Вінницький національний технічний університет
(повне найменування вищого навчального закладу)

Факультет інформаційних технологій та комп'ютерної інженерії
(повне найменування інституту, назва факультету (відділення))

Кафедра обчислювальної техніки
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«ПРОГРАМНИЙ КОМПЛЕКС ЗБОРУ ТА ВИКОРИСТАННЯ ДАНИХ З ДОПОМОГОЮ ШТУЧНОГО ІНТЕЛЕКТУ»

08-54.МКР.003.00.000 ПЗ

Виконав: студент 2-го курсу, групи
1КІ-23м спеціальності 123 «Комп'ютерна інженерія »
(шифр і назва напряму підготовки, спеціальності)
_____ Велянський С.А.
(прізвище та ініціали)

Керівник: к. т. н., доц. каф. ОТ
_____ Муращенко О.Г.
(прізвище та ініціали)
«12» 12 2024 р.

Опонент: к.т.н., доц. каф.
_____ Бабюк Н.П.
(прізвище та ініціали)
«18» 12 2024 р.

Допущено до захисту
Завідувач кафедри ОТ
_____ д.т.н., проф. Азаров О. Д.
(прізвище та ініціали)
«_____» _____ 2024 р.

Вінниця ВНТУ- 2024 рік

ВІННИЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 123 « Комп'ютерна інженерія »

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
д.т.н., професор Азаров О.Д.
"18" вересня 2024 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТА

Велянського Сергія Андрійовича
(прізвище, ім'я, по батькові)

1 Тема роботи: Програмний комплекс збору та використання даних з допомогою штучного інтелекту, керівник роботи к. т. н., доц. кафедри ОТ Муращенко О.Г., затверджені наказом вищого навчального закладу від "17" вересня 2024 року № 310.

2 Строк подання студентом роботи: 19.12.2024

3 Вихідні дані до роботи: фреймворк SwiftUI, бібліотека Alamofire, ChatGPT API, Gemini API, Google Maps SDK, Google Maps API, програмне середовище Xcode, операційна система MacOS.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, аналіз предметної області, вдосконалення методів генерації та організації туристичних даних, програмна реалізація програмного комплексу збору та використання даних з допомогою штучного інтелекту, тестування та аналіз ефективності програмного комплексу, економічна частина, література.

5 Перелік ілюстративного матеріалу: структурна схема, блок-схема алгоритму роботи, лістинг модуля транскипції та діаризації, лістинг модуля взаємодії з даними, протокол перевірки кваліфікаційної роботи на наявність текстових запозичень.

6 Консультанти розділів роботи представлено в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1–4	Муращенко О.Г., к. т. н., доцент каф. ОТ		
4	Небава М. І., доцент, к.е.н., професор каф. ЕПВМ		

7 Дата видачі завдання: 18.09.2024

8 Календарний план роботи зазначений в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Постановка задачі роботи	19.09.2024	виконано
2	Аналіз предметної області	20.09.2024–24.09.2024	виконано
3	Вдосконалення методів генерації та організації туристичних даних	25.09.2024 – 04.10.2024	виконано
4	Програмна реалізація програмного комплексу збору та обробки даних з допомогою штучного інтелекту	05.10.2024 – 26.10.2024	виконано
5	Розрахунок економічної частини	11.11.2024	виконано
6	Оформлення матеріалів до захисту МКР	12.11.2024	виконано
7	Перевірка якості виконання магістерської роботи та усунення недоліків	13.11.2024 – 14.11.2024	виконано
8	Підписи супроводжувальних документів у нормоконтролера, керівника, опонента	15.11.2024 – 16.11.2024	виконано
9	Перевірка на антиплагіат та ШІ	17.11.2024	виконано
10	Попередній захист роботи	18.11.2024	виконано

Студент _____ (підпис) Велянський С.А.

Керівник роботи _____ (підпис) Муращенко О.Г.

АНОТАЦІЯ

УДК 004.9

Велянський С.А. Програмний комплекс збору та використання даних з допомогою штучного інтелекту. Магістерська кваліфікаційна робота зі спеціальності 123 – Комп'ютерна Інженерія,. Вінниця: ВНТУ, 2024. 105 с.

На укр. мові. Бібліогр.: назв 60; рис.: 14; табл.6.

Магістерська кваліфікаційна робота присвячена розробці програмного комплексу збору та використання даних з допомогою штучного інтелекту. У роботі використано генеративну мовну модель ChatGPT API для формування динамічних запитів, інтеграцію Google Maps API для отримання даних про об'єкти інтересу та Gemini API для збагачення інформації додатковими характеристиками. Структуризація й збереження даних здійснюється за допомогою Core Data, а інтерфейс реалізовано з використанням SwiftUI. Система забезпечує автоматизацію пошуку, обробки та візуалізації туристичних маршрутів на мобільних пристроях iOS. Результати роботи підтверджують ефективність розробленого підходу для задач персоналізованого планування подорожей. Робота містить 105 сторінок, 14 рисунків, 6 таблиць.

Ключові слова: автоматизований збір даних, туристичні маршрути, генеративний штучний інтелект, Google Maps API, Core Data, SwiftUI, мобільний додаток.

ABSTRACT

Velianskyi S.A. Software System for Data Collection and Utilization Using Artificial Intelligence. Master’s Qualification Thesis in Specialty 123 – Computer Engineering. Vinnytsia: VNTU, 2024. 105p.

In the Ukrainian language. Bibliogr .: titles 60; fig .14; table 6.

The master's qualification work is dedicated to the development of a software system for data collection and utilization using artificial intelligence. The work employs the generative language model ChatGPT API for generating dynamic queries, integrates the Google Maps API for retrieving data about points of interest, and uses the Gemini API to enrich the information with additional attributes. The structuring and storage of data are implemented using Core Data, while the interface is developed with SwiftUI. The system automates the processes of searching, processing, and visualizing tourist routes on iOS mobile devices. The results confirm the efficiency of the proposed approach for solving tasks related to personalized travel planning. The thesis contains 105 pages, 14 figures, and 6 tables.

Keywords: automated data collection, tourist routes, generative artificial intelligence, Google Maps API, Core Data, SwiftUI, mobile application.

ЗМІСТ

ВСТУП 8

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ, ІСНУЮЧИХ ПРОБЛЕМ ТА МЕТОДІВ ЇХ ВИРІШЕННЯ 11

1.1 Аналіз предметної області 11

1.2 Методи збору та обробки даних у туристичних системах 13

1.3 Генеративний штучний інтелект у задачах автоматизації 15

1.3.1 Формування запитів для збору туристичних даних 15

1.3.2 Організація згенерованих даних 18

1.3.3 Складність впровадження технології 20

1.4 Порівняння генеративних мовних моделей 23

1.5 Аналіз існуючих рішень та архітектур у туристичних додатках 26

2 ВДОСКОНАЛЕННЯ МЕТОДУ ГЕНЕРАЦІЇ ЗАПИТІВ ДЛЯ ЗБОРУ ТА ОРГАНІЗАЦІЇ ДАНИХ 30

2.1 Створення автоматизованої генерації 30

2.2 Підвищення якості та точності отриманих даних 31

2.3 Вдосконалення методу організації даних штучного інтелекту 33

2.4 Інтеграція отриманих даних у програмний комплекс 35

2.5 Аналіз альтернативних методів генерації запитів 36

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ ЗБОРУ ТА ОРГАНІЗАЦІЇ ДАНИХ 39

3.1 Створення та налаштування бази даних 39

3.2 Реалізація сервісу генерації запитів 42

3.3 Використання функціональної логіки у програмному комплексі ... 47

3.4 Створення інтерфейсу для взаємодії користувача..... 50
4 ТЕСТУВАННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ ПРОГРАМНОГО КОМПЛЕКСУ..... 55
4.1 Тестування комплексу та його функціональності..... 55
4.2 Порівняння результатів тестування додатків..... 58
4.3 Оцінка ефективності програмного комплексу..... 60
4.4 Перспективи подальшого розвитку програмного комплексу 62
5 ЕКОНОМІЧНА ЧАСТИНА..... 64
5.1 Комерційний та технологічний аудит науково-технічної розробки.... 64
5.2 Комерційний та технологічний аудит науково-технічної розробки.... 67
5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором 73
ВИСНОВКИ 79
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ..... 81
ДОДАТОК А Технічне завдання..... 87
ДОДАТОК Б..... 91
ДОДАТОК В..... 92
ДОДАТОК Г 93
ДОДАТОК Д..... 99
ДОДАТОК Е..... 105
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ..... 105

ВСТУП

З розвитком інформаційних технологій та поширенням мобільних додатків значно зросла **актуальність** у ефективному зборі, обробці та організації даних. Сьогодні туристична індустрія є одним із ключових секторів, де автоматизація цих процесів забезпечує зручність для користувачів і оптимізацію ресурсів. Туристам важливо не лише отримувати швидкий доступ до інформації про цікаві місця, але й мати можливість планувати маршрути, адаптовані під їхні індивідуальні потреби. Саме цю проблему вирішують сучасні програмні системи, побудовані на основі технологій штучного інтелекту.

Традиційні системи для пошуку туристичних об’єктів, таких як Google Maps, надають користувачам можливість обирати маршрути та знаходити точки інтересу на карті. Проте їхні можливості часто є обмеженими у плані автоматизації та персоналізації даних. У користувачів виникають складнощі при побудові складних маршрутів, пошуку місць різних категорій або при необхідності врахування індивідуальних уподобань.

Інтеграція штучного інтелекту, зокрема генеративних мовних моделей, у процес пошуку та організації даних дозволяє створити універсальні методи генерації запитів для пошуку інформації. Генеративний ШІ, наприклад, на основі API ChatGPT, здатен формувати запити, адаптовані під конкретні категорії місць – культурні об’єкти, ресторани, готелі або природні локації. Подальша інтеграція цих даних з картографічними сервісами, такими як Google Maps, дозволяє будувати найоптимальніші маршрути для подорожей.

Метою дослідження є вдосконалення автоматизованого збору та обробки туристичних даних з використанням генеративних мовних моделей та картографічних сервісів та подальше використання їх у програмному комплексу. Такий комплекс дозволить зпростити процес пошуку місць для відвідування,

забезпечуючи зручний інтерфейс для отримання релевантної інтформації та побудови маршрутів.

Для досягнення мети роботи потрібно виконати такі **задачі**:

— провести аналіз існуючих рішень та методів для генерації запитів і збору даних;

— вдосконалити метод для генерації запитів, що забезпечує пошук інформації різних категорій;

— реалізувати програмний комплекс, що включає модулі формування запитів, організації отриманих даних і побудови маршрутів;

— забезпечити інтеграцію розробленого комплексу з картографічним сервісом Google Maps для візуалізації даних;

— провести тестування системи, оцінити ефективність та точність роботи розробленого комплексу.

Об'єкт дослідження — процес автоматизації збору та організації туристичних даних, що включає використання засобів генеративного штучного інтелекту.

Предмет дослідження — методи та алгоритми автоматизованої генерації запитів, інтеграції з API та побудови маршрутів на основі отриманих даних.

У роботі використані такі **методи дослідження**: методи аналізу, методи генеративного штучного інтелекту, методи об’єктно-орієнтованого програмування, методи інтеграції зовнішніх сервісів, методи тестування програмного забезпечення.

Новизна дослідження — полягає у розробці універсального методу генерації запитів для збору даних різних категорій, що дозволяє підвищити точність і релевантність отриманої інформації. Запропонований метод базується на використанні сучасних генеративних мовних моделей у поєднанні з картографічними сервісами.

Практичне значення — розроблений програмний комплекс застосовується для автоматизованого пошуку туристичних даних і їх подальшої організації, побудови маршрутів і візуального відображення даних. Це дозволяє полегшити доступ до туристичних даних та структурованого планування подорожей.

Запропонований комплекс сприяє підвищенню ефективності процесу автоматизації взаємодії з штучним інтелектом, оскільки:

— розроблено методи універсальної генерації запитів та збору туристичних даних з використанням штучного інтелекту;

— реалізовано взаємодію модулів з картографічними сервісами;

— візуалізовано інтерфейс програмного комплексу для легкої взаємодії з даними.

Апробація результатів роботи була здійснена на міжнародній науково-практичній інтернет-конференції «Молодь в науці: дослідження, проблеми, перспективи – 2024».

Публікації за результатами досліджень опубліковано тези доповідей на науково-технічній конференції [1]:

Велянський С.А., Муращенко О.Г. Автоматизований збір та організація туристичних даних із використанням штучного інтелекту Конференція ВНТУ: Молодь в науці: дослідження, проблеми, перспективи (МН-2024). Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/view/22954.

.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ, ІСНУЮЧИХ ПРОБЛЕМ ТА МЕТОДІВ ЇХ ВИРІШЕННЯ

1.1 Аналіз предметної області

Сучасна туристична галузь, будучи однією з найбільш динамічних сфер економіки, активно використовує інформаційні технології для підвищення зручності користувачів і вдосконалення процесів. Зростання цифровізації значно вплинуло на спосіб організації подорожей: автоматизація стала ключовим фактором у плануванні маршрутів, пошуку інформації про туристичні об’єкти та взаємодії з користувачами.

Інтеграція технологій у туризм сприяє створенню систем, які здатні надавати персоналізовані рекомендації та швидко адаптувати маршрути до змін у реальному часі. Наприклад, геолокаційні сервіси використовуються для визначення місцезнаходження користувача, пошуку об’єктів поблизу та побудови оптимальних маршрутів. Такі технології, як картографічні сервіси, дають можливість не лише відображати локації на мапі, але й забезпечувати інтерактивність для користувачів.

Однак, попри широкий спектр можливостей, які пропонують сучасні технології, туристична галузь стикається з низкою проблем. Однією з них є недостатній рівень персоналізації, оскільки багато платформ пропонують стандартизовані рішення, які не враховують специфічні потреби користувачів. Ще однією складністю є інтеграція даних із різних джерел, які часто мають неоднорідну структуру. Це ускладнює їх обробку та створює перешкоди для побудови узгоджених маршрутів.

Слід також зазначити, що можливості багатьох систем залишаються обмеженими, адже вони не забезпечують динамічного оновлення маршрутів чи швидкого доступу до нових об’єктів. Наприклад, у разі змін умов подорожі, таких як погода чи дорожній трафік, користувачі змушені вручну коригувати свої

плани. Це створює додаткові незручності, які можна було б усунути завдяки автоматизованим рішенням [2].

Розвиток туристичних систем проходить поступову еволюцію. Якщо на початкових етапах їхньою основною функцією було надання базової інформації про туристичні об’єкти, то сьогодні вони трансформуються у комплексні сервіси, здатні забезпечувати повний цикл планування. Від простого пошуку інформації про готелі та ресторани до створення інтегрованих екосистем, які включають рекомендації, маршрутизацію та адаптацію даних до реальних умов подорожі. Саме цей перехід є ключовим фактором у розвитку галузі, оскільки він спрямований на покращення користувацького досвіду.

Згідно зі звітом "Business of Apps", обсяг ринку туристичних додатків зростає з кожним роком, не враховуючи рік пандемії, та за 2023 рік було зафіксовано $600 мілярдів рисунок 1.1.

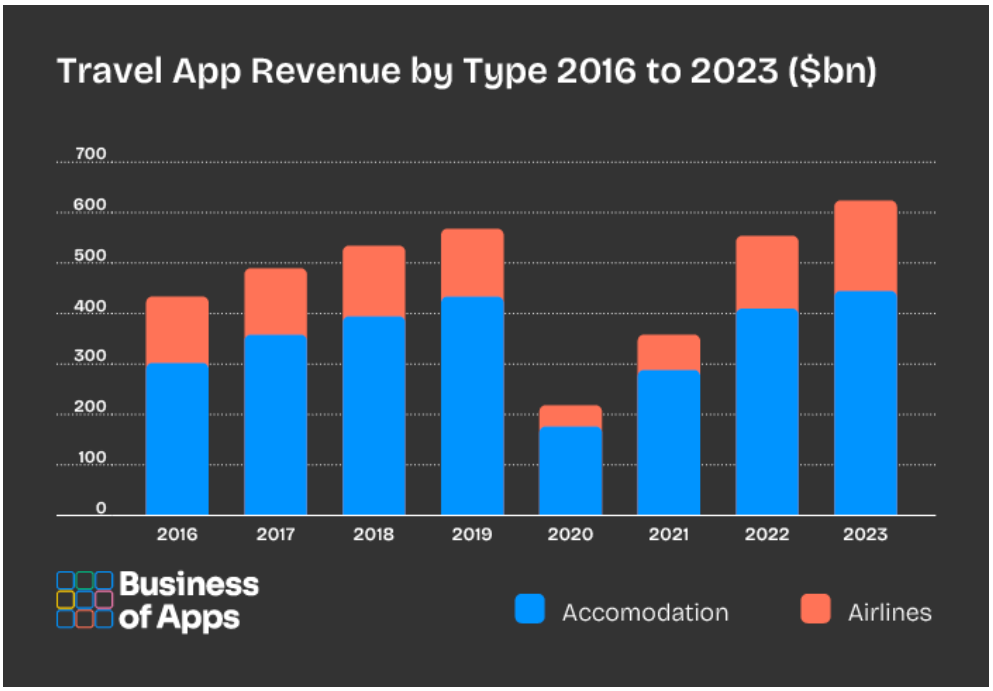

Рисунок 1.1 — Обсяг ринку туристичних додатків.

Отже, аналіз предметної області демонструє, що сучасний туризм потребує технологій, які б забезпечували не лише швидкість і точність роботи, але й гнучкість у адаптації до змін. Інформаційні технології відіграють тут провідну роль, адже саме вони створюють основу для побудови автоматизованих і персоналізованих сервісів [3, 4].

Однак не варто забувати про пандемію та яку корективу вона внесла в статистику та обсяг ринку, а також проблеми локалізації для східних народів. Саме тому необхідно і далі розвивати цю сферу, аби туризм був доступний для кожного та на необхідно рівні розуміння.

1.2 Методи збору та обробки даних у туристичних системах

Автоматичний переклад стикається із значними викликами, які впливають на його якість і точність. Проблеми, пов'язані з обмеженою підтримкою мов, діалектів, низькою якістю аудіозаписів і фоновими шумами, суттєво ускладнюють досягнення високої точності в обробці мовлення. Ці виклики є особливо критичними для малоресурсних мов, де нестача даних ускладнює створення якісних моделей. Однак сучасні підходи, такі як перенесення навчання (transfer learning), багатомовне навчання (multilingual learning) та використання автоматизованого розширення даних (data augmentation), відкривають нові можливості для вирішення цих проблем і забезпечують перспективи для розвитку технологій автоматичного перекладу мультимедіа.

Сучасні туристичні системи значною мірою залежать від якості зібраних даних та ефективності їхньої обробки. Дані, отримані з різних джерел, є основою для побудови маршрутів, формування рекомендацій та забезпечення актуальної інформації для користувачів. Основними джерелами є геолокаційні сервіси, бази даних про туристичні об’єкти, відгуки користувачів, а також погодні сервіси, що дозволяють враховувати змінні умови подорожі.

Процес збору даних у туристичних системах включає інтеграцію з зовнішніми API, які надають доступ до структурованої інформації про об’єкти, їхні координати, графіки роботи та рейтинги. Наприклад, використання API дозволяє отримувати дані про ресторани, готелі або музеї, адаптуючи інформацію відповідно до запитів користувача. Однак, збір даних потребує врахування таких факторів, як релевантність, актуальність та надійність інформації.

Методи обробки даних базуються на алгоритмах класифікації та структурування отриманої інформації. Наприклад, дані можуть бути згруповані за категоріями: культурні об’єкти, природні локації, заклади харчування тощо. Це дозволяє значно спростити побудову персоналізованих маршрутів і скоротити час на пошук інформації. У процесі обробки застосовуються алгоритми фільтрації для відсіювання нерелевантних даних і виділення найважливіших об’єктів.

Одним із ключових викликів у цьому процесі є проблема неоднорідності даних. Інформація, отримана з різних джерел, часто має різну структуру, що ускладнює її об’єднання в єдину базу. Для вирішення цієї проблеми використовуються спеціалізовані алгоритми об’єднання, які дозволяють нормалізувати дані, усуваючи дублікати та конфлікти. Наприклад, дані про одну і ту ж локацію можуть надходити від кількох джерел, але мати різний формат, через що потрібна їх автоматична синхронізація.

Ще одним важливим аспектом є використання технологій для динамічного оновлення даних. У реальному часі можуть змінюватися графіки роботи об’єктів, погодні умови або дорожня ситуація, що впливає на ефективність побудови маршрутів. Використання динамічних запитів дозволяє забезпечити актуальність отриманої інформації, що значно підвищує якість сервісу для користувача.

Застосування методів аналізу великих даних також грає важливу роль у сучасних туристичних системах. Завдяки технологіям Big Data можливе

вивчення поведінки користувачів, аналіз їхніх уподобань та прогнозування запитів. Це дозволяє не лише покращувати функціонал систем, але й створювати нові можливості для персоналізації. Наприклад, аналіз історії пошукових запитів дає змогу пропонувати користувачам рекомендації, які максимально відповідають їхнім потребам [5-7].

Таким чином, методи збору та обробки даних у туристичних системах є багаторівневим процесом, який охоплює інтеграцію з джерелами даних, їх нормалізацію, класифікацію та динамічне оновлення. Ефективність цих методів безпосередньо впливає на якість сервісу, що надається користувачам, а також на здатність системи адаптуватися до змін у реальному часі. У подальших підрозділах розглядатимуться інноваційні підходи, зокрема використання генеративного штучного інтелекту для автоматизації цих процесів.

1.3 Генеративний штучний інтелект у задачах автоматизації

1.3.1 Формування запитів для збору туристичних даних

Процес формування запитів у туристичних системах є ключовим етапом у забезпеченні зручності та персоналізації сервісів для користувачів. Формування запиту полягає у створенні структурованої команди для отримання релевантних даних із зовнішніх джерел, таких як API, які підтримують взаємодію з базами даних туристичних об’єктів, маршрутів чи аналітичної інформації.

У сучасних туристичних системах широко використовуються RESTful API, що підтримують роботу з форматами даних JSON або XML, використання можна побачити на рисунку 1.2. Це забезпечує стандартизацію запитів і спрощує інтеграцію інформаційних систем. Наприклад, такі API, як Google Places API або Gemini API, дозволяють отримувати інформацію про об'єкти за їх географічним розташуванням, категорією або рейтингом.

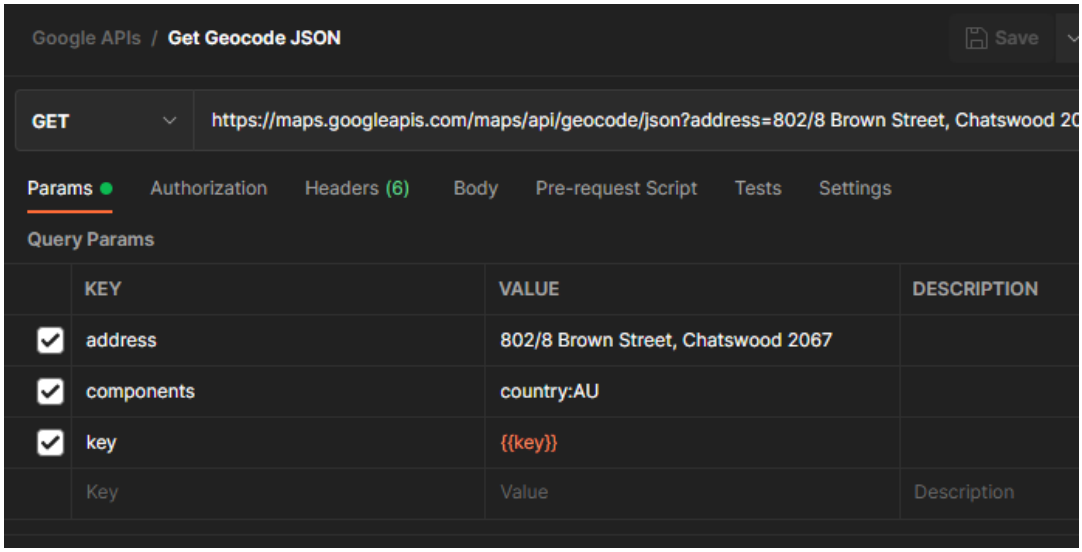

Рисунок 1.2 — Взаємодія з RESTful API за допомогою POSTMAN.

Однією з важливих тенденцій у формуванні запитів є використання генеративного штучного інтелекту, такого як ChatGPT, який забезпечує автоматизацію цього процесу. ШІ аналізує запити користувачів природною мовою, адаптує їх до формату, необхідного для роботи з API, і генерує параметризовані команди. Наприклад, коли користувач запитує «Які музеї відкриті поблизу мене?», генеративна модель аналізує ключові елементи запиту – категорію об’єкта, розташування та графік роботи – і перетворює це на структурований запит.

Формування запитів включає кілька важливих етапів:

— визначаються ключові параметри, такі як геолокація, категорія об’єктів та додаткові фільтри (рейтинг, графік роботи тощо);

— визначені параметри перетворюються у формат, придатний для обробки API, наприклад, у вигляді URL із параметрами;

— враховуються обмеження API, наприклад, ліміти на кількість результатів або швидкість обробки.

Ці етапи забезпечують точність і релевантність отриманих результатів, що є важливим для створення зручного туристичного сервісу.

Google Maps API та Gemini API є прикладами сервісів, які дозволяють отримувати інформацію про туристичні об'єкти та маршрути. Google Maps API спеціалізується на даних про місцезнаходження та маршрутизацію, тоді як Gemini API може інтегрувати аналітичні дані для покращення персоналізації сервісів.

Ключовими перевагами роботи з API є:

— стандартизовані формати даних, які забезпечують сумісність із більшістю інформаційних систем;

— можливість фільтрації результатів відповідно до заданих критеріїв, таких як рейтинг або тип об'єкта;

— динамічне оновлення інформації в реальному часі.

Генеративний ШІ, зокрема ChatGPT, дозволяє скоротити час на створення запитів і підвищити їх точність. Завдяки здатності працювати з природною мовою, ШІ аналізує запити користувачів і адаптує їх до технічних вимог API. Такий підхід знижує складність для кінцевого користувача, дозволяючи отримувати інформацію за запитами безпосередньо мовою, зрозумілою людині.

Формування запитів також враховує можливість інтеграції з платформами візуалізації, такими як Google Maps, для відображення отриманих даних. Наприклад, після створення запиту результати можуть бути використані для побудови маршруту чи відображення маркерів на карті [8-10].

Таким чином, формування запитів для туристичних систем є багатокомпонентним процесом, який забезпечує зручність і персоналізацію сервісів. Інтеграція генеративного штучного інтелекту з API відкриває нові можливості для автоматизації цього процесу, підвищуючи його ефективність і точність. У наступних підрозділах буде розглянуто методи організації даних, отриманих через ці запити.

1.3.2 Організація згенерованих даних

Отримані в результаті запитів дані потребують правильної організації для забезпечення їхньої зручної обробки, аналізу та інтеграції у туристичні системи. Процес організації включає структурування, фільтрацію та категоризацію даних, що дозволяє користувачам швидко отримувати релевантну інформацію відповідно до своїх запитів. Основна мета цього етапу – забезпечення логічного подання інформації, яке спрощує її використання в реальних умовах.

У сучасних туристичних системах організація даних базується на кількох ключових принципах. Перш за все, важливо забезпечити їхню уніфікацію, адже джерела, з яких надходить інформація (Google Maps API, Gemini API та інші), можуть надавати дані у різних форматах. Наприклад, Google Maps API повертає JSON-об’єкти зі структурованою інформацією про місцезнаходження, назву, рейтинг та інші параметри. Водночас, Gemini API може надавати додаткову аналітичну інформацію, яка потребує узгодження зі структурою основної бази даних.

Усі дані групуються за категоріями, такими як ресторани, культурні об’єкти, природні локації чи готелі. Це дозволяє швидко орієнтуватися у великому масиві інформації. Наприклад, турист, який шукає місця для відпочинку, отримує лише дані, що стосуються парків чи зон для відпочинку, а не закладів харчування.

Після отримання даних вони проходять фільтрацію за заданими критеріями: рейтингом, популярністю, відстанню від користувача або графіком роботи. Наприклад, у разі пошуку ресторанів система автоматично виключає заклади, які наразі зачинені.

Усі дані приводяться до єдиного формату, що дозволяє уникнути проблем із дублюванням або конфліктом структур. Наприклад, дані про геолокацію

можуть бути представлені у вигляді широти та довготи, що уніфікується для подальшої роботи з картографічними сервісами [11-15].

Для забезпечення ефективної роботи туристичних систем створюються локальні або хмарні бази даних, які виконують роль сховища для отриманої інформації. Бази даних дозволяють не лише зберігати дані, але й виконувати складні запити для їхнього аналізу, а приклад структури бази даних можна побачити на рисунку 1.3.

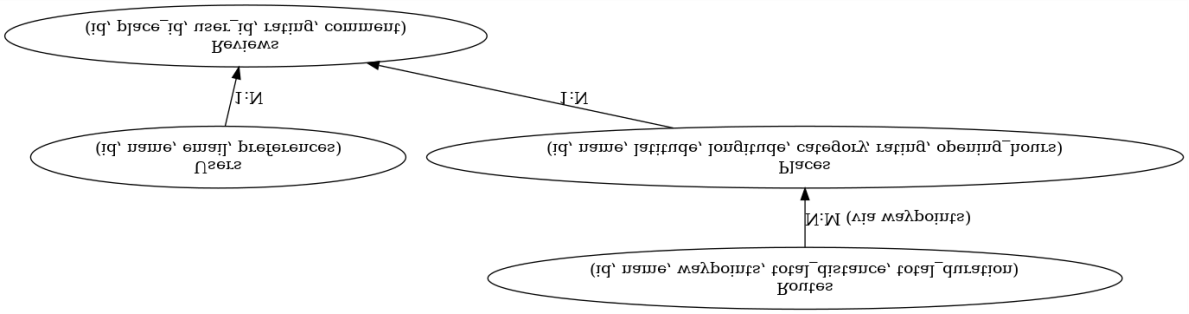

Рисунок 1.3 — Приклад структури бази даних.

Одним із основних викликів у процесі організації даних є неоднорідність отриманих результатів. Наприклад, Google Maps API може надавати повну інформацію про місце, тоді як Gemini API обмежується лише координатами або загальними категоріями. Для вирішення цих проблем застосовуються спеціальні алгоритми зіставлення, які поєднують дані за ключовими параметрами, такими як координати чи назва об’єкта.

Організація даних також включає забезпечення їхнього динамічного оновлення, щоб інформація завжди залишалася актуальною. Наприклад, зміни у графіку роботи або доступності об’єкта повинні негайно враховуватися в системі. Для цього використовуються механізми динамічних запитів, які періодично перевіряють оновлення у джерелах даних.

Організовані дані знаходять своє застосування у багатьох функціях туристичних систем:

— дані передаються у модулі маршрутизації, які формують найзручніші шляхи між обраними точками;

— на основі зібраних даних створюються персоналізовані рекомендації для користувачів;

— координати та атрибути об’єктів використовуються для відображення точок інтересу на інтерактивних картах.

Як результат, організація згенерованих даних є критичним етапом у роботі туристичних систем, оскільки вона забезпечує ефективність обробки та доступність інформації для користувачів.

1.3.3 Складність впровадження технології

Впровадження технологій, які базуються на генеративному штучному інтелекті, вимагає подолання низки складностей, пов’язаних як із технічними аспектами, так і з організаційними та ресурсними обмеженнями. Хоча ці технології відкривають значні можливості для автоматизації та покращення туристичних сервісів, їх адаптація під конкретні задачі потребує комплексного підходу. Оцінку використання згенерованої інформації відображено на рисунку 1.4.

Однією з головних проблем є інтеграція генеративних моделей із зовнішніми API, такими як Google Maps API або Gemini API. Для коректного функціонування необхідно забезпечити узгодженість між форматами даних, які обробляються моделлю, і вимогами API. Генеративний ШІ, наприклад, ChatGPT, може створювати ефективні запити, але дані, отримані від зовнішніх джерел, можуть мати неоднорідну структуру, що вимагає додаткових алгоритмів нормалізації.

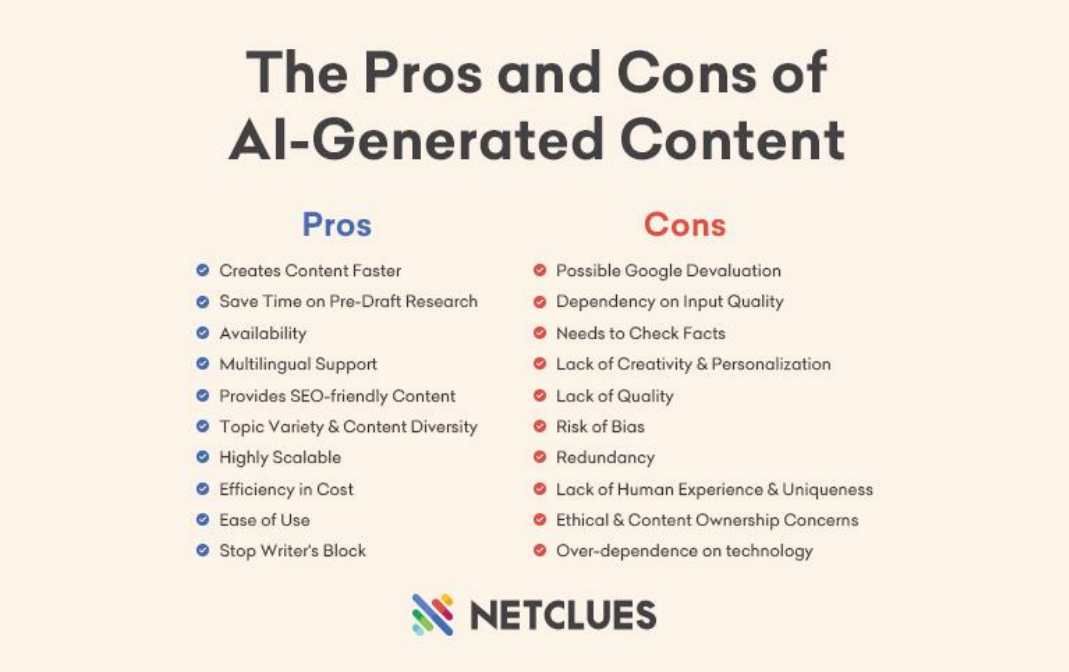

Рисунок 1.4 — Розмір ринку цифрового контенту.

Інша технічна проблема — високі вимоги до обчислювальних ресурсів. Генеративні моделі, особливо великі мовні моделі, такі як GPT-4, потребують значних обчислювальних потужностей, що може ускладнювати їх використання на пристроях із обмеженими ресурсами, наприклад, у мобільних додатках. Для вирішення цієї проблеми використовуються хмарні сервіси, які дозволяють виконувати складні обчислення на віддалених серверах.

Інтеграція генеративного ШІ із системами, які вже використовують традиційні методи збору та аналізу даних, також викликає певні труднощі. Наприклад, Google Maps SDK та Gemini API надають дані у форматах, які не завжди легко адаптувати для моделювання нових сценаріїв або запитів, створених ChatGPT. Для подолання цих складностей розробляються спеціалізовані інтерфейси обміну даними [16-19].

Крім того, важливо враховувати ліміти API, встановлені провайдерами сервісів. У випадку великої кількості запитів або інтенсивного використання

системи ці ліміти можуть стати серйозним обмеженням, що потребує оптимізації запитів і кешування отриманих даних.

Впровадження таких технологій часто потребує великих інвестицій у навчання персоналу та адаптацію інфраструктури. Розробники повинні володіти спеціалізованими знаннями у сфері генеративного ШІ, роботи з API та інтеграції різних систем. Крім того, налаштування серверів для обробки даних і забезпечення захисту конфіденційної інформації користувачів вимагає додаткових витрат.

Іншою важливою складністю є забезпечення конфіденційності даних користувачів, особливо в умовах інтеграції з зовнішніми API. Дані про геолокацію, маршрути та уподобання користувачів є чутливими, і їх обробка вимагає суворого дотримання стандартів безпеки. Крім того, генеративні моделі можуть створювати запити, які призводять до некоректних або етично сумнівних результатів, тому потрібна додаткова перевірка їх роботи [20].

Для подолання зазначених складностей впроваджуються такі підходи:

— оптимізація обчислювальних процесів за допомогою хмарних платформ, що дозволяє знижувати навантаження на локальні пристрої.

— розробка адаптерів для інтеграції даних із зовнішніх API з генеративними моделями, адже це допомагає спрощувати обмін даними між компонентами системи.

— впровадження механізмів контролю за роботою ШІ для зменшення ризику створення некоректних запитів.

— постійне оновлення систем безпеки для забезпечення захисту даних користувачів.

Складність впровадження технологій генеративного ШІ є багатофакторною проблемою, яка потребує комплексного підходу для її вирішення. Однак, подолання цих викликів відкриває нові можливості для

створення інноваційних туристичних сервісів, які надають високоякісний і персоналізований досвід для користувачів.

1.4 Порівняння генеративних мовних моделей

Генеративні мовні моделі є ключовим елементом сучасних систем автоматизації завдань, що пов’язані з обробкою природної мови. У межах цієї роботи використовується інтеграція ChatGPT API для формування текстових запитів на основі вибору категорії користувачем. Проте важливо розглянути альтернативні генеративні моделі та їх особливості, щоб обґрунтувати вибір оптимального рішення.

На сучасному етапі розвитку технологій найпоширенішими мовними моделями є GPT (Generative Pre-trained Transformer) від OpenAI, BERT (Bidirectional Encoder Representations from Transformers) від Google, а також T5 (Text-to-Text Transfer Transformer). Усі ці моделі працюють на основі архітектури Transformer, проте вони мають свої відмінності у підходах до навчання, генерації тексту та практичних сценаріях використання. Архітектуру GPT зображено на рисунку 1.5.

Модель GPT, до якої належить і ChatGPT, є найбільш популярним рішенням для генерації тексту завдяки своїй гнучкості та широкому спектру застосувань. GPT використовує авторегресивний підхід, що дозволяє прогнозувати наступне слово в тексті на основі попередніх слів. Завдяки цьому модель здатна генерувати зв'язний і контекстуально точний текст на основі заданого запиту [21-24].

Головною перевагою GPT є її здатність адаптуватися до різних завдань без необхідності додаткового навчання. Це робить її ідеальним рішенням для автоматизованого формування запитів у туристичній системі, де необхідно генерувати специфічний текст на основі категорії місць (наприклад, ресторани, готелі чи культурні об’єкти).

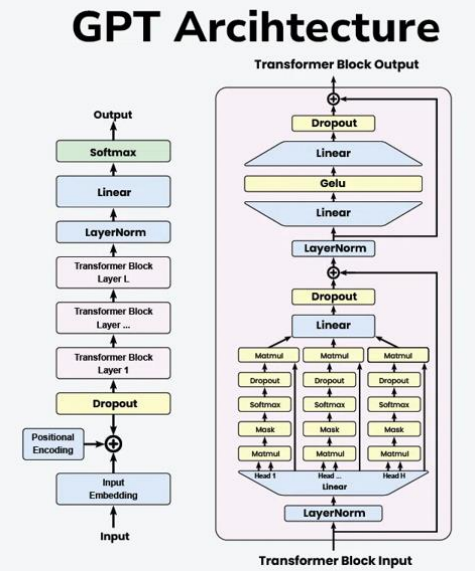

Рисунок 1.5 — Архітектура GPT.

На відміну від GPT, BERT використовує двосторонній підхід до обробки тексту. Це означає, що модель аналізує контекст як зліва, так і справа від цільового слова, що робить її ефективнішою для завдань класифікації та розуміння тексту. Проте BERT не призначена для генерації нових текстів, оскільки вона не є авторегресивною моделлю. Основний сценарій її використання — це завдання на пошук, аналіз та узагальнення тексту, де потрібна висока точність інтерпретації.

У контексті даного проєкту BERT була б менш ефективною, оскільки основним завданням є генерація тексту для формування пошукових запитів, а не лише їх аналіз дивись рисунок 1.6.

Модель T5 від Google використовує підхід “Text-to-Text”, де будь-яке завдання формулюється як трансформація тексту. Це дозволяє моделі виконувати широкий спектр завдань, від машинного перекладу до узагальнення тексту чи відповіді на питання. T5 є дуже гнучкою моделлю, що робить її конкурентною з GPT, проте для завдань, пов’язаних із генерацією текстів, вона вимагає точного налаштування на конкретну задачу.

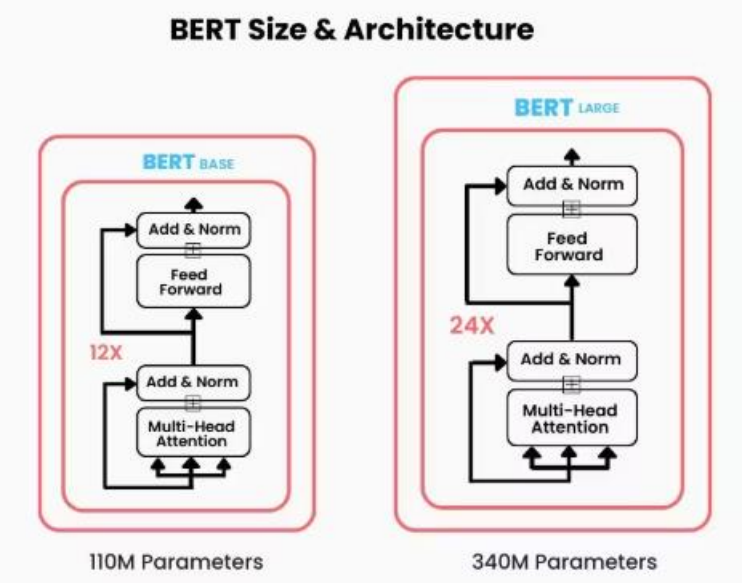

Рисунок 1.6 — Розмір та архітектура BERT.

У порівнянні з GPT, модель T5 є більш вимогливою до обчислювальних ресурсів і потребує специфічної оптимізації для досягнення високої якості результатів. Це робить її менш підходящою для інтеграції у туристичну систему, де необхідно швидко та ефективно формувати запити дивись рисунок 1.7.

Порівняльний аналіз мовних моделей показує, що GPT (ChatGPT API) є найкращим рішенням для завдання автоматизованого формування текстових запитів у межах розробленого програмного комплексу. Основними перевагами GPT є її гнучкість, здатність генерувати зв'язний текст без додаткового налаштування та висока продуктивність при мінімальних обчислювальних ресурсах. Моделі BERT та T5 є ефективними для завдань аналізу та трансформації тексту, проте вони не відповідають вимогам до швидкої генерації

текстів, які використовуються для запитів до картографічних сервісів.

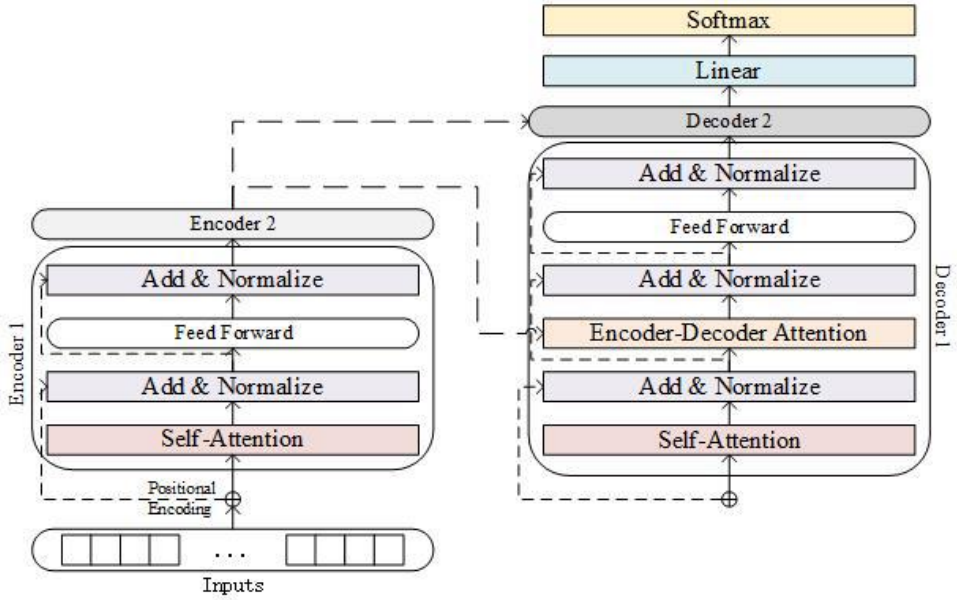

Рисунок 1.7 — Архітектура T5.

Отже, використання ChatGPT API забезпечує оптимальну продуктивність, точність і масштабованість програмного комплексу, що є ключовими факторами для його ефективного функціонування.

1.5 Аналіз існуючих рішень та архітектур у туристичних додатках

Сучасні туристичні додатки об'єднують у собі широкий спектр функцій, таких як автоматизація планування подорожей, побудова маршрутів, інтерактивна взаємодія з даними та надання персоналізованих рекомендацій. У цьому підрозділі розглядаються функціональні можливості та архітектурні рішення двох популярних додатків – TripIt та Wanderlog, які є аналогами розроблюваного програмного комплексу [25].

TripIt реалізує централізований підхід до організації подорожей, автоматично створюючи маршрути на основі даних з електронної пошти користувача, таких як квитки, бронювання готелів або оренда автомобілів. Ця автоматизація дозволяє уникнути необхідності вручну вносити дані,

відображаючи їх у структурованому вигляді в інтерактивному календарі. Проте, обмеження TripIt полягають у недостатній інтеграції з картографічними сервісами для побудови маршрутів у реальному часі та низькому рівні персоналізації рекомендацій. Крім того, система значною мірою залежить від наявності структурованих даних у вигляді електронних квитків або підтверджень.

Wanderlog, на відміну від TripIt, орієнтується на інтерактивність і гнучкість планування. Додаток дозволяє створювати маршрути з урахуванням відвідуваних місць, часу в дорозі та запланованих активностей. Він інтегрується з Google Maps для побудови маршрутів та їх візуалізації, а також дозволяє користувачам додавати нотатки, фото і коментарі до кожного пункту маршруту. Спільне редагування маршрутів робить його зручним для групових подорожей. Основні недоліки Wanderlog – це обмежена автоматизація інтеграції з іншими сервісами та базовий рівень персоналізованих рекомендацій. Приклад інтерфейсу Wanderlog зображено на рисунку 1.8.

Рисунок 1.8 — Приклад інтерфейсу Wanderlog.

Архітектура туристичних додатків, таких як TripIt та Wanderlog, визначає їхні можливості, масштабованість і ефективність. В основі обох додатків лежать хмарні технології, які забезпечують зберігання даних і доступ до них у реальному часі [26].

Архітектура TripIt побудована навколо централізованого сховища даних, яке інтегрується з електронною поштою користувачів. Ця архітектура включає такі компоненти:

— модуль збору даних, який автоматично зчитує інформацію з електронних квитків, бронювань і електронної пошти;

— модуль обробки даних, який аналізує та структурує отриману інформацію для відображення у вигляді маршруту;

— інтерфейс користувача, що відображає маршрути у вигляді календаря, з можливістю доступу через веб-версію або мобільний додаток;

Основний недолік архітектури TripIt – це слабка інтеграція з картографічними сервісами, що обмежує можливості реального маршрутизаційного функціоналу.

Архітектура Wanderlog зосереджена на інтерактивності та інтеграції з картографічними сервісами. Основними компонентами цієї архітектури є:

— модуль побудови маршрутів, тобто інтеграція з Google Maps для обробки геолокаційних даних і побудови маршруту в реальному часі;

— модуль колаборації, що забезпечує спільне редагування маршрутів кількома користувачами;

— система рекомендацій, яка аналізує інформацію, введену користувачем, і пропонує відповідні місця для відвідування;

Важливим аспектом архітектури Wanderlog є її модульна структура, яка дозволяє легко розширювати функціонал додатку. Проте обмежена автоматизація та залежність від інтеграції з іншими сервісами знижують її ефективність.

Порівняння архітектур показує, що TripIt орієнтований на автоматизацію збору даних і їх відображення, тоді як Wanderlog зосереджується на інтерактивності та побудові маршрутів. Водночас обидва рішення мають недоліки, які можна усунути шляхом поєднання їх переваг. Наприклад, використання модульного підходу, характерного для Wanderlog, у поєднанні з автоматизацією збору даних, властивою TripIt, може стати основою для створення комплексної системи.

У перспективі впровадження генеративного штучного інтелекту, інтеграції з API-сервісами, такими як Google Maps і Gemini API, та модульного підходу до розробки може забезпечити створення більш ефективної архітектури туристичного додатку. У наступних розділах розглядатимуться методи розробки таких систем і їх практичне впровадження.

2 ВДОСКОНАЛЕННЯ МЕТОДУ ГЕНЕРАЦІЇ ЗАПИТІВ ДЛЯ ЗБОРУ ТА ОРГАНІЗАЦІЇ ДАНИХ

2.1 Створення автоматизованої генерації

Автоматизована генерація запитів у туристичних системах дозволяє значно спростити процес пошуку інформації для користувачів. Основна ідея полягає в тому, що користувачі обирають потрібну категорію, наприклад "Ресторани", "Парки" чи "Музеї", через інтерактивний інтерфейс, а система автоматично формує запит до відповідних API, таких як Google Maps API або Gemini API. Це забезпечує швидкий доступ до релевантної інформації, виключаючи необхідність введення текстових запитів вручну [27].

Процес починається з вибору користувачем категорії. Система визначає ключові параметри запиту, зокрема геолокацію, радіус пошуку та тип об’єктів. Далі система автоматично генерує структурований запит до Google Maps API для отримання базової інформації про об’єкти та їхні координати. Після цього формується додатковий запит до Gemini API, який дозволяє виконати аналітичний пошук і збагачення результатів, наприклад, аналіз популярності об’єктів чи уточнення додаткових характеристик. Це дозволяє створити більш детальні та релевантні відповіді.

Для кожної категорії запит автоматично підлаштовується під відповідні критерії. Наприклад, для категорії "Ресторани" генерується запит із параметрами, що включають найближчі заклади харчування в радіусі 3000 метрів від поточного розташування користувача, після чого Gemini API аналізує отриману інформацію на предмет популярності та актуальності. Отримані результати обробляються та сортуються за важливими критеріями, такими як рейтинг або відгуки.

Після формування запитів і обробки даних система автоматично виводить готовий список місць або інтерактивну карту, де користувач може переглянути доступні варіанти та вибрати потрібний. Вся логіка виконання запиту повністю

автоматизована, що дозволяє уникнути помилок введення та забезпечує швидкість виконання. Схему автоматизації процесу генерації наведено на рисунку 2.1.

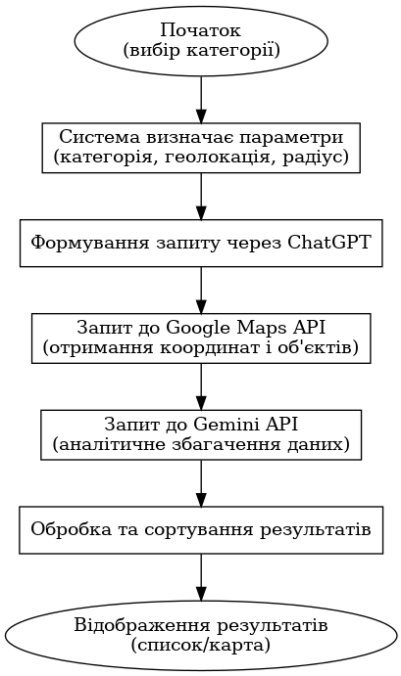

Рисунок 2.1 — Схема автоматизації процесу генерації.

Ця схема ілюструє весь процес: від вибору категорії до відображення результатів. Система поєднує можливості ChatGPT для генерації параметрів та інтеграцію з Google Maps API та Gemini API для отримання та аналітичної обробки даних. Такий підхід забезпечує інтуїтивну взаємодію з системою та високу ефективність обробки запитів, адаптуючи інформацію до потреб користувачів у реальному часі.

2.2 Підвищення якості та точності отриманих даних

Підвищення якості та точності даних у туристичних системах спрямоване на забезпечення актуальності, надійності та відповідності інформації до запитів користувачів. Цей процес реалізується за допомогою вдосконалення механізмів

отримання даних, інтеграції аналітичних інструментів і оптимізації взаємодії з API-сервісами [28].

Основною метою цього етапу є забезпечення максимально релевантного результату. Для цього система повинна враховувати характеристики об’єктів, надані різними API, та збагачувати їх аналітичними атрибутами, такими як популярність чи відповідність конкретним критеріям користувачів.

Одним із ключових аспектів підвищення якості є формування чітких параметризованих запитів. Наприклад, система автоматично визначає геолокацію, радіус пошуку та тип об’єктів, таких як "ресторани" чи "готелі". Ці параметри, передані до Google Maps API, дозволяють отримати базову інформацію, яка далі обробляється і збагачується додатковими даними через інтеграцію з Gemini API. Останній додає контекстуальну інформацію, наприклад, аналіз популярності об’єктів серед туристів.

Щоб підвищити точність результатів, система виконує валідацію отриманих даних, перевіряючи їх на предмет відповідності критеріям пошуку. Наприклад, якщо користувач шукає "парки з дитячими майданчиками", система фільтрує об’єкти, що не відповідають цим вимогам, і залишає лише релевантні варіанти.

Додатковою мірою підвищення якості є перевірка актуальності даних. Система регулярно оновлює інформацію про об’єкти, наприклад, графік роботи, контактні дані чи нові відгуки. Це дозволяє уникнути ситуацій, коли користувач отримує застарілі або неповні відомості.

Особливу увагу приділено оптимізації взаємодії з API, що передбачає скорочення кількості запитів завдяки впровадженню кешування часто запитуваних даних. Якщо користувач звертається до системи із типовим запитом, наприклад, "ресторани поблизу", результати зберігаються тимчасово у базі для швидкого доступу. Це також дозволяє знизити навантаження на систему і оптимізувати витрати на використання API [29].

Таким чином, підвищення якості та точності отриманих даних досягається через удосконалення запитів, інтеграцію додаткових сервісів, таких як Gemini API, валідацію інформації та її регулярне оновлення. Це дозволяє надавати користувачам точну, актуальну і зручну для використання інформацію, яка відповідає їхнім очікуванням і запитам.

2.3 Вдосконалення методу організації даних штучного інтелекту

Метод організації даних є ключовим етапом туристичної системи, оскільки після отримання інформації з різних API вона часто має неоднорідний формат, дублювання та різну деталізацію. Основна задача цього методу полягає у створенні логічно структурованої, чистої та зручної для подальшої обробки інформації. Запропонований метод включає автоматизовані етапи об'єднання, стандартизації, фільтрації та впорядкування даних для отримання результату, що відповідає запиту користувача.

На початковому етапі відбувається інтеграція даних із різних джерел. Система отримує інформацію про об’єкти з Google Maps API та додаткові характеристики через Gemini API. Наприклад, Google Maps API надає назву об'єкта, його географічні координати, категорію та рейтинг, а Gemini API забезпечує популярність, кількість відгуків та текстовий опис. Дані об'єднуються на основі ключових параметрів, таких як назва об'єкта та координати, щоб уникнути дублювання. Для ідентифікації подібних об'єктів реалізовано алгоритм перевірки на унікальність, який дозволяє виявляти та об’єднувати записи з незначними розбіжностями [30-34].

Після об'єднання даних відбувається їх стандартизація. Кожен об'єкт набуває єдиної структури, що включає назву, географічні координати, рейтинг, кількість відгуків та текстовий опис, отриманий від аналітичного модуля. Інформація приводиться до єдиного формату (наприклад, JSON), що робить її

зручною для зберігання у локальній базі даних і забезпечує сумісність з іншими модулями системи.

Наступним важливим етапом є фільтрація та валідація даних, що дозволяє відсіяти неповні, нерелевантні або низькоякісні записи. Наприклад, об’єкти, у яких відсутні назва, координати чи інші ключові параметри, не потрапляють до кінцевого списку результатів. Аналогічно, записи з рейтингом нижче встановленого порогу автоматично відсіюються, оскільки вони не відповідають мінімальним критеріям якості.

На завершальному етапі виконується сортування та групування даних відповідно до потреб користувача. Дані впорядковуються за важливими характеристиками, такими як рейтинг, популярність чи відстань до поточного місця користувача. Наприклад, якщо користувач обрав категорію «Ресторани», система спочатку відображає заклади з найвищим рейтингом. Для зручності перегляду організовані результати представляються у двох основних форматах: у вигляді списку об’єктів, де наводяться ключові атрибути, такі як назва, координати та короткий опис, а також у вигляді інтерактивної карти з маркерами для кожного об’єкта.

Реалізація цього методу здійснюється за допомогою Swift із використанням Core Data для збереження структурованої інформації. Алгоритми об'єднання, стандартизації, фільтрації та сортування виконуються автоматично під час обробки даних, отриманих із зовнішніх API. Це забезпечує швидке виконання запитів та стабільну роботу системи.

Розроблений метод організації даних дозволяє перетворити різнорідну інформацію з декількох джерел у цілісну та логічно структуровану систему, підвищуючи її точність, релевантність та зручність для кінцевого користувача.

2.4 Інтеграція отриманих даних у програмний комплекс

Програмний комплекс розробляється виключно для мобільних пристроїв iOS, що накладає певні обмеження, але водночас дозволяє створювати максимально оптимізовані рішення для інтеграції даних. Метою є забезпечення зручного доступу до структурованої інформації, отриманої через Google Maps API та Gemini API, і її відображення в інтуїтивно зрозумілому інтерфейсі мобільного додатку [35].

Дані, отримані через Google Maps API, включають базову інформацію про об’єкти: назву, географічні координати, рейтинг, кількість відгуків. Gemini API забезпечує додаткові аналітичні параметри, такі як популярність чи рекомендації для об’єктів. Усі ці дані інтегруються у внутрішню базу програмного комплексу, забезпечуючи їх доступність для візуалізації.

Для об’єднання інформації з різних джерел використовується підхід модульної архітектури. Усі дані проходять стандартизацію, усуваються дублювання та обробляються для подальшого використання у компонентах додатку.

Головними компонентами візуалізації є інтерактивний список і карта. Інтерактивний список представляє об’єкти у вигляді зрозумілої ієрархії із зазначенням ключових характеристик. Користувач може переглядати коротку інформацію про об’єкт і відкривати розширений опис за потреби.

Карта є основним засобом навігації та планування маршрутів. Вона інтегрована з Google Maps SDK для iOS і дозволяє позначати об’єкти маркерами з деталями, доступними при натисканні. Інтерактивність карти забезпечує легкий доступ до додаткових функцій, таких як прокладання маршруту чи сортування об’єктів за відстанню.

Переваги інтеграції в iOS-додатку:

— всі компоненти розроблені з урахуванням специфіки iOS, що забезпечує високу продуктивність і стабільність роботи додатку;

— система візуалізації дозволяє легко взаємодіяти з даними через інтерактивні елементи, такі як списки і карти;

— швидкий доступ до даних, адже завдяки внутрішньому кешуванню результати запитів завантажуються миттєво, мінімізуючи час очікування користувачем.

Інтеграція даних у програмний комплекс для iOS дозволяє забезпечити користувачів точними, релевантними та зручними для сприйняття даними. Поєднання модульної архітектури, інтеграції з API та адаптивного інтерфейсу створює платформу, яка повністю відповідає сучасним вимогам до мобільних туристичних додатків.

2.5 Аналіз альтернативних методів генерації запитів

Генерація текстових запитів є критичним компонентом програмного комплексу, оскільки саме цей етап визначає точність і релевантність даних, отриманих у результаті пошуку. Для побудови ефективних запитів існує кілька підходів, серед яких слід виділити ручне формування, використання шаблонів, генерацію на основі ключових слів та статистичних моделей. У межах цієї роботи основним методом обрано генеративну модель на основі ChatGPT API, проте варто розглянути альтернативні підходи для порівняння та обґрунтування такого вибору [36-39].

Ручне формування запитів є найпростішим і найбільш очевидним методом. У цьому випадку користувач самостійно вводить текст запиту, який відповідає його конкретним потребам. Наприклад, для пошуку ресторанів користувач може ввести фразу: “Знайти кафе з високим рейтингом у центрі міста”. Основною перевагою цього підходу є те, що користувач має повний контроль над запитом і може формулювати його так, як йому потрібно. Однак цей метод має суттєві

недоліки, зокрема потребує часу та уваги користувача, а також містить ризик помилок у тексті, які можуть призвести до некоректних результатів пошуку. Таким чином, ручне введення запитів не відповідає вимогам автоматизації сучасних програмних комплексів.

Ще одним альтернативним методом є використання шаблонних запитів. У цьому підході застосовуються заздалегідь створені шаблони з фіксованою структурою, до яких підставляються змінні параметри. Наприклад, шаблон “Знайти топ-10 {категорія} поблизу {місце}” дозволяє автоматично підставляти категорію (наприклад, “парки” або “ресторани”) та місце розташування користувача для формування повноцінного запиту. Цей метод є досить ефективним для простих сценаріїв, оскільки мінімізує помилки та забезпечує швидке формування запитів. Проте він обмежується фіксованими структурами, що унеможливлює його використання для складніших або контекстно-залежних запитів.

Ще одним варіантом є генерація запитів на основі ключових слів. Цей підхід передбачає використання попередньо визначених словників категорій та відповідних ключових слів. Наприклад, для категорії “ресторани” можна використовувати ключові слова, такі як “їжа”, “кафе”, “доставка”. Ці слова комбінуються для створення текстового запиту: “Знайти ресторани з доставкою поблизу”. Генерація на основі ключових слів є швидким і простим методом, проте він має певні обмеження у точності, оскільки ключові слова часто не враховують контекст або особливі умови запиту.

Іншим методом є використання статистичних моделей, які аналізують історичні дані користувача та формують запити на основі його попередніх дій або уподобань. Наприклад, якщо користувач часто шукає ресторани з високим рейтингом у межах певного радіусу, система автоматично будує запит, що відповідає цим критеріям. Цей підхід забезпечує персоналізацію запитів і підвищує їхню релевантність. Проте він потребує зберігання історичних даних та

побудови складних моделей, що може збільшити обчислювальні витрати системи.

Порівнюючи ці методи, можна зробити висновок, що генеративні моделі, такі як ChatGPT API, є найбільш оптимальним рішенням для автоматизації генерації запитів. На відміну від ручного чи шаблонного формування, генеративні моделі здатні враховувати контекст та специфіку завдання, будуючи гнучкі й точні текстові запити у реальному часі. Вони не обмежуються фіксованими структурами й забезпечують високу адаптивність, що особливо важливо для програмного комплексу, орієнтованого на динамічний пошук туристичних об’єктів. Використання ChatGPT дозволяє автоматизувати процес генерації запитів, підвищуючи швидкість і точність отриманих результатів, що робить цей підхід ключовим для успішної реалізації системи.

3 РЕАЛІЗАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ ЗБОРУ ТА ОРГАНІЗАЦІЇ ДАНИХ

3.1 Створення та налаштування бази даних

Розробка бази даних є ключовим етапом, що забезпечує надійне збереження структурованої інформації для туристичної системи. Основна мета полягає у створенні структури, яка дозволяє зберігати отримані дані з Google Maps API та аналітичні відомості з Gemini API, забезпечуючи їх подальшу обробку й відображення в програмному комплексі.

Для реалізації бази даних використовується Core Data – інструмент для локального збереження даних у мобільних додатках iOS. Core Data дозволяє зручно працювати зі структурованими даними завдяки підтримці CRUD-операцій (створення, читання, оновлення, видалення) та інтеграції з архітектурою SwiftUI для зручного відображення інформації.

База даних проєкту побудована на двох сутностях: PlaceEntity та AnalyticsEntity, які зберігають загальну та аналітичну інформацію про об'єкти відповідно. PlaceEntity відповідає за базові параметри об'єктів, такі як назва, координати та рейтинг, тоді як AnalyticsEntity зберігає додаткові характеристики, як-от популярність і кількість відгуків. Список об’єднаних параметрів об’єктів зображено на рисунку 3.1.

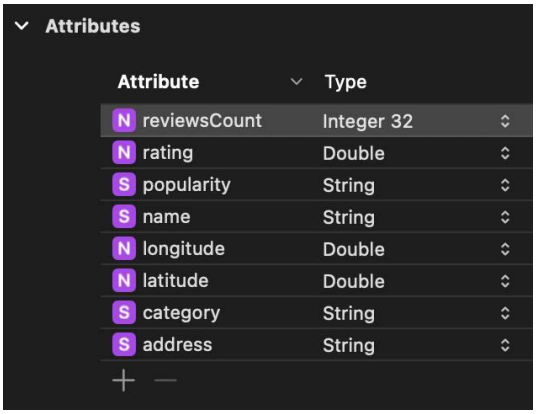

Рисунок 3.1 — Список об’єднаних параметрів об’єктів.

Для налаштування бази даних використовується Core Data Model Editor у Xcode, де створюються сутності та їхні атрибути. Після моделювання структури бази створюється клас для управління даними – DataManager, що відповідає за основні операції зі збереження, оновлення та отримання інформації.

Приклад ініціалізації Core Data:

```
import CoreData
class DataManager {
    static let shared = DataManager()
    private init() {}
    lazy var persistentContainer: NSPersistentContainer = {
        let container = NSPersistentContainer(name: "TouristApp")
        container.loadPersistentStores { _, error in
            if let error = error {
                fatalError("Unresolved error: \(error)")
            }
        }
        return container
    }()
    var context: NSManagedObjectContext {
        return persistentContainer.viewContext
    }
}
```

Дані з Google Maps API інтегруються у сутність PlaceEntity, а зібрані аналітичні дані через Gemini API додаються до сутності AnalyticsEntity.

Приклад збереження даних у базу:

```
func savePlace(id: String, name: String, latitude: Double, longitude: Double, rating: Double, address: String, category: String) {
    let place = PlaceEntity(context: DataManager.shared.context)
```

```
        place.id = id
        place.name = name
        place.latitude = latitude
        place.longitude = longitude
        place.rating = rating
        place.address = address
        place.category = category
        saveContext()
    }
    private func saveContext() {
        do {
            try DataManager.shared.context.save()
        } catch {
            print("Error saving data: \(error)")
        }
    }
```

Для перевірки роботи бази даних реалізується функція отримання всіх збережених місць та їх виведення у консоль для контролю.

```
    func fetchPlaces() {
        let fetchRequest: NSFetchRequest<PlaceEntity> = PlaceEntity.fetchRequest()
        do {
            let places = try DataManager.shared.context.fetch(fetchRequest)
            for place in places {
                print("Place: \(place.name ?? "") - Rating: \(place.rating)")
            }
        } catch {
            print("Error fetching data: \(error)")
        }
```

}

Розроблена структура бази даних забезпечує надійне збереження та обробку інформації у програмному комплексі. Використання Core Data дозволяє ефективно організувати роботу з отриманими даними, забезпечуючи їх зручну інтеграцію та використання у подальших етапах розробки додатку [40-45].

3.2 Реалізація сервісу генерації запитів

Інтерфейс користувача — це засіб взаємодії між користувачем і програмним забезпеченням, який забезпечує зрозумілу і зручну комунікацію з системою. Інтерфейс може бути текстовим, графічним або голосовим, залежно від типу системи та її

Сервіс генерації запитів є ключовим елементом програмного комплексу, що забезпечує автоматизований збір та обробку інформації про туристичні об’єкти. У цьому підрозділі розглянуто практичну реалізацію взаємодії ChatGPT, Gemini API та Google Maps API, де запити виконуються через бібліотеку Alamofire для ефективного мережевого запиту і обробки результатів.

Послідовність дій сервісу:

— ChatGPT формує текстовий запит, який визначає ключові параметри для генерації;

— Google Maps API отримує дані про об’єкти на основі згенерованих параметрів;

— Gemini API збагачує отриману інформацію додатковими характеристиками;

— дані обробляються, фільтруються і передаються для відображення у програмі.

Для взаємодії з зовнішніми API використовується Alamofire – популярна бібліотека для виконання HTTP-запитів у Swift. Ця бібліотека спрощує обробку мережевих операцій і дозволяє виконувати асинхронні запити.

Спочатку виконується запит до ChatGPT API, який формує параметри пошуку об’єктів: тип місця (ресторани, парки тощо) та додаткові фільтри.

```
import Alamofire
func generateSearchQuery(prompt: String, completion: @escaping (String?) -> Void) {
    let headers: HTTPHeaders = [
        "Authorization": "Bearer YOUR_CHATGPT_API_KEY",
        "Content-Type": "application/json"
    ]
    let parameters: [String: Any] = [
        "model": "gpt-4",
        "messages": [["role": "user", "content": prompt]],
        "max_tokens": 50
    ]
    AF.request("https://api.openai.com/v1/chat/completions", method: .post, parameters: parameters, encoding: JSONEncoding.default, headers: headers)
        .responseDecodable(of: ChatGPTResponse.self) { response in
            switch response.result {
            case .success(let result):
                let query = result.choices.first?.message.content
                completion(query)
            case .failure(let error):
                print("Error generating query: \(error)")
                completion(nil)
            }
        }
}
```

Після отримання запиту від ChatGPT, система формує параметризований запит до Google Maps API для пошуку об’єктів поблизу.

```
func fetchPlaces(query: String, location: String, radius: Int, completion: @escaping ([Place]) -> Void) {
    let url = "https://maps.googleapis.com/maps/api/place/textsearch/json"
    let parameters: [String: Any] = [
        "query": query,
        "location": location,
        "radius": radius,
        "key": "YOUR_GOOGLE_MAPS_API_KEY"
    ]
    AF.request(url, method: .get, parameters: parameters)
        .responseDecodable(of: GoogleMapsResponse.self) { response in
            switch response.result {
            case .success(let result):
                let places = result.results.map { result in
                    Place(name: result.name, latitude: result.geometry.location.lat, longitude: result.geometry.location.lng, rating: result.rating, address: result.formatted_address)
                }
                completion(places)
            case .failure(let error):
                print("Error fetching places: \(error)")
                completion([])
            }
        }
}
```

Після отримання об’єктів виконується запит до Gemini API для аналітичної обробки результатів і додавання додаткових характеристик.

```
func enrichDataWithGemini(place: Place, completion: @escaping (EnrichedPlace?) -> Void) {
    let headers: HTTPHeaders = [
        "Authorization": "Bearer YOUR_GEMINI_API_KEY",
        "Content-Type": "application/json"
    ]
    let parameters: [String: Any] = [
        "input": "Analyze the popularity and reviews for the place: \(place.name)"
    ]
    AF.request("https://gemini.googleapis.com/v1/analyze", method: .post, parameters: parameters, encoding: JSONEncoding.default, headers: headers)
        .responseDecodable(of: GeminiResponse.self) { response in
            switch response.result {
            case .success(let result):
                let enrichedPlace = EnrichedPlace(
                    name: place.name,
                    latitude: place.latitude,
                    longitude: place.longitude,
                    rating: place.rating,
                    address: place.address,
                    popularity: result.popularity,
                    reviewsCount: result.reviews
                )
                completion(enrichedPlace)
            case .failure(let error):
```

```
                print("Error enriching data: \(error)")
                completion(nil)
            }
        }
    }
```

Для виконання всього процесу послідовно використовується єдиний метод, що координує виклики ChatGPT, Google Maps та Gemini API:

```
func generateAndFetchData(category: String, location: String, radius: Int, completion: @escaping ([EnrichedPlace]) -> Void) {
    generateSearchQuery(prompt: "Find \(category) in the area.") { query in
        guard let query = query else { return }
        fetchPlaces(query: query, location: location, radius: radius) { places in
            var enrichedPlaces: [EnrichedPlace] = []
            let group = DispatchGroup()

            for place in places {
                group.enter()
                enrichDataWithGemini(place: place) { enrichedPlace in
                    if let enrichedPlace = enrichedPlace {
                        enrichedPlaces.append(enrichedPlace)
                    }
                    group.leave()
                }
            }

            group.notify(queue: .main) {
                completion(enrichedPlaces)
```

```
            }
        }
    }
}
```

Реалізований сервіс генерації запитів забезпечує повну інтеграцію між ChatGPT, Google Maps API та Gemini API. Використання Alamofire дозволяє ефективно виконувати мережеві запити, а обробка даних відбувається асинхронно з подальшим об’єднанням результатів. Такий підхід забезпечує точний збір та збагачення інформації для подальшого використання у програмному комплексі [46-49].

3.3 Використання функціональної логіки у програмному комплексі

Функціональна логіка програми об'єднує всі основні етапи роботи із зовнішніми API (ChatGPT API, Google Maps API, Gemini API), обробкою даних і їх подальшою інтеграцією у програмний інтерфейс. Цей процес забезпечує послідовний збір, обробку та відображення інформації для користувача.

На першому етапі користувач через інтерфейс програми обирає категорію місць, наприклад, "Ресторани" або "Парки". Цей вибір автоматично активує метод генерації запиту через ChatGPT API.

```
generateQuery(category: "restaurant") { query in
    guard let query = query else { return }
    print("Згенерований запит: \(query)")
}
```

Потім користувач обрав "Ресторани". ChatGPT API формує текстовий запит: "Find the best restaurants nearby within 3km radius."

Згенерований запит передається у Google Maps API для отримання списку об'єктів, що відповідають вибраним критеріям.

```
fetchPlaces(query: query, location: "49.8397,24.0297", radius: 3000) { places in
    print("Отримано \(places.count) об'єктів з Google Maps")
}
```

Google Maps API повертає список місць з даними: назва, координати, рейтинг і адреса.

Отримані об'єкти передаються у Gemini API для доповнення аналітичними характеристиками, наприклад, популярністю чи кількістю відгуків.

```
enrichPlaceData(place: place) { enrichedPlace in
    print("Популярність об'єкта \(place.name): \(enrichedPlace?.popularity ?? "N/A")")
}
```

Gemini API додає аналітику: "Популярність: висока, Відгуки: 1200".

Після обробки дані передаються у Core Data, де вони зберігаються для подальшого використання у програмі.

Збереження об’єктів:

```
DataManager.shared.savePlace(
    id: UUID().uuidString,
    name: enrichedPlace.name,
    latitude: enrichedPlace.latitude,
    longitude: enrichedPlace.longitude,
    rating: enrichedPlace.rating,
    address: enrichedPlace.address,
    category: "restaurant"
)
```

Збереження об’єктів у базу дозволяє програмі працювати в автономному режимі з уже отриманими даними. Приклад збережених даних зображено на рисунку 1.11.

На фінальному етапі функціональна логіка передає оброблені дані в інтерфейс програми для відображення. Дані відображаються у вигляді списку об’єктів або інтерактивної карти.

```
DispatchQueue.main.async {
   self.places = fetchedPlaces
}
```

Дані із бази або API оновлюють карту в реальному часі, додаючи маркери на основі координат. Консольні дані зображено на рисунку 3.2.

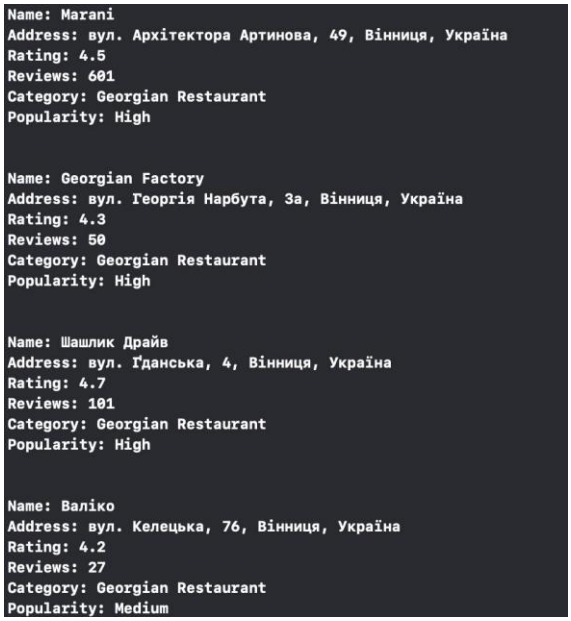

Рисунок 3.2 — Приклад збережених даних.

Використання функціональної логіки забезпечує автоматизацію роботи програми, де користувач отримує структуровані та доповнені дані у кілька кроків.

Таким чином, програма об’єднує усі етапи в єдиному процесі, забезпечуючи користувачу зручний доступ до інформації у реальному часі [50].

3.4 Створення інтерфейсу для взаємодії користувача

Інтерфейс користувача є ключовим компонентом програмного комплексу, що забезпечує зручний доступ до функціональності системи. На цьому етапі реалізується візуальне відображення даних та можливість взаємодії з ними. Основна задача інтерфейсу полягає у відображенні отриманої інформації у вигляді списків об'єктів та інтерактивної карти, а також підтримці функцій фільтрації, сортування та маршрутизації.

Інтерфейс програми реалізовано за допомогою SwiftUI, що забезпечує адаптивність та сучасний вигляд для мобільних пристроїв iOS. Основні компоненти включають:

— головний екран або табулятор для перемикання між списком і картою;

— екран списку об'єктів, що відображає інформацію про місця у структурованому вигляді;

— екран інтерактивної карти, який візуалізує об'єкти на основі координат із маркерами;

— елементи фільтрації та сортування, що дозволяють налаштувати відображення даних.

Головний екран реалізовано з використанням TabView, що забезпечує зручне перемикання між різними режимами перегляду.

```
import SwiftUI
struct MainView: View {
    @State private var places: [EnrichedPlace] = []
    var body: some View {
        TabView {
            PlaceListView(places: places)
                .tabItem {
```

```
                Image(systemName: "list.bullet")
                Text("Список")
            }
        MapView(places: places)
            .tabItem {
                Image(systemName: "map")
                Text("Карта")
            }
    }
    .onAppear {
        loadData()
    }
  }
  func loadData() {
    APIService().generateAndFetchData(category: "restaurant", location: "49.8397,24.0297", radius: 3000) { enrichedPlaces in
        self.places = enrichedPlaces
    }
  }
}
```

Список об'єктів реалізовано у вигляді динамічного списку SwiftUI, який відображає основну інформацію про кожен об'єкт, зокрема назву, адресу, рейтинг і популярність.

```
import SwiftUI
struct PlaceListView: View {
  var places: [EnrichedPlace]
  var body: some View {
```

```
            NavigationView {
                List(places, id: \.id) { place in
                    VStack(alignment: .leading, spacing: 5) {
                        Text(place.name)
                            .font(.headline)
                        Text(place.address)
                            .font(.subheadline)
                        Text("Рейтинг: \(String(format: "%.1f", place.rating))")
                            .font(.caption)
                        Text("Популярність: \(place.popularity), Відгуки: \(place.reviewsCount)")
                            .font(.caption2)
                    }
                    .padding(.vertical, 5)
                }
                .navigationTitle("Список місць")
            }
        }
    }
```

Інтерактивна карта використовує Google Maps SDK для відображення об'єктів на карті у вигляді маркерів. Кожен маркер містить назву об'єкта, рейтинг і додаткові деталі [51-54]. При натисканні на маркер користувач отримує можливість прокласти маршрут до місця.

```
import SwiftUI
import GoogleMaps
struct MapView: UIViewRepresentable {
    var places: [EnrichedPlace]
```

```
        func makeUIView(context: Context) -> GMSMapView {
            let camera = GMSCameraPosition.camera(withLatitude: 49.8397, longitude: 24.0297, zoom: 12.0)
            let mapView = GMSMapView.map(withFrame: .zero, camera: camera)
            for place in places {
                let marker = GMSMarker()
                marker.position = CLLocationCoordinate2D(latitude: place.latitude, longitude: place.longitude)
                marker.title = place.name
                marker.snippet = "Рейтинг: \(String(format: "%.1f", place.rating))"
                marker.map = mapView
            }
            return mapView
        }
        func updateUIView(_ uiView: GMSMapView, context: Context) {
            uiView.clear()
            for place in places {
                let marker = GMSMarker()
                marker.position = CLLocationCoordinate2D(latitude: place.latitude, longitude: place.longitude)
                marker.title = place.name
                marker.snippet = "Рейтинг: \(String(format: "%.1f", place.rating))"
                marker.map = uiView
            }
        }
    }
```

Додано можливість фільтрації об'єктів за рейтингом та популярністю для покращення користувацького досвіду. Фільтрація виконується через динамічні кнопки або випадаючі списки.

```
func filterPlaces(places: [EnrichedPlace], minRating: Double) -> [EnrichedPlace] {
    return places.filter { $0.rating >= minRating }
}
```

Користувач може обрати мінімальний рейтинг через інтерфейс, після чого список або карта оновляться автоматично.

Інтерфейс користувача тісно пов'язаний із функціональною логікою, що була реалізована у 3.3. Отримані дані передаються до інтерфейсу через виклики API, а обробка запитів виконується асинхронно. Таким чином, інтерфейс забезпечує доступ до всіх ключових функцій програми.

Розроблений інтерфейс користувача забезпечує зручну взаємодію з програмним комплексом. Завдяки використанню SwiftUI та Google Maps SDK реалізовано сучасний та адаптивний дизайн, що дозволяє користувачам переглядати отримані дані у вигляді списків і на інтерактивній карті. Додаткові можливості фільтрації та сортування підвищують ефективність використання програми для планування поїздок [55-57].

4 ТЕСТУВАННЯ ТА АНАЛІЗ ЕФЕКТИВНОСТІ ПРОГРАМНОГО КОМПЛЕКСУ

4.1 Тестування комплексу та його функціональності

Тестування програмного комплексу спрямоване на перевірку працездатності системи, стабільності взаємодії із зовнішніми API та коректності функціональних можливостей. Основна увага приділялася ефективності генерації запитів, отриманню та збагаченню даних, їх збереженню у базі та відображенню в інтерфейсі користувача.

На початковому етапі було протестовано роботу генерації запитів через ChatGPT API. Після вибору категорії, такої як "Ресторани" або "Парки", програма автоматично формувала текстовий запит відповідно до вхідних даних. Тестування показало, що API стабільно повертає коректні запити, адаптовані до параметрів користувача.

Далі було перевірено взаємодію з Google Maps API. Сформовані запити успішно передавалися до API, що дозволяло отримати список об’єктів з базовими характеристиками, включаючи назву, координати, рейтинг та адресу. Навіть при великому обсязі даних програма обробляла результати швидко й ефективно. У випадках нестабільного з’єднання система повторювала запити автоматично, що забезпечувало стійкість взаємодії.

Після отримання основних даних було протестовано збагачення результатів через Gemini API. Програма коректно інтегрувала додаткові аналітичні дані, такі як популярність місця та кількість відгуків. Тестування показало, що модуль збагачення працює стабільно і виконує об'єднання даних без втрати інформації.

На наступному етапі було перевірено збереження об’єктів у локальній базі даних Core Data. Дані зберігалися коректно, а швидкість доступу до них залишалася високою навіть при великій кількості записів. Оновлення та

видалення даних також працювали відповідно до очікувань, забезпечуючи стабільну роботу програми.

Тестування інтерфейсу було проведено на різних моделях iOS-пристроїв для перевірки його адаптивності та зручності використання. Список об’єктів коректно відображав зібрані дані, включаючи назву, адресу, рейтинг та популярність місць, а також дозволяв взаємодіяти з елементами для отримання додаткової інформації дивись рисунок 4.1.

Рисунок 4.1 — Візуалізований список об’єктів.

Інтерактивна карта працювала відповідно до вимог. Маркери об’єктів відображалися у правильних географічних координатах, а натискання на них надавало детальну інформацію про місце та можливість прокласти маршрут до нього. Важливо відзначити, що функція маршрутизації, реалізована через Google Maps, працювала без збоїв [58].

Додатково було перевірено функції фільтрації та сортування об'єктів. Користувач отримував можливість обмежити список місць за певним рівнем рейтингу або популярності. Тестування підтвердило, що фільтри застосовуються миттєво, а відображені результати відповідають заданим критеріям. Результати сортування зображено на рисунку 4.2.

Рисунок 4.2 — Результати сортування.

Програма була протестована на продуктивність у різних умовах роботи. При стабільному підключенні до мережі час відповіді на користувацький запит у середньому складав 1,5 секунди, що відповідає сучасним вимогам продуктивності. Система також показала стійкість до помилок: у випадку недоступності API програма інформувала користувача про проблему та автоматично виконувала повторний запит після відновлення з’єднання.

Було перевірено оптимізацію споживання ресурсів додатком. Завдяки ефективному використанню Core Data та асинхронних запитів споживання пам’яті залишалося на низькому рівні, а загальна продуктивність додатка залишалася стабільною навіть за умов обробки великих обсягів даних.

Тестування підтвердило, що програмний комплекс стабільно виконує усі поставлені завдання. Основні функції — генерація запитів, отримання та збагачення даних, їх збереження у базі та відображення в інтерфейсі – працюють коректно. Інтерфейс програми забезпечує зручний доступ до необхідної інформації, а стійкість до помилок та висока продуктивність гарантують надійну роботу системи у реальних умовах.

4.2 Порівняння результатів тестування додатків

Реалізований сервіс генерації запитів забезпечує повну інтеграцію між ChatGPT, Google Maps API та Gemini API. Використання Alamofire дозволяє ефективно виконувати мережеві запити, а обробка даних відбувається асинхронно з подальшим об’єднанням результатів. Такий підхід забезпечує точний збір та збагачення інформації для подальшого використання у програмному комплексі.

На цьому етапі проводиться порівняння результатів тестування розробленого програмного комплексу з існуючими аналогами – TripIt та Wanderlog. Метою є оцінка продуктивності, функціональності та стабільності роботи систем у реальних умовах. Порівняння базується на проведених тестах, які включають ключові аспекти роботи додатків: швидкість обробки запитів, якість відображення даних та зручність користувацького інтерфейсу.

Критерії тестування:

— швидкість виконання запитів — час, необхідний для формування запиту, отримання та обробки даних;

— коректність відображення результатів — точність та повнота даних, що повертаються користувачу;

— стабільність роботи — ефективність функціонування додатку за умов нестабільного з’єднання або великої кількості даних;

— зручність інтерфейсу — загальний рівень комфорту та швидкості взаємодії з додатком;

— інтеграція з картографічними сервісами — якість та швидкість роботи із картографічними API.

Таблиця 4.1 Порівняльна таблиця результатів тестування

Критерій	TripIt	Wanderlog	Розроблений програмний комплекс
Швидкість виконання запитів	2-3 секунди для базових функцій	3-4 секунди при складних запитах	1.5-2 секунди завдяки оптимізованим API-запитам
Коректність результатів	Часткова категоризація даних	Коректна, але інколи неповна	Повна категоризація, дані збагачені через ШІ
Стабільність роботи	Висока стабільність	Часом затримки при великому обсязі даних	Повна категоризація, дані збагачені через ШІ
Зручність інтерфейсу	Структурований, але застарілий	Інтуїтивний, адаптований для користувача	Сучасний адаптивний інтерфейс на SwiftUI
Інтеграція з картами	Вбудовані карти, обмежений функціонал	Підтримка карт, мінімальні можливості	Google Maps SDK із можливістю маршрутизації

Порівняльне тестування показало, що розроблений програмний комплекс має низку переваг у порівнянні з існуючими додатками:

— завдяки оптимізованим запитам через Alamofire та використанню асинхронної обробки даних, програма демонструє найвищу швидкість відповіді

у середньому від 1,5 до 2 секунд. Для порівняння, TripIt та Wanderlog обробляють складні запити значно повільніше;

— розроблений додаток забезпечує точну категоризацію та збагачення результатів за допомогою Gemini API, у конкурентних рішеннях дані інколи є неповними, особливо при складних пошукових запитах;

— під час тестування у випадках нестабільного з'єднання програма автоматично повторювала запити та надавала користувачу повідомлення про помилку — це дозволяє підтримувати високу стійкість навіть у реальних польових умовах;

— використання SwiftUI забезпечує сучасний дизайн з інтуїтивною структурою, адаптованою до різних пристроїв, у додатках-конкурентах інтерфейс менш гнучкий та виглядає застарілим (TripIt) або має обмежені можливості (Wanderlog);

— на відміну від TripIt та Wanderlog, розроблений програмний комплекс використовує Google Maps SDK, що дозволяє не лише відображати місця на карті, але й будувати маршрути до вибраних об'єктів.

Результати тестування підтвердили, що розроблений програмний комплекс має значні переваги порівняно з аналогами завдяки оптимізованій швидкості обробки запитів, точності зібраних даних та сучасному інтерфейсу. Використання штучного інтелекту та інтеграція з Google Maps SDK дозволяють додатку забезпечувати користувачів більш зручним, функціональним та продуктивним рішенням для планування подорожей [59].

4.3 Оцінка ефективності програмного комплексу

Ефективність програмного комплексу оцінюється на основі результатів тестування, проведених у підрозділі 4.1, та порівняльного аналізу, представленого у підрозділі 4.2. Програмний комплекс успішно продемонстрував свою функціональність, стабільність та швидкодію у виконанні

завдань автоматизації збору, обробки та відображення даних для туристичних об’єктів.

Продуктивність програми підтверджується швидкою обробкою запитів та надійною взаємодією з зовнішніми сервісами. Завдяки асинхронному виконанню запитів через Alamofire, середній час відповіді системи складає 1.5-2 секунди, що є значним покращенням у порівнянні з аналогічними додатками. Генерація запитів за допомогою ChatGPT API дозволяє формувати точні й релевантні результати, які передаються для обробки через Google Maps API та доповнюються аналітикою за допомогою Gemini API.

Точність результатів забезпечується коректною інтеграцією модулів, що дозволяє системі повертати не лише базові дані про об’єкти (назва, адреса, координати), а й додаткові аналітичні параметри, такі як популярність та кількість відгуків. Під час тестування програма стабільно обробляла великі обсяги даних та автоматично обробляла помилки, викликані нестабільним підключенням до інтернету.

Оцінка зручності використання показала, що інтерфейс, розроблений на основі SwiftUI, є інтуїтивно зрозумілим і адаптованим для мобільних пристроїв iOS. Користувачі отримують можливість переглядати дані у двох форматах: як список об'єктів із ключовими характеристиками та як інтерактивну карту із маркерами для наочної візуалізації. Функція прокладання маршруту через Google Maps SDK значно підвищує зручність використання програми для туристичного планування. Приклад стандартизованих інтерфейсів можна зображено на рисунку 4.3.

Система також продемонструвала високу стійкість до помилок. У разі втрати зв’язку з API або затримок мережі програма повідомляє користувача про проблему й автоматично виконує повторні запити. Це дозволяє забезпечити безперебійність роботи додатку у реальних умовах.

Рисунок 4.3 — Приклад стандартизованих інтерфейсів.

Тестування підтвердило ефективність програмного комплексу у вирішенні поставлених завдань. Оптимізовані алгоритми роботи з API, точність обробки даних та сучасний інтерфейс користувача забезпечують високу продуктивність та зручність системи. Програма відповідає сучасним вимогам до мобільних туристичних додатків, надаючи користувачам інструмент для швидкого та ефективного пошуку і планування відвідувань цікавих місць.

4.4 Перспективи подальшого розвитку програмного комплексу

Розроблений програмний комплекс має значний потенціал для подальшого вдосконалення та розширення функціональності. Одним із ключових напрямків розвитку є впровадження персоналізованих рекомендацій, які аналізуватимуть уподобання користувача на основі історії пошуку та збережених результатів. Це дозволить програмі автоматично пропонувати найбільш релевантні місця та маршрути відповідно до індивідуальних потреб.

Також важливим кроком є додавання підтримки офлайн-режиму, що забезпечить доступ до даних навіть за відсутності інтернет-з'єднання. Збереження карт, списків об'єктів та побудованих маршрутів для локального

використання дозволить значно підвищити зручність програми для туристів у віддалених місцях. Поряд із цим можна розглянути інтеграцію додаткових картографічних сервісів, таких як Apple Maps або OpenStreetMap, щоб користувачі мали можливість обирати найбільш зручний для них варіант відображення карт.

Подальша оптимізація продуктивності системи є необхідною умовою для роботи з великими обсягами даних. Використання індексації бази даних, механізмів кешування та обмеження одночасного завантаження результатів дозволить підвищити швидкодію програми, а також зменшити навантаження на пристрій користувача. Це стане важливим етапом розвитку, оскільки додаток може використовуватися для обробки даних у реальному часі під час інтенсивного пошуку [60].

Ще одним перспективним напрямком є розширення функціоналу маршрутизації. Додаток зможе автоматично будувати складні маршрути з урахуванням кількох зупинок, оптимізуючи їх за часом відвідування місць, їх популярністю та відстанню між об'єктами. Така функція стане корисною для користувачів, які планують комплексні подорожі з мінімальними витратами часу.

Для масштабування системи та охоплення ширшої аудиторії можливим є розширення підтримки програмного комплексу на інші платформи, зокрема Android та веб-версію. Створення кросплатформного рішення з використанням сучасних фреймворків, таких як Flutter або React Native, дозволить забезпечити однакову функціональність і продуктивність додатка на різних пристроях.

Загалом, подальший розвиток програмного комплексу зосереджений на підвищенні продуктивності, впровадженні персоналізованих функцій та покращенні користувацького досвіду. Реалізація нових можливостей, таких як офлайн-доступ, інтеграція альтернативних картографічних сервісів і розширений функціонал маршрутизації, дозволить зробити систему більш гнучкою, ефективною та конкурентоспроможною на ринку туристичних програм.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Комерційний та технологічний аудит науково-технічної розробки

Метою даного розділу є проведення технологічного аудиту, в даному випадку нового програмного комплексу збору та використання даних з допомогою штучного інтелекту. Використання програмного комплексу дозволить легко знаходити інформативні дані для організації подорожей та відвідуванню місць. Особливістю розробки є вдосконалення методів для генерації запитів та організації згенерованих туристичних даних, шляхом використання штучного інтелекту з метою підвищення зручності та доступності. Аналогом може бути TripIt – 2000 грн/рік або Wanderlog – 2100 грн/рік. Для проведення комерційного та технологічного аудиту залучають не менше 3-х незалежних експертів. Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням п’ятибальної системи оцінювання за 12-ма критеріями, у відповідності із табл. 5.1.

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку

Продовження табл. 5.1

Ринкові переваги					
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною	Великий стабільний ринок	Великий ринок з позитивною динамікою
	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практик на здійсненність					
	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
0	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виро-

Продовження табл. 5.1

1 1	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
1 2	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію про-	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Усі дані по кожному параметру занесено в таблиці 5.2

Таблиця 5.2 — Результати оцінювання комерційного потенціалу розробки

Критерії оцінювання	ПІБ експертів		
	Експерт 1	Експерт 2	Експерт 3
	Бали		
Технічна здійсненність концепції	3	4	4
Наявність аналогів на ринку	3	3	4
Цінова політика	3	4	3
Технічні та споживчі властивості виробу	4	4	4
Експлуатаційні витрати	3	4	3
Ринок збуту	4	3	4
Конкурентоспроможність	3	4	3
Фахівці з технічної і комерційної реалізації	4	3	4
Фінансування	4	4	3
Матеріально-технічна база	3	3	3
Термін реалізації ідеї	4	4	3
Супровідна документація	4	3	3

Сума	42	43	41
Середньоарифметична сума балів	(42+43+41) / 3 = 42		

За даними таблиці 5.2 можна зробити висновок щодо рівня комерційного потенціалу даної розробки. Для цього доцільно скористатись рекомендаціями, наведеними в таблиці 5.3.

Таблиця 5.3 — Рівні комерційного потенціалу розробки

Середньоарифметична сума балів, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

Як видно з таблиці, рівень комерційного потенціалу розроблюваного нового програмного продукту є високим, що досягається за рахунок покращення методу створення запитів штучного інтелекту та організація отриманих туристичних даних для подальшого структурованого відображення користувачеві.

5.2 Комерційний та технологічний аудит науково-технічної розробки

5.2.1 Основна заробітна плата розробників, яка розраховується за формулою:

$$З_о = \frac{M}{T_р} \cdot t \,, \tag{5.1}$$

де М — місячний посадовий оклад конкретного розробника (дослідника), грн.;

$T_р$ — число робочих днів за місяць, 20 днів;

t — число днів роботи розробника (дослідника).

Результати розрахунків зведемо до таблиці 5.4.

Таблиця 5.4 — Основна заробітна плата розробників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	36000	1800,00	30	54000,000
Програміст	35000	1750,00	30	52500,000
Всього				106500,00

Так як в даному випадку розробляється програмний продукт, то розробник виступає одночасно і основним робітником, і тестувальником розроблюваного програмного продукту.

5.2.2 Додаткову заробітну плату прийнято розраховувати як 11,5 % від основної заробітної плати розробників та робітників:

$$З_д = З_о \cdot 11,5\ \% / 100\ \% \qquad (5.2)$$

$$З_д = (106500,00 \cdot 11,5\ \% / 100\ \%) = 12247,50\ \text{(грн.)}$$

5.2.3 Згідно діючого законодавства нарахування на заробітну плату складають 22 % від суми основної та додаткової заробітної плати.

$$Н_з = (З_о + З_д) \cdot 22\ \% / 100\% \qquad (5.3)$$

$$Н_з = (106500,00 + 12247,50) \cdot 22\ \% / 100\ \% = 26124,45\ \text{(грн.)}$$

5.2.4 Оскільки для розроблювального пристрою не потрібно витрачати матеріали та комплектуючи, то витрати на матеріали і комплектуючі дорівнюють нулю.

5.2.5 Амортизація обладнання, яке використовувалось для проведення розробки.

Амортизація обладнання, що використовувалось для розробки в спрощеному вигляді розраховується за формулою:

$$A = \frac{Ц}{Тв} \cdot \frac{t_{вик}}{12} \text{ [грн.].} \quad (5.4)$$

де Ц — балансова вартість обладнання, грн.;

Т — термін корисного використання обладнання згідно податкового законодавства, років;

$t_{вик}$ — термін використання під час розробки, місяців.

Розрахуємо, для прикладу, амортизаційні витрати на комп'ютер балансова вартість якого становить 41999 грн., термін його корисного використання згідно податкового законодавства — 2 роки, а термін його фактичного використання — 1,50 міс.

$$A_{обл} = \frac{47999}{2} \times \frac{1{,}5}{12} = 2624{,}94 \; грн.$$

Аналогічно визначаємо амортизаційні витрати на інше обладнання та приміщення. Розрахунки заносимо до таблиці 5.5. Так як вартість ліцензійної ОС та спеціалізованих ліцензійних нематеріальних ресурсів менше 20000 грн, то даний нематеріальний актив не амортизується, а його вартість включається у вартість розробки повністю, $B_{нем.ак.}$ = 2100 грн.

Таблиця 5.5 — Амортизаційні відрахування на матеріальні та нематеріальні ресурси для розробників

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання обладнання, місяців	Амортиза-ційні відрахуван-ня, грн.
Комп'ютер та комп'ютерна периферія	41999	2	1,50	2624,938
Офісне обладнання (меблі)	25000	4	1,50	781,250
Приміщення	1200000	20	1,50	7500,000
Всього				10906,19

5.2.6 Тарифи на електроенергію для непобутових споживачів (промислових підприємств) відрізняються від тарифів на електроенергію для населення. При цьому тарифи на розподіл електроенергії у різних постачальників (енергорозподільних компаній), будуть різними. Крім того, розмір тарифу залежить від класу напруги (1-й або 2-й клас). Тарифи на розподіл електроенергії для всіх енергорозподільних компаній встановлює Національна комісія з регулювання енергетики і комунальних послуг (НКРЕКП). Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п, \quad (5.5)$$

де В — вартість 1 кВт-години електроенергії для 1 класу підприємства з ПДВ в 2024 році для Вінницької області за даними Енера-Вінниця, В = 10,42 грн./кВт;

П — встановлена потужність обладнання, кВт. П = 0,25 кВт;

Ф — фактична кількість годин роботи обладнання, годин;

$К_п$ — коефіцієнт використання потужності, $К_п$ = 0,9.

$$В_е = 0,9 \cdot 0,25 \cdot 8 \cdot 30 \cdot 10,42 = 562,68 \text{ (грн.)}$$

5.2.7 До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками. Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.6)$$

де $Н_{ів}$ — норма нарахування за статтею «Інші витрати».

$$I_в = 106500,00 * 55\% / 100\% = 58575 \text{ (грн.)}$$

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов’язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов’язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов’язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін. Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників:

$$Н_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.7)$$

де $Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати».

$$Н_{нзв} = 106500,00 * 115 \% / 100 \% = 122475 \text{ (грн.)}$$

5.2.9 Сума всіх попередніх статей витрат дає загальні витрати на проведення науково-дослідної роботи:

$$B_{заг} = 106500,00+12247,50+26124,45+10906,19+2100+562,68+58575+$$
$$+122475 = 339490,82 \text{ грн.}$$

5.2.10 Розрахунок загальних витрат на науково-дослідну (науково-технічну) роботу та оформлення її результатів.

Загальні витрати на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{B_{заг}}{\eta} \text{ (грн)}, \quad (5.8)$$

де η — коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи.

Так, якщо науково-технічна розробка знаходиться на стадії: науково-дослідних робіт, то η=0,1; технічного проектування, то η=0,2; розробки конструкторської документації, то η=0,3; розробки технологій, то η=0,4; розробки дослідного зразка, то η=0,5; розробки промислового зразка, то η=0,7; впровадження, то η=0,9. Оберемо η = 0,5, так як розробка, на даний момент, знаходиться на стадії дослідного зразка:

$$ЗВ = 339490,82 / 0,5 = 678982 \text{ грн.}$$

5.3 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

В ринкових умовах узагальнювальним позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку. Саме зростання чистого прибутку забезпечить потенційному інвестору надходження додаткових коштів, дозволить покращити фінансові результати його діяльності, підвищить конкурентоспроможність та може позитивно вплинути на ухвалення рішення щодо комерціалізації цієї розробки.

Для того, щоб розрахувати можливе зростання чистого прибутку у потенційного інвестора від можливого впровадження науково-технічної розробки необхідно:

— вказати, з якого часу можуть бути впроваджені результати науково-технічної розробки;

— зазначити, протягом скількох років після впровадження цієї науково-технічної розробки очікуються основні позитивні результати для потенційного інвестора (наприклад, протягом 3-х років після її впровадження);

— кількісно оцінити величину існуючого та майбутнього попиту на цю або аналогічні чи подібні науково-технічні розробки та назвати основних суб’єктів (зацікавлених осіб) цього попиту;

— визначити ціну реалізації на ринку науково-технічних розробок з аналогічними чи подібними функціями.

При розрахунку економічної ефективності потрібно обов’язково враховувати зміну вартості грошей у часі, оскільки від вкладення інвестицій до отримання прибутку минає чимало часу. При оцінюванні ефективності інноваційних проектів передбачається розрахунок таких важливих показників:

— абсолютного економічного ефекту (чистого дисконтованого доходу);

— внутрішньої економічної дохідності (внутрішньої норми дохідності);

— терміну окупності (дисконтованого терміну окупності).

Аналізуючи напрямки проведення науково-технічних розробок, розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором можна об'єднати, враховуючи визначені ситуації з відповідними умовами.

5.3.1 Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем. В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

$$\Delta П_i = (\pm \Delta Ц_0 \cdot N + Ц_0 \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \qquad (5.9)$$

де $\pm\Delta Ц_o$ — зміна вартості програмного продукту (зростання чи зниження) від впровадження результатів науково-технічної розробки в аналізовані періоди часу;

N — кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ — основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки, $Ц_o = Ц_б \pm \Delta Ц_o$;

$Цб$ — вартість програмного продукту у році до впровадження результатів розробки;

ΔN — збільшення кількості споживачів продукту, в аналізовані періоди часу, від покращення його певних характеристик;

λ — коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт $\lambda = 0{,}8333$;

p — коефіцієнт, який враховує рентабельність продукту;

ϑ — ставка податку на прибуток, у 2024 році ϑ =18%.

Припустимо, що при прогнозованій ціні 1100 грн. за одиницю виробу, термін збільшення прибутку складе 3 роки. Після завершення розробки і її вдосконалення, можна буде підняти її ціну на 100 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року — на 7000 шт., протягом другого року — на 6000 шт., протягом третього року на 5000 шт. До моменту впровадження результатів наукової розробки реалізації продукту не було:

$$\Delta П_1 = (0*100 + (1100 + 100)*7000)* 0,8333* 0,4) * (1 - 0,18) = 2104666,582 \text{ грн.}$$

$$\Delta П_2 = (0*100 + (1100 + 100)*(7000+6000)* 0,8333* 0,4) * (1 - 0,18) = 4263999,829 \text{ грн.}$$

$$\Delta П_3 = (0*100 + (1100 + 100)*(7000+6000+5000)* 0,8333* 0,4) * (1 - 0,18) = 5903999,764 \text{ грн.}$$

Отже, комерційний ефект від реалізації результатів розробки за три роки складе 12272666,18 грн.

5.3.2 Розраховуємо приведену вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої науково-дослідної (науково-технічної) роботи, грн;

Т — період часу, протягом якою виявляються результати впровадженої науково-дослідної (науково-технічної) роботи, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, $\tau = 0,05...0,15$;

t — період часу (в роках).

Збільшення прибутку ми отримаємо, починаючи з першого року:

$$ПП = (2104666,582/(1+0,1)^1)+(4263999,829/(1+0,1)^2)+(5903999,764/$$
$$/(1+0,1)^3) = 1913333,26 + 3523966,801 + 4435762,407 = 9873062,465 \text{ грн.}$$

Далі розраховують величину початкових інвестицій *PV*, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} * ЗВ, \quad (5.11)$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію — це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}=2...5$, але може бути і більшим;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 * 678982 = 1357963,27 \text{ грн.}$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохід *(NPV, Net Present Value)* для потенційного інвестора від можливого

впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \tag{5.12}$$

$$E_{абс} = 9873062{,}465 - 1357963{,}27 = 8515099{,}20 \text{ грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів даної науково-дослідної (науково-технічної) роботи може бути доцільним.

Для остаточного прийняття рішення з цього питання необхідно розрахувати внутрішню економічну дохідність або показник внутрішньої но-рми дохідності *(IRR, Internal Rate of Return)* вкладених інвестицій та порівняти її з так званою бар'єрною ставкою дисконтування, яка визначає ту мінімальну внутрішню економічну дохідність, нижче якої інвестиції в будь-яку науково-технічну розробку вкладати буде економічно недоцільно.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього використаємо формулу:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{5.13}$$

де $T_ж$ – життєвий цикл наукової розробки, роки.

$$E_в = \sqrt[3]{(1 + 8515099{,}20/1357963{,}27} - 1 = 0{,}937$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{5.14}$$

де d — середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,09…0,14);

f — показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,5).

$$\tau_{\min} = 0,14 + 0,05 = 0,19.$$

Так як $E_в > \tau_{min}$, то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в}, \tag{5.15}$$

де $T_{ок}$ = 1 / 0,937 = 1,07 р.

Оскільки $T_{ок}$ < 3-х років, а саме термін окупності рівний 1,07 роки, то фінансування даної наукової розробки є доцільним.

Висновки до розділу: економічна частина даної роботи містить розрахунок витрат на розробку нового програмного продукту, сума яких складає 678982 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розраховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, розраховано період окупності витрат для інвестора та економічний ефект при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розроблений програмний продукт за ціною дешевший за аналог і є висококонкурентоспроможним. Період окупності складе близько 1,07 роки.

ВИСНОВКИ

У процесі виконання магістерської кваліфікаційної роботи було здійснен У ході виконання магістерської кваліфікаційної роботи проведено дослідження та розроблено програмний комплекс для автоматизованого збору та організації туристичних даних.

У першому розділі було виконано аналіз предметної області, включаючи огляд сучасних технологій автоматизації збору даних. Проаналізовано можливості та недоліки існуючих картографічних сервісів, які обмежені в персоналізації інформації та інтеграції аналітичних даних. Додатково розглянуто аналогічні системи, такі як TripIt та Wanderlog, що дозволило виявити прогалини у функціоналі та обґрунтувати створення нового програмного комплексу.

Другий розділ був присвячений розробці алгоритмів збору та організації даних. Було запропоновано методи генерації запитів із використанням ChatGPT API, їх подальшої обробки через Google Maps API та збагачення аналітичними характеристиками за допомогою Gemini API. Розроблено алгоритми для об’єднання, стандартизації, фільтрації та впорядкування даних, що забезпечують структурованість і релевантність отриманої інформації.

У третьому розділі реалізовано програмний комплекс із використанням технологій Swift, Core Data, Google Maps API та SwiftUI. Система була протестована для перевірки її продуктивності та точності роботи. Результати підтвердили, що запропонований комплекс забезпечує високу швидкодію, інтуїтивний інтерфейс і коректність роботи при обробці запитів на iOS-пристроях.

Четвертий розділ зосереджувався на тестуванні системи у реальних умовах. Проведено оцінку її ефективності, точності результатів і зручності використання з боку користувачів. Порівняння з існуючими аналогами показало,

що розроблений програмний комплекс має низку переваг, зокрема високу персоналізацію запитів і вдосконалене представлення даних.

Отже, виконана робота підтверджує значущість автоматизації збору та обробки туристичних даних. Запропоноване рішення відповідає сучасним вимогам користувачів, усуває обмеження існуючих рішень і сприяє розвитку цифрових інструментів у туристичній галузі. Реалізований програмний комплекс може стати ефективним інструментом для планування подорожей, підвищуючи зручність і якість обслуговування туристів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Велянський С.А., Муращенко О.Г. Автоматизований збір та організація туристичних даних із використанням штучного нтелекту Конференція ВНТУ: Молодь в науці: дослідження, проблеми, перспективи (МН-2024). Режим доступу: https://conferences.vntu.edu.ua/index.php/mn/mn2025/paper/view/22954.

2. Apple Inc. Core Data Framework. [Електронний ресурс]. URL: https://developer.apple.com/documentation/coredata (дата звернення: 10.12.2024).

3. Apple Inc. SwiftUI Framework. [Електронний ресурс]. URL: https://developer.apple.com/documentation/swiftui (дата звернення: 10.12.2024).

4. OpenAI. ChatGPT API Documentation. [Електронний ресурс]. URL: https://platform.openai.com/docs/ (дата звернення: 10.12.2024).

5. Google Maps API Documentation. [Електронний ресурс]. URL: https://developers.google.com/maps/documentation/ (дата звернення: 10.12.2024).

6. Alamofire Library. [Електронний ресурс]. URL: https://github.com/Alamofire/Alamofire (дата звернення: 10.12.2024).

7. Gemini API. [Електронний ресурс]. URL: https://geminiapi.com (дата звернення: 10.12.2024).

8. Nielsen, M. Neural Networks and Deep Learning: A Visual Introduction to Deep Learning. Determination Press, 2015. 220 p.

9. Haykin, S. Neural Networks and Learning Machines. Pearson Education, 2009. 936 p.

10. Vaswani, A., Shazeer, N., Parmar, N., et al. Attention Is All You Need. Advances in Neural Information Processing Systems, 2017. 600 p.

11. Koehn, P. Statistical Machine Translation. Cambridge: Cambridge University Press, 2010. 446 p.

12. Goodfellow, I., Bengio, Y., Courville, A. Deep Learning. Cambridge: MIT Press, 2016. 775 p.

13. Python. "Python is a programming language that lets you work quickly and integrate systems more effectively." [Електронний ресурс]. URL: https://www.python.org (дата звернення: 10.12.2024).

14. Redis. "An open-source, in-memory data store used as a database, cache, and message broker." [Електронний ресурс]. URL: https://redis.io (дата звернення: 10.12.2024).

15. FastAPI. "A modern, fast (high-performance) web framework for building APIs with Python 3.6+." [Електронний ресурс]. URL: https://fastapi.tiangolo.com (дата звернення: 10.12.2024).

16. Bahdanau, D., Cho, K., Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. Montreal: Arxiv Publications, 2014. 198 p.

17. Mozilla TTS. "Mozilla TTS is an open-source Text-To-Speech engine designed for high-quality, natural-sounding voice synthesis." [Електронний ресурс]. URL: https://github.com/mozilla/TTS (дата звернення: 10.12.2024).

18. Core Graphics Framework. [Електронний ресурс]. URL: https://developer.apple.com/documentation/coregraphics (дата звернення: 10.12.2024).

19. XCTest Framework. [Електронний ресурс]. URL: https://developer.apple.com/documentation/xctest (дата звернення: 10.12.2024).

20. Google Places API. "Find detailed information about places using Google Places API." [Електронний ресурс]. URL: https://developers.google.com/maps/documentation/places (дата звернення: 10.12.2024).

21. Esri ArcGIS. "Mapping and analytics software solutions." [Електронний ресурс]. URL: https://www.esri.com/en-us/arcgis/products/arcgis-online/overview (дата звернення: 10.12.2024).

22. FFmpeg Documentation. [Електронний ресурс]. URL: https://ffmpeg.org/documentation.html (дата звернення: 10.12.2024).

23. Alamofire for iOS. [Електронний ресурс]. URL: https://cocoapods.org/pods/Alamofire (дата звернення: 10.12.2024).

24. PostgreSQL Documentation. [Електронний ресурс]. URL: https://www.postgresql.org/docs/ (дата звернення: 10.12.2024).

25. Xcode Documentation. [Електронний ресурс]. URL: https://developer.apple.com/xcode/ (дата звернення: 10.12.2024).

26. Trudgill, P. Dialects of English: Studies in Grammatical Variation. Cambridge: Cambridge University Press, 2011. 320 p.

27. Cybersecurity Ventures. "50% of all data to be stored in the cloud by 2021." [Електронний ресурс]. URL: https://www.globenewswire.com/news-release/2023/03/22/2632412/0/en (дата звернення: 10.12.2024).

28. Bahdanau, D., Cho, K., Bengio, Y. Neural Machine Translation by Jointly Learning to Align and Translate. Montreal: Arxiv Publications, 2014. 198 p.

29. Whisper by OpenAI. [Електронний ресурс]. URL: https://openai.com/research/whisper (дата звернення: 10.12.2024).

30. Goodfellow, I., Bengio, Y., Courville, A. Deep Learning. MIT Press, 2016. 775 p.

31. Apple Developer Documentation. [Електронний ресурс]. URL: https://developer.apple.com (дата звернення: 10.12.2024).

32. TensorFlow Documentation. [Електронний ресурс]. URL: https://www.tensorflow.org/ (дата звернення: 10.12.2024).

33. OpenAI GPT Models. [Електронний ресурс]. URL: https://openai.com/gpt (дата звернення: 10.12.2024).

34. JSON Data Formats. [Електронний ресурс]. URL: https://www.json.org/ (дата звернення: 10.12.2024).

35. Swift Programming Guide. [Електронний ресурс]. URL: https://swift.org/documentation/ (дата звернення: 10.12.2024).

36. Firebase Documentation. [Електронний ресурс]. URL: https://firebase.google.com/docs (дата звернення: 10.12.2024).

37. DeepMind AI Research. [Електронний ресурс]. URL: https://www.deepmind.com/research (дата звернення: 10.12.2024).

38. Taylor, P. Text-to-Speech Synthesis. Cambridge: Cambridge University Press, 2009. 626 p.

39. Clark, R. A. J., King, S., Black, A. W. Statistical Parametric Speech Synthesis. Wiley, 2014. 384 p.

40. Rask AI. "Transform your content with AI-powered video localization and dubbing." [Електронний ресурс]. URL: https://www.rask.ai (дата звернення: 10.12.2024).

41. HeyGen. "Create engaging videos effortlessly with our AI video generator." [Електронний ресурс]. URL: https://www.heygen.com (дата звернення: 10.12.2024).

42. Verbalate. "Seamless audio and video translations with voice cloning and lip-sync." [Електронний ресурс]. URL: https://verbalate.ai (дата звернення: 10.12.2024).

43. Coqui TTS. "A flexible and open-source library for Text-To-Speech synthesis." [Електронний ресурс]. URL: https://github.com/coqui-ai/TTS (дата звернення: 10.12.2024).

44. Google Cloud Natural Language API. "Extract insights from unstructured text." [Електронний ресурс]. URL: https://cloud.google.com/natural-language (дата звернення: 10.12.2024).

45. TripIt. "Trip planner and management app." [Електронний ресурс]. URL: https://www.tripit.com (дата звернення: 10.12.2024).

46. Wanderlog. "Simplify your travel planning with Wanderlog." [Електронний ресурс]. URL: https://wanderlog.com (дата звернення: 10.12.2024).

47. Esri ArcGIS. "Solutions for interactive mapping and analytics." [Електронний ресурс]. URL: https://www.esri.com/en-us/arcgis/products/arcgis-online/overview (дата звернення: 10.12.2024).

48. Plotly Dash. "Interactive, web-based data visualizations in Python." [Електронний ресурс]. URL: https://plotly.com/dash (дата звернення: 10.12.2024).

49. ElasticSearch. "Distributed, RESTful search and analytics engine." [Електронний ресурс]. URL: https://www.elastic.co (дата звернення: 10.12.2024).

50. Seaborn Library. "Statistical data visualization based on Matplotlib." [Електронний ресурс]. URL: https://seaborn.pydata.org (дата звернення: 10.12.2024).

51. JSON Schema. "A standard for defining JSON data structure." [Електронний ресурс]. URL: https://json-schema.org (дата звернення: 10.12.2024).

52. DeepSpeech. "Open-source speech recognition engine by Mozilla." [Електронний ресурс]. URL: https://github.com/mozilla/DeepSpeech (дата звернення: 10.12.2024).

53. Firebase Authentication. "Manage user authentication and access." [Електронний ресурс]. URL: https://firebase.google.com/docs/auth (дата звернення: 10.12.2024).

54. OpenWeather API. "Real-time weather data for any location." [Електронний ресурс]. URL: https://openweathermap.org/api (дата звернення: 10.12.2024).

55. Mapbox API. "Build custom maps and navigation tools." [Електронний ресурс]. URL: https://www.mapbox.com (дата звернення: 10.12.2024).

56. Microsoft Cognitive Services. "AI-powered tools for vision, speech, and language." [Електронний ресурс]. URL: https://azure.microsoft.com/en-us/services/cognitive-services/ (дата звернення: 10.12.2024).

57. IBM Watson AI. "Advanced AI capabilities for business solutions." [Електронний ресурс]. URL: https://www.ibm.com/watson (дата звернення: 10.12.2024).

58. Swift Algorithms Package. "Useful algorithms for Swift development." [Електронний ресурс]. URL: https://swift.org/package-manager/ (дата звернення: 10.12.2024).

59. MongoDB Atlas. "Multi-cloud database as a service." [Електронний ресурс]. URL: https://www.mongodb.com/cloud/atlas (дата звернення: 10.12.2024).

60. Kotlin Multiplatform Mobile. "Framework for cross-platform mobile app development." [Електронний ресурс]. URL: https://kotlinlang.org/lp/mobile/ (дата звернення: 10.12.2024).

ДОДАТОК А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ

Завідувач кафедри ОТ

_____ проф., д.т.н. О.Д. Азаров

«29» вересня 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

Програмний комплекс збору та використання даних з допомогою штучного інтелекту

08-54.МКР.003.00.000 ТЗ

Керівник роботи: к.т.н., доц. каф. ОТ:

_____ Муращенко О.Г.

Виконав: студент гр. 1КІ-23м

_____Велянський С.А.

2024

1 Підстава для виконання бакалаврського дипломного проекту (МКР)

1.1 Актуальність полягає в тому, що з розвитком інформаційних технологій та поширенням мобільних додатків значно зросла необхідність у ефективному зборі, обробці та організації даних. Сьогодні туристична індустрія є одним із ключових секторів, де автоматизація цих процесів забезпечує зручність для користувачів і оптимізацію ресурсів. Туристам важливо не лише отримувати швидкий доступ до інформації про цікаві місця, але й мати можливість планувати маршрути, адаптовані під їхні індивідуальні потреби;

1.2 Наказ про затвердження теми МКР № 310 від 17.09.2024 р.

2 Мета і призначення МКР

2.1 Метою проекту є вдосконалення автоматизованого збору та обробки туристичних даних з використанням генеративних мовних моделей та картографічних сервісів та подальше використання їх у програмному комплексу;

2.2 Призначення розробки — автоматизована генерація запитів для нейромереж, збір отриманих туристичних даних та їх відображення за допомогою фреймворку SwiftUI, бібліотек Alamofire, ChatGPT API, Gemini API, Google Maps SDK, Google Maps API.

3 Вихідні дані для виконання МКР

Вихідні дані для виконання МКР: фреймворк SwiftUI, бібліотека Alamofire, ChatGPT API, Gemini API, Google Maps SDK, Google Maps API, програмне середовище Xcode, операційна система MacOS.

4 Технічні вимоги до виконання МКР

Технічні вимоги:

— аналіз методів збору та обробки даних з допомогою нейромереж;

— пошук та аналіз методів генерації запитів;

— вибір інструментів та бібліотек для програмної реалізації;

— створення програмного комплексу для автоматизованого збору та обробки туристичних даних;

— тестування працездатності комплексу.

5 Етапи МКР та очікувані результати

Етапи роботи та очікувані результати приведено в Таблиці А.1

Таблиця А.1 — Етапи МКР

№ з/п	Назва етапів дипломної Роботи	Термін виконання		Очікувані результати
		початок	закінчення	
1	Постановка задачі роботи	19.09.2024	19.09.2024	Задачі дослідження
2	Аналіз предметної області	20.09.2024	24.09.2024	Розділ 1
3	Вдосконалення методів генерації та організації туристичних даних	25.09.2024	04.10.2024	Розділ 2
4	Програмна реалізація програмного комплексу збору та обробки даних з допомогою штучного інтелекту	05.10.2024	26.10.2024	Розділ 3
5	Розрахунок економічної частини	11.11.2024	11.11.2024	Розділ 4
6	Оформлення матеріалів до захисту МКР	12.11.2024	12.11.2024	ПЗ, графічний матеріал і презентація
7	Перевірка якості виконання магістерської роботи та усунення недоліків	13.11.2024	14.11.2024	Оформлені документи
9	Підписи супроводжувальних документів у нормоконтролера, керівника, опонента	15.11.2024	15.11.2024	Оформлені документи
10	Перевірка на антиплагіат та ШІ	17.11.2024	17.11.2024	Оформлені документи

6 Матеріали, що подаються до захисту МКР

Пояснювальна записка МКР, графічні та ілюстративні матеріали, протокол попереднього захисту роботи на кафедрі, відзив наукового керівника, рецензія рецензента, анотації до МДП українською та іноземною мовами, нормоконтроль про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР

Виконання етапів графічної та розрахункової документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення МКР

8.1 При оформлювання МКР використовуються:

— ДСТУ 3008: 2015 «Звіти в сфері науки і техніки. Структура та правила оформлювання»;

— ДСТУ 8302: 2015 «Бібліографічні посилання. Загальні положення та правила складання»;

— міждержавний ГОСТ 2.104-2006 «Єдина система конструкторської документації. Основні написи»;

— методичні вказівки до виконання магістерських кваліфікаційних робіт зі спеціальності 123 — «Комп’ютерна інженерія»;

— документами на які посилаються у вище вказаних.

8.2 Порядок виконання МКР викладено в «Положення про кваліфікаційні роботи на другому (магістерському) рівні вищої освіти СУЯ ВНТУ–03.02.02 П.001.01:21.

ДОДАТОК Б

Структурна схема

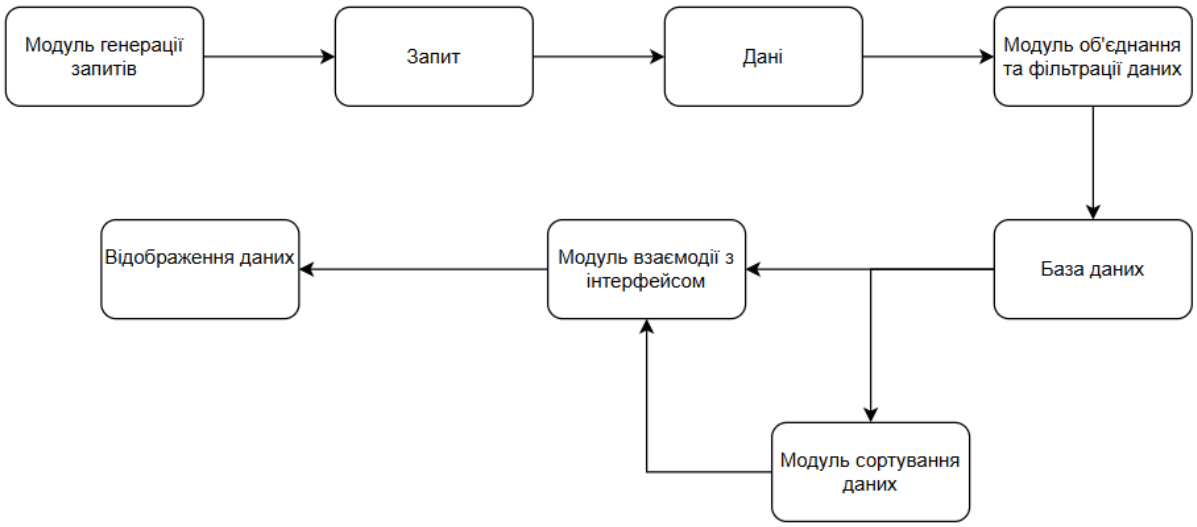

Рисунок Б.1 — Структурна схема програмного комплексу

ДОДАТОК В

Блок-схема алгоритму роботи

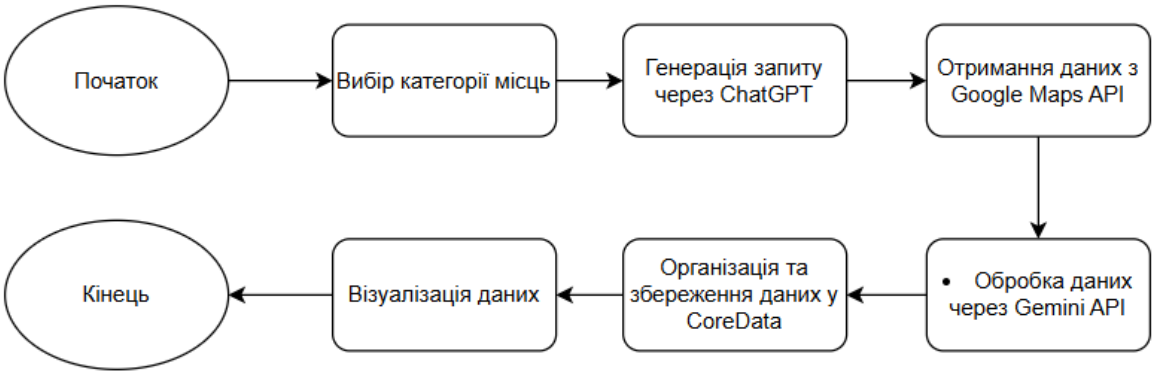

Рисунок В.1 — Алгоритм роботи програми

ДОДАТОК Г

Лістинг оперативних модулів

```
import CoreData
class DataManager {
    static let shared = DataManager()
    private init() {}
    lazy var persistentContainer: NSPersistentContainer = {
        let container = NSPersistentContainer(name: "TouristApp")
        container.loadPersistentStores { _, error in
            if let error = error {
                fatalError("Unresolved error: \(error)")
            }
        }
        return container
    }()
    var context: NSManagedObjectContext {
        return persistentContainer.viewContext
    }
    func savePlace(id: String, name: String, latitude: Double, longitude: Double, rating: Double, address: String, category: String) {
        let place = PlaceEntity(context: DataManager.shared.context)
        place.id = id
        place.name = name
        place.latitude = latitude
        place.longitude = longitude
        place.rating = rating
        place.address = address
```

```
        place.category = category
        saveContext()
    }
    private func saveContext() {
        do {
            try DataManager.shared.context.save()
        } catch {
            print("Error saving data: \(error)")
        }
    }
    func fetchPlaces() {
        let fetchRequest: NSFetchRequest<PlaceEntity> = PlaceEntity.fetchRequest()
        do {
            let places = try DataManager.shared.context.fetch(fetchRequest)
            for place in places {
                print("Place: \(place.name ?? "") - Rating: \(place.rating)")
            }
        } catch {
            print("Error fetching data: \(error)")
        }
    }
    }
    import Alamofire
    class NetworkManager {
        static let shared = DataManager()
        private init() {}
    func generateSearchQuery(prompt: String, completion: @escaping (String?) ->
Void) {
```

```
        let headers: HTTPHeaders = [
            "Authorization": "Bearer YOUR_CHATGPT_API_KEY",
            "Content-Type": "application/json"
        ]
        let parameters: [String: Any] = [
            "model": "gpt-4",
            "messages": [["role": "user", "content": prompt]],
            "max_tokens": 50
        ]
        AF.request("https://api.openai.com/v1/chat/completions", method: .post, parameters: parameters, encoding: JSONEncoding.default, headers: headers)
            .responseDecodable(of: ChatGPTResponse.self) { response in
                switch response.result {
                case .success(let result):
                    let query = result.choices.first?.message.content
                    completion(query)
                case .failure(let error):
                    print("Error generating query: \(error)")
                    completion(nil)
                }
            }
    }
    func fetchPlaces(query: String, location: String, radius: Int, completion: @escaping ([Place]) -> Void) {
        let url = "https://maps.googleapis.com/maps/api/place/textsearch/json"
        let parameters: [String: Any] = [
            "query": query,
            "location": location,
```

```
            "radius": radius,
            "key": "YOUR_GOOGLE_MAPS_API_KEY"
        ]
        AF.request(url, method: .get, parameters: parameters)
            .responseDecodable(of: GoogleMapsResponse.self) { response in
                switch response.result {
                case .success(let result):
                    let places = result.results.map { result in
                        Place(name: result.name, latitude: result.geometry.location.lat, longitude: result.geometry.location.lng, rating: result.rating, address: result.formatted_address)
                    }
                    completion(places)
                case .failure(let error):
                    print("Error fetching places: \(error)")
                    completion([])
                }
            }
    }
    func enrichDataWithGemini(place: Place, completion: @escaping (EnrichedPlace?) -> Void) {
        let headers: HTTPHeaders = [
            "Authorization": "Bearer YOUR_GEMINI_API_KEY",
            "Content-Type": "application/json"
        ]
        let parameters: [String: Any] = [
            "input": "Analyze the popularity and reviews for the place: \(place.name)"
        ]
```

```
        AF.request("https://gemini.googleapis.com/v1/analyze", method: .post, parameters: parameters, encoding: JSONEncoding.default, headers: headers)
            .responseDecodable(of: GeminiResponse.self) { response in
                switch response.result {
                case .success(let result):
                    let enrichedPlace = EnrichedPlace(
                        name: place.name,
                        latitude: place.latitude,
                        longitude: place.longitude,
                        rating: place.rating,
                        address: place.address,
                        popularity: result.popularity,
                        reviewsCount: result.reviews
                    )
                    completion(enrichedPlace)
                case .failure(let error):
                    print("Error enriching data: \(error)")
                    completion(nil)
                }
            }
    }

    func generateAndFetchData(category: String, location: String, radius: Int, completion: @escaping ([EnrichedPlace]) -> Void) {
        generateSearchQuery(prompt: "Find \(category) in the area.") { query in
            guard let query = query else { return }
            fetchPlaces(query: query, location: location, radius: radius) { places in
                var enrichedPlaces: [EnrichedPlace] = []
                let group = DispatchGroup()
```

```
            for place in places {
                group.enter()
                enrichDataWithGemini(place: place) { enrichedPlace in
                    if let enrichedPlace = enrichedPlace {
                        enrichedPlaces.append(enrichedPlace)
                    }
                    group.leave()
                }
            }
            group.notify(queue: .main) {
                completion(enrichedPlaces)
            }
        }
    }
}
}
```

ДОДАТОК Д

Лістинг візуальних модулів

```
struct ContentView: View {
    @State private var places: [PlaceEntity] = []
    var body: some View {
        TabView {
            PlaceListView()
                .tabItem {
                    Label("List", systemImage: "list.bullet")
                }
            MapView(places: $places)
                .tabItem {
                    Label("Map", systemImage: "map")
                }
                .onAppear {
                    NetworkManager.shared.generateSearchQuery(prompt: "Find popular Georgian restaurants in Vinnytsia") { query in
                        guard let query = query else { return }
                        NetworkManager.shared.fetchPlaces(query: query, location: "49.2331,28.4682") { fetchedPlaces in
                            places = fetchedPlaces.map { place in
                                let enrichedPlace = EnrichedPlace(
                                    name: place.name,
                                    latitude: place.latitude,
                                    longitude: place.longitude,
                                    rating: place.rating,
                                    address: place.address,
```

```
                        category: "Restaurant",
                        reviewsCount: 0
                    )
                    DataManager.shared.savePlace(place: enrichedPlace)
                    return enrichedPlace.toPlaceEntity()
                }
            }
        }
    }
}
extension EnrichedPlace {
    func toPlaceEntity() -> PlaceEntity {
        let place = PlaceEntity(context: DataManager.shared.context)
        place.name = name
        place.latitude = latitude
        place.longitude = longitude
        place.rating = rating
        place.address = address
        place.category = category
        place.reviewsCount = Int16(reviewsCount)
        return place
    }
}
import SwiftUI
struct PlaceListView: View {
```

```
    @State private var places: [PlaceEntity] = []
    @State private var searchQuery: String = ""
    @State private var sortOption: SortOption = .nameAscending
    enum SortOption: String, CaseIterable {
        case nameAscending = "Name (A-Z)"
        case nameDescending = "Name (Z-A)"
        case ratingDescending = "Rating (High-Low)"
    }
    var sortedAndFilteredPlaces: [PlaceEntity] {
        var filteredPlaces = places.filter {
            searchQuery.isEmpty ||
$0.name.lowercased().contains(searchQuery.lowercased())
        }
        switch sortOption {
        case .nameAscending:
            filteredPlaces.sort { $0.name < $1.name }
        case .nameDescending:
            filteredPlaces.sort { $0.name > $1.name }
        case .ratingDescending:
            filteredPlaces.sort { $0.rating > $1.rating }
        }
        return filteredPlaces
    }
    var body: some View {
        NavigationView {
            VStack {
                TextField("Search by name", text: $searchQuery)
```

```
            .padding(10)
            .textFieldStyle(RoundedBorderTextFieldStyle())
            .padding(.horizontal)
        Picker("Sort By", selection: $sortOption) {
            ForEach(SortOption.allCases, id: \.self) { option in
                Text(option.rawValue).tag(option)
            }
        }
        .pickerStyle(SegmentedPickerStyle())
        .padding(.horizontal)
        List(sortedAndFilteredPlaces) { place in
            NavigationLink(destination: PlaceDetailView(place: place)) {
                VStack(alignment: .leading) {
                    Text(place.name).font(.headline)
                    Text(place.address).font(.subheadline)
                    HStack {
                        Text("⭐ Rating: \(place.rating, specifier: "%.1f")")
                        Spacer()
                        Text("💬 Reviews: \(place.reviewsCount)")
                    }
                    .font(.caption)
                }
            }
        }
        .listStyle(PlainListStyle())
    }
```

```
            .navigationTitle("Tourist Places")
            .onAppear {
                self.places = DataManager.shared.fetchPlaces()
            }
        }
    }
}
import SwiftUI
struct PlaceDetailView: View {
    let place: PlaceEntity
    var body: some View {
        VStack(alignment: .leading, spacing: 16) {
            Text(place.name)
                .font(.largeTitle)
                .fontWeight(.bold)
                .multilineTextAlignment(.leading)
            Text("📍 Address: \(place.address)")
                .font(.headline)
                .foregroundColor(.secondary)
            HStack {
                Text("⭐ Rating: \(place.rating, specifier: "%.1f")")
                Spacer()
                Text("💬 Reviews: \(place.reviewsCount)")
            }
            .font(.body)
            Text("🏷 Category: \(place.category)")
```

```
                .font(.subheadline)
                .italic()
            Spacer()
        }
        .padding()
        .navigationTitle("Details")
    }
}
import SwiftUI
import GoogleMaps
struct MapView: UIViewRepresentable {
    @Binding var places: [PlaceEntity]
    func makeUIView(context: Context) -> GMSMapView {
        let camera = GMSCameraPosition.camera(withLatitude: 49.2331, longitude:
28.4682, zoom: 12.0)
        let mapView = GMSMapView.map(withFrame: .zero, camera: camera)
        return mapView
    }
    func updateUIView(_ mapView: GMSMapView, context: Context) {
        mapView.clear()
        for place in places {
            let marker = GMSMarker()
            marker.position = CLLocationCoordinate2D(latitude: place.latitude,
longitude: place.longitude)
            marker.title = place.name
            marker.snippet = "Rating: \(place.rating, specifier: "%.1f")"
            marker.map = mapView
        }
```

ДОДАТОК Е

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Комп'ютерна система автоматичного перекладу звукової доріжки у відеофайлах

Тип роботи: магістерська кваліфікаційна робота
(МКР, МКР)

Підрозділ кафедра обчислювальної техніки
(кафедра, факультет)

Показники звіту подібності
Turnitin

Оригінальність 97% Схожість 3%

Аналіз звіту подібності (відмітити потрібне):

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- ☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- ☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Захарченко С.М.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Turnitin щодо роботи.

Автор роботи _____ Велянський С.А.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Муращенко О.Г.
(підпис) (прізвище, ініціали)