обробку даних про свою геолокацію запускається найважливіший процес - обробка запиту. Спочатку аналізується запит за допомого ШІ , та формується відповідь в зрозумілому для API форматі.

Система бере текст що ввів користувач в пошукове поле на головному екрані. У якості запиту може прийматися будь-що до прикладу «Тихе кафе поруч зі мною» або запит з багатьма деталями система буде інтерпретувати будь-який запит та виділяти з нього основну інформацію що потрібна для подальшої роботи застосунку. Оскільки тут можливо ввести будь-який запит то це надає перевеагу над існуючими картографічними рішеннями , адже вони не приймають такі запити і не надають коректної інформації, а підключення ШІ вирішує таку проблему.

З бекенду використовується модель штучного інтелекту Gemini. Модель аналізує текст запиту від користувача та поділяє його на дві частини - категорія локації та ключові слова – характеристики місця[30]. Під категоріями мається на увазі – класифікація локації , до прикладу кафе, ресторан і тд. А ключові слова – це може бути «романтичне» «інтерактивне» «спокійне».

Побудована логіка маж низку переваг , адже вона інтерпретує запит користувача та розбиває на категорії які можна використати в API Google Maps та використовує принцип фільтрації даних за допомогою ключових слів що бере також з тексту запиту користувача. Що дозволяє відобразити маркери на карті та відобразити місця що найбільше підходять до запиту користувача.

Ці дані зберігаються у компоненті та відображаються на екрані щоб користува бачив , що його запит відображається коректно , а також для використання у подальших процесах. Вони в подальшому будуть використані у процесі фільтрації та у створенні опису обраного місця.

Ці елементи є вкрай важливими , адже без AI цей додаток був би просто поганою версією Google Maps, що ще й не видає всіх даних(Рис. 3.15).

```
// ── 2. Коли є локація + запит → виконуємо AI-аналіз і пошук ──────────
useEffect(() => {
  if (!userLocation || !searchQuery.trim()) return;

  (async () => {
    try {
      setLoading(true);
      setLoadingText('Аналізуємо запит користувача за допомогою ШІ…');

      // 2.1. Аналізуємо запит через Gemini
      const aiResult = await analyzeQuery(searchQuery);
      const category = aiResult.category || null;
      const keywords: string[] = aiResult.keywords || [];

      setAiCategory(category);
      setAiKeywords(keywords);
```

Рис. 3.15 AI-аналіз запиту користувача

Коли штучний інтелект уже проаналізував текстовий запит, отримав категорію та ключові фрази, ми можемо переходити до етапу первинного пошуку за допомогою Google Places API.

Цей процес запускається тільки коли виконані попередні умови , та дані записані у стан , на екрані відображається текст де показано що пошук йде через API Google , що дає змогу користувачам зрозуміти що запит в обробці та скоро повинний з’явитися результат.

Реалізація та виклик API вбудована в backend це зроблено через декілька причин. По-перше це захист API ключів для того щоб вони не світились в клієнтськів частині , а також через оптимізацю, бо код стаж менш навантаженим та все буде завантадуватися швидше.

Пошук працює по такому принципу – до API ми дасилаємо координати користувача які отримали раніше , це потрібно для того, щоб отримати дані про реально найближчі місця , а не просто місця в межах міста чи поза його територією чи взагалі в іншій країні , це можна налаштувати за допомогою фільтру по радіусу пошуку. Також окрім цього ми передаємо сам текстовий запит , разом з категорією , адже інколи Google maps також може інтерпретувати прості запити , і видати доволі точну інформацію.

В результаті чого ми отримуємо список потрібної інформації про місця , а саме : назву , координати, відгуки, робочі години і тд. Єдине що ми отримуємо фотографію не на пряму як URL , тож це також потрібно форматувати під потібний нам формат. Це робиться в на сервері , відповідь API форматується в URL формат, після чого відображається в додатку.

Після того як всі минулі кроки завершили свою роботу , ми уже маємо потрібний нам список інформації в конкретному діапазоні що може бути відображений на карті. Проте цього на разі не відбувається в переді ще процес фільтрації , він буде викликатися після того як всі локації прогрузяться та ми отримаємо всю потрібну інформаці(Рис. 3.16).

```
// 2.2. Шукаємо місця через Google Places (бекенд /places/search)
setLoadingText('Шукаємо місця через Google Places…');

const placesRes = await fetch(`${BACKEND_URL}/places/search`, {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    lat: userLocation.lat,
    lon: userLocation.lon,
    keyword: searchQuery,
  }),
});

const rawPlaces = await placesRes.json();
console.log('Raw Google Places response:', rawPlaces);

if (!Array.isArray(rawPlaces) || rawPlaces.length === 0) {
  setPlaces([]);
  setLoading(false);
  return;
}
```

Рис. 3.16 Пошук місць через Google Places API

Після того як цей процес завершив свою роботу ми можемо перейти до наступного процесу який ж також вкрай важливим. API дає відповідь у форматі JSON , але в ньому велика кількість потрібних та не потрібних атрибутів , з яких потрібно витягнути основні дані , обробити їх і тоді уже можна відображати на карті , та створювати опис , це потрібно знову ж таки для того щоб сильно не перенавантажувати систему . Ці дні зводяться до тих що потрібні в нашій системі, Словник Places якраз й містить потрібні нам атрибути що отримуємо з

відповід API . І на основі даних з цього словника ми можемо проводити подальшу обробку.

Процес обробки сирих даних проходить так що ми отримуємо тільки головне , а саме : назву закладу , рейтинг , координати , відгуки і тд. Це дозволяє зменшити навантаження , а якщо якісь певні дані відсутні то виконується fallback для того щоб застосунок не припиняв свою роботу в разі відсутності якихось даних.

Наступним етапом є отримання фото, як ми уже говорили API не видає URL фотографій , а на томість дає photo_refence, тож для того щоб відобразити їх на сторінці , нам потрібно перетворити ці дані на URL. Перетворення робиться за допомогою POST запиту до нашого Backend де реалізована функція претворення інформації з API на URL що буде відоюражена в додатку

Ця логіка дозволяє нам уникнути додаткових запитів до API , що змуншує грошові витрати , також таким чином ми отримуємо фото в хорошій якості , проте якщо фото відсутнє додаток також не перестане працювати адже ми додали fallback що уникнути таких ситуацій.

Після завантаження всіх потрібних даних в наш Place, ми можемо їх використовувати для відображення маркерів на карті , для створення карток закладів, а окрім цього вони будуть використовувати й в подальших процесах(Рис. 3.17).

```
setLoadingText('Готуємо результати…');

const parsed: Place[] = await Promise.all(
  rawPlaces.map(async (p: any) => {
    let imageUrl = '';

    // Фото через бекенд /places/photo (photo_reference → URL)
    if (p.photos && p.photos[0]?.photo_reference) {
      try {
        const photoRes = await fetch(`${BACKEND_URL}/places/photo`, {
          method: 'POST',
          headers: { 'Content-Type': 'application/json' },
          body: JSON.stringify({
            photoReference: p.photos[0].photo_reference,
          }),
        });
        const photoData = await photoRes.json();
        if (photoData?.url) {
          imageUrl = photoData.url;
        }
      } catch (e) {
        console.log('Фото не завантажилось:', e);
      }
    }

    return {
      id: p.place_id,
      name: p.name || 'Невідомий заклад',
      rating: typeof p.rating === 'number' ? p.rating : 4.3,
      distance: '—',
      address: p.vicinity || '',
      category: Array.isArray(p.types) && p.types.length > 0 ? p.types[0] : 'place',
      openNow: p.opening_hours?.open_now ?? false,
      image: imageUrl,
      description: '', // детальний опис краще генерувати на екрані деталізації
      phone: '',
      workingHours: '',
      latitude: p.geometry?.location?.lat ?? userLocation.lat,
      longitude: p.geometry?.location?.lng ?? userLocation.lon,
    };
  })
);
```

Рис. 3.17 Перетворення сирих даних Google Places

Наступним кроком у нас настає фільтрація уже нормалізованих даних за допомогою ШІ. Пілся того як ми отрималі ці данні ми маємо можливість співставитти знайдені заклади з реальною потребою користувача, що дає змогу розширити стандартні фільтки, що використовуються в API на розумну AI логіку.

Google Maps має базові алгоритми пошуку , які є статичними це значить що є влаштований набір фільтрів який неможливо перевищити. Проте коли

користувач вводить свій специфічний запит це реалізувати не завжди реально такими методами. Тож на цьому етапі якраз і включається роботу наш алгоритм інтелектуальної фільтрації .

Якщо користувач пристворенні свого запити ввів якісь ключові слова, до прикладу «тихе», «ромнтичне» і ШІ зміг записати ці дані в ключові слова , то ми передаємо наш нормалізований список до backend де й проводимо фільтрацію , так як нормалізовані дані містять назву , відгуки , та текстовий опис , ми можемо за допомогою ШІ інтепретувати ці дані згідно запиту користувача , та відобразити найбільш валідну інформацію.

Тож такий аналіз працює за допомогою ключових слів , коли користувач вводить «тихий» чи «романтичний» ШІ отримує цю інформацію , аналізує всі отримані данні від картографічоного сервісу , та якщо знаходить, до прикладу у відгуках потрібне нам ключове слово , то цей заклад автоматично переходить у статус тих що втілюють очікування користувача.

Як результат обробки ми отримуємо масив даних , що максимально втілюють всі потреби користувача, якщо розглядати UX сторону , то користувач отримує одразу потрібну йому інформацію замість того щоб гортати великий список та аналізувати дані самому.

В UI частині користувач не помічає цієї обробки ніяк , окрім не великої затримки , що відображається за допомогою текстових підказок , які описують поточний стан роботи додатку.

Якщо розглядати подальші плани на розробку , то цей функціонал може бути масштабованим , адже сюди можна додати , аналіз годин роботи та навантаження , аналіз фотографій, та погодних умов. До прикладу якщо користувачу потрібно знайти ресторан з терасою щоб посидіти ввечері в природній обстановці , а погода в цей час буде дощова , ШІ зможе проаналізувати ці показники та дати рекомендації що плани було б добре перенести або навпаки помітити ,що на фото видно , як на терасі стоять зонтики , тож це не помішає втілити плани користувача.

В результаті за допомогою функції ітелектуальної фільтрації додаток престає бути просто ще одною обгорткою для Google Maps, а додає реально новий функціонал який можна практично безкінечно розширяти, додавати нові функції та втілювати запити користувача в реальність надючи максимально релевантну інформацію згідно його потреб (Рис. 3.18).

```
      let rankedPlaces = parsed;

      if (aiKeywords.length > 0) {
        setLoadingText('Ранжуємо результати за допомогою ШІ…');
        try {
          const ranked = await rankPlacesByRelevance(parsed, aiKeywords);
          if (Array.isArray(ranked) && ranked.length > 0) {
            rankedPlaces = ranked;
          } else {
            console.log('⚠️ rankPlacesByRelevance повернув некоректні дані:', ranked);
          }
        } catch (e) {
          console.log('⚠️ Помилка при ранжуванні:', e);
        }
      }

      console.log('Ranked places:', rankedPlaces);
      setPlaces(rankedPlaces);
    } catch (err) {
      console.log('AI/Google Places Error:', err);
      setPlaces([]);
    } finally {
      setLoading(false);
      setLoadingText('Пошук…');
    }
  })();
}, [searchQuery, userLocation]);
```

Рис. 3.18 AI-ранжування результатів за ключовими словами користувача

Після того як додаток закінчив пошук локацій та фільтрацію згідно ключових слів , відобразив всі елементи на карті та створив короткі кратки знайдених місць. Ми переходимо до наступного етапу де користувач при переході на заклад , отримує розширений список інформації про обрану локацію , включаючи опис закладу згідно з його потребами.

Цей процес складається з двох основних частин:

1. отримання детальної інформації від Google Places (Details API);

2. генерація унікального AI-опису на основі реальних характеристик закладу та ключових слів користувача.

Перший етап — звернення до Google Places Details API. Пошук в Google який використовується по стандарту повертає обмежений список даних : адрес , відгуки , години роботи, чого нам не достатнього для детального аналізу. Саме API Details містить усе те, що очікує побачити користувач:

- повну адресу у форматі Google;
- статус роботи закладу в різні дні тижня;
- телефон;
- точні координати;
- повний список типів місць;
- посилання на вебсайт;
- години роботи;
- додаткові атрибути[25].

Тому при натисканні на локацію у блоці карток , ми відправляємо запит на бекенд за допомого ID що використовує Google, отримуємо всі доступні дані у форматі Json та відправляємо у клієнтську частину додатку.

Після цього ми можемо приступити до формування AI опису закладу , згідно потреб користувача. За рахунок отриманих даних від Google Maps та ключових слів користувача ШІ порівнює всі можливі варіанти та шукає те що найбільш відповідає потребам запиту.

Це працює так , якщо користувачу потрібно знайти “романтичне”,“зал з біговими доріжками” чи “тихе” місце , ШІ буде аналізувати всю наявну інформцію та підбирати опис що буде відповідати всім потрібним вимогам. Якщо знайдено збіг , до прикладу для «спротивного залу» ШІ побачи що в даних

отриманих від Google ми маємо в залі наявну кардіо зону , то він сформує опис так щоб користувач бачив – все відповідає потребам.

Це все дає змогу сформувати опис , який буде задовільняти всі потреби користувача і буде набагато кращим ніж стандарним наявнив в Google , адже він буде вирізнятися тим що написаний простою мовою , та видавати одразу всю потрібну і й повну інформацію навідміну від Google де щоб знайти всі потрібні дані потрібно витрати ще й час на їх пошуки.

Крім того якщо збіги все таки не знайшли я додав fallback , завдяки цьому ШІ всерівно поверне загальний опис і дасть змогу додатку продовжити свою роботу.

Завдяки цій архітектурі екран деталізації виглядає не як проста сторінка з даними Google Maps, а як повноцінна інтелектуальна картка місця з унікальним текстом, адаптованим під конкретний пошуковий запит (Рис. 3.19).

```
const handlePlaceSelect = async (place: Place) => {
  setLoading(true);
  setLoadingText("Завантажуємо деталі місця…");

  const fullPlace = await fetchPlaceDetails(place);

  setLoading(false);

  router.push({
    pathname: '/place-detail',
    params: { place: JSON.stringify(fullPlace) },
  });
};

const fetchPlaceDetails = async (place: Place) => {
try {
  // 1. Google Details
  const detailsRes = await fetch(`${BACKEND_URL}/places/details`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ placeId: place.id })
  });

  const details = await detailsRes.json();

  // 2. AI-опис
  const aiRes = await fetch(`${BACKEND_URL}/ai/describePlace`, {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      name: details.name,
      address: details.formatted_address,
      rating: details.rating,
      keywords: aiKeywords,
      placeDetails: details
    })
  });

  const aiData = await aiRes.json();
```

```
return {
  ...place,
  description: aiData.description || "Опис недоступний",
  phone: details.formatted_phone_number || "",
  workingHours: details.opening_hours?.weekday_text || [],
  address: details.formatted_address || place.address,
  image: place.image,
};

  } catch (e) {
    console.log("❌ Error loading details:", e);
    return {
      ...place,
      description: "Опис тимчасово недоступний",
    };
  }
};
```

Рис. 3.19 Завантаження деталей місця

3.3 Розробка серверної частини (Backend)

Бекенд виконує такоє одну з ключових ролей в цьому проекті, незважаючи на те що прктично вся логіка написана у клієнтскій частіні. Під час використання стороніх API ключів постає питання безпеки цих данних , тож в нашому випадку всі виклики до API Google Maps , Gemini робляться через бек, де всі токени приховані[26].

Приховання ключів є важливою процедурою, адже якщо вони б знаходились на клієнській частині це означало б що будь-хто міг би ними скористатись у своїї цілях. А так як ліміти на використання API не велики та й до того ж кожен виклик поверх ліміту повинен оплачуватись , то це може спичинити багато проблем

Саме тому бекенд став контейнером для зберігання таких токенів. Вони в програмі допускаються через бібліотеку dotenv[29]. А самі ключі записані в змінні файлу .env в результаті, змінні не потрапляють до репозиторію з якого запускається сервер, а додаються як змінні отечення в самому хостингу , тож їх ніколи не буде видно у клієнтській частині, а всі виклики здійснюються через backend

На серверній частні проходать такі операції як виклики API Google звідки ми отримуємо велику кількість інформації яка нормалізується та в результаті ми отримуємо потрібні нам дані що можна використати далі в проекті.

Також важливими є виклики Gemini API тут ми обробляємо запит користувача отримуємо категорії та ключові слова , а також виконуємо функцію канжування отриманих локацій згідно з потребами користувача. Тож бекенд в нашому випадку відіграє такі ключові ролі:

1. захист AI-ключів;
2. високі обсяги обчислень, які не повинні виконуватися у мобільному клієнті;

3. стандартизація роботи з AI, що забезпечує стабільні та повторювані результати.

Таким чином, бекенд є критичним проміжним шаром між мобільним застосунком і зовнішніми API. Він забезпечує повну безпеку секретних токенів, виконує складні операції з очищення та трансформації даних, а також централізує AI-логіку. Завдяки цьому фронтенд не потребує жодних доступів до Google або Gemini безпосередньо, що значно підвищує безпеку та керованість системи (Рис. 3.20).

```
import express from "express";
import cors from "cors";
import dotenv from "dotenv";
import fetch from "node-fetch";
import { GoogleGenerativeAI } from "@google/generative-ai";

dotenv.config();

const app = express();
app.use(cors());
app.use(express.json());

const PORT = process.env.PORT || 3000;
const GOOGLE_KEY = process.env.GOOGLE_MAPS_API_KEY;
const GEMINI_KEY = process.env.GEMINI_API_KEY;

if (!GOOGLE_KEY) {
  console.warn("⚠️ GOOGLE_MAPS_API_KEY не заданий в .env");
}
if (!GEMINI_KEY) {
  console.warn("⚠️ GEMINI_API_KEY не заданий в .env");
}
```

Рис. 3.20 Логіка приховання API ключів

Точкою взоду ШІ в систему є система семантичної обробки тесксових запитів користувача. Вони потрібна для того що правильно реалізувати аналіз наданих даних та перетворити їх в потрібну нам інформацію, а саме ключові слова та категорії. Ендпоінт gemini/analyze якраз й реалізує цю логіку.

Це працює таким чином , ми від клієнтської частини отримуємо текст у довільному форміті , далі штучний інтелект за допомого Gemini API надає запит

ШІ з необхідними параметрами і як відповідь ми отримуємо потрібні та структуровані дані.

Щоб це реалізувати сервервер викликає модель Gemini 2.0 Flash . Вона на відміну від інших моделей має змогу не просто проводити аналіз даних , але й враховувати контекст , що дає змогу нам краще зрозуміти користувача. Далі з серверу передаємо запит у вигляді промпту (набору інструкцій для ШІ) всі потрібні нам дані , та обмежуємо його роботу , ставимо строгі налаштування адже відповідь нам потрібна структурована і у форматі JSON.

AI повертає два ключові елементи:

1. **category** — узагальнений тип місця (наприклад, "cafe", "bar", "park", "gym" тощо);
2. **keywords** — масив ключових характеристик, витягнутих із запиту (наприклад, "тихе", "з Wi-Fi", "атмосферне", "для роботи").

Саме ці дані потім дозволяють системі:

- будувати коректні запити до Google Places;
- розуміти, які саме атрибути є важливими для користувача;
- ефективно ранжувати результати;
- формувати AI-опис місця на іншому етапі.

Також хотів би розлянути функцію яка нормалізує відповідь ШІ , адже він може надавати свою відповідь у текстовому форматі а не у JSON, тож для того щоб звести ці данні до потрібного нам формату , я реалізував функцію що очищує всі непотрібні символи окрім Json.

Тож пілся детального огляду функції gemini/analyze можна дійти до висновку , що ця функція є основним джерелом обробки клієнтської інформації для подального використання у інших процесах таких як ранжування та отримання опису від штучного інтелекту.(Рис. 3.21).

```
app.post("/gemini/analyze", async (req, res) => {
  try {
    const { query } = req.body;

    if (!query || typeof query !== "string") {
      return res.json({ category: null, keywords: [] });
    }

    const model = genAI.getGenerativeModel({ model: "gemini-2.0-flash" });

    const prompt = `
      Ти — система аналізу пошукових запитів для мобільного застосунку з картою.

      Користувач вводить запит природною мовою, наприклад:
      - "затишне кафе з Wi-Fi та розетками для роботи"
      - "бар з живою музикою і коктейлями"
      - "місце для ранкової пробіжки в парку"

      Твоє завдання:
      1) Виділити тип закладу (категорію) коротким рядком, наприклад:
         "cafe", "bar", "restaurant", "park", "gym", "co-working".
      2) Виділити 2–6 ключових слів, які описують побажання користувача
         (атмосфера, бюджет, Wi-Fi, тиша, спорт, краєвид, романтика, тощо).

      Формат відповіді строго JSON без пояснень:
      {
        "category": "cafe",
        "keywords": ["затишне", "wifi", "розетки", "для роботи"]
      }

      Користувацький запит:
      "${query}"
    `;

    const result = await model.generateContent(prompt);
    let text = result.response.text().trim();

    // Прибираємо можливі ```json ... ```
    text = text
      .replace(/```json/gi, "")
      .replace(/```/g, "")
      .trim();
```

```
    const start = text.indexOf("{");
    const end = text.lastIndexOf("}") + 1;
    if (start === -1 || end === -1) {
      throw new Error("JSON not found in Gemini response");
    }

    const jsonPart = text.substring(start, end);
    const parsed = JSON.parse(jsonPart);

    const category = typeof parsed.category === "string" ? parsed.category : null;
    const keywords = Array.isArray(parsed.keywords) ? parsed.keywords : [];

    res.json({
      category,
      keywords,
    });
  } catch (e) {
    console.error("❌ Помилка /gemini/analyze:", e);
    res.json({ category: null, keywords: [] });
  }
});
```

Рис. 3.21 Ендпоінт /gemini/analyze — аналіз текстового запиту користувача через AI

Після того як миотримали ключові слова та категорію з запиту користувача , з’вляється потреба відфільтрувати додатково всю інформацію що ми отримали від Google для того щоб відокремити тільки ту, що потрібна нам . Google Maps вміє фільтрувати дані тільки прив’язуючись до ключових слів та категорій , на томість ШІ иоже враховувати весь контекст і цю всю логіку ми реалізуємо за допомогою ендпоінту gemini/rank.

Ранжування потрібне тому, що:

- Google Places не враховує стиль атмосфери (“затишне”, “романтичне”, “для роботи”);
- не розуміє ключові бажання типу “з Wi-Fi”, “з терасою” чи “в тиші”;
- не може зіставити улюблені характеристики користувача з контентом про місце;
- результати часто містять заклади «в тему», але не точні за очікуваннями.

Ендпоінт /gemini/rank допомагає провести оцінку всім отриманим даним : він отримує дані від Google, да дані запиту користувача що ми отримали на попередньому кроці, заклади надсилаються в промпті у вилгяді нумерованого списку. ШІ аналізує отримні дані від Google : адреса, години роботи , опис та велика кількість іншої інформації , знаходить дотичність до запиту користувача , та повертає впорядкований масив даних який сортується так що першим йде локація яка є найбльш підходящою для втілення задуму користувача.

Після того як система бек отримав результа обробки він надсилає його на клієнтську частину у вигляді оновленого списку що відображається на карті та картках локацій , якщо ШІ не зміг видати ніякої інформації , або те що він повернув не відповідає формату JSON , то програма не завершує свою роботу , а переходить до fallback де просто повертає до клієнта оригінальний список даних.

Таким чином, /gemini/rank виконує функцію рекомендаційного ядра, перетворюючи класичний геопошук на персоналізований інтелектуальний механізм (Рис. 3.22).

```
app.post("/gemini/rank", async (req, res) => {
  try {
    const { places, keywords } = req.body;

    if (!Array.isArray(places) || !Array.isArray(keywords) || keywords.length === 0) {
      return res.json(places || []);
    }

    const model = genAI.getGenerativeModel({ model: "gemini-2.0-flash" });

    const placeList = places
      .map((p, i) => {
        const line = [
          `#${i + 1}`,
          `Назва: ${p.name || "Невідомо"}`,
          `Адреса: ${p.address || p.vicinity || ""}`,
          `Категорія: ${p.category || (p.types && p.types[0]) || ""}`,
        ].join("; ");
        return line;
      })
      .join("\n");

    const prompt = `
      Ти допомагаєш відсортувати список закладів за релевантністю до запиту користувача.

      Ключові слова користувача:
      ${keywords.join(", ")}

      Нижче список закладів (кожен з номером #N):

      ${placeList}

      Поверни тільки JSON-масив з номерами закладів у порядку від найбільш релевантного до найменш релевантного.
      Приклад:
      [3, 1, 2, 4]

      Без пояснень, тільки JSON масив чисел.
    `;

    const result = await model.generateContent(prompt);
    let text = result.response.text().trim();

    text = text
      .replace(/```json/gi, "")
      .replace(/```/g, "")
      .replace(/\s+/g, " ")
      .trim();
```

```
    const start = text.indexOf("[");
    const end = text.lastIndexOf("]") + 1;
    if (start === -1 || end === -1) {
      throw new Error("JSON array not found in Gemini response");
    }

    const arrPart = text.substring(start, end);
    const indices = JSON.parse(arrPart);

    if (!Array.isArray(indices)) {
      throw new Error("Gemini returned non-array");
    }

    const ranked = indices
      .map((idx) => {
        const i = Number(idx) - 1;
        return places[i];
      })
      .filter(Boolean);

    if (!ranked.length) {
      return res.json(places);
    }

    res.json(ranked);
  } catch (e) {
    console.error("❌ Помилка /gemini/rank:", e);
    res.json(req.body.places || []);
  }
});
```

Рис. 3.22 Ендпоінт /gemini/rank — ранжування місць за користувацьким запитом

Ендпоінт /places/search також відіграє дуже важливу функцію , адже саме тут ми шукаємо локації згідно оброблених даних отриманих після виконання функції gemini/analyze. Тут уже надсилається запит до Google API

використовуючи координати користувача , ключові слова та категорію , і після отримання результатів обробки , ми бачимо мітки на карті та картки локацій.

Робота цього ендпоінту розпочинається з того що ми отримуємо дані про координати від клієнта , а також текстовий запит користувача що передається як ключові слова та категорії , пілся чого надсилається HTTP запит до Google Places API , важливо звернути увагу на те що тут також працює логіка закритих API токенів , що не відображаються в клієнті завдаки їх прихованню в .env змінних.

Далі ми формуємо URL для запиту до Google API , куди ми передаємо сам запит користувача , координати , зону пошуку, а також API ключ. Таким чином ми й знаходимо місця поряд з координатими користувача. Важливим пунктом тут є те що якийсь із параметрів URL може бути не валідним , відповідно перед тим як надсилати запит ми перевіряємо чи користувач надав доступ до координат , та запит не пустий.

Після того як запит надійшов до Google ми отримуємо структуровану відповідь де міститься велика кількість даних про локації які були знайдені по нашим параметрам.Проте тут також можуть виникати проблеми , і бек для того щоб не ломати код у разі виникнення проблем з відповіддю Google отримує помилку , та відправляє пустий масив до клієнту. Помилки можуть виникнути через проблеми з запитом, інтернет з’єднаннями чи через те що ліміт запитів вичерпано , тож потрібно пердбачати подібні ситуації та створювати fallback.

У результаті цей ендпоінт стає ключовим компонентом у всьому процесі роботи користувача з додатком. Він отримує координати, формує запит, надсилає його, отримує сирі дані, обробляє їх і повертає у структурованому вигляді, адаптованому до потреб фронтенду. Його робота практично непомітна з точки зору UX, але є дуже важливою для функціонування додатку, оскільки це визначає, які місця будуть відображені на карті (Рис. 3.23).

```
app.post("/places/search", async (req, res) => {
  try {
    const { lat, lon, keyword, radius = 2500 } = req.body;

    if (!lat || !lon) {
      return res.status(400).json({ error: "lat/lon required" });
    }

    const url =
      `https://maps.googleapis.com/maps/api/place/nearbysearch/json?` +
      `location=${lat},${lon}` +
      `&radius=${radius}` +
      (keyword ? `&keyword=${encodeURIComponent(keyword)}` : "") +
      `&key=${GOOGLE_KEY}`;

    console.log("🌐 Google Places Nearby URL:", url);

    const resp = await fetch(url);
    const data = await resp.json();

    if (data.error_message) {
      console.error("Google Places error:", data.error_message);
    }

    res.json(data.results || []);
  } catch (err) {
    console.error("❌ /places/search error:", err);
    res.status(500).json({ error: "places search failed" });
  }
});
```

Рис. 3.23 логіка роботи ендпоінта /places/search

Після того як система знайшла всю потрібну інформацію про заклади через Google Place нам потрібно отримати їх фото. Тут постає одна проблема , так як Google не повертає фото як URL а у форматі Photo_reference , нам потрібно його дотатково обробити щоб отримати URL використовуючи API ключ.

Тож в бекенді ми створили ендпоінт places/photo який допомагає нам сформувати потрібний URL для зображення. Він не повертає ніякого фото в двійковому форматі , а передає URL фото до компоненту <Image> в клієтській частині.

Знову ж таки логіка обробка фото до правильного формату написана саме в серверній частині для того , щоб уникнути засвічення API ключа в клієнті. Адже й цей процес використовує апі , тому було прийняте таке рішення.

Алгоритм перетворення photo_reference працює таким чином , що ми отримуємо ці данні з відповіді від Google, перевіряємо чи це поле доступне , й надсилаємо запит до Google Places Photo з параметрами шрини та висоти зображення , після чого отримане зображення відправляємо в клієнт.

Такий підхід не тільки захищає проект від загроз але й дозволяє вномити зміни без модифікацій на клієнтській стороні. До прикладу нам потрібно змінити розміри отримих фото , ми можемо це зробити без модифікації клієнта(Рис. 3.24).

```
app.post("/places/photo", async (req, res) => {
  try {
    const { photoReference, maxwidth = 800 } = req.body;

    if (!photoReference) {
      return res.json({ url: "" });
    }

    const photoUrl =
      `https://maps.googleapis.com/maps/api/place/photo?` +
      `maxwidth=${maxwidth}` +
      `&photo_reference=${photoReference}` +
      `&key=${GOOGLE_KEY}`;

    res.json({ url: photoUrl });
  } catch (e) {
    console.error("❌ /places/photo error:", e);
    res.json({ url: "" });
  }
});
```

Рис. 3.24 отримання фото через Google Places Photo API

В архітектурі цього додатку ці 3 алгоритми відповідають за пошук та фільтрацію даних , коротко про формування списку локацій , тоді як наступний алгоритм що ми розглянемо починає будувати новий процес , а саме процес надання опису для відображених локацій.

Починається процес із ендпоінту /places/details. Він звертається до Google places для того щоб отримати максимальну інформацію про поточні локації. Nearby Search повертає лише базовий набір даних — найчастіше назву, ID, приблизну адресу, рейтинг, типи та фото. Проте для нашого додатку цієї інформації замало , нам також потрібно що ШІ аналізував всі надані йому данні та надавав опис що буде втілювати потреби користувача та виглядав живим а не штучним.

Ендпоінт /places/details дістанє цю всі інформацію що в подальшому відображається у клієнті. Завдяки тому що це все знаходиться на сервері , ми як говорилося раніше по-перше захищаємо наш додаток , по-друге не перенавантажуємо клієнт зайвими обчисленнями.(Рис. 3.25).

```
app.post("/places/details", async (req, res) => {
  try {
    const { placeId } = req.body;

    if (!placeId) return res.status(400).json({ error: "placeId required" });

    const url =
      `https://maps.googleapis.com/maps/api/place/details/json?place_id=${placeId}` +
      `&fields=name,rating,formatted_address,opening_hours,photos,geometry,formatted_phone_number,website,types` +
      `&key=${GOOGLE_KEY}`;

    const resp = await fetch(url);
    const data = await resp.json();

    res.json(data.result || {});
  } catch (e) {
    console.error("❌ /places/details error:", e);
    res.status(500).json({ error: "details failed" });
  }
});
```

Рис. 3.25 – фрагмент коду Ендпоінту /places/details

/ai/describePlace – це наступний ендпоінт який продовжує попередньо зазначену логіку. Тут ми знову звертаємося до Gemini , який в свою чергу співставляє отриманий запит від користувача та розширену інформацію що ми отримали на попередньому етапі з ендпоінту places/details , порівнює всі отримані дані та формує опис що буде втілювати всі потреби користувача , якщо це реально.Це не просто копіювання даних отриманих від Google , ШІ створює опис підбираючи тон та задючи настрій та атмосферу закладу , що буде дуже привабливим для користувача.

Промт побудовано так що ШІ давав більш художній опис , а не видавав просто суху інформацію , цим самим ми робимо сприйняття тексу більш легким для користувача.

Важливим в цьому етапі є те що AI враховує ключові слова з запиту користувача і видає персоналізовану інформацію, що відповідає його запиту. (Рис. 3.26).

```
app.post("/ai/describePlace", async (req, res) => {
  try {
    const { name, address, rating, keywords = [], placeDetails = {} } = req.body;

    const model = genAI.getGenerativeModel({ model: "gemini-2.0-flash" });

    const prompt = `
      Створи унікальний, гарно написаний опис закладу українською мовою.
      Опиши атмосферу, враження, тип аудиторії та причини, чому місце може підійти користувачу.

      Використай такі дані:
      Назва: ${name}
      Адреса: ${address}
      Рейтинг: ${rating || "без рейтингу"}
      Побажання користувача: ${keywords.join(", ")}
      Google details: ${JSON.stringify(placeDetails)}

      Вимоги:
      - Тон: дружній, професійний, живий, без канцелярщини.
      - Обсяг: 3–6 речень.
      - Ніяких списків, маркерів, нумерацій.
      - НЕ перераховувати дані із JSON (типу "рейтинг 4.7" або "Адреса така-то").
      - Просто створи гарний, емоційний текст, який читається природно.
      `;

    const r = await model.generateContent(prompt);
    const description = r.response.text().trim();

    res.json({ description });
  } catch (e) {
    console.error("❌ /ai/describePlace error:", e);
    res.json({ description: "" });
  }
});
```

Рис. 3.26 Фрагмент коду енпоінту /ai/describePlace

Тож поєднання останніх двох ендпоінтів і створюють повноцінну логіку сторінки з детальною інформацією про локаці. Тут надаються розширені дані про заклад а також опис що персоналізований під потреби користувача , що робить

додаток не просто пошуковим , а також й асистентом що дозволяє легше знаходити інформацію і надає рекомендації для прийняття подальших рішень.

Після того як я завершив розробке бекенд частини , постало питання розгортання бекенд частини на хостингу, для того щоб уникнути проблем з мережею чи девайсами на якому ми працюємо , та й щоб сервер була постійно запущений і ми могли просто відкрити клієнт і спокійно працювати уже в готовому додатку. Для втілення цієї мети я обрав хмарний сервіс Render , він дозволяє розгортати node.js застосунки використовуючи репозитой гіт , та автоматично оновлюватись при кожному коміті[27]. Render є дуже простим та зручним в роботі його можна налаштувати задекілька хвилин , потрібно обрати репозиторій , ввести змінні середовища , у нашому випадку це ключі API , та й запустити(Рис. 3.27).

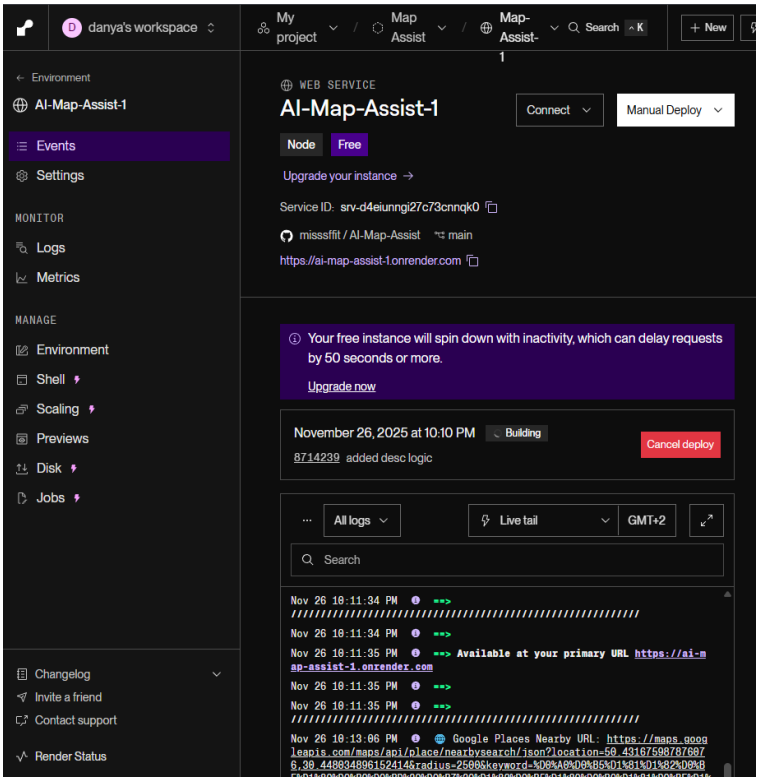


Рис. 3.27 – Інтерфейс хостингу Render

Процес розгортання розпочався зі створення нового веб-сервісу у панелі Render. Після чого я створив репозиторій та закомітив як сервер так і клієнт до репозиторію. В Renderer обрав потрібний репозиторій , налаштував мову

виконання коду в моєму випадку це node.js[28] налаштував команду для build та start, а саме це “npm install” – build та “node server.js” – start.

Далі так як наші API ключі були сховані для безпеки користування додатком , постало питання як сервер буде виконувати код , якщо він не знає ключів що сховані в .env. так як ми використовували API Gemini та Google Maps була потреба ініціалізувати іх якось у render. На допомогу прийшли змінні середовище , де ти ініціалізуєш ключі як звичайні змінні в коді та й усе.

Після запуску серверу на Render ми отримуємо URL за допомогою якого ми й можемо звертатись до backend та формувати запити до API. Також окремо хотів би виділити, що Render автоматично оновлює всі нові отримані дані коли надзодить новий коміт до репозиторію , це дозволяє працювати з сервером без його перезапуску.

Таким чином як на мене Render є найкращим рішенням для хостингу серверної частини , адже він є безкоштовним , легким в налаштуванні, та дозволяє вносити оновлення без перезапуску.

4. ОПИС РОБОТИ МОБІЛЬНОГО ЗАСТОСУНКУ

4.1. Головний екран та взаємодія із системою вводу запитів

Робота мобільного застосунку починається з початкового екрану де відображено текстове поле для вводу інформації , рекомендіції що до можливих запитів що при натискані одразу відображаються в текстовому полі. Система здатна сприймати багаторядкові запити. Цей інтерфейс фокусує всю увагу користувача саме на текстовому полі , і дає змогу користувачу зрозуміти що для нього тут головне . Інтефейс виглядає інтуєтивно зрозуміло , адже він не навантажений великою кількістю елементів.

На цьому етапі користувач запускає логіку виклику ШІ та аналізу отриманої інформації від користувача , для користувача всі ці процеси не видні , він тільки бачить зміну стану кнопки пошуку.

Тож давайте перевіримо роботу додатку написавши якийсь запит , в моєму випадку я обрав такий «ресторан з терасою»(Рис. 4.1)

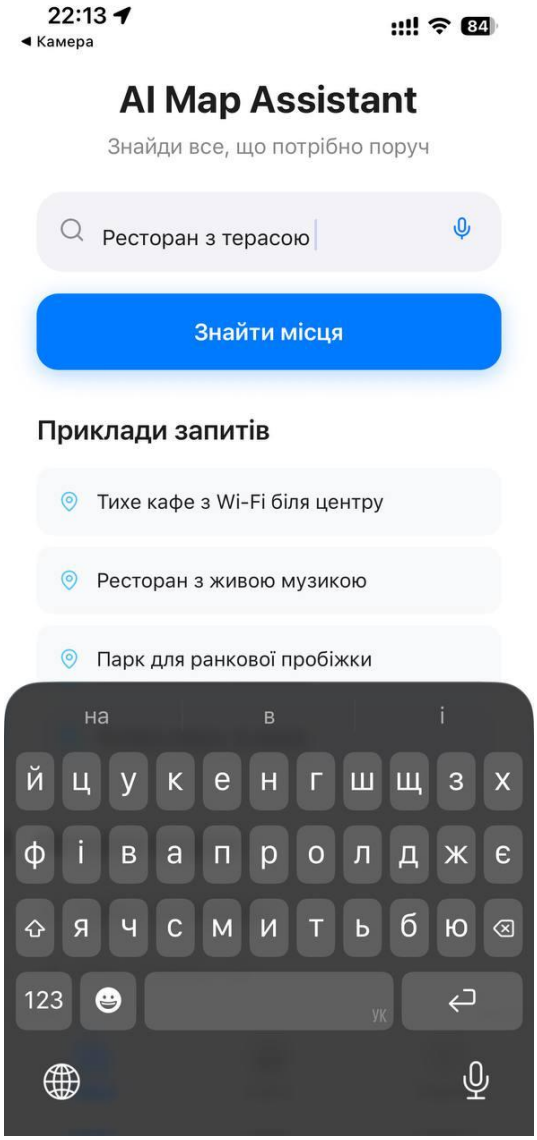


Рис. 4.1 Тестування пошуку

4.2. Екран із картою та робота з результатами пошуку

Після того як користувач ввів запит та натиснув кнопку пошуку, нас перекидує у вікно з картою , де ми бачимо саму карту яка є основним елементом. На ній зображені маркери та локація користувача , завдяки чому користувач бачить яка локація є найближчою до нього, також карту можна переміщати та мастшабувати що додає інтерактивностів просцесу , окрім того доступна кнопка яка одразу фокусує карту на координатах користувача(Рис. 4.2).

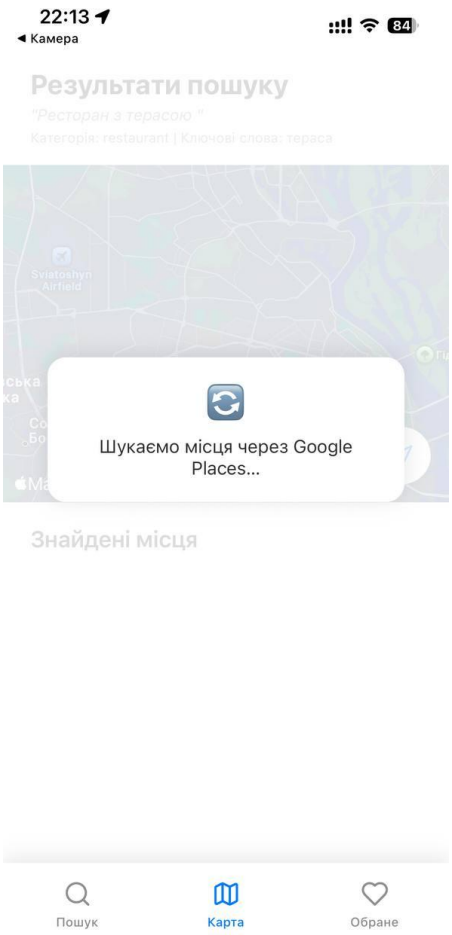


Рис. 4.2 Процес обробки запиту користувача

Далі під картою знаходиться блок з картакми локацій де стисло описана основна інформація , а також відображене фото закладу. Це є одним з основних елементів тут , адже користувач спочатку консентрує свою увагу на візуальній інформації а вже потім на текстовій. Тут можна натиснути на певну локацію та перейти на настуну сторінку(Рис. 4.3).

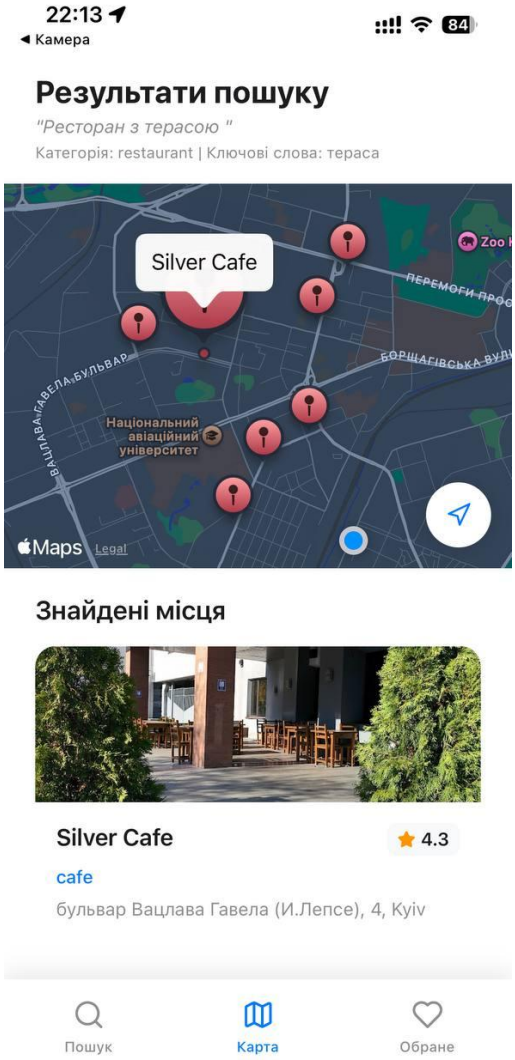


Рис. 4.3 Відображення маркерів на карті та карток локацій

4.3. Екран детальної інформації про заклад

Останнім екраном в нашому додатку буде детальна інформація про заклад. Після того як користувач натиснув на якусь картку локацій система перекидує його на цю сторінку.

Тут знаходиться детальна інформація про заклад , а саме :

- години роботи
- рейтинг
- фото

- назва
- кнопка переходу в гугл мапс для побудови маршруту
- номер телефону
- кнопка що дозволяє набрати заклад
- Опис від ШІ

Як на мене основним тут є якраз опит від ШІ який є й головним принципом.UX на цьому екрані , адже він задає тон всій сторінці. Користувач отримує не просто сухі факти , а інформацію яку він потребує та щей описану в зрозумілим стилем , що полегшує прийняття рішення (Рис. 4.4).

Рис. 4.4 Відображення екрану детальної інформації

У ході тестування ми змогли пройти всі етапи роботи застосунку успішно й добилися поставленої перед нами мети , тож можна зробити висновок що додаток працює коректно.

ВИСНОВОКИ

Під час виконання дипломної роботи було розроблено мобільний застосунок метою якого було створити іноваційний сервіс що об’єднує використання картографії та ШІ, для створення інтелектуального пошуку о’бєктів та надання певних рекомендацій згідно окчікуванням користувача. Тож для виконання ціє цілі потрібно було написати як backend так і frontend частини коду , для того щоб додаток працював без зайвих затримок та був безпечним середовищем для використання API. Говорячи про API потрібно було використати якийсь сервіс картографії в моєму випадку це Google Maps та модель ШІ тут це Gemini , для виконання поставлених цілей.

Результат роботи показав що додаток працює як й задумувалось і додав мені нові ідеї для масштабування. Головної ідеєю тут було те щоб не користувач підлаштовувався під систему і описував свої потреби під мову зрозумілу для системи , а навпаки , щоб системе сприймала запит користувача сформаваний природною мовою і перетворювала його запит сама так що він міг використовуватися в системі й надалі.

Важливим в цьому додатку є те, що штучний інтелект використовуються одразу на двох рівнях функціонування програми , спочатку він аналізує запит користувача та отримує потрібну інформацію для роботи системи, а потім уже фільтрує його з урахуванням потреб користувача , щоб видати йому максимально персоналізовану інформацію під кожен запит , це дуже вирізняє цей додаток поміж інших.

Також по проекту було додано багато fallback функціоналу , що дозволяє додатку функціонувати навіть якщо виникнуть помилки під час виконання того чи іншого етапу.

Важлимив стало також розробка backend за допомогою node.js та деплой беку на хостинг. Завдяки цьому ми сховали всі ключі API для того щоб вони не відображалися в клієнті це убезпечить мене як розробника , адже ці ключі могли

б бути використанні сторонніми людми , що є величезним мінусом адже , існують ліміти на запити та API є платними. Деплой серверу на Render в свою чергу дозволив нам зручно вносити зміни без перезапуску серверу та дав можливість запускати додаток з будь-якої точки світу.

Клієнтська чачтина коду реалізована за допомогою бібліотеки для JavaScript – React native та Expo – бібліотка для реакт , що дозволяє розробити додаток одразу сумісний для всі платформ , як Android так і IOS. Інтерфейс побудований так, щоб користувач міг лейко й швидко користуватися додатком та знаходити йому всю необхідну інформацію.Кольори що були підібрані під час розробки UI/UX інтерфейсу як на мене ідеально підходять до стилістики всього додатку , також анімації та стани кнопок відіграють тут теж дуже важливу роль, адже вони роблять інтерфейс більш плавним.

Також хотілось би вибілити алгоритм надання опису від штучного інтелекту. Завдяки цьому користувач отримує персоналізований опис , який оживляє додаток , та дозволяє користувачу приймати рішення швидше.

Під час тестування роботи додатку ми побачили що всі функції функціонують правильно , текстове поле передає інформацію і перенаправляє нас на сторінку з картою , карта є рухомою та відображає маркери , кнопка центрування працює, відповіді від API ми отримуємо коректно , картки відображаються так як було задумано , а також сторінка з тетальною інформаціє та ШІ описом функціонує без проблем.

Тож можна зробити висновок, що поставлені цілі в цій дипломній роботі виконані , та додаток функціою так як потрібно. Інтелектуальна частина в комбінації з простив інтерфейсов що орієнтований на користувача виконує всі поставлені функції. Але завжди є простір до вдосконалення , та масштабування. Цей додаток легко масштабувати та додавати нові функці , на приклад : можливість не просто шукати заклади , а різні розважальні чи культурні активності , до прикладу алпнувати походи до театру чи катання на квадроциклі , зробити так щоб додаток також міг аналізувати таку інформацію та давати якісь свої рекомендації.У перспективі цей застосунок може бути розширений до

рекомендаційної платформи, туристичного асистента, сервісу для локального бізнесу або навіть інтегрований у ширші екосистеми на основі AI. Розроблена архітектура дозволяє продовжити масштабування без кардинальної зміни логіки.

ПЕРЕЛІК ВИКОРИСТАНИХ ПОСИЛАНЬ

1.Topic: mobile internet usage worldwide. *Statista*. URL: https://www.statista.com/topics/779/mobile-internet/ (date of access: 01.12.2025).

2.Розробка геолокаційних програм - компанія Wezom. *IT-компания полного цикла разработки программных продуктов WEZOM - Киев, Украина*. URL: https://wezom.com.ua/ua/blog/razrabotka-geolokatsionnyh-prilozhenij (дата звернення: 01.12.2025).

3.Використання штучного інтелекту у розробці мобільних додатків. *IT-компания полного цикла разработки программных продуктов WEZOM - Киев, Украина*. URL: https://wezom.com.ua/ua/blog/shtuchniy-intelekt-u-mobilnih-dodatkah (дата звернення: 01.12.2025).

4.Mobile network subscriptions worldwide 2028| Statista. *Statista*. URL: https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/ (date of access: 01.12.2025).

5.Stryker C., Holdsworth J. What is NLP (natural language processing)? | IBM. *IBM*. URL: https://www.ibm.com/topics/natural-language-processing (date of access: 01.12.2025).

6.Geolocation accuracy. *MaxMind Knowledge Base*. URL: https://support.maxmind.com/knowledge-base/articles/maxmind-geolocation-accuracy (date of access: 01.12.2025).

7.5 альтернатив Google Maps для вашого бізнесу. *Ranktracker:The all-in-one platform for effective SEO*. URL: https://www.ranktracker.com/uk/blog/5-alternatives-to-google-maps-for-your-business/ (дата звернення: 01.12.2025).

8.Mapbox vs Google Maps: What's best for mapping?. *Red Rocket*. URL: https://redrocket.software/blog/mapbox-vs-google-maps-whats-best-for-mapping-in-logistics (date of access: 01.12.2025).

9.10 Best Foursquare Alternatives & Competitors in 2025. *Software Podium*. URL: https://softwarepodium.com/foursquare-alternatives (date of access: 01.12.2025).

10.Blog - huwise. *Huwise*. URL: https://www.opendatasoft.com/blog/what-is-openstreetmap (date of access: 01.12.2025).

11.The Evolution of Mobile Apps: 1994 to 2025 | AppsRhino. *Custom App Development Company - AppsRhino*. URL: https://www.appsrhino.com/blogs/evolution-of-mobile-apps (date of access: 01.12.2025).

12.Topic: Smartphones. *Statista*. URL: https://www.statista.com/topics/840/smartphones/?srsltid=AfmBOoo9tRbKxryM5Dty5wmQbP81qC5r837RzdMt2Xv2kVh4jwVw8BcB (date of access: 01.12.2025).

13.Contributors to Wikimedia projects. Apache cordova - wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/Apache_Cordova (date of access: 01.12.2025).

14.Contributors to Wikimedia projects. PhoneGap – Википедия. *Википедия – свободная энциклопедия*. URL: https://ru.wikipedia.org/wiki/PhoneGap (дата звернення: 01.12.2025).

15.Contributors to Wikimedia projects. React Native - Wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/React_Native (date of access: 01.12.2025).

16.What is Flutter? A Comprehensive Overview | Holdapp. *Holdapp*. URL: https://www.holdapp.com/blog/what-is-flutter-an-overview (date of access: 01.12.2025).

17.Mapbox vs. Google Maps API: 2026 comparison (and better options). *Radar*. URL: https://radar.com/blog/mapbox-vs-google-maps-api (date of access: 01.12.2025).

18.OpenStreetMap Foundation – Supporting the work of the OpenStreetMap project. *OpenStreetMap Foundation – Supporting the work of the OpenStreetMap project*. URL: https://supporting.openstreetmap.org/ (date of access: 01.12.2025).

19. Google Maps API - SerpApi. *SerpApi*. URL: https://serpapi.com/google-maps-api?gad_source=1&gad_campaignid=1061187028&gbraid=0AAAADD8kqNiXewWFwcbwsPFxBdl9eT-5&gclid=CjwKCAiAlrXJBhBAEiwA-5pgwvdbWSC-bHzVbWI_DmLDZRPg40m6l2ncQ1dtRTh56_pqXwFl9WeBuhoCkUMQAvD_BwE (date of access: 01.12.2025).

20.Singh S. LLM development process: key insights and overview. *Labellerr AI*. URL: https://www.labellerr.com/blog/overview-of-development-of-large-larnguage-models/ (date of access: 01.12.2025).

21.React Native · Learn once, write anywhere. *React Native · Learn once, write anywhere*. URL: https://reactnative.dev/ (date of access: 01.12.2025).

22.Expo documentation. *Expo Documentation*. URL: https://docs.expo.dev/ (date of access: 01.12.2025).

23.GitHub - react-native-maps/react-native-maps: react native mapview component for ios + android. *GitHub*. URL: https://github.com/react-native-maps/react-native-maps (date of access: 01.12.2025).

24.Google maps platform documentation | google for developers. *Google for Developers*. URL: https://developers.google.com/maps/documentation (date of access: 01.12.2025).

25.Google maps platform documentation | places API | google for developers. *Google for Developers*.

URL: https://developers.google.com/maps/documentation/places/web-service (date of access: 01.12.2025).

26.Google maps platform security guidance | google for developers. *Google for Developers*. URL: https://developers.google.com/maps/api-key-best-practices (date of access: 01.12.2025).

27.Docs + quickstarts | render. *Cloud Application Platform | Render*. URL: https://render.com/docs (date of access: 01.12.2025).

28.Index | node.js v25.2.1 documentation. *Node.js – Run JavaScript Everywhere*. URL: https://nodejs.org/en/docs (date of access: 01.12.2025).

29.GeeksforGeeks. NPM dotenv - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/node-js/npm-dotenv/ (date of access: 01.12.2025).

30.Gemini API | google AI for developers. *Google AI for Developers*. URL: https://ai.google.dev/gemini-api/docs (date of access: 01.12.2025).

КОПІЇ ДЕМОНСТРАЦІЙНИХ МАТЕРІАЛІВ

Державний університет інформаційно-комунікаційних технологій

Кафедра Інженерії програмного забезпечення автоматизованих систем

КВАЛІФІКАЦІЙНА РОБОТА
на тему:

«Мобільний застосунок для пошуку та рекомндаїй локацій з інтеграцією мовних моделей штучного інтелекту»

на здобуття освітнього ступеня магістра
зі спеціальності 126 Інформаційні системи та технології
освітньо-професійної програми Інформаційні системи та технології

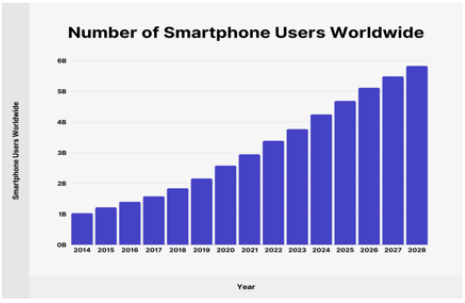
Виконав : Буджак Данило Віталійович
Науковий керівник роботи:
Данильченко В.М.

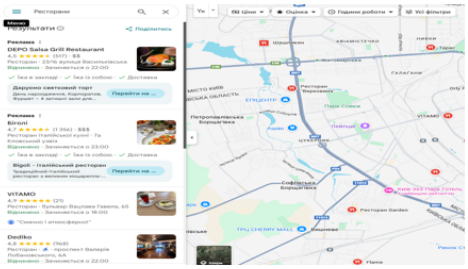
Київ – 2025

- ***Актуальність теми:*** Сервіси на основі ШІ стрімко розвиваються, тому створення зручних мобільних застосунків для пошуку локацій є актуальним та затребуваним.
- ***Наукова новизна:*** Проаналізовано роботу мобільних застосунків із використанням Google Maps API і штучного інтелекту та реалізовано власний застосунок із розширеним функціоналом пошуку.
- ***Об`єкт дослідження:*** Мобільні застосунки з використанням геолокаційних сервісів та штучного інтелекту.
- ***Предмет дослідження:*** Методи та технології інтеграції картографічних API та ШІ у мобільні застосунки.
- ***Мета дослідження:*** Створення мобільного застосунку для пошуку та відображення локацій за допомогою Google Maps API та Gemini API.
- ***Завдання дослідження:***
  - **1.** Проаналізувати можливості Google Maps API та Gemini API.
  - **2.** Розробити мобільний застосунок на основі React Native та backend-сервісом.
  - **3.** Оцінити ефективність роботи застосунку в реальних умовах.

1. Актуальність та проблеми сучасних гео-застосунків

Чому тема актуальна?
•Мобільні застосунки — основний спосіб пошуку місць та сервісів.
•Класичний пошук через фільтри → повільний, незручний, не персоналізований.
•ШІ дозволяє формувати запити звичайною мовою: *«Покажи тихе кафе з Wi-Fi поруч»*.
Основні проблеми ринку
•Дані про заклади розкидані між різними сервісами.
•Відсутня інтерпретація природної мови.
•Різна повнота даних у різних країнах.
•Часто застаріла інформація(зміна графіків, локацій тощо).



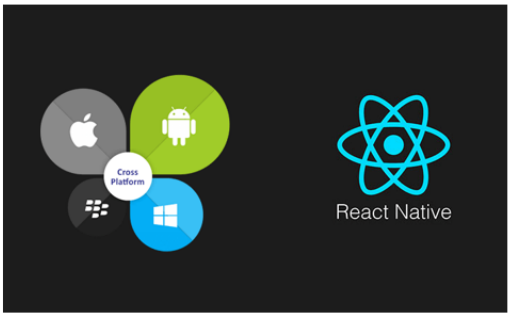


1. Актуальність та проблеми сучасних гео-застосунків

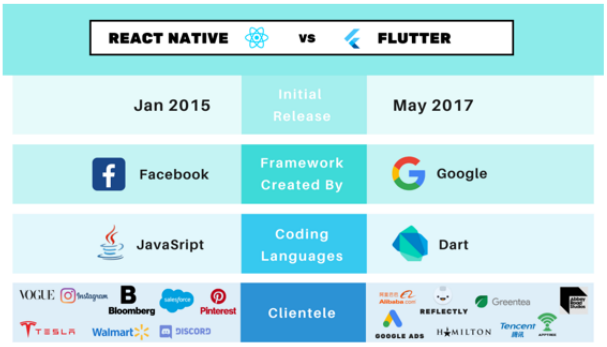
Що вже існує?
•Google Maps — найбільша база, але дорогий API.
•Apple Maps — слабше наповнення.
•Foursquare / Yelp / Tripadvisor — хороші фото й описи, але обмеженість по регіонах.
•OpenStreetMap — безкоштовний, але без фото/описів.
Виявлені недоліки
•Немає сервісу, який закриває *всі* потреби користувача.
•Користувач змушений вручну уточнювати запит, читати відгуки та перемикатися між платформами.
•Не враховується контекст та попередні вподобання.
•Не вистачає актуальних і повних даних.
Що вирішує застосунок?
•Єдиний інтерфейс з даними з Google + інших сервісів.
•Природномовний пошук через AI (Gemini / OpenAI).
•Актуальна інформація, фото, описи в одному місці.
•Автоматичний підбір закладів за контекстом та потребами користувача.

2. АНАЛІЗ ТЕХНОЛОГІЙ ТА ОБҐРУНТУВАННЯ ВИБОРУ СТЕКУ

React Native — це open-source фреймворк на JavaScript для створення нативних мобільних застосунків під IOS та Android з однієї кодової бази. Розроблений компанією Meta, він дозволяє будувати кросплатформні інтерфейси з високою продуктивністю.
Основні переваги:
Кросплатформність: один код для кількох платформ.
Нативні UI-компоненти: <View>, <Text>, <Image> рендеряться у справжні нативні елементи, забезпечуючи швидкодію та природний вигляд.
концепції React: компоненти, JSX, props, state та Hooks, що спрощує роботу розробникам з досвідом у React.



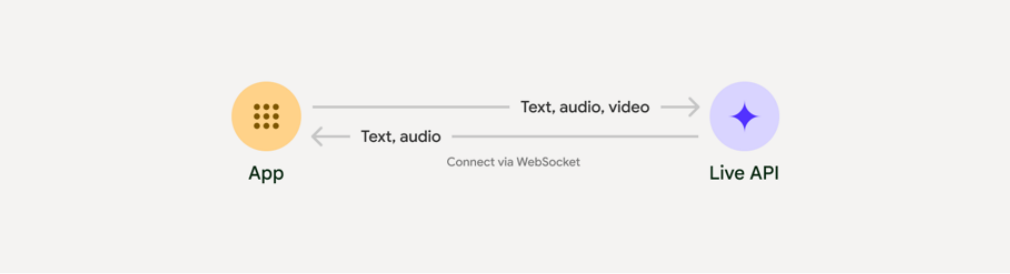
2. АНАЛІЗ ТЕХНОЛОГІЙ ТА ОБҐРУНТУВАННЯ ВИБОРУ СТЕКУ



Хоча Flutter забезпечує стабільний інтерфейс і високу продуктивність, для даного проєкту більш раціональним рішенням став React Native. Його головна перевага — можливість швидко розробляти інтерфейси завдяки знайомим принципам React та JavaScript, що значно скорочує час створення застосунку. React Native має велику екосистему бібліотек і спрощує інтеграцію зовнішніх сервісів, зокрема Google Maps API, що було ключовою вимогою проєкту.

2. АНАЛІЗ ТЕХНОЛОГІЙ ТА ОБҐРУНТУВАННЯ ВИБОРУ СТЕКУ

Google Maps API було обрано завдяки його високій точності геолокації, стабільності роботи та широкому набору інструментів для картографії й пошуку локацій. Він легко інтегрується у React Native та є найнадійнішим рішенням для подібних застосунків. Gemini використано для обробки природномовних запитів, оскільки модель добре структурує користувацькі запити та підвищує зручність взаємодії. Додатковою перевагою є гнучка система оплати API, що дозволяє оптимізувати витрати на розробку та подальшу підтримку додатка.



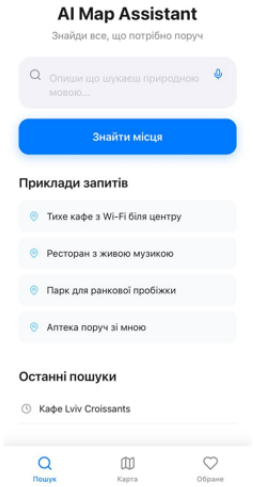
2. АНАЛІЗ ТЕХНОЛОГІЙ ТА ОБҐРУНТУВАННЯ ВИБОРУ СТЕКУ

Node.js було обрано для бекенд-частини через його високу швидкодію, просту інтеграцію з API та зручність роботи з JavaScript як на фронтенді, так і на сервері. Це дозволило створити легкий, гнучкий та швидкий сервер для обробки запитів застосунку.

Платформа Render обрана для хостингу завдяки простоті розгортання, автоматичним деплоям, стабільності та безкоштовному тарифу, достатньому для навчального та тестового застосунку. У поєднанні ці технології забезпечують надійну роботу сервера та мінімальні витрати.

3. РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

Дизайн застосунку побудований на простому та інтуїтивному інтерфейсі. Користувач швидко отримує всю необхідну інформацію про заклад: назва, категорія, рейтинг, фото та опис. Основні дії, такі як “Подзвонити” та “Маршрут”, завжди на видному місці. Інтерфейс адаптивний під різні розміри екранів, а кольори допомагають легко орієнтуватися: синій для активних дій, зелений і червоний для статусу відкриття закладу. Візуальна ієрархія побудована так, щоб користувач спершу бачив фото та назву закладу, потім категорію, рейтинг, опис і контакти. Додатково, опис місць генерується AI на основі переваг користувача, а карта інтерактивно відображає заклади, ранжовані за релевантністю.



3. РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

У мобільному додатку ми не викликаємо сторонні сервіси напряму з клієнта. Натомість фронтенд звертається до власного бекенду, який виступає посередником. Бекенд робить запити до Google Places та AI-сервісів, отримує дані і повертає вже оброблений результат. Такий підхід дозволяє повністю приховати ключі API та забезпечує безпеку додатку.

Переваги цього підходу:

1. API ключі не видно в клієнті – зменшення ризику зловживань.
2. Централізована обробка даних і логіки пошуку.
3. Можливість додати додаткову обробку або кешування на бекенді.
4. Легко підключати нові сервіси без змін на клієнті.

3. РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

- Коли користувач вводить запит у додатку (наприклад, у полі пошуку карти), функція **analyzeQuery** відправляє його на бекенд до ендпоінта /**gemini**/**analyze**. На сервері цей запит обробляє модель Gemini через клас **GoogleGenerativeAI**, яка аналізує текст і повертає два основні результати: категорію закладу (category) і набір ключових слів (keywords).
- Фронтенд отримує вже готову структуру JSON і використовує її для формування запиту до Google Places або для ранжування результатів. Такий підхід дозволяє розділити обробку запиту між фронтендом і бекендом, приховуючи ключі доступу до AI та зберігаючи безпеку даних.

```
app.post("/gemini/analyze", async (req, res) => {
  try {
    const { query } = req.body;

    if (!query || typeof query !== "string") {
      return res.json({ category: null, keywords: [] });
    }

    const model = genAI.getGenerativeModel({ model: "gemini-2.0-flash" });

    const prompt = `
      Ти — система аналізу пошукових запитів для мобільного застосунку з картою.

      Користувач вводить запит природною мовою, наприклад:
      - "затишне кафе з Wi-Fi та розетками для роботи"
      - "бар з живою музикою і коктейлями"
      - "місце для ранкової пробіжки в парку"

      Твоє завдання:
      1) Виділити тип закладу (категорію) коротким рядком, наприклад:
         "cafe", "bar", "restaurant", "park", "gym", "co-working".
      2) Виділити 2–5 ключових слів, які описують побажання користувача
         (атмосфера, бюджет, Wi-Fi, тиша, спорт, красивд, романтика, тощо).

      Формат відповіді строго JSON без пояснень:
      {
        "category": "cafe",
        "keywords": ["затишне", "wifi", "розетки", "для роботи"]
      }

      Користувацький запит:
      "${query}"
    `;

    const result = await model.generateContent(prompt);
    let text = result.response.text().trim();
```

```
export async function analyzeQuery(query: string) {
  try {
    const response = await fetch(`${BASE_URL}/gemini/analyze`, {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({ query }),
    });

    const data = await response.json();
    return {
      category: data.category || null,
      keywords: Array.isArray(data.keywords) ? data.keywords : [],
    };
  } catch (e) {
    console.error("❌ Помилка виклику backend /gemini/analyze:", e);
    return { category: null, keywords: [] };
  }
}
```


3. РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

- Додаток спершу запитує дозвіл на геолокацію і отримує координати користувача через GPS та Wi-Fi. Потім ці координати разом із ключовими словами відправляються на бекенд, де виконується пошук у Google Places API. Бекенд формує запит із зазначенням радіусу пошуку та ключового слова, отримує масив закладів із базовими даними (назва, адреса, категорія, рейтинг) і повертає його клієнту.

```
// ── 1. Отримуємо поточну локацію користувача ──────────────
useEffect(() => {
  (async () => {
    const { status } = await Location.requestForegroundPermissionsAsync();
    if (status !== 'granted') return;

    const loc = await Location.getCurrentPositionAsync({});
    setUserLocation({ lat: loc.coords.latitude, lon: loc.coords.longitude });
  })();
}, []);

// ── 2. Коли є локація + запит → виконуємо AI-аналіз і пошук ──────
useEffect(() => {
  if (!userLocation || !searchQuery.trim()) return;

  (async () => {
    try {
      setLoading(true);
      setLoadingText('Аналізуємо запит користувача за допомогою ШІ…');

      // 2.1. Аналізуємо запит через Gemini
      const aiResult = await analyzeQuery(searchQuery);
      const category = aiResult.category || null;
      const keywords: string[] = aiResult.keywords || [];

      setAiCategory(category);
      setAiKeywords(keywords);
```

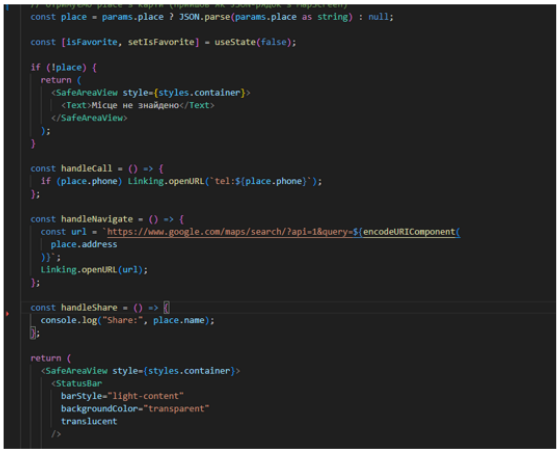
3. РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

- Після отримання списку місць із Google Places, функція **rankPlacesByRelevance** надсилає їх разом із ключовими словами користувача на бекенд у **/gemini/rank**. Сервер формує спеціальний промпт для моделі Gemini, де описує кожне місце (назва, адреса, категорія) та перелічує ключові слова запиту.
- Модель повертає JSON-масив із номерами закладів у порядку від найбільш релевантного до найменш релевантного. Фронтенд отримує вже відсортований масив, який використовується для відображення карток на екрані **MapScreen**, де користувач бачить найкращі варіанти першими.

```
const rawPlaces = await placesRes.json();
console.log('Raw Google Places response:', rawPlaces);

if (!Array.isArray(rawPlaces) || rawPlaces.length === 0) {
  setPlaces([]);
  setLoading(false);
  return;
}

setLoadingText('Готуємо результати…');

const parsed: Place[] = await Promise.all(
  rawPlaces.map(async (p: any) => {
    let imageUrl = '';
```


3. РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

- Коли користувач відкриває карту, компонент **MapScreen** викликає функцію **/places/search** на бекенді, передаючи координати користувача та ключові слова з запиту. Сервер формує запит до Google Places API, отримує список місць у радіусі, а потім повертає їх фронтенду.
- Для отримання додаткової інформації про конкретне місце використовується ендпоінт **/places/details**. Він повертає повний набір даних: адресу, номер телефону, години роботи, рейтинг, фотографії та категорії. За допомогою цих даних на екрані **PlaceDetailScreen** формується детальна картка закладу для користувача, де можна подзвонити, побудувати маршрут або поділитися інформацією.



```
const place = params.place ? JSON.parse(params.place as string) : null;

const [isFavorite, setIsFavorite] = useState(false);

if (!place) {
  return (
    <SafeAreaView style={styles.container}>
      <Text>Місце не знайдено</Text>
    </SafeAreaView>
  );
}

const handleCall = () => {
  if (place.phone) Linking.openURL(`tel:${place.phone}`);
};

const handleNavigate = () => {
  const url = `https://www.google.com/maps/search/?api=1&query=${encodeURIComponent(
    place.address
  )}`;
  Linking.openURL(url);
};

const handleShare = () => {
  console.log("Share:", place.name);
};

return (
  <SafeAreaView style={styles.container}>
    <StatusBar
      barStyle="light-content"
      backgroundColor="transparent"
      translucent
    />
```

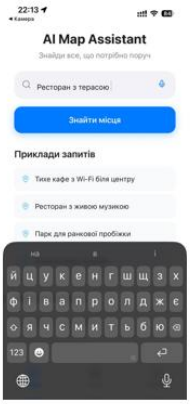
3. РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ

- Після отримання деталей закладу компонент **PlaceDetailScreen** може показати опис, створений штучним інтелектом. Для цього фронтенд викликає ендпоінт **/ai/describePlace** на бекенді, передаючи назву, адресу, рейтинг, ключові слова користувача та додаткові дані з Google Places.
- Серверна функція використовує модель **gemini-2.0-flash** через об'єкт **genAI**, формує запит до AI з інструкцією створити унікальний текст українською мовою, який описує атмосферу закладу, цільову аудиторію та враження від місця. Важливо, що AI не перераховує дані з JSON, а створює природний, емоційний опис у кілька речень.
- Отриманий опис повертається фронтенду, де відображається у секції "Опис" в **PlaceDetailScreen**, разом із фотографіями, рейтингом та контактами. Це робить інтерфейс живим і персоналізованим, підвищує залучення користувача та допомагає швидко оцінити заклад без довгих списків технічної інформації.

```
app.post("/ai/describePlace", async (req, res) => {
  try {
    const { name, address, rating, keywords = [], placeDetails = {} } = req.body;

    const model = genAI.getGenerativeModel({ model: "gemini-2.0-flash" });

    const prompt = `
      Створи унікальний, гарно написаний опис закладу українською мовою.
      Опиши атмосферу, враження, тип аудиторії та причини, чому місце може підійти користувачу.

      Використай такі дані:
      Назва: ${name}
      Адреса: ${address}
      Рейтинг: ${rating || "без рейтингу"}
      Побажання користувача: ${keywords.join(", ")}
      Google details: ${JSON.stringify(placeDetails)}

      Вимоги:
      - Тон: дружній, професійний, живий, без канцелярщини.
      - Обсяг: 3–6 речень.
      - Ніяких списків, маркерів, нумерацій.
      - НЕ перераховувати дані із JSON (типу "рейтинг 4.7" або "Адреса така-то").
      - Просто створи гарний, емоційний текст, який читається природно.
      `;

    const r = await model.generateContent(prompt);
    const description = r.response.text().trim();

    res.json({ description });
  } catch (e) {
    console.error("❌ /ai/describePlace error:", e);
    res.json({ description: "" });
  }
});
```


4. ОПИС РОБОТИ МОБІЛЬНОГО ЗАСТОСУНКУ

- давайте перевіримо роботу додатку написавши якийсь запит , в моєму випадку я обрав такий «ресторан з терасою»
- Після того як користувач ввів запит та натиснув кнопку пошуку, нас перекидує у вікно з картою , де ми бачимо саму карту яка є основним елементом. На ній зображені маркери та локація користувача
- Далі під картою знаходиться блок з картакми локацій де стисло описана основна інформація , а також відображене фото закладу.
- Останнім екраном в нашому додатку буде детальна інформація про заклад. Як на мене основним тут є якраз опит від ШІ який є й головним принципом.



Висновки

- Досліджено методи інтеграції AI і зовнішніх API у мобільний застосунок для роботи з картами та місцями.
- Показано що використання AI для аналізу запитів користувачів і генерації описів підвищує зручність і персоналізацію пошуку.
- Встановлено що поєднання AI та картографічних сервісів значно покращує користувацький досвід і ефективність роботи застосунку.
- Апробація
- III всеукраїнська науково-технічна конференція «Технологічні горизонти: дослідження та застосування інформаційних технологій для технологічного прогресу України і світу», 18 листопада 2025 року, ДУКТ – «Обмеження штучного інтелекту, використаного через API, у мобільних застосунках» , «Інтеграція чат-ботів у мобільні застосунки для покращення взаємодії користувача з системою»
- VIII Всеукраїнська науково- технічна конференція «Комп'ютерні технології: інновації, проблеми, рішення», 02-03 жовтня 2025 року, Житомирська політехніка - «Інтелектуальні методи фільтрації даних у сучасних інформаційних системах»