

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ ДЛЯ ОРГАНІЗАЦІЇ
ТА УЧАСТІ У СУСПІЛЬНИХ ПОДІЯХ
DEVELOPMENT OF A MOBILE APPLICATION FOR SOCIAL EVENT
MANAGEMENT AND PARTICIPATION

Виконав(ла): студент(ка) 2 курсу, групи
КНТ-114м

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ТИЩЕНКО С.С.

(ПРИЗВИЩЕ та ініціали)

Керівник СТЕПАНЕНКО О.О.

(ПРИЗВИЩЕ та ініціали)

Рецензент БАБЕНКО Н.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2025 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ТИЩЕНКА Сергія Сергійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка мобільного застосунку для організації та участі у суспільних подіях.

Development of a Mobile Application for Social Event Management and Participation.

керівник проєкту (роботи) к.т.н., доцент, СТЕПАНЕНКО Олександр Олексійович

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 30 ” вересня 2025 року № 447

2. Строк подання студентом проєкту (роботи) 08 грудня 2025 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.

2. Матеріали і методи. 3. Розробка архітектури програми. 4. Розробка програми.

5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) _____

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	СТЕПАНЕНКО О.О., доцент		
Нормоконтролер	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання “ 30 ” вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	2-3 тижні	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	4-5 тижні	Розділ 2
4	Розробка архітектури програми.	6 тиждень	Розділ 3
5	Розробка програми.	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програмного забезпечення.	9 тиждень	Розділ 5
7	Оформлення пояснювальної записки та документів до неї.	10-11 тижні	Додатки
8	Нормоконтроль та рецензування.	12 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Сергій ТИЩЕНКО
(Імя ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Олександр СТЕПАНЕНКО
(Імя ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
90 с., 5 табл., 29 рис., 3 дод., 30 джерел.

МОБІЛЬНИЙ ЗАСТОСУНОК, СОЦІАЛЬНІ ПОДІЇ,
ІВЕНТ-МЕНЕДЖМЕНТ, КЛІЄНТ-СЕРВЕР, REACT NATIVE, NODE.JS,
POSTGRESQL, ГЕОЛОКАЦІЯ.

Об'єкт дослідження – процес розробки програмного забезпечення для організації, координації та управління суспільними подіями в цифровому середовищі.

Предмет дослідження – методи та програмні засоби автоматизації взаємодії учасників та організаторів заходів з використанням мобільних технологій.

Мета роботи – підвищення ефективності процесів організації та координації суспільних подій шляхом розробки спеціалізованої програмної системи, що забезпечує скорочення часу на планування заходів та пошук релевантного контенту.

Матеріали, методи та технічні засоби: системний аналіз предметної області, проектування клієнт-серверної архітектури, моделювання реляційних баз даних, використання REST API для обміну даними, мови програмування TypeScript та JavaScript, фреймворк React Native (фронтенд), середовище Node.js (бекенд), система керування базами даних PostgreSQL.

Результати. Проведено аналіз існуючих рішень у сфері івент-менеджменту, обґрунтовано вибір архітектурного патерну та технологічного стека. Розроблено та імплементовано програмну систему, що складається з серверної частини та кросплатформенного мобільного клієнта, яка дозволяє створювати події, керувати списками учасників та здійснювати пошук заходів.

Висновки. Досліджено та програмно реалізовано систему для організації суспільних подій, яка вирішує проблему децентралізованої координації груп людей та спрощує пошук актуальних заходів.

Галузь використання – сфера дозвілля та розваг, організація професійних зустрічей, соціальна координація громад.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 90 pages, 5 tables, 29 figures, 3 appendixes, 30 sources.

MOBILE APPLICATION, SOCIAL EVENTS, EVENT MANAGEMENT, CLIENT-SERVER, REACT NATIVE, NODE.JS, POSTGRESQL, GEOLOCATION.

The object of study is a process of software development for organization, coordination, and management of public events in the digital environment.

The subject of study are methods and software tools for automating the interaction of participants and event organizers using mobile technologies.

The purpose of the work is to improve the efficiency of organizing and coordinating public events by developing a specialized software system that reduces the time required for event planning and for finding relevant content.

Materials, methods, and tools: system analysis of the domain area, design of client-server architecture, relational database modeling, use of REST API for data exchange, TypeScript and JavaScript programming languages, React Native framework (frontend), Node.js environment (backend), PostgreSQL database management system.

Results. An analysis of existing solutions in the field of event management was carried out, and the choice of architectural pattern and technology stack was substantiated. A software system consisting of a server part and a cross-platform mobile client has been developed and implemented, allowing creating events, managing participant lists, and searching for activities.

Conclusions. A system for organizing public events has been researched and implemented, solving the problem of decentralized group coordination and simplifying the search for relevant events.

The field of use - leisure and entertainment, organization of professional meetings, social community coordination.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	9
Вступ.....	10
1 Аналіз предметної області та постановка завдань дослідження.....	12
1.1 Аналіз особливостей організації та координації суспільних подій у цифровому середовищі.....	13
1.2 Огляд існуючих аналогів та платформ для івент-менеджменту.....	14
1.2.1 Платформа Meetup.....	14
1.2.2 Інструмент Facebook Events.....	15
1.2.3 Сервіс Eventbrite.....	16
1.2.4 Локальні Telegram-канали та чати.....	17
1.3 Висновки за розділом 1.....	20
2 Матеріали і методи.....	21
2.1 Обґрунтування вибору мови програмування та середовища виконання системи.....	22
2.2 Вибір системи керування базами даних.....	23
2.3 Вибір фреймворку для мобільної розробки.....	27
2.4 Діаграма прецедентів.....	28
2.5 Висновки за розділом 2.....	30
3 Розробка архітектури програми.....	31
3.1 Аналіз архітектурних рішень для високонавантажених систем організації подій.....	32
3.2 Проектування схеми бази даних.....	33
3.3 Структура програмних компонентів.....	35
3.4 Висновки за розділом 3.....	36

4 Розробка програми.....	37
4.1 Реалізація серверної частини системи.....	37
4.1.1 Розробка API та логіки авторизації.....	37
4.1.2 Реалізація механізму створення та управління подіями.....	38
4.1.3 Реалізація системи рекомендацій подій.....	39
4.2 Реалізація клієнтської частини системи.....	40
4.2.1 Інтерфейс користувача та навігація.....	40
4.2.2 Взаємодія з геолокацією та картами.....	41
4.3 Висновки за розділом 4.....	41
5 Експлуатація, тестування та експериментальне дослідження програми.....	43
5.1 Призначення та умови виконання системи.....	43
5.1.1 Вимоги до апаратного забезпечення сервера.....	43
5.1.2 Вимоги до апаратного забезпечення клієнта.....	43
5.1.3 Вимоги до програмного забезпечення розробника.....	44
5.2 Підготовка середовища та розгортання.....	44
5.2.1 Клонування репозиторію та налаштування змінних середовища.....	44
5.2.2 Запуск серверної частини.....	45
5.2.3 Запуск клієнтської частини.....	45
5.3 Інструкція користувача.....	46
5.3.1 Реєстрація та авторизація.....	47
5.3.2 Перегляд та пошук подій.....	48
5.3.3 Створення нової події.....	50
5.3.4 Перегляд деталей та участь.....	52
5.3.5 Профіль користувача.....	53
5.4 Тестування програмного забезпечення.....	54
5.4.1 Тестування серверної частини.....	54

5.4.2 Тестування клієнтської частини.....	55
5.5 Висновки за розділом 5.....	56
Висновки.....	58
Перелік джерел посилання.....	59
Додаток А Технічне завдання.....	62
Додаток Б Фрагмент тексту програми.....	68
Додаток В Слайди презентації.....	82

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
GPS	– Global Positioning System;
HTTP	– HyperText Transfer Protocol;
JSON	– JavaScript Object Notation;
JWT	– JSON Web Token;
MVP	– Minimum Viable Product;
REST	– Representational State Transfer;
SDK	– Software Development Kit;
SQL	– Structured Query Language;
UI	– User Interface;
UX	– User Experience;
ORM	– Object-Relational Mapping;
БД	– база даних;
ВВЛ	– віддалені та віртуальні лабораторії;
ІС	– інформаційна система;
ОС	– операційна система;
ПЗ	– програмне забезпечення;
СКБД	– система керування базами даних;

ВСТУП

Соціальна взаємодія та організація спільних заходів є невід'ємною частиною життя сучасного суспільства. В умовах прискореного ритму життя та цифровізації комунікацій виникає проблема ефективної координації груп людей для проведення дозвілля, професійних зустрічей чи громадських акцій. Існуючі інструменти, такі як месенджери або соціальні мережі, часто не забезпечують достатнього рівня структурованості інформації: важливі деталі губляться у стрічці повідомлень, а планування подій вимагає використання сторонніх сервісів. Спеціалізовані платформи часто є громіздкими, орієнтованими на великі комерційні заходи або мають обмежений функціонал у безкоштовних версіях.

Тому актуальним є дослідження методів оптимізації процесів організації подій та розробка спеціалізованого програмного забезпечення, яке б поєднувало зручність мобільного інтерфейсу, можливості геолокації для пошуку подій поблизу та надійну клієнт-серверну архітектуру для забезпечення актуальності даних у реальному часі.

Об'єктом дослідження є процес розробки програмного забезпечення для організації, координації та управління суспільними подіями в цифровому середовищі.

Предметом дослідження є методи та програмні засоби автоматизації взаємодії учасників та організаторів заходів з використанням мобільних технологій та клієнт-серверної архітектури.

Мета роботи – підвищення ефективності соціальної взаємодії та скорочення часових витрат користувачів на організацію і пошук подій шляхом розробки мобільної інформаційної системи з використанням геолокації та алгоритмів рекомендацій.

Для досягнення поставленої мети необхідно вирішити такі задачі:

- виконати аналіз предметної області, дослідити існуючі аналоги та виявити їх недоліки;
- провести порівняльний аналіз інструментарію розробки та обґрунтувати вибір технологічного стека;
- спроектувати архітектуру системи, розробити діаграми прецедентів, класів та схему бази даних;
- виконати програмну реалізацію серверної частини з використанням REST API;
- виконати програмну реалізацію мобільної клієнтської частини для ОС Android/iOS;
- провести тестування розробленого програмного забезпечення та оцінити його ефективність.

У роботі використано методи системного аналізу для визначення вимог до системи [1], методи об'єктно-орієнтованого проєктування для розробки структури ПЗ [2], методи моделювання баз даних для організації зберігання інформації [3], емпіричні методи тестування для перевірки працездатності системи.

Наукова новизна одержаних результатів полягає в удосконаленні підходу до організації локальних суспільних подій шляхом інтеграції геолокаційних сервісів з системою миттєвих сповіщень та гнучким керуванням ролями учасників в межах єдиної мобільної платформи, що дозволяє зменшити час на координацію груп.

Практичне значення одержаних результатів полягає у створенні діючої програмної системи, яка може бути використана спільнотами, організаторами заходів або окремими користувачами для планування та проведення зустрічей.

Розроблене рішення є кросплатформенним, що забезпечує широке охоплення аудиторії.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Аналіз особливостей організації та координації суспільних подій у цифровому середовищі

Сучасний урбанізований простір характеризується парадоксальним явищем: при високій щільності населення та доступності цифрових комунікацій [4] рівень соціальної ізоляції та дезінтеграції локальних спільнот зростає [5]. Традиційні механізми організації дозвілля та громадських ініціатив трансформуються під впливом цифровізації [6].

Процес організації суспільної події — від спонтанної зустрічі за інтересами (наприклад, спільне тренування на спортивному майданчику) до масштабних запланованих акцій (міські марафони, екологічні ініціативи) — вимагає вирішення низки комунікаційних та логістичних задач, а саме:

- інформування цільової аудиторії: донесення інформації до зацікавлених осіб, які знаходяться в певному географічному ареалі [7];
- координація у часі та просторі: чітке визначення місця проведення з прив'язкою до карти та часу початку/завершення;
- управління учасниками: оцінка кількості присутніх, розподіл ролей (організатор, учасник, волонтер, тощо).

Існуюча екосистема цифрових інструментів для організації подій є вкрай фрагментованою. Інформація про заходи розпорошена між різними платформами, які мають різну специфіку та бар'єри входу: месенджери, соціальні мережі, спеціалізовані квиткові сервіси.

Таким чином, існує невирішена проблема відсутності універсального інструменту, який би дозволяв користувачеві миттєво отримати відповідь на запитання: «Що відбувається навколо мене прямо зараз?», незалежно від масштабу та тематики події.

1.2 Огляд існуючих аналогів та платформ для івент-менеджменту

Для визначення функціональних вимог до розроблюваної системи було проведено порівняльний аналіз існуючих рішень на ринку, що дозволило виявити їхні сильні та слабкі сторони.

1.2.1 Платформа Meetup

Одна з найпопулярніших глобальних платформ для пошуку спільнот за інтересами та організації регулярних зустрічей (рис. 1.1) [8].

До переваг платформи можна віднести:

- чітку структуру груп та ієрархію спільнот;
- розвинену систему автоматичних сповіщень про нові події для учасників групи;
- фокус на регулярних тематичних зустрічах, що сприяє формуванню стійких спільнот.

Серед недоліків рішення варто виділити:

- високу вартість використання для організаторів (обов'язкова платна підписка для створення подій);
- орієнтацію переважно на професійні спільноти та довгострокові групи, а не на спонтанні разові активності;
- перевантажений інтерфейс, який ускладнює швидкий пошук подій за принципом «тут і зараз».

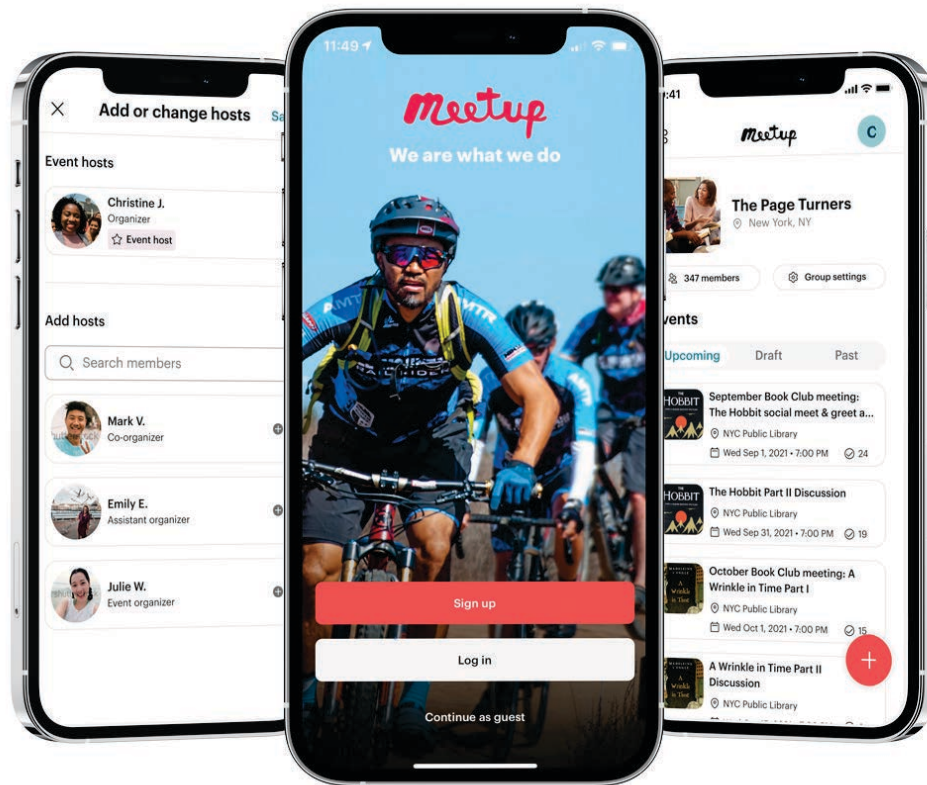


Рисунок 1.1 – Програмне забезпечення Meetup

1.2.2 Інструмент Facebook Events

Потужний інструмент, інтегрований у соціальну мережу Facebook, який використовує соціальний граф користувача для рекомендацій (рис. 1.2) [9].

До переваг платформи можна віднести:

- велика база активних користувачів, що вже зареєстровані в соціальній мережі;
- високий рівень віральності завдяки механізму поширення інформації через друзів (користувачі бачать, куди йдуть їхні знайомі).

Основними недоліками є:

- алгоритмічна стрічка, яка часто приховує локальні події малого масштабу на користь популярних або рекламних заходів;

- складність пошуку подій поза межами власного «соціального графа» (кола друзів та підписок);
- інтерфейс, перевантажений рекламою та стороннім контентом;
- загальна тенденція до зниження популярності платформи Facebook серед молодіжної аудиторії.



Рисунок 1.2 – Програмне забезпечення Facebook Events

1.2.3 Сервіс Eventbrite

Платформа, що спеціалізується на управлінні квитками, реєстрацією відвідувачів та організацією масштабних заходів (рис. 1.3) [10].

Перевагами сервісу є:

- потужний функціонал для продажу квитків та обробки платежів;
- детальна аналітика відвідуваності та інструменти для маркетингу;
- надійність роботи при високих навантаженнях.

До недоліків можна віднести:

- чітку орієнтацію на конференції, семінари та комерційні заходи;

- формалізований та складний процес створення події, що є надмірним для організації простих активностей (наприклад, «прибирання в парку» чи «гра у футбол»);
- високі комісійні збори при продажу квитків.

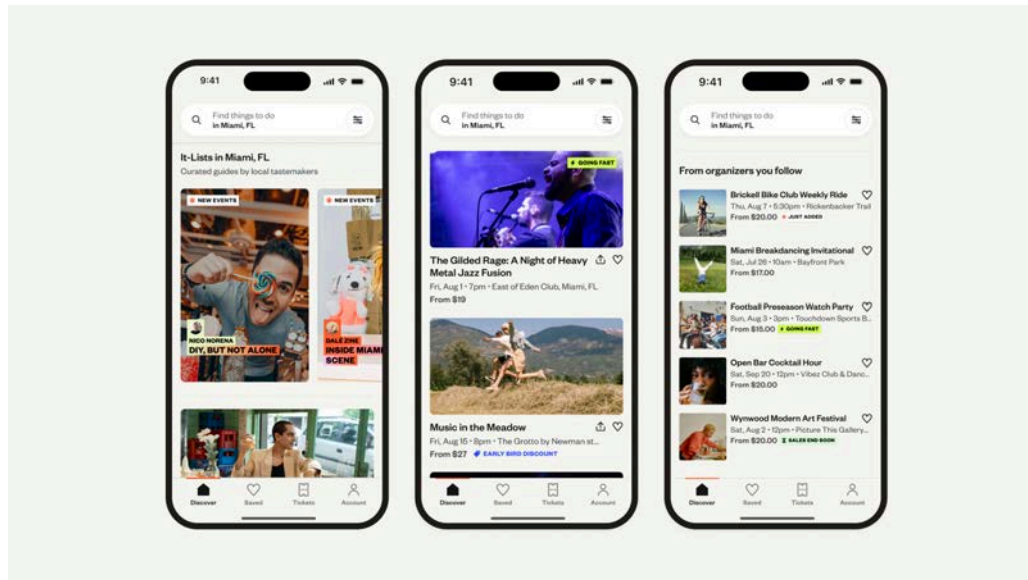


Рисунок 1.3 – Програмне забезпечення Eventbrite

1.2.4 Локальні Telegram-канали та чати

Популярний спосіб координації у містах та районах, що базується на використанні месенджера Telegram (рис. 1.4).

До переваг такого підходу належать:

- висока швидкість поширення інформації серед підписників;
 - глибоке занурення в локальний контекст та відсутність алгоритмічної фільтрації;
 - простота використання та низький поріг входу.
- Суттєвими недоліками є:
- повна відсутність структурування даних (події губляться у стрічці серед новин, обговорень та реклами);

- неможливість фільтрації контенту за категоріями або видами подій;
- відсутність інструментів для відображення подій на карті;
- складність пошуку інформації про подію через певний час після її публікації.



Рисунок 1.4 – Програмне забезпечення Telegram

Для систематизації результатів аналізу та виявлення функціональних розривів на ринку існуючих рішень було складено порівняльну характеристику розглянутих аналогів. Ключовими критеріями порівняння обрано: наявність географічної прив'язки, фінансові умови для організаторів, рівень структурованості даних, тип подій на які орієнтується платформа, а

також можливість пошуку подій поблизу користувача. Результати порівняння наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих платформ для організації подій

Критерій порівняння	Meetup	Facebook Events	Eventbrite	Telegram
Відображення на карті	Присутнє (обмежене)	Присутнє	Відсутнє (список)	Відсутнє
Поріг входу для організатора	Високий (платна підписка)	Середній (вимагає профіль/сторінку)	Високий (складна реєстрація)	Низький (простота створення)
Вартість використання	Платна для організаторів	Безкоштовно (є реклама)	Комісія з квитків	Безкоштовно
Структурованість даних	Висока	Середня	Висока	Низька (стрічка повідомлень)
Тип подій	Регулярні спільноти	Соціальні зв'язки / Реклама	Конференції / Квитки	Будь-які / Новини
Пошук "поруч зі мною"	Є	Ускладнений алгоритмами	Обмежений містом	Відсутній

Із порівняння, жодна з існуючих платформ не поєднує в собі низький поріг входу, безкоштовність для організаторів некомерційних подій та зручну навігацію на карті. Telegram забезпечує швидкість, але не має структури та карти. Спеціалізовані сервіси як Meetup та Eventbrite створюють фінансові та організаційні бар'єри для простих користувачів. Це підтверджує доцільність розробки власної системи, яка заповнить цю нішу.

1.3 Висновки за розділом 1

Проведений у попередніх підрозділах порівняльний аналіз існуючих інструментів для організації подій дозволив виявити системні недоліки поточного стану цифрового середовища у сфері івент-менеджменту. Жоден із розглянутих аналогів не забезпечує комплексного вирішення проблеми локальної соціальної координації, створюючи бар'єри як для організаторів, так і для учасників.

На основі аналізу можна сформулювати ключові проблеми, які вирішуватиме розроблювана система.

Першою є проблема фрагментації інформаційного простору:

- користувачі змушені використовувати різні платформи для різних типів подій (месенджери — для зустрічей з друзями, квиткові оператори — для концертів, соцмережі — для афіш);
- відсутня єдина точка входу для пошуку активностей за географічним принципом («що відбувається поруч зі мною»);
- інформація у чатах та групах не є структурованою та швидко втрачає актуальність через нашарування нових повідомлень.

Наступною є проблема високого порогу входу для організаторів:

- існуючі спеціалізовані платформи вимагають складної реєстрації, налаштування профілів організатора або оплати підписки, що є надмірним для організації простих, некомерційних подій (наприклад, спільне тренування, настільні ігри чи суботник);
- соціальні мережі алгоритмічно обмежують охоплення публікацій, вимагаючи від організатора навичок SMM або бюджету на рекламу для залучення навіть локальної аудиторії.

Також проблемою є неактуальність даних та слабка координація:

- у статичних афішах або постах у соцмережах складно оперативно повідомити про зміну локації чи часу зустрічі всім зацікавленим;

- відсутні інструменти для відстеження реального місцезнаходження події в динаміці (наприклад, якщо група переміщується парком під час прибирання).

Враховуючи вищезазначене, доцільною є розробка спеціалізованої програмної системи, яка займе вільну нішу між простими месенджерами та складними професійними івент-платформами.

Система повинна базуватися на таких ключових принципах:

- геоцентричність: основним інтерфейсом взаємодії має бути інтерактивна карта, що дозволяє візуалізувати події у просторі та фільтрувати їх за радіусом доступності;
- низький поріг входу: спрощена процедура створення події (мінімальна кількість полів, автоматичне визначення координат), що дозволяє організувати зустріч за лічені хвилини;
- універсальність контенту: архітектура системи має підтримувати будь-які типи подій — від приватних зустрічей до масових заходів, із гнучкою системою категорій;
- актуальність у реальному часі: використання сучасних клієнт-серверних технологій для миттєвої синхронізації даних між усіма учасниками.

Таким чином, розробка власної системи є не лише інженерним завданням створення програмного продукту, але й науково-прикладним дослідженням методів покращення соціальної взаємодії у містах через використання мобільних технологій.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Обґрунтування вибору мови програмування та середовища виконання системи

Для розробки серверної частини обрано середовище виконання Node.js. Це кросплатформне середовище з відкритим кодом, яке дозволяє виконувати код JavaScript поза межами браузера.

Основні причини вибору:

- неблокуюча модель вводу/виводу: завдяки подійній моделі [11], Node.js ефективно обробляє велику кількість одночасних з'єднань, що є критичним для соціальних застосунків з високою активністю користувачів [12];
- єдина мова програмування: використання JavaScript чи TypeScript як на клієнті, так і на сервері спрощує розробку, дозволяє перевикористовувати код та знижує когнітивне навантаження на розробника;
- висока продуктивність: рушій V8 від Google компілює JavaScript безпосередньо в машинний код, забезпечуючи високу швидкість виконання операцій.

Окрім вибору середовища виконання, критично важливим етапом проєктування було визначення основної мови програмування. Для реалізації як серверної, так і клієнтської частин системи було обрано TypeScript.

TypeScript є типізованою надбудовою над мовою JavaScript [13], що компілюється у чистий JavaScript, забезпечуючи сумісність з існуючими бібліотеками [14]. Вибір саме цієї технології обумовлений наступними факторами:

- статична типізація: на відміну від JavaScript, TypeScript дозволяє явно визначати типи даних. Це дає змогу виявляти потенційні помилки ще на етапі написання коду, а не під час виконання програми, що значно підвищує надійність системи;

- єдиний технологічний стек: використання TypeScript для написання серверної та клієнтської частини дозволяє створити спільну базу інтерфейсів та типів даних. Це гарантує, що структура даних, яку відправляє сервер, точно відповідає тій, яку очікує отримати мобільний застосунок, мінімізуючи ризики помилок інтеграції;
- підтримка інструментарію: TypeScript забезпечує високий рівень підтримки в сучасних середовищах розробки. Функції автодоповнення, швидкої навігації по коду та безпечного рефакторингу суттєво пришвидшують процес розробки;
- читабельність та масштабованість: типи виступають у ролі самодокументації коду. Чітко описані інтерфейси дозволяють швидше орієнтуватися в архітектурі проєкту, що спрощує його подальшу підтримку та розширення функціоналу.

Таким чином, використання TypeScript є стандартом сучасної промислової розробки, який забезпечує баланс між гнучкістю JavaScript та надійністю статично типізованих мов.

2.2 Вибір системи керування базами даних

При проєктуванні системи організації подій ключовим питанням є вибір підходу до зберігання даних. Основними кандидатами на роль СКБД були реляційна база даних PostgreSQL та документо-орієнтована NoSQL база даних MongoDB.

MongoDB є популярним вибором для стартапів завдяки своїй гнучкості [15], що дозволяє швидко змінювати структуру даних без необхідності виконання міграцій. Вона демонструє високу продуктивність при операціях запису та горизонтальному масштабуванні. Проте, специфіка предметної області проєкту передбачає наявність чітких, структурованих

зв'язків між сутностями: один користувач може створити багато подій, одна подія має багато учасників, учасники мають ролі тощо.

Реалізація таких зв'язків у MongoDB вимагає або дублювання даних, що ускладнює їх оновлення, або виконання декількох запитів для отримання пов'язаних даних, що нівелює переваги у швидкості. Натомість PostgreSQL, будучи об'єктно-реляційною системою, нативно підтримує складні зв'язки [16] та гарантує цілісність даних, що є критичним для функцій реєстрації та управління учасниками.

Ключовим аргументом на користь PostgreSQL стала наявність розширення PostGIS, яке є стандартом де-факто для геоінформаційних систем. Хоча MongoDB також підтримує геопросторові індекси, PostGIS надає значно ширший набір функцій для складних геометричних обчислень [17] безпосередньо в базі даних.

Порівняльна характеристика розглянутих систем наведена у таблиці 2.1.

Таблиця 2.1 – Порівняльний аналіз систем керування базами даних

Критерій порівняння	PostgreSQL (з PostGIS)	MongoDB
Тип моделі даних	Реляційна (таблиці)	Документно-орієнтована (JSON-подібні документи)
Підтримка зв'язків	Повна, нативна підтримка складних запитів	Обмежена, часто вимагає дублювання даних
Геопросторові можливості	Розширені (PostGIS): підтримка складних типів даних та операцій	Базові: пошук у радіусі, перетин полігонів
Гарантія цілісності	Так, повна відповідність	Підтримується на рівні документа (з v4.0 транзакції додані, але складніші)
Гнучкість схеми	Менш гнучка (вимагає міграцій), але є підтримка JSONB	Висока, схема визначається даними

Продовження таблиці 2.1

Критерій порівняння	PostgreSQL (з PostGIS)	MongoDB
Сфера застосування	Системи зі складними зв'язками та високими вимогами до надійності	Системи з великими обсягами неструктурованих даних, кешування, логи

Враховуючи необхідність зберігання структурованих даних про користувачів та події, а також складних зв'язків між ними, було обрано PostgreSQL.

Для забезпечення ефективної та безпечної взаємодії серверного застосунку на Node.js з реляційною базою даних PostgreSQL доцільно використовувати технологію ORM [18]. Це дозволяє працювати із записами бази даних як з об'єктами в коді, уникаючи написання "сирих" SQL-запитів вручну, що підвищує швидкість розробки та захищеність системи.

В екосистемі Node.js/TypeScript найпоширенішими рішеннями є Sequelize, TypeORM та Prisma. Для обґрунтування вибору було проведено їх порівняльний аналіз.

Sequelize — одна з найстаріших ORM для Node.js. Вона має велику спільноту та документацію. Проте, її архітектура застаріла: підтримка TypeScript реалізована не нативному рівні, що часто призводить до помилок типізації, а продуктивність складних запитів часто є низькою через надмірну абстракцію.

TypeORM — потужна ORM, що тривалий час була стандартом для TypeScript-проектів. Вона використовує декоратори та класи для опису моделей. Хоча TypeORM надає широкі можливості, її недоліком є складність налаштування міграцій та певна непередбачуваність поведінки при роботі зі складними відношеннями між таблицями.

Порівняльна характеристика розглядаємих ORM наведена у таблиці 2.2.

Таблиця 2.2 – Порівняльний аналіз ORM для Node.js

Критерій порівняння	Prisma	TypeORM	Sequelize
Підтримка TypeScript	Найвища	Висока	Середня
Підхід до моделювання	Schema-first (декларативний файл схеми)	Code-first (класи та декоратори)	Code-first (визначення моделей у коді)
Продуктивність у запитів	Висока (оптимізований рушій Query Engine на Rust)	Середня	Середня / Низька (залежить від складності)
Керування міграціями	Вбудований інструмент Prisma Migrate (автоматичний)	Вбудований CLI (вимагає ручної перевірки)	Окремий інструмент (складніший у налаштуванні)
Безпека запитів	Повний захист від SQL-ін'єкцій на рівні клієнта	Захист є, але можливі вразливості при використанні QueryBuilder	Базовий захист
Зручність (Developer Experience)	Висока (автодоповнення коду в IDE)	Середня	Низька

На основі аналізу для розробки системи було обрано Prisma ORM. Ключовими факторами вибору стали:

- повна типобезпека: оскільки проєкт розробляється на TypeScript, можливість Prisma автоматично генерувати типи на основі схеми бази даних дозволяє виявляти помилки ще на етапі написання коду, а не під час виконання;

- інтуїтивно зрозумілий API: Prisma Client надає зручний інтерфейс для виконання CRUD-операцій та складних вибірок з вкладеними даними, що пришвидшує розробку;
- prisma Studio: наявність вбудованого графічного інтерфейсу для перегляду та редагування даних у базі, що значно спрощує процес відлагодження;
- продуктивність: використання рушія запитів, написаного на мові Rust, забезпечує високу швидкість обробки запитів та ефективне використання ресурсів сервера.

2.3 Вибір фреймворку для мобільної розробки

Оскільки цільова аудиторія користується пристроями на базі як iOS, так і Android, розробка двох окремих нативних застосунків є економічно недоцільною в умовах обмежених ресурсів [19]. Тому було розглянуто кросплатформені рішення, що дозволяють оптимізувати процес розробки [20]: React Native, Flutter та Ionic.

Ionic використовує веб-технології (HTML, CSS, JS) і фактично відображає сайт у WebView. Це найпростіший шлях, але він часто програє у продуктивності та відчутті "нативності" інтерфейсу.

Flutter, розроблений Google, використовує мову Dart і власний рушій рендерингу Skia. Це забезпечує чудову продуктивність, але вимагає вивчення нової мови програмування та має меншу екосистему бібліотек порівняно з JavaScript.

React Native дозволяє використовувати одну кодову базу [21] для створення застосунків під обидві платформи, при цьому інтерфейс відображається за допомогою нативних компонентів мобільної ОС [22]. Це забезпечує баланс між швидкістю розробки та якістю кінцевого продукту.

Наявність функції «Fast Refresh» дозволяє миттєво бачити зміни в коді, що значно прискорює процес розробки.

Порівняння характеристик розглянутих фреймворків наведено у таблиці 2.3.

Таблиця 2.3 – Порівняння популярних мобільних фреймворків

Критерій порівняння	React Native	Flutter	Ionic
Мова програмування	JavaScript / TypeScript	Dart	HTML, CSS, JavaScript
Принцип рендерингу	Нативні компоненти через міст (Bridge)	Власний графічний рушій (Skia)	WebView (браузерний двигун)
Продуктивність	Висока, близька до нативної	Дуже висока	Середня (залежить від WebView)
Розмір екосистеми	Величезна (NPM), багато готових бібліотек	Швидко зростаюча, але менша за JS	Велика, орієнтована на веб
Крива навчання	Низька (для тих, хто знає React/JS)	Середня (потрібно вивчити Dart)	Низька (для веб-розробників)

На основі проведеного аналізу для розробки клієнтської частини системи обрано фреймворк React Native через використання єдиної мови програмування з серверною частиною та наявність розвиненої екосистеми картографічних бібліотек.

2.4 Діаграма прецедентів

Для формалізації функціональних вимог до системи та визначення меж її можливостей було розроблено діаграму прецедентів. Ця модель дозволяє візуалізувати взаємодію між зовнішніми сутностями (акторами) та функціональними модулями системи (прецедентами).

Основним актором у системі виступає Користувач. В залежності від контексту взаємодії, Користувач може набувати однієї з двох динамічних ролей:

- Організатор — ініціатор події, який створює захід, визначає його локацію, час та опис, а також керує списком учасників;
- Учасник — користувач, який шукає події, переглядає деталі та подає заявку на участь.

Варто зазначити, що в межах одного облікового запису користувач може одночасно виступати організатором власних подій та учасником інших.

Основні сценарії використання (прецеденти), реалізовані в системі:

- реєстрація та авторизація: є обов'язковою умовою для створення подій та приєднання до них (реалізує захист від спаму та ідентифікацію особи);
- перегляд карти подій: базовий сценарій, доступний одразу після запуску застосунку. Включає автоматичне визначення геолокації користувача;
- створення події: включає введення текстового опису, вибір категорії та встановлення маркера на карті. Цей прецедент включає обов'язкову перевірку авторизації;
- пошук та фільтрація: дозволяє відсіювати події за радіусом, датою або тематикою.

Взаємозв'язки між актором та прецедентами зображено на рисунку 2.1.

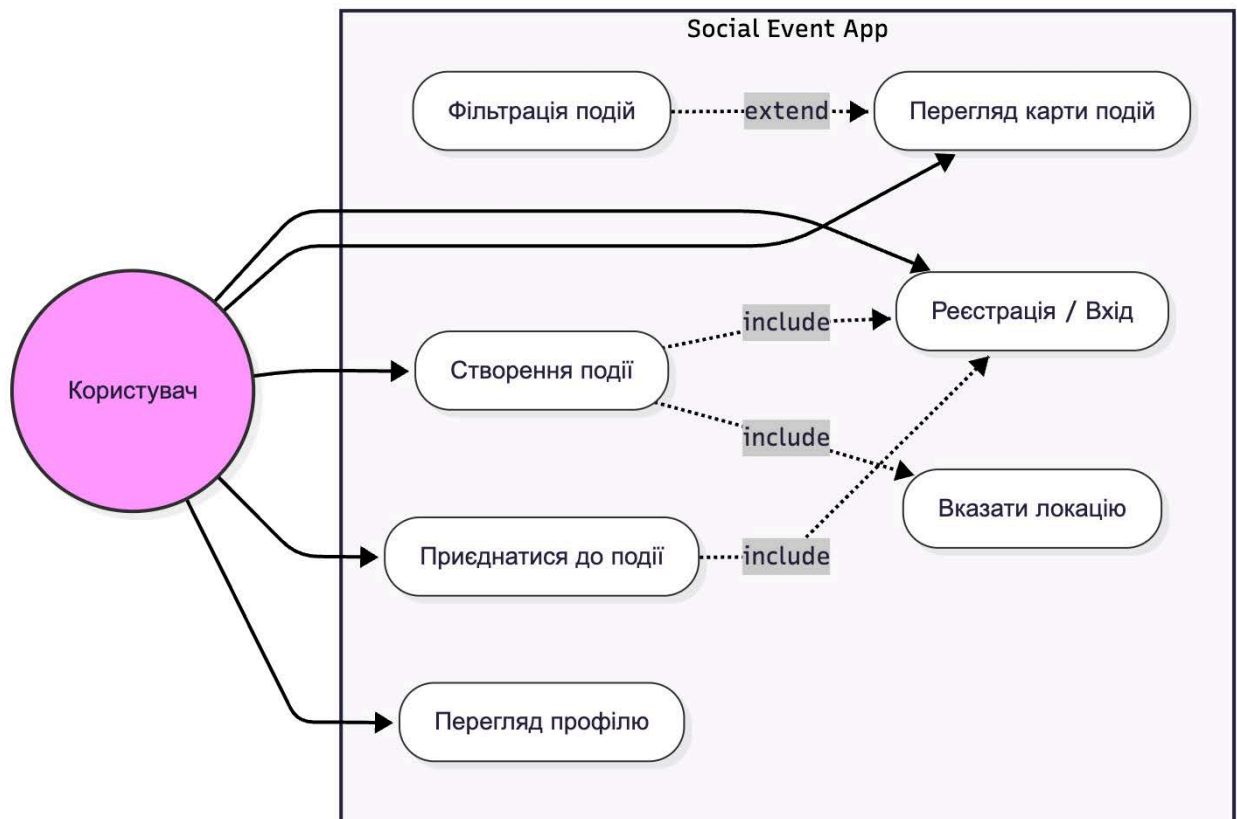


Рисунок 2.1 – Діаграма прецедентів системи

На діаграмі використано відношення <<include>> для позначення обов'язкових дій (наприклад, створення події включає в себе авторизацію) та <<extend>> для опціональних дій (наприклад, фільтрація розширює можливості перегляду карти). Така деталізація дозволяє чітко розуміти логіку роботи майбутнього застосунку.

2.5 Висновки за розділом 2

У другому розділі було проведено обґрунтування вибору архітектурних рішень та інструментарію для реалізації системи організації суспільних подій.

На основі аналізу вимог до системи було обрано клієнт-серверну архітектуру з використанням REST API.

За результатами порівняльного аналізу визначено оптимальний технологічний стек:

- серверна частина: середовище Node.js, яке забезпечує високу продуктивність при обробці численних одночасних запитів завдяки неблокуючій моделі вводу/виводу;
- база даних: обрано реляційну СКБД PostgreSQL з розширенням PostGIS. Це рішення гарантує цілісність складних зв'язків між даними та надає потужний інструментарій для виконання ефективних геопросторових запитів;
- для клієнтської частини обрано фреймворк React Native, що дозволяє створити кросплатформенний мобільний застосунок з нативною продуктивністю та спільним кодом для iOS та Android, використовуючи єдину мову програмування TypeScript для всієї системи.

Також було виконано моделювання програмного забезпечення: розроблено діаграму прецедентів, яка формалізує сценарії взаємодії користувача із системою. Обрані рішення створюють надійний фундамент для подальшої програмної реалізації системи.

3 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

3.1 Аналіз архітектурних рішень для високонавантажених систем організації подій

При розробці сучасних програмних систем, що передбачають взаємодію великої кількості користувачів у реальному часі з прив'язкою до геолокації, критично важливим є вибір правильної архітектури. Традиційні монолітні підходи часто не справляються з вимогами до масштабованості та гнучкості, тому для розроблюваної системи було обрано клієнт-серверну архітектуру.

Основним патерном взаємодії між клієнтською частиною та сервером обрано архітектурний стиль REST. Використання RESTful API дозволяє:

- забезпечити чітке розділення інтерфейсу користувача та бізнес-логіки;
- зробити систему масштабованою, оскільки сервер не зберігає стан сесії клієнта між запитами;
- використовувати стандартні методи протоколу HTTP (GET, POST, PUT, DELETE) [23] для управління ресурсами системи, що відповідає принципам архітектурного стилю REST [24].

Для роботи з геопросторовими даними, що є ключовим функціоналом для пошуку подій «поруч зі мною», необхідно використовувати спеціалізовані розширення баз даних. Звичайні реляційні запити є неефективними для розрахунку відстаней на сфері. Тому архітектура повинна включати підтримку геоінформаційних систем на рівні СКБД, що дозволяє виконувати просторові запити з високою швидкістю.

Для забезпечення актуальності даних (наприклад, миттєве повідомлення про зміну часу події) доцільно використовувати не лише класичну модель «запит-відповідь», але й механізми асинхронної доставки повідомлень або технологію WebSockets для підтримки постійного з'єднання.

3.2 Проектування схеми бази даних

Розробка структури бази даних базувалася на вимогах до нормалізації (приведення до третьої нормальної форми — 3NF) для уникнення надлишковості даних та забезпечення їх цілісності.

У якості системи керування базами даних обрано PostgreSQL з розширенням PostGIS. Це зумовило використання специфічних типів даних для збереження геопросторової інформації.

Інфологічна модель системи складається з наступних ключових сутностей:

- User (Користувач) — базова сутність, що містить облікові дані та профіль учасника системи;
- Event (Подія) — центральна сутність системи, яка описує параметри заходу (назва, опис, час проведення) та його географічне розташування;
- Category (Категорія) — довідкова сутність для класифікації подій (наприклад, «Спорт», «Освіта», «Розваги»);
- Participant (Учасник) — асоціативна сутність, що реалізує зв'язок "багато-до-багатьох" між користувачами та подіями, а також визначає роль користувача в конкретній події (організатор, гість).

Структура зв'язків між сутностями зображена на рисунку 3.1.

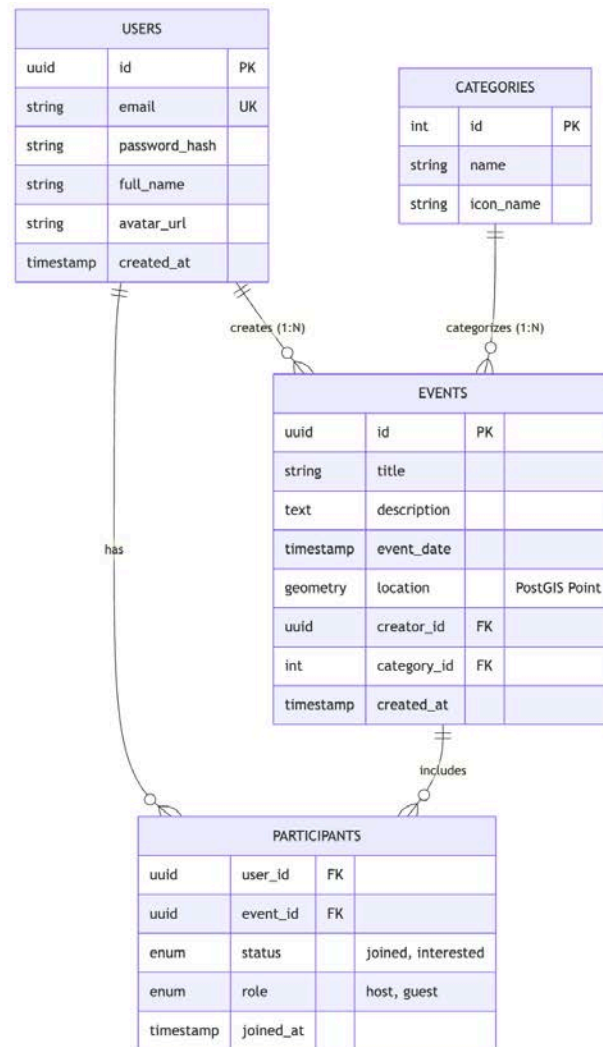


Рисунок 3.1 – ER-діаграма бази даних системи

Для фізичної реалізації моделі у середовищі PostgreSQL було розроблено структуру таблиць, наведену нижче. Особливістю схеми є використання типу даних `GEOMETRY(Point, 4326)` для атрибуту `location` у таблиці `events`, що дозволяє виконувати індексацію та швидкий пошук об'єктів на мапі світу (SRID 4326 відповідає стандарту WGS 84, що використовується в GPS).

Для реалізації було використано 3 основні таблиці. Нижче наведено опис їх атрибутного складу.

Таблиця `users`:

- `id` (UUID, PK) — унікальний ідентифікатор користувача;

- email (VARCHAR, Unique) — електронна пошта для авторизації;
- password_hash (VARCHAR) — хеш пароля (з міркувань безпеки паролі не зберігаються у відкритому вигляді);

- full_name (VARCHAR) — ім'я користувача для відображення;
- created_at (TIMESTAMP) — дата реєстрації.

Таблиця events:

- id (UUID, PK) — унікальний ідентифікатор події;
- title (VARCHAR) — заголовок;
- description (TEXT) — детальний опис;
- event_date (TIMESTAMP) — час початку;
- location (GEOMETRY) — геопросторові координати точки проведення;

- creator_id (UUID, FK) — посилання на організатора;
- category_id (INT, FK) — посилання на категорію.

Таблиця participants:

- user_id (UUID, FK) — посилання на користувача;
- event_id (UUID, FK) — посилання на подію;
- status (ENUM) — статус участі ('joined', 'declined', 'interested');
- role (ENUM) — роль ('host', 'guest').

Така структура забезпечує гнучкість системи та можливість масштабування. Використання зовнішніх ключів гарантує посилальну цілісність даних: при видаленні користувача система може автоматично видалити або анонімізувати його події.

3.3 Структура програмних компонентів

Архітектура серверного застосунку побудована за принципом "Controller-Service-Repository" [25], що забезпечує чіткий поділ відповідальності та полегшує масштабування системи [26]:

- **Controllers** (Контролери): відповідають за обробку вхідних HTTP-запитів, валідацію вхідних даних та формування HTTP-відповідей;
- **Services** (Сервіси): містять бізнес-логіку застосунку (наприклад, логіка перевірки доступності події або розрахунок відстані);
- **Repositories** (Репозиторії): відповідають за безпосередню взаємодію з базою даних PostgreSQL.

Такий підхід дозволяє легко тестувати окремі компоненти та змінювати логіку без впливу на інші частини системи.

3.4 Висновки за розділом 3

У третьому розділі спроектовано архітектуру програмного забезпечення, що базується на клієнт-серверній моделі взаємодії.

Основні результати проєктування:

- обґрунтовано використання RESTful API як стандарту обміну даними, що забезпечує масштабованість системи, незалежність клієнтської та серверної частин, а також стандартизацію інтерфейсів взаємодії;
- розроблено інфологічну та даталогічну схеми бази даних. Структура БД приведена до третьої нормальної форми, що мінімізує надлишковість даних. Визначено ключові сутності та типи зв'язків між ними. Особливу увагу приділено використанню типу даних GEOMETRY для зберігання координат, що є основою геоцентричного функціоналу системи;
- для серверної частини застосовано архітектурний патерн Controller-Service-Repository. Такий поділ відповідальності (відокремлення обробки HTTP-запитів, бізнес-логіки та доступу до даних) підвищує надійність коду, спрощує його тестування та подальшу підтримку.

Запропонована архітектура повністю задовольняє вимогам до високонавантажених систем соціальної координації та створює надійний фундамент для етапу програмної реалізації.

4 РОЗРОБКА ПРОГРАМИ

У цьому розділі описано технічні вимоги, процедуру налаштування середовища розробки, процес розгортання серверної та клієнтської частин системи, а також методику тестування програмного забезпечення.

4.1 Реалізація серверної частини системи

Серверна частина системи реалізована як RESTful API на базі платформи Node.js з використанням фреймворку Express.js. Вибір цієї технології дозволив побудувати масштабовану та швидку систему обробки запитів.

4.1.1 Розробка API та логіки авторизації

Безпека даних користувачів забезпечується впровадженням механізму авторизації на основі токенів з використанням стандарту JWT.

Процес аутентифікації реалізовано наступним чином:

- клієнт надсилає запит на ендпоінт `/api/auth/login` з email та паролем;
- сервер перевіряє наявність користувача в БД та валідність пароля (паролі зберігаються у вигляді хешу, згенерованого алгоритмом bcrypt);
- у разі успішної перевірки сервер генерує Access Token (короткостроковий) та Refresh Token (довгостроковий) і повертає їх клієнту;
- Для захисту приватних маршрутів реалізовано Middleware (проміжне програмне забезпечення). Воно перехоплює запит, перевіряє наявність токена в заголовку Authorization та декодує його. Якщо токен невалідний або прострочений, сервер повертає помилку 401 Unauthorized.

Основні реалізовані маршрути API наведено в таблиці 4.1.

Таблиця 4.1 – Основні ендпоінти REST API системи

Метод	Маршрут (Endpoint)	Опис	Рівень доступу
POST	/api/auth/register	Реєстрація нового користувача	Публічний
POST	/api/auth/login	Вхід у систему (отримання токенів)	Публічний
GET	/api/events	Отримання списку подій (з фільтрами)	Публічний
POST	/api/events	Створення нової події	Приватний
GET	/api/events/:id	Отримання деталей конкретної події	Публічний
POST	/api/events/:id/join	Приєднатися до події	Приватний

4.1.2 Реалізація механізму створення та управління подіями

Ключовою функцією бекенду є робота з геопросторовими даними. Для реалізації функції "пошук подій поруч" використано можливості розширення PostGIS для PostgreSQL.

При створенні події клієнт надсилає координати. Сервер перетворює ці дані у формат WKT (Well-Known Text) та зберігає у колонці з типом GEOMETRY(Point, 4326).

Для вибірки подій у заданому радіусі використовується SQL-запит із функцією ST_DWithin. Логіка запиту виглядає наступним чином:

- отримати координати користувача;
- відфільтрувати записи в таблиці events, де відстань між точкою події та точкою користувача менша за заданий радіус. Використання просторового індексу (GIST index) дозволяє виконувати цю операцію за мілісекунди навіть при великій кількості записів.

Також реалізовано механізм завантаження зображень для подій. Файли завантажуються на хмарне сховище, а в базу даних записується лише посилення на зображення.

4.1.3 Реалізація системи рекомендацій подій

Для підвищення залученості користувачів та спрощення пошуку релевантного контенту в системі реалізовано механізм рекомендацій. Оскільки на початкових етапах запуску системи накопиченої історії дій користувачів може бути недостатньо для використання методів колаборативної фільтрації, було обрано підхід контент-орієнтованої фільтрації у поєднанні з гео-ранжуванням [27].

Алгоритм рекомендацій працює на стороні сервера і виконує розрахунок «індексу релевантності» для активних подій. Формування стрічки рекомендованих подій базується на трьох ключових факторах:

- географічна близькість: використовуючи можливості PostGIS, система розраховує відстань від користувача до місця проведення події. Чим менша відстань, тим вищий пріоритет отримує подія у видачі;
- категорійна відповідність: система аналізує інтереси користувача (виявлені на основі попередніх приєднань до подій) та порівнює їх з категорією події. Збіг категорій додає ваговий коефіцієнт до загального рейтингу події;
- часовий фактор: події, що розпочинаються у найближчий час (наприклад, протягом найближчих 24 годин), отримують вищий пріоритет порівняно з тими, що заплановані на віддалені дати.

Технічна реалізація цього механізму виконана шляхом побудови оптимізованого SQL-запиту за допомогою Query Builder в середовищі Node.js. Це дозволяє виконувати сортування та фільтрацію даних безпосередньо на рівні СКБД PostgreSQL, що забезпечує високу швидкість

формування персоналізованої стрічки навіть при значній кількості подій у базі даних.

Впровадження запропонованого гібридного механізму фільтрації дозволяє суттєво підвищити ефективність пошуку інформації [28]. Математично це виражається у звуженні простору пошуку S до підмножини релевантних подій $|S_{rel}|$, де $|S_{rel}| \ll |S|$. Завдяки попередній фільтрації за радіусом відсіюється до 80-90% нерелевантних подій, що знаходяться поза зоною досяжності користувача. Наступний етап ранжування за категоріями та часом дозволяє представити користувачеві найбільш вірогідні варіанти у перших рядках видачі. Такий підхід мінімізує когнітивне навантаження на користувача та скорочує час, необхідний для прийняття рішення про участь у події, трансформуючи процес із активного пошуку бажаного контенту в легкий вибір із готових результатів, що точно задовольняють інтереси користувача.

4.2 Реалізація клієнтської частини системи

Клієнтська частина розроблена з використанням фреймворку React Native. Структура проєкту базується на компонентному підході, де інтерфейс розбито на незалежні, перевикористовувані елементи [29] (кнопки, картки подій, поля вводу), що відповідає сучасним стандартам розробки мобільних застосунків.

4.2.1 Інтерфейс користувача та навігація

Навігація в застосунку реалізована за допомогою бібліотеки react-navigation. Використано комбінацію двох типів навігаторів:

- нижня панель вкладок: забезпечує швидкий доступ до основних розділів: "Карта", "Стрічка", "Створити", "Профіль";

- стекова навігація: використовується для переходів вглиб, наприклад, відкриття екрану "Деталі події" зі стрічки новин.

Для управління станом застосунку (зберігання даних користувача, списку завантажених подій) використано бібліотеку Redux Toolkit. Це дозволяє кешувати завантажені дані та зменшити кількість запитів до сервера при повторному відкритті екранів.

4.2.2 Взаємодія з геолокацією та картами

Основним екраном застосунку є інтерактивна карта. Для її реалізації використано бібліотеку react-native-maps.

Алгоритм роботи з картою:

- при запуску застосунків запитує дозвіл на доступ до геолокації користувача;
- отримані координати записуються у стан застосунку;
- карта центрується на локації користувача;
- відправляється запит на сервер для отримання подій у видимій області карти;
- отримані події відображаються у вигляді маркерів.

Для оптимізації продуктивності при великій кількості маркерів на карті (наприклад, у центрі міста) може бути застосовано кластеризацію (об'єднання близьких точок в один маркер з цифрою).

4.3 Висновки за розділом 4

У четвертому розділі виконано програмну реалізацію компонентів системи відповідно до спроектованої архітектури.

В ході розробки було реалізовано серверну частину, а саме:

- розроблено REST API на базі Express.js, що забезпечує обробку запитів від мобільних клієнтів;
- реалізовано механізм безпечної аутентифікації користувачів з використанням стандартів JWT, що гарантує захист персональних даних та розмежування прав доступу;
- реалізовано логіку геопросторового пошуку з використанням SQL-запитів до PostGIS, що дозволяє ефективно фільтрувати події у заданому радіусі від користувача;
- реалізовано систему рекомендації подій для вищої залученості користувачів.

Також в ході розробки було реалізовано клієнтську частину у вигляді мобільного застосунку, а саме:

- створено інтерфейс користувача на React Native з використанням компонентного підходу;
- налаштовано навігацію в застосунку для зручного переміщення між екранами;
- інтегровано інтерактивні карти та сервіси геолокації, що дозволяє відображати маркери подій та визначати поточне місцезнаходження користувача;
- впроваджено менеджер станів Redux Toolkit для кешування даних та синхронізації стану інтерфейсу з відповідями сервера;

Створений програмний код є модульним, структурованим та готовим до розгортання і тестування.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Призначення та умови виконання системи

Розроблена програмна система призначена для автоматизації процесів створення, пошуку та координації суспільних подій. Система функціонує у клієнт-серверній архітектурі.

5.1.1 Вимоги до апаратного забезпечення сервера

Для розгортання серверної частини та бази даних мінімальні вимоги до апаратного забезпечення є такими:

- процесор: 2 ядра з частотою не менше 2.0 ГГц (рекомендовано архітектуру x86-64);
- оперативна пам'ять: не менше 4 ГБ (для стабільної роботи Node.js та PostgreSQL у контейнерах);
- дисковий простір: не менше 20 ГБ вільного місця, рекомендовано використання SSD рекомендовано для пришвидшення роботи бази даних;
- мережеве з'єднання: стабільний доступ до мережі Інтернет з пропускною здатністю не менше 100 Мбіт/с.

5.1.2 Вимоги до апаратного забезпечення клієнта

Клієнтський застосунок розрахований на роботу на мобільних пристроях користувачів:

- смартфон під управлінням ОС Android версії 7.0 або новіша, або iOS версії 13.0 чи новіша;
- наявність модуля GPS для коректної роботи геолокаційних сервісів;

- підключення до мережі інтернет.

5.1.3 Вимоги до програмного забезпечення розробника

Для розгортання проєкту та внесення змін у вихідний код на локальній машині розробника необхідно мати встановленим наступне ПЗ:

- операційна система: Linux (Ubuntu 20.04+), macOS або Windows 10/11 (з WSL2);
- середовище виконання Node.js версії 18.x LTS або новішої;
- система контейнеризації Docker Engine (версія 20.10+) та Docker Compose;
- система контролю версій Git;
- менеджер пакетів npm або yarn;
- будь яке середовище розробки.

5.2 Підготовка середовища та розгортання

Для спрощення процесу розгортання та забезпечення ідентичності середовищ було використано технологію контейнеризації Docker.

5.2.1 Клонування репозиторію та налаштування змінних середовища

Першим кроком є отримання вихідного коду проєкту. Для цього необхідно скопувати репозиторій проєкту.

Після переходу в директорію проєкту необхідно створити файл конфігурації `.env` у кореневій папці сервера. Цей файл містить чутливі дані та налаштування, які не повинні потрапити до системи контролю версій. Приклад змісту файлу `.env`:

```
PORT=3000
```

```
DATABASE_URL=postgresql://user:password@db:5432/events_db
JWT_SECRET=your_secret_key_here
AWS_ACCESS_KEY_ID=your_aws_key
AWS_SECRET_ACCESS_KEY=your_aws_secret
```

5.2.2 Запуск серверної частини

Серверна інфраструктура описана у файлі `docker-compose.yml`, який оркеструє запуск двох контейнерів: `backend` (API) та `db` (PostgreSQL + PostGIS).

Для запуску системи необхідно виконати команду `docker-compose up --build -d`.

Ця команда виконує наступні дії:

- завантажує необхідні образи Node.js та PostgreSQL;
- збирає образ серверного застосунку, встановлюючи залежності з `package.json`;
- створює віртуальну мережу для взаємодії контейнерів;
- запускає контейнери у фоновому режимі.

Перевірити статус контейнерів можна командою `docker ps`. Успішний запуск означає, що API доступне за адресою `http://localhost:3000`.

5.2.3 Запуск клієнтської частини

Для запуску мобільного застосунку в режимі розробки необхідно відкрити нове вікно терміналу та перейти в папку `mobile`:

- встановити залежності використовуючи команду `npm install`;

- для запуску на Android необхідно мати запущений емулятор Android Studio або підключений реальний пристрій та виконати команду `npm run android`;

- для запуску на iOS необхідно мати запущений емулятор iOS з XCode або підключений реальний пристрій та виконати команду `npm run ios`.

Система Metro Bundler автоматично збере клієнтський JavaScript-код та завантажить його на пристрій.

5.3 Інструкція користувача

У цьому розділі наведено інструкцію користувача (оператора) щодо практичного використання розробленої програмної системи. Описано основні сценарії взаємодії з інтерфейсом мобільного застосунку для виконання завдань пошуку, створення та координації суспільних подій.

Клієнтська частина системи призначена для надання користувачам зручного інструменту доступу до єдиної бази міських подій. Інтерфейс застосунку адаптовано для використання людьми з різним рівнем технічної підготовки.

Для коректної роботи із застосунком користувач повинен забезпечити виконання наступних умов:

- наявність мобільного пристрою: смартфон на базі операційної системи Android версії 7.0 чи новіше або iOS версії 13.0 чи новіше;
- доступ до мережі Інтернет: стабільне з'єднання через Wi-Fi або мобільні дані для завантаження мапи та синхронізації списку подій;
- активація геолокації: надання дозволу застосунку на використання служби GPS для визначення актуального місцезнаходження та коректного відображення подій поблизу.

Робота із системою будується навколо чотирьох основних сценаріїв: авторизація, пошук подій (через карту або стрічку), створення власної події та керування профілем.

5.3.1 Реєстрація та авторизація

При першому запуску застосунку користувач потрапляє на екран привітання. Для доступу до функціоналу необхідно пройти процедуру авторизації.

Для реєстрації користувач повинен ввести адресу електронної пошти та пароль (рис. 5.1). Система перевіряє унікальність пошти та створює новий обліковий запис.

Вже зареєстрований користувач повинен ввести свої облікові дані (рис. 5.2). Після успішної перевірки система завантажує головний екран.

23:59

←

Create Account

Join the community today

Email Address

hello@example.com

Choose Password

Sign Up

By signing up, you agree to our Terms of Service and Privacy Policy.

Already have an account? [Log In](#)

Рисунок 5.1 - Інтерфейс реєстрації користувача

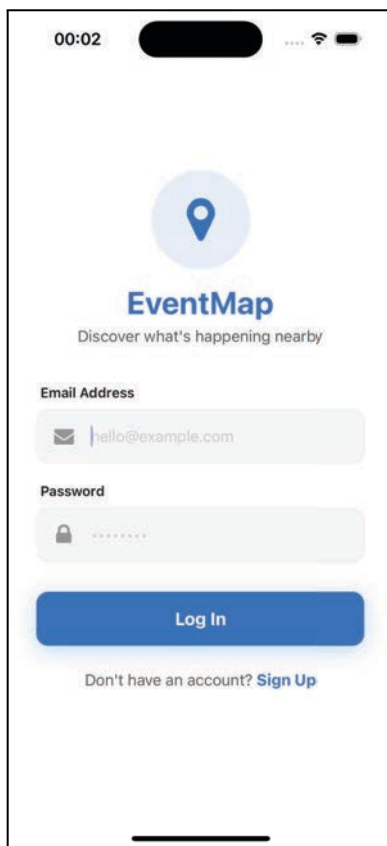


Рисунок 5.2 - Інтерфейс авторизації користувача

5.3.2 Перегляд та пошук подій

Після авторизації користувачеві доступні два режими перегляду контенту, перемикання між якими здійснюється через нижню навігаційну панель.

Режим "Карта" є основним режимом, де події відображаються у вигляді маркерів на географічній мапі міста (рис. 5.3). Карта автоматично центрується на поточній позиції користувача. При натисканні на маркер відкривається коротке прев'ю події. Користувач може змінювати масштаб карти для охоплення більшої території.

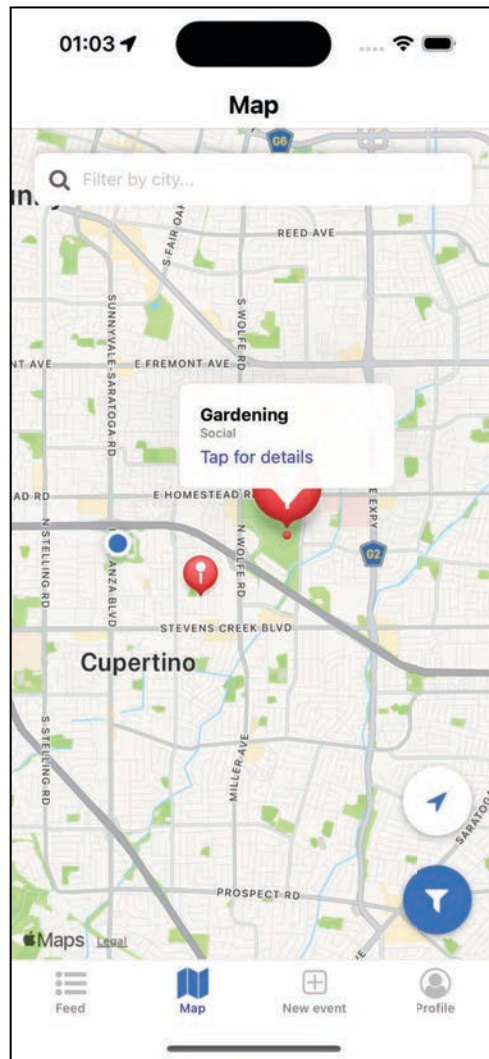


Рисунок 5.3 — Відображення подій на інтерактивній карті

Режим "Стрічка подій" є альтернативним режимом, реалізованим у вигляді вертикального списку карток (рис. 5.4). Цей режим зручний для швидкого перегляду деталей без прив'язки до точної локації.

Події у стрічці відсортовані за часом початку та відстанню до місця події (найближчі — зверху). Кожна картка містить зображення, назву, дату та відстань до користувача. Реалізовано механізм "нескінченної прокрутки" для зручного завантаження великої кількості подій.

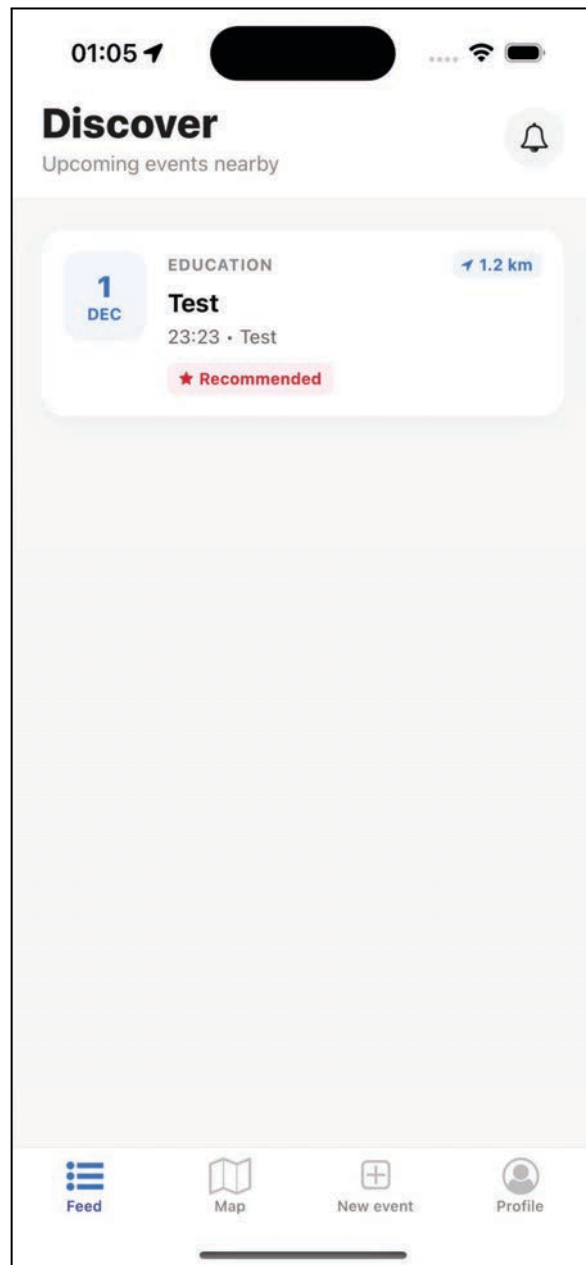


Рисунок 5.4 — Інтерфейс стрічки подій

5.3.3 Створення нової події

Для організації власного заходу користувач натискає кнопку "+" у навігаційній панелі (рис. 5.5). Процес створення складається з наступних кроків:

- введення основних даних: користувач вказує назву події та обирає категорію;

- інтелектуальне автозаповнення опису: для прискорення процесу реалізовано функцію генерації контенту на базі штучного інтелекту. Користувачеві достатньо ввести назву та натиснути кнопку "Згенерувати опис з AI". Система автоматично створить детальний та привабливий текст опису, що відповідає тематиці події. Отриманий текст можна відредагувати вручну;
- вибір часу: встановлення дати та часу початку події;
- вказання локації: користувач може або використати своє поточне місцезнаходження, або вибрати точку на карті вручну;
- підтвердження: після натискання кнопки "Створити", подія миттєво з'являється на карті та у стрічках інших користувачів.

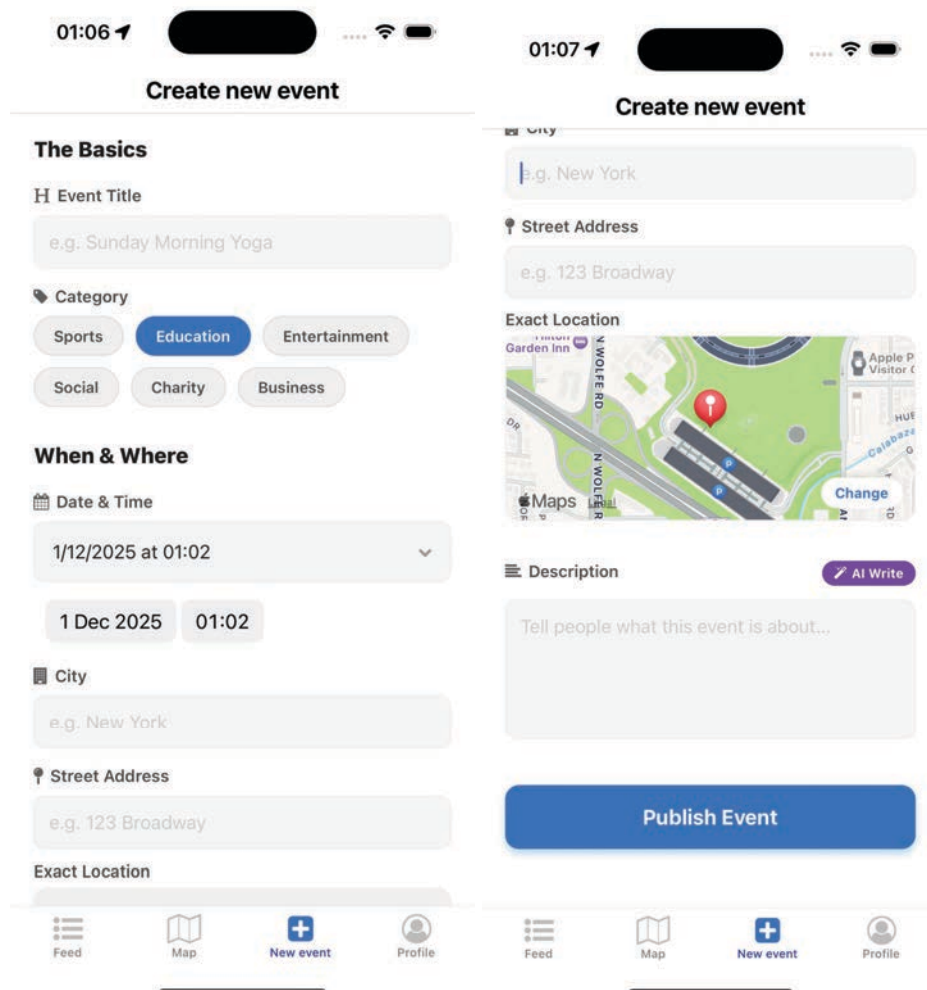


Рисунок 5.5 — Форма створення нової події

5.3.4 Перегляд деталей та участь

Натиснувши на подію на карті або у стрічці, користувач переходить на екран детальної інформації (рис. 5.6). Тут доступні розширений опис, точна адреса та список учасників. Натисканням кнопки "Приєднатися" користувач додає себе до списку учасників. Для організатора на цьому екрані доступна опція видалення або редагування події.

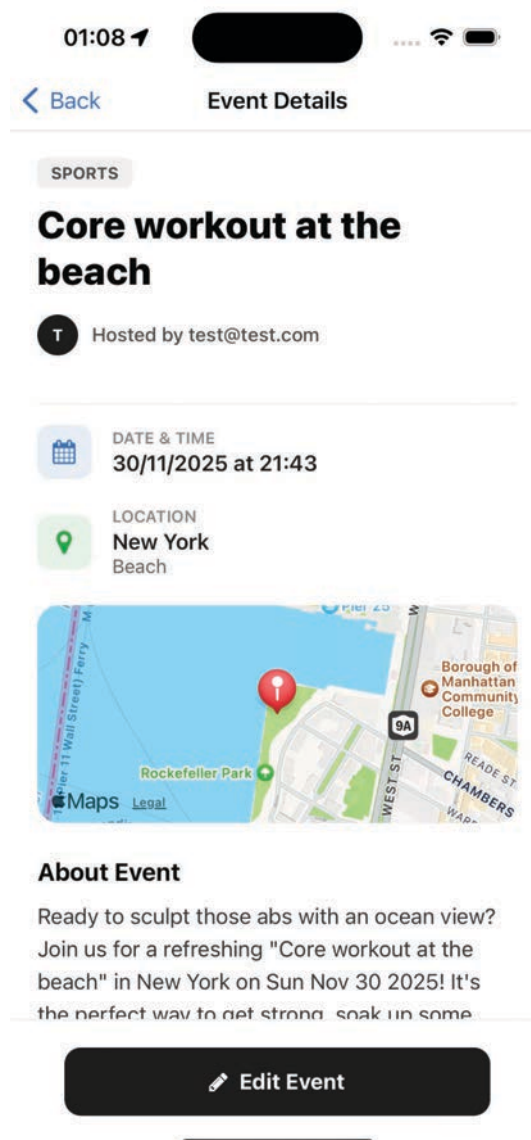


Рисунок 5.6 — Екран детальної інформації про подію

5.3.5 Профіль користувача

Для керування особистою інформацією та перегляду історії активності призначена вкладка «Profile» у нижній навігаційній панелі. Цей екран виконує функцію особистого кабінету користувача.

На сторінці профілю реалізовано перегляд подій, всі заходи розділені на дві категорії — «Створені мною» та «Беру участь». Це дозволяє користувачеві швидко знайти потрібну подію без необхідності шукати її на загальній карті та перевірити свій розклад.

Також доступне редагування профілю користувач має можливість оновити своє відображуване ім'я, завантажити нове фото профілю або змінити пароль для входу в систему.

Інтерфейс профілю спроектовано таким чином, щоб забезпечити швидкий доступ до найбільш важливої інформації про соціальну активність користувача (рис. 5.7).

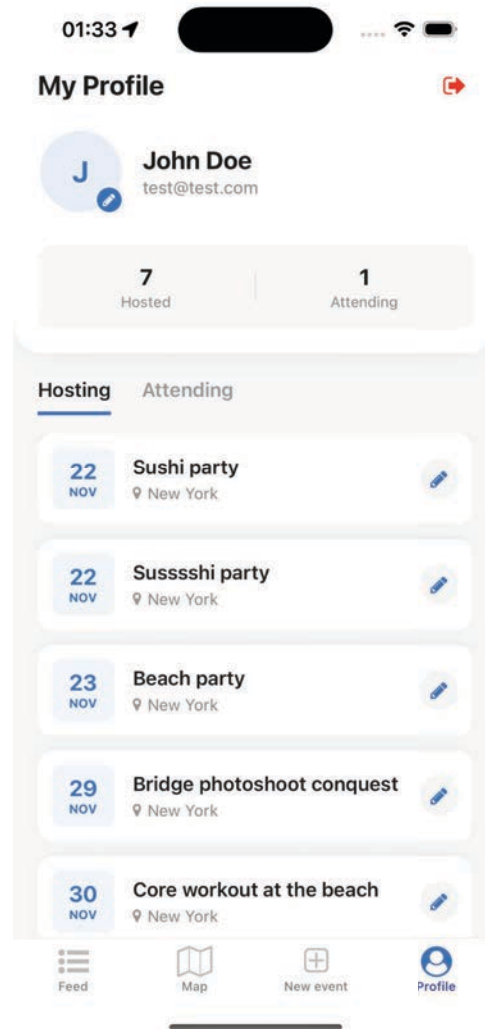


Рисунок 5.7 — Інтерфейс профілю користувача

5.4 Тестування програмного забезпечення

Забезпечення якості розробленого програмного продукту реалізовано через комплексне тестування, яке включає модульне та інтеграційне тестування [30].

5.4.1 Тестування серверної частини

Для тестування API використано фреймворк Jest у поєднанні з бібліотекою Supertest.

Модульні тести покривають бізнес-логіку сервісів, наприклад, коректність розрахунку відстані між координатами, валідація вхідних даних при реєстрації.

Інтеграційні тести перевіряють роботу ендпоінтів API в цілому, включаючи взаємодію з тестовою базою даних.

Результати тестування відображаються у консолі з інформацією про кількість пройдених та кількість успішних тестів (рис.5.8).

```
[vite:dts] Start generate declaration files...
dist/index.js 0.07 kB | gzip: 0.08 kB
[vite:dts] Declaration files built in 356ms.

✓ built in 391ms

> nx run @social-app-mono/api:test

> jest --passWithNoTests=true

PASS @social-app-mono/api tests/events.integration.spec.ts
PASS @social-app-mono/api tests/auth.service.spec.ts

Test Suites: 2 passed, 2 total
Tests:       30 passed, 30 total
Snapshots:   0 total
Time:        0.208 s, estimated 1 s
Ran all test suites.
```

Рисунок 5.8 — Результати тестування

5.4.2 Тестування клієнтської частини

Тестування інтерфейсу мобільного застосунку виконується за допомогою Jest та React Native Testing Library. Основні сценарії перевірки:

- рендеринг компонентів: перевірка, чи коректно відображаються кнопки, поля вводу та списки;
- обробка подій: імітація натискання на кнопки, наприклад, перевірка, чи викликається функція `handleLogin` при натисканні кнопки "Увійти";

- снапшот-тестування: фіксація еталонного вигляду компонентів для запобігання випадковим змінам у верстці.

Також проведено ручне тестування на реальних пристроях (Xiaomi Mi 9 та iPhone 15 Pro) для перевірки роботи геолокації, камери та плавності анімацій.

5.5 Висновки за розділом 5

У п'ятому розділі розглянуто питання практичної експлуатації, розгортання та перевірки якості розробленого програмного забезпечення.

Основні підсумки етапу:

- умови експлуатації: визначено вимоги до апаратного та програмного забезпечення як для серверної сторони, так і для мобільних пристроїв користувачів. Це дозволяє гарантувати стабільну роботу системи у реальних умовах;

- розгортання системи: описано процес підготовки середовища з використанням технології контейнеризації Docker. Створено конфігурації docker-compose, що дозволяють швидко розгорнути серверну інфраструктуру на будь-якому сумісному сервері, мінімізуючи проблеми з залежностями;

- керівництво користувача: розроблено детальну інструкцію, що описує основні сценарії роботи з мобільним застосунком: реєстрацію, пошук подій на карті та у стрічці, створення власних заходів та управління участю. Інтерфейс підтвердив свою інтуїтивність та зручність;

- тестування: проведено комплексне тестування системи. Модульні тести (Unit tests) підтвердили коректність роботи окремих функцій бізнес-логіки, а інтеграційні тести перевірили взаємодію API з базою даних. Тестування клієнтської частини забезпечило відсутність критичних помилок у відображенні інтерфейсу та обробці дій користувача.

Результати експериментального дослідження підтвердили працездатність системи, її відповідність технічному завданню та готовність до використання кінцевими споживачами.

ВИСНОВКИ

У магістерській кваліфікаційній роботі вирішено актуальне науково-прикладне завдання розробки спеціалізованої програмної системи для організації та координації суспільних подій у міському середовищі.

У ході виконання роботи отримано такі основні результати:

- проведено аналіз предметної області та існуючих ринкових рішень. Виявлено проблему фрагментації інформаційного простору, яка призводить до надмірних часових витрат на координацію груп. Порівняльний аналіз підтвердив відсутність на ринку універсального інструменту, що поєднує низький поріг входу з геоцентричною навігацією;
- обґрунтовано вибір архітектурного підходу та технологічного стека. Визначено, що поєднання середовища Node.js для серверної частини, бази даних PostgreSQL з розширенням PostGIS для геопросторових обчислень та фреймворку React Native для мобільного клієнта є оптимальним рішенням, що забезпечує високу продуктивність, масштабованість та кросплатформеність;
- спроектовано та реалізовано серверну частину системи, з використанням алгоритмів просторового пошуку та контент-орієнтованої фільтрації. Система надає RESTful API для керування даними, забезпечує безпечну авторизацію користувачів та виконує ефективний пошук подій у заданому радіусі. Реалізація цих алгоритмів дозволила автоматизувати процес відбору подій, скоротивши простір пошуку для кінцевого користувача;
- розроблено клієнтський мобільний застосунок з інтуїтивним інтерфейсом, що включає інтерактивну карту, стрічку подій та функцію створення заходів з використанням штучного інтелекту для автоматичної генерації описів, що дозволяє зменшити час на організацію;

— проведено комплексне тестування розробленої системи, яке підтвердило її працездатність, стабільність та відповідність сформульованим функціональним вимогам.

Наукова новизна роботи полягає в удосконаленні методів соціальної координації шляхом комбінування геолокаційних сервісів реального часу з персоніфікованим ранжуванням контенту, що, на відміну від існуючих підходів, дозволяє зменшити час пошуку релевантних подій та знизити комунікаційні затримки при організації спонтанних зустрічей.

Практичне значення полягає у створенні готового програмного продукту, впровадження якого дозволяє скоротити час на планування дозвілля та підвищити рівень залученості громадян у локальні спільноти за рахунок спрощення доступу до інформації.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Субботін, С. О. Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень : навчальний посібник / С. О. Субботін. – Запоріжжя : ЗНТУ, 2008. – 341 с.
2. Методи та засоби оброблення великих даних в системах діагностування та розпізнавання образів : монографія / [Субботін С. О., Олійник А. О. та ін.] ; під заг. ред. С. О. Субботіна, А. О. Олійника. – Запоріжжя : ЗНТУ, 2018. – 228 с.
3. Олійник, А. О. Еволюційні обчислення та програмування : навчальний посібник / А. О. Олійник, С. О. Субботін, О. О. Олійник. – Запоріжжя : ЗНТУ, 2010. – 324 с.
4. Smith, A. P. The Internet and Loneliness / A. P. Smith, H. Alheneidi // AMA Journal of Ethics. – 2023. – Vol. 25, No. 11. – P. 833–838.
5. Nansen, B. Navigating isolation: mobilising a digital social connection ecosystem on the urban fringe / B. Nansen [et al.] // Information, Communication & Society. – 2025. – P. 1–18.
6. Biermann, G. Mechanics of Location-Based Social Networks: An Analysis of the Foursquare Phenomenon / G. Biermann, S. Hobert // Proceedings of the 24th Americas Conference on Information Systems. – New Orleans, 2018. – P. 1–10.
7. Lutkevich, I. The Role of Geolocation in Modern Social Networking Apps / I. Lutkevich // Journal of Location Based Services. – 2021. – Vol. 15, No. 2. – P. 89–104.
8. Meetup Platform [Electronic resource]. – Access mode: <https://www.meetup.com/about/>
9. Facebook Events [Electronic resource] // Довідковий центр Facebook. – Access mode: <https://www.facebook.com/help/1076296042409786>
10. Eventbrite, The ultimate toolkit for event organizers [Electronic

resource]. – Access mode: <https://www.eventbrite.com/l/all-features/>

11. Tilkov, S. Node.js: Using JavaScript to Build High-Performance Network Programs / S. Tilkov, S. Vinoski // IEEE Internet Computing. – 2010. – Vol. 14, No. 6. – P. 80–83.

12. Node.js Documentation [Electronic resource]. – Access mode: <https://nodejs.org/en/docs/>

13. Simpson, K. You Don't Know JS: Types & Grammar / K. Simpson. – O'Reilly Media, 2015. – 198 p.

14. Haverbeke, M. Eloquent JavaScript: A Modern Introduction to Programming / M. Haverbeke. – 3rd ed. – No Starch Press, 2018. – 472 p.

15. Han, J. Survey on NoSQL database / J. Han, E. Haihong, G. Le, J. Du // 6th International Conference on Pervasive Computing and Applications (ICPCA). – IEEE, 2011. – P. 363–366.

16. Drake, J. D. PostgreSQL 14 Administration Cookbook / J. D. Drake. – Packt Publishing, 2022. – 540 p.

17. Obe, R. O. PostGIS in Action / R. O. Obe, L. S. Hsu. – 3rd ed. – Manning Publications, 2021. – 600 p.

18. Prisma ORM Documentation [Electronic resource]. – Access mode: <https://www.prisma.io/docs>

19. Jabeen, F. A Systematic Review on Cross-Platform Mobile App Development Frameworks / F. Jabeen [et al.] // IEEE Access. – 2023. – Vol. 11. – P. 4567–4589.

20. Heitkötter, H. Evaluating Cross-Platform Development Approaches for Mobile Applications / H. Heitkötter, S. Hanschke, T. A. Majchrzak // Web Information Systems and Technologies. – Springer, 2013. – P. 120–138.

21. Banks, A. Learning React: Modern Patterns for Developing React Apps / A. Banks, E. Porcello. – 2nd ed. – O'Reilly Media, 2020. – 310 p.

22. React Native Documentation [Electronic resource]. – Access mode: <https://reactnative.dev/docs/getting-started>

23. Pautasso, C. REST: Advanced Research Topics and Practical Applications / C. Pautasso, E. Wilde, R. Alarcon. – Springer, 2014. – 236 p.
24. Fielding, R. T. Architectural Styles and the Design of Network-based Software Architectures : Dissertation for the degree of Doctor of Philosophy / R. T. Fielding. – Irvine : University of California, 2000. – 180 p.
25. Martin, R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design / R. C. Martin. – Prentice Hall, 2017. – 432 p.
26. Richardson, C. Microservices Patterns: With examples in Java / C. Richardson. – Manning Publications, 2018. – 520 p.
27. Schiller, J. Location-Based Services / J. Schiller, A. Voisard. – San Francisco : Morgan Kaufmann, 2004. – 257 p.
28. Aggarwal, C. C. Recommender Systems: The Textbook / C. C. Aggarwal. – Cham : Springer, 2016. – 498 p.
29. Freeman, E. Mobile First Design: A Case Study in Event Management Systems / E. Freeman // International Journal of Interactive Mobile Technologies. – 2022. – Vol. 16, No. 4. – P. 112–125.
30. Sommerville, I. Software Engineering / I. Sommerville. – 10th ed. – Pearson, 2015. – 816 p.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення призначене для автоматизації процесів організації, пошуку та координації суспільних подій за допомогою мобільних технологій.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Розробка мобільного застосунку для організації та участі у суспільних подіях», затверджене наказом Національного університету «Запорізька політехніка» № 447 від 30 вересня 2025 р.

A.2 Призначення розробки

Програмний продукт призначений для надання користувачам інструментів створення власних заходів, пошуку активностей за геолокацією та управління участю у подіях

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Програмне забезпечення для організації та участі у суспільних подіях забезпечує виконання таких функцій:

- реєстрація та авторизація користувачів з використанням безпечних протоколів;
- відображення подій на інтерактивній карті з прив'язкою до геолокації;

- перегляд списку подій у вигляді стрічки з можливістю фільтрації;
- створення нових подій із зазначенням назви, категорії, часу та місця проведення;
- автоматична генерація опису події за допомогою штучного інтелекту;
- управління профілем користувача (редагування даних, перегляд історії подій);
- можливість приєднання до подій та управління списком учасників.

A.3.2 Вимоги до інтерфейсу програми

Інтерфейс мобільного застосунку має бути інтуїтивно зрозумілим, адаптованим для сенсорних екранів та відповідати сучасним стандартам UI/UX. Повинна бути забезпечена коректна робота навігації між екранами. Відображення елементів на карті повинно бути чітким та інформативним.

A.3.3 Вимоги до надійності

Програмне забезпечення повинно забезпечувати надійне зберігання даних користувачів та цілісність інформації про події. Система повинна коректно обробляти помилки з'єднання з сервером та нештатні ситуації (наприклад, відсутність доступу до GPS).

A.3.4 Умови експлуатації

Для експлуатації клієнтської частини програмного забезпечення необхідна наявність смартфона з підключенням до мережі Інтернет та активованим модулем геолокації.

А.3.5 Вимоги до складу та параметрів технічний засобів

Для безперебійного функціонування системи необхідно виконати певні вимоги.

Для серверної частини це:

- процесор: 2 ядра з частотою не менше 2.0 ГГц;
- оперативна пам'ять: не менше 4 ГБ;
- дисковий простір: не менше 20 ГБ (SSD);
- програмне середовище: Node.js, СКБД PostgreSQL з розширенням PostGIS.

Для клієнтської частини це:

- смартфон під управлінням ОС Android версії 7.0 або новішої, або iOS версії 13.0 чи новішої;
- наявність модуля GPS;
- стабільне підключення до мережі Інтернет.

А.3.6 Вимоги до маркування і пакування

Програмний продукт пакується у вигляді інсталяційних файлів з розширенням .apk (для операційної системи Android) та .ipa (для операційної системи iOS).

Передача дистрибутивів програми може здійснюватися будь-яким підтримуваним способом обміну цифровою інформацією (через хмарні сховища, електронну пошту, файлообмінні мережі або на фізичних носіях).

Маркування здійснюється в електронному вигляді шляхом присвоєння файлам унікальних імен, що містять назву програмного продукту – «Мобільний застосунок для організації та участі у суспільних подіях» та номер версії збірки.

A.4 Вхідні дані до роботи

Вхідними даними до програмного забезпечення є інформація, яку вводить користувач: облікові дані, параметри подій, зокрема їх назва, категорія, дата, координати, опис, а також геодані, отримані від GPS-модуля пристрою.

Вихідними даними є візуалізовані на карті маркери подій, сформовані списки учасників, згенеровані описи подій та інформація у профілі користувача.

ДОДАТОК Б
Фрагмент тексту програми

```

import { Router } from 'express';
import bcrypt from 'bcrypt';
import jwt from 'jsonwebtoken';
import prisma from './prisma';
import { Prisma } from '@prisma/client';

const router = Router();

router.post('/register', async (req, res) => {
  try {
    const { email, password } = req.body;

    if (!email || !password) {
      return res
        .status(400)
        .json({ message: 'Email and password are required' });
    }

    const saltRounds = 10;
    const passwordHash = await bcrypt.hash(password, saltRounds);

    const user = await prisma.user.create({
      data: {
        email: email,
        passwordHash: passwordHash,
      },
    });

    return res.status(201).json({ id: user.id, email: user.email });
  } catch (error) {
    if (
      error instanceof Prisma.PrismaClientKnownRequestError &&
      error.code === 'P2002'
    ) {
      if (
        Array.isArray(error.meta?.target) &&
        error.meta.target.includes('email')
      ) {
        return res.status(409).json({ message: 'Email already taken' });
      }
    }

    console.error('Registration error:', error);
    return res.status(500).json({ message: 'Internal server error' });
  }
});

router.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;

    if (!email || !password) {
      return res
        .status(400)
        .json({ message: 'Email and password are required' });
    }

    const user = await prisma.user.findUnique({
      where: { email },
    });

    if (!user) {

```

```

    return res.status(401).json({ message: 'Invalid credentials' });
  }

  const isPasswordValid = await bcrypt.compare(password, user.passwordHash);

  if (!isPasswordValid) {
    return res.status(401).json({ message: 'Invalid credentials' });
  }

  const token = jwt.sign(
    { userId: user.id },
    process.env.JWT_SECRET as string,
    { expiresIn: '7d' }
  );

  return res.json({
    token,
    user: {
      id: user.id,
      email: user.email,
      username: user.username,
    },
  });
} catch (error) {
  console.error('Login error:', error);
  return res.status(500).json({ message: 'Internal server error' });
}
});

export default router;

import { NextFunction, Request, Response } from 'express';
import jwt from 'jsonwebtoken';

declare global {
  namespace Express {
    export interface Request {
      user?: { userId: string };
    }
  }
}

export const authMiddleware = (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  const authHeader = req.headers.authorization;
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) {
    return res.status(401).json({ message: 'No token provided' });
  }

  try {
    const payload = jwt.verify(token, process.env.JWT_SECRET as string) as {
      userId: string;
    };

    req.user = payload;
    return next();
  } catch (error) {

```

```

    if (error instanceof jwt.TokenExpiredError) {
      return res.status(401).json({ message: 'Token expired' });
    }
    return res.status(403).json({ message: 'Invalid token' });
  }
};

import { Router } from 'express';
import multer from 'multer';
import path from 'path';
import fs from 'fs';
import prisma from '../prisma';
import { authMiddleware } from '../auth/middleware';

const router = Router();

const storage = multer.memoryStorage();
const upload = multer({ storage });
const uploadDir = path.join(__dirname, 'assets/uploads');
if (!fs.existsSync(uploadDir)) {
  fs.mkdirSync(uploadDir, { recursive: true });
}

router.put(
  '/profile',
  authMiddleware,
  upload.single('avatar'),
  async (req, res) => {
    try {
      const userId = req.user?.userId;
      const { firstName, lastName } = req.body;

      const updateData: any = {};
      if (firstName) updateData.firstName = firstName;
      if (lastName) updateData.lastName = lastName;

      if (req.file) {
        updateData.avatarData = req.file.buffer;
        updateData.avatarMimeType = req.file.mimetype;
      }

      const updatedUser = await prisma.user.update({
        where: { id: userId },
        data: updateData,
        select: {
          id: true,
          email: true,
          firstName: true,
          lastName: true,
        },
      });

      res.json(updatedUser);
    } catch (error) {
      console.error('Update profile error:', error);
      res.status(500).json({ message: 'Failed to update profile' });
    }
  }
);

router.get('/:id/avatar', async (req, res) => {
  try {

```

```

const { id } = req.params;

const user = await prisma.user.findUnique({
  where: { id },
  select: { avatarData: true, avatarMimeType: true },
});

if (!user || !user.avatarData) {
  return res.status(404).send('No avatar found');
}

res.setHeader('Content-Type', user.avatarMimeType || 'image/jpeg');
return res.send(user.avatarData);
} catch (error) {
  return res.status(500).send('Error fetching avatar');
}
});

router.get('/me', authMiddleware, async (req, res) => {
  const user = await prisma.user.findUnique({
    where: { id: req.user?.userId },
    select: {
      id: true,
      email: true,
      username: true,
      firstName: true,
      lastName: true,
    },
  });
  res.json(user);
});

export default router;

import React from 'react';
import { NavigationContainer } from '@react-navigation/native';
import { createNativeStackNavigator } from '@react-navigation/native-stack';
import { Text, View } from 'react-native';
import { useAuth } from '../context/AuthContext';
import { LoginScreen } from '../screens/LoginScreen';
import { RegisterScreen } from '../screens/RegisterScreen';
import { AppTabs } from '../AppTabs';
import { EventDetailsScreen } from '../screens/EventDetailsScreen';
import { EditProfileScreen } from '../screens/EditProfileScreen';
import { PickLocationScreen } from '../screens/PickLocationScreen';
import { EditEventScreen } from '../screens/EditEventScreen';

const Stack = createNativeStackNavigator();

export const Navigation = () => {
  const { token, isLoading } = useAuth();

  if (isLoading) {
    return (
      <View style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
        <Text>Loading...</Text>
      </View>
    );
  }

  return (

```

```

<NavigationContainer>
  <Stack.Navigator
    screenOptions={{
      headerBackTitle: 'Back',
    }}
  >
    {token ? (
      // --- LOGGED IN STACK ---
      <>
        <Stack.Screen
          name="AppTabs"
          component={AppTabs}
          options={{ headerShown: false }}
        />
        {/* Add the Details Screen here */}
        <Stack.Screen
          name="EventDetails"
          component={EventDetailsScreen}
          options={{ title: 'Event Details' }}
        />
        <Stack.Screen
          name="EditProfile"
          component={EditProfileScreen}
          options={{ title: 'Edit Profile' }}
        />
        <Stack.Screen
          name="PickLocation"
          component={PickLocationScreen}
          options={{ title: 'Select Location' }}
        />
        <Stack.Screen
          name={'EditEvent'}
          component={EditEventScreen}
          options={{ title: 'Edit Event' }}
        />
      </>
    ) : (
      // --- AUTH STACK ---
      <Stack.Group screenOptions={{ headerShown: false }}>
        <Stack.Screen name="Login" component={LoginScreen} />
        <Stack.Screen name="Register" component={RegisterScreen} />
      </Stack.Group>
    )}
  </Stack.Navigator>
</NavigationContainer>
);
};

import React, { useEffect, useState } from 'react';
import {
  ActivityIndicator,
  Alert,
  KeyboardAvoidingView,
  Platform,
  ScrollView,
  StyleSheet,
  Text,
  TextInput,
  TouchableOpacity,
  View,
} from 'react-native';
import DateTimePicker from '@react-native-community/datetimepicker';

```

```

import MapView, { Marker } from 'react-native-maps';
import { FontAwesomeIcon } from '@expo/vector-icons';
import api from '../services/api';

export const CreateEventScreen = ({ navigation, route }: any) => {
  const [loading, setLoading] = useState(false);
  const [aiLoading, setAiLoading] = useState(false);

  // Form State
  const [title, setTitle] = useState('');
  const [description, setDescription] = useState('');
  const [city, setCity] = useState('');
  const [address, setAddress] = useState('');

  // Location
  const [latitude, setLatitude] = useState<number | null>(null);
  const [longitude, setLongitude] = useState<number | null>(null);

  // Date
  const [date, setDate] = useState(new Date());
  const [showDatePicker, setShowDatePicker] = useState(false);

  // Categories
  const [categories, setCategories] = useState([]);
  const [selectedCategoryId, setSelectedCategoryId] = useState<number | null>(
    null
  );

  // Fetch Categories
  useEffect(() => {
    api.get('/categories').then((res) => setCategories(res.data));
  }, []);

  // Handle Location Return
  useEffect(() => {
    if (route.params?.pickedLocation) {
      setLatitude(route.params.pickedLocation.latitude);
      setLongitude(route.params.pickedLocation.longitude);
    }
  }, [route.params?.pickedLocation]);

  const generateDescription = async () => {
    if (!title || !city) {
      Alert.alert('Missing Info', 'Enter a Title and City first!');
      return;
    }
    setAiLoading(true);
    try {
      const response = await api.post('/ai/generate-description', {
        title,
        city,
        date: date.toDateString(),
      });
      setDescription(response.data.description);
    } catch (error) {
      Alert.alert('Error', 'AI is taking a nap. Try again later.');
```

```

    } finally {
      setAiLoading(false);
    }
  };

  const handleCreate = async () => {

```

```

if (
  !title ||
  !description ||
  !city ||
  !address ||
  !latitude ||
  !longitude
) {
  Alert.alert('Missing Fields', 'Please fill in everything to continue.');
```

return;

```

}

setLoading(true);
try {
  await api.post('/events', {
    title,
    description,
    city,
    address,
    date: date.toISOString(),
    latitude,
    longitude,
    categoryId: selectedCategoryId,
  });

  Alert.alert('Success', 'Event created!');
  // Reset
  setTitle('');
  setDescription('');
  setCity('');
  setAddress('');
  setLatitude(null);
  setLongitude(null);
  setSelectedCategoryId(null);

  navigation.navigate('Map');
} catch (error) {
  Alert.alert('Error', 'Failed to create event');
} finally {
  setLoading(false);
}
};

// --- Render Helpers ---

const renderLabel = (text: string, icon?: any) => (
  <View style={styles.labelRow}>
    {icon && (
      <FontAwesome
        name={icon}
        size={14}
        color="#666"
        style={{ marginRight: 6 }}
      />
    )}
    <Text style={styles.label}>{text}</Text>
  </View>
);

return (
  <KeyboardAvoidingView
    behavior={Platform.OS === 'ios' ? 'padding' : undefined}

```

```

    style={{ flex: 1, backgroundColor: '#fff' }}
  >
  <ScrollView contentContainerStyle={styles.container}>
    {/* SECTION 1: THE BASICS */}
    <View style={styles.section}>
      <Text style={styles.sectionHeader}>The Basics</Text>

      {renderLabel('Event Title', 'header')}
      <TextInput
        style={styles.input}
        value={title}
        onChangeText={setTitle}
        placeholder="e.g. Sunday Morning Yoga"
        placeholderTextColor="#ccc"
      />

      {renderLabel('Category', 'tag')}
      <View style={styles.categoryContainer}>
        {categories.map((cat: any) => (
          <TouchableOpacity
            key={cat.id}
            onPress={() => setSelectedCategoryId(cat.id)}
            style={[
              styles.categoryChip,
              selectedCategoryId === cat.id && styles.categoryChipActive,
            ]}
          >
            <Text
              style={[
                styles.categoryText,
                selectedCategoryId === cat.id && styles.categoryTextActive,
              ]}
            >
              {cat.name}
            </Text>
          </TouchableOpacity>
        ))}
      </View>
    </View>

    {/* SECTION 2: WHEN & WHERE */}
    <View style={styles.section}>
      <Text style={styles.sectionHeader}>When & Where</Text>

      {/* Date Picker Trigger */}
      {renderLabel('Date & Time', 'calendar')}
      <TouchableOpacity
        style={styles.dateInput}
        onPress={() => setShowDatePicker(true)}
      >
        <Text style={styles.dateText}>
          {date.toLocaleDateString()} at{' '}
          {date.toLocaleTimeString([], {
            hour: '2-digit',
            minute: '2-digit',
          })}
        </Text>
        <FontAwesome name="chevron-down" size={12} color="#999" />
      </TouchableOpacity>

      {/* Actual Pickers */}
      {showDatePicker && Platform.OS === 'android' && (

```

```

    <DateTimePicker
      value={date}
      mode="date"
      display="default"
      onChange={ (e, d) => {
        setShowDatePicker(false);
        if (d) setDate(d);
      }}
    />
  )}
  {Platform.OS === 'ios' && (
    <View style={{ alignItems: 'flex-start', marginBottom: 15 }}>
      <DateTimePicker
        value={date}
        mode="datetime"
        display="default"
        onChange={ (e, d) => d && setDate(d) }
      />
    </View>
  )}

  {renderLabel('City', 'building')}
  <TextInput
    style={styles.input}
    value={city}
    onChangeText={setCity}
    placeholder="e.g. New York"
    placeholderTextColor="#ccc"
  />

  {renderLabel('Street Address', 'map-pin')}
  <TextInput
    style={styles.input}
    value={address}
    onChangeText={setAddress}
    placeholder="e.g. 123 Broadway"
    placeholderTextColor="#ccc"
  />

  { /* Map Preview */ }
  <Text style={styles.label}>Exact Location</Text>
  <View style={styles.mapWrapper}>
    {latitude && longitude ? (
      <View style={{ flex: 1 }}>
        <MapView
          style={{ flex: 1 }}
          region={{
            latitude,
            longitude,
            latitudeDelta: 0.005,
            longitudeDelta: 0.005,
          }}
          scrollEnabled={false}
          zoomEnabled={false}
        >
          <Marker coordinate={{ latitude, longitude }} />
        </MapView>
        <TouchableOpacity
          style={styles.changeLocBtn}
          onPress={() => navigation.navigate('PickLocation')}
        >
          <Text style={styles.changeLocText}>Change</Text>

```

```

        </TouchableOpacity>
      </View>
    ) : (
      <TouchableOpacity
        style={styles.pickLocPlaceholder}
        onPress={() => navigation.navigate('PickLocation')}
      >
        <FontAwesome name="map-o" size={32} color="#007AFF" />
        <Text style={styles.pickLocText}>
          Tap to pin location on map
        </Text>
      </TouchableOpacity>
    )}
  </View>
</View>

{/* SECTION 3: DETAILS */}
<View style={styles.section}>
  <View
    style={{
      flexDirection: 'row',
      justifyContent: 'space-between',
      alignItems: 'center',
      marginBottom: 8,
    }}
  >
    <View style={styles.labelRow}>
      <FontAwesome
        name="align-left"
        size={14}
        color="#666"
        style={{ marginRight: 6 }}
      />
      <Text style={styles.label}>Description</Text>
    </View>

    {/* AI Button */}
    <TouchableOpacity
      style={styles.aiButton}
      onPress={generateDescription}
      disabled={aiLoading}
    >
      <FontAwesome
        name="magic"
        size={12}
        color="white"
        style={{ marginRight: 4 }}
      />
      <Text style={styles.aiButtonText}>
        {aiLoading ? 'Thinking...' : 'AI Write'}
      </Text>
    </TouchableOpacity>
  </View>

  <TextInput
    style={[styles.input, styles.textArea]}
    value={description}
    onChangeText={setDescription}
    placeholder="Tell people what this event is about..."
    placeholderTextColor="#ccc"
    multiline
  />

```

```

    </View>

    { /* SUBMIT */ }
    <TouchableOpacity
      style={styles.createButton}
      onPress={handleCreate}
      disabled={loading}
    >
      {loading ? (
        <ActivityIndicator color="white" />
      ) : (
        <Text style={styles.createButtonText}>Publish Event</Text>
      )}
    </TouchableOpacity>

  </ScrollView>
</KeyboardAvoidingView>
);
};

const styles = StyleSheet.create({
  container: { padding: 20, paddingBottom: 50 },

  // Sections
  section: { marginBottom: 30 },
  sectionHeader: {
    fontSize: 18,
    fontWeight: '800',
    color: '#333',
    marginBottom: 15,
  },

  // Labels
  labelRow: {
    flexDirection: 'row',
    alignItems: 'center',
    marginBottom: 8,
    marginTop: 5,
  },
  label: { fontSize: 14, fontWeight: '600', color: '#555' },

  // Inputs
  input: {
    backgroundColor: '#F5F5F5',
    borderRadius: 10,
    padding: 12,
    fontSize: 16,
    borderWidth: 1,
    borderColor: '#EEE',
    marginBottom: 10,
    color: '#333',
  },
  textArea: { height: 120, textAlignVertical: 'top' },
  dateInput: {
    backgroundColor: '#F5F5F5',
    borderRadius: 10,
    padding: 15,
    borderWidth: 1,
    borderColor: '#EEE',
    marginBottom: 15,
    flexDirection: 'row',
    justifyContent: 'space-between',
  },

```

```

    alignItems: 'center',
  },
  dateText: { fontSize: 16, color: '#333' },

  // Categories
  categoryContainer: { flexDirection: 'row', flexWrap: 'wrap', gap: 10 },
  categoryChip: {
    paddingVertical: 8,
    paddingHorizontal: 16,
    borderRadius: 20,
    backgroundColor: '#F0F0F0',
    borderWidth: 1,
    borderColor: '#DDD',
  },
  categoryChipActive: { backgroundColor: '#007AFF', borderColor: '#007AFF' },
  categoryText: { color: '#555', fontWeight: '600', fontSize: 13 },
  categoryTextActive: { color: 'white' },

  // Map Preview
  mapWrapper: {
    height: 160,
    borderRadius: 12,
    overflow: 'hidden',
    marginTop: 5,
    backgroundColor: '#F0F0F0',
    borderWidth: 1,
    borderColor: '#EEE',
  },
  pickLocPlaceholder: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
  },
  pickLocText: { color: '#007AFF', fontWeight: '600', marginTop: 10 },
  changeLocBtn: {
    position: 'absolute',
    bottom: 10,
    right: 10,
    backgroundColor: 'white',
    paddingHorizontal: 12,
    paddingVertical: 6,
    borderRadius: 15,
    shadowColor: '#000',
    shadowOpacity: 0.1,
    elevation: 2,
  },
  changeLocText: { fontSize: 12, fontWeight: 'bold', color: '#007AFF' },

  // AI Button
  aiButton: {
    backgroundColor: '#8A2BE2', // Purple for "Magic"
    flexDirection: 'row',
    alignItems: 'center',
    paddingHorizontal: 10,
    paddingVertical: 4,
    borderRadius: 12,
  },
  aiButtonText: { color: 'white', fontSize: 11, fontWeight: 'bold' },

  // Submit
  createButton: {
    backgroundColor: '#007AFF',

```

```
height: 55,  
borderRadius: 12,  
justifyContent: 'center',  
alignItems: 'center',  
shadowColor: '#007AFF',  
shadowOpacity: 0.3,  
shadowOffset: { width: 0, height: 4 },  
elevation: 5,  
},  
createButtonText: { color: 'white', fontSize: 18, fontWeight: 'bold' },  
});
```

ДОДАТОК В
Слайди презентації



Рисунок В.1 - Слайд 1

Загальна характеристика роботи

Об'єкт дослідження

Процес розробки програмного забезпечення для організації, координації та управління суспільними подіями в цифровому середовищі.

Предмет дослідження

Методи та програмні засоби автоматизації взаємодії учасників та організаторів заходів з використанням мобільних технологій.

Мета роботи

Підвищення ефективності соціальної взаємодії та скорочення часових витрат користувачів на організацію і пошук подій шляхом розробки мобільної інформаційної системи з використанням геолокації та алгоритмів рекомендацій.

Рисунок В.2 - Слайд 2

Завдання роботи

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- виконати аналіз предметної області та існуючих аналогів;
- проаналізувати та обґрунтувати вибір програмних засобів для реалізації системи;
- здійснити проектування програмного забезпечення та розробити архітектуру системи;
- створити програмне забезпечення для організації та участі у суспільних подіях;
- виконати тестування розробленого програмного забезпечення.

Рисунок В.3 - Слайд 3

Порівняння аналогів

Критерій	Розроблена система	Meetup	Facebook Events	Telegram
Гео-навігація	Присутня	Присутня (обмежено)	Присутня	Відсутня
Поріг входу	Низький	Високий	Середній	Низький
Вартість	Безкоштовно	Платна	Безкоштовно	Безкоштовно
Фокус	Локальні події	Проф. спільноти	Соц. зв'язки	Новини/Чати

Висновок: Існуючі рішення або складні для швидкої організації, або не мають структурування та мапи

Рисунок В.4 - Слайд 4

Обґрунтування вибору серверної платформи

Критерій порівняння	Node.js (Обрано)	Python (Django/Flask)	Java (Spring)
Модель I/O	Неблокуюча, асинхронна (ефективна для багатьох з'єднань)	Блокуюча (синхронна) у стандартній реалізації	Багатопотокова (вимоглива до ресурсів)
Мова розробки	TypeScript/JS (спільна з клієнтом)	Python	Java
Швидкодія (JSON)	Висока (нативна робота V8)	Середня (витрати на серіалізацію)	Висока, але складніша обробка
Екосистема	NPM - величезна база готових рішень	PyPI	Maven/Gradle

Висновок: Вибір Node.js дозволяє використовувати єдину мову (TypeScript) на всьому стеку та забезпечує високу продуктивність при обробці численних одночасних запитів від мобільного додатку

Рисунок В.5 - Слайд 5

Обґрунтування вибору СКБД

Критерій порівняння	PostgreSQL + PostGIS (Обрано)	MongoDB (NoSQL)
Структура даних	Чітка схема, підтримка складних зв'язків (SQL)	Гнучка схема, складність у підтримці зв'язків "багато-до-багатьох"
Геопросторові дані	PostGIS: професійний стандарт для складних гео-запитів	Базова підтримка GeoJSON, обмежений функціонал
Цілісність даних	ACID транзакції (гарантія надійності запису)	ACID реалізовано, але з обмеженнями продуктивності
Дублювання даних	Нормалізація (відсутність дублів)	Денормалізація (ризик неузгодженості даних)

Висновок: PostgreSQL обрано через необхідність підтримки складної реляційної моделі (користувачі-події-учасники) та потужні інструменти PostGIS для пошуку подій у радіусі локації.

Рисунок В.6 - Слайд 6

Обґрунтування вибору мови програмування

Критерій порівняння	TypeScript (Обрано)	JavaScript
Типізація	Статична (дозволяє визначати типи даних явно)	Динамічна (типи визначаються автоматично, ризик помилок)
Виявлення помилок	На етапі компіляції (до запуску коду)	Під час виконання (Runtime errors)
Масштабованість	Висока	Складна підтримка у великих проєктах
Підтримка IDE	Максимальна (IntelliSense, автодоповнення, рефакторинг)	Базова (обмежені підказки) (ризик неузгодженості даних)
Спільний код	Єдині моделі для Frontend та Backend	Потрібне дублювання логіки перевірки даних

Висновок: Використання TypeScript дозволяє виявляти помилки на ранніх етапах розробки, забезпечує самодокументованість коду та дає можливість використовувати спільні інтерфейси між сервером та клієнтом.

Рисунок В.7 - Слайд 7

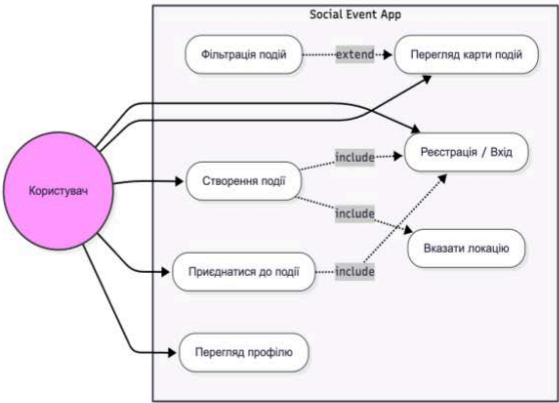
Обґрунтування засобів мобільної розробки

Критерій порівняння	React Native (Обрано)	Flutter	Нативна розробка (Swift/Kotlin)
Мова програмування	TypeScript (єдина екосистема з Backend)	Dart (нова мова)	Swift / Kotlin (дві різні мови)
Кодова база	Спільна для iOS та Android (до 90% коду)	Спільна, власний рушій рендерингу	Окрема для кожної платформи
Інтерфейс	Використовує нативні компоненти UI	Емулює нативні компоненти (Canvas)	Повністю нативний
Швидкість розробки	Висока	Висока	Низька

Висновок: React Native дозволяє суттєво зекономити час завдяки спільній кодовій базі та використанню TypeScript, що забезпечує строгу типізацію та спільні інтерфейси з серверною частиною.

Рисунок В.8 - Слайд 8

Діаграма прецедентів



Основні актори:

- Користувач (Організатор/Учасник):
 - Перегляд інтерактивної карти
 - Пошук та фільтрація подій
 - Створення власної події
 - Участь у заходах
 - Керування профілем

Рисунок В.9 - Слайд 9

Схема бази даних

Ключові сутності (ER-діаграма):

Users
Користувачі, профілі, авторизаційні дані.

Events
Дані про події + геометрія.

Participants
Зв'язок "багато-до-багатьох" (учасники подій).

Categories
Довідник категорій подій.

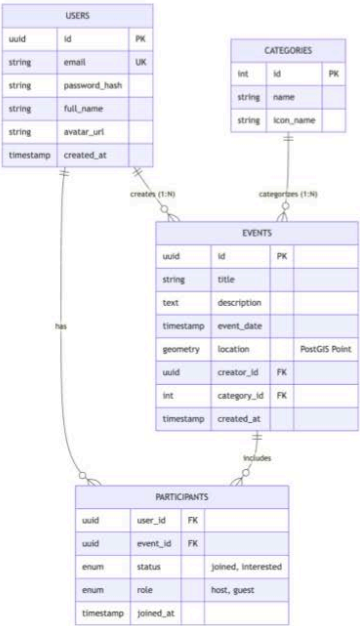


Рисунок В.10 - Слайд 10

Інтерфейс

користувача:

- авторизація
- реєстрація

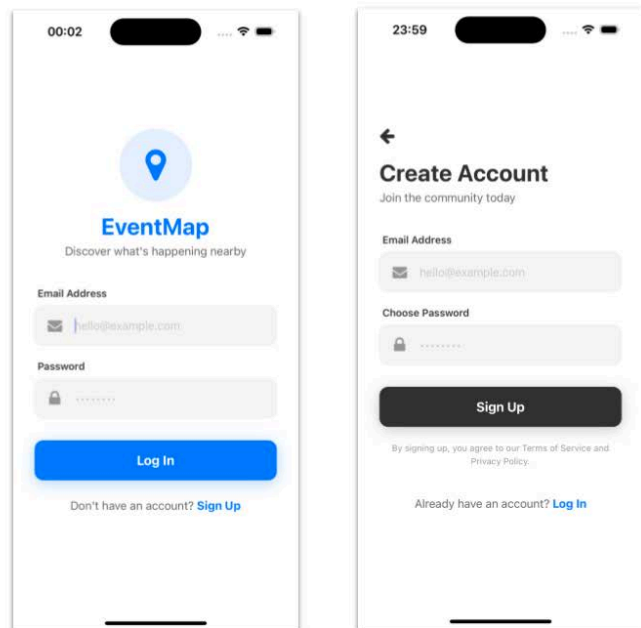


Рисунок В.11 - Слайд 11

Інтерфейс

користувача:

- карта
- стрічка подій

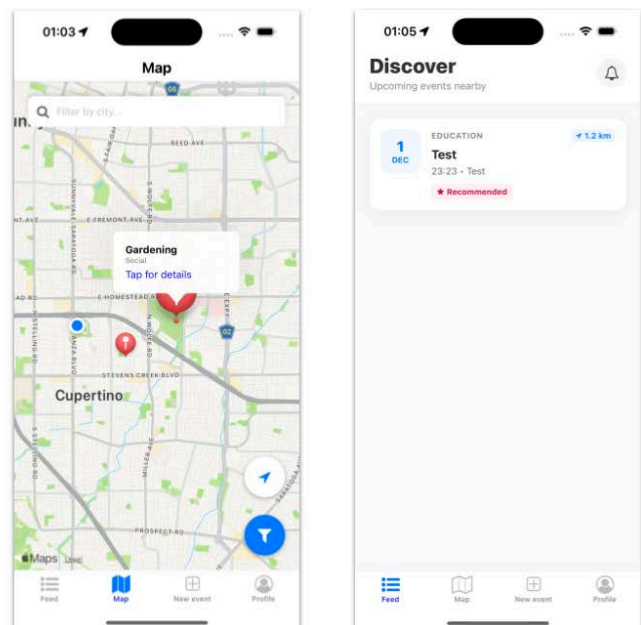


Рисунок В.12 - Слайд 12

Створення події

Ключовий функціонал:

- **Інтеграція AI:** Автоматична генерація опису події на основі назви.
- **Геолокація:** Вибір точки проведення безпосередньо на мапі.
- **Параметри:** Вибір категорії, дати та часу.

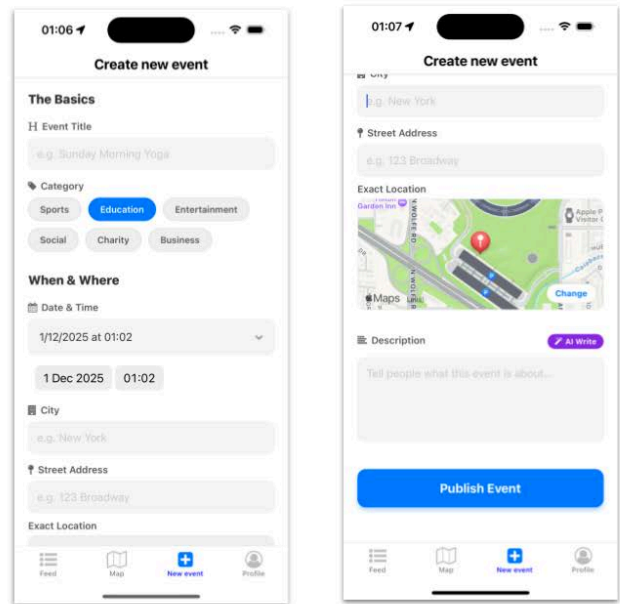


Рисунок В.13 - Слайд 13

Інтерфейс користувача:

- деталі події
- профіль

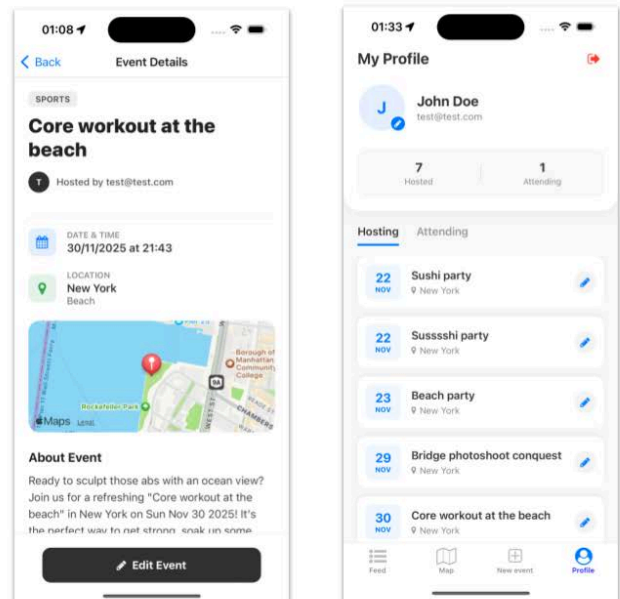


Рисунок В.14 - Слайд 14

Висновки

В ході виконання дипломної кваліфікаційної роботи магістра було проаналізовано та досліджено процес розробки програмного забезпечення для організації та участі у суспільних подіях.

Наукова новизна роботи полягає в удосконаленні методів соціальної координації шляхом комбінування геолокаційних сервісів реального часу з персоніфікованим ранжуванням контенту, що, на відміну від існуючих підходів, дозволяє зменшити час пошуку релевантних подій та знизити комунікаційні затримки при організації спонтанних зустрічей.

Практичне значення полягає у створенні готового програмного продукту, впровадження якого дозволяє скоротити час на планування дозвілля та підвищити рівень залученості громадян у локальні спільноти за рахунок спрощення доступу до інформації.

Проведене тестування встановило, що програма для організації та участі у суспільних подіях відповідає технічним вимогам та готова до впровадження.

Рисунок В.15 - Слайд 15