

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 проєкту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ДЛЯ ОРГАНІЗАЦІЇ ТА ПРОВЕДЕННЯ ОФЛАЙН-ЗАХОДІВ
DEVELOPMENT OF SOFTWARE FOR
ORGANIZING AND RUNNING OFFLINE EVENTS

Виконав(ла): студент(ка) 4 курсу, групи КНТ-113сп

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

ГРИГОРОВ М.С.

(ПРИЗВИЩЕ та ініціали)

Керівник СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

Рецензент ШИТІКОВА О.В.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
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
“ ” 2026 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ГРИГОРОВА Михайла Сергійовича
(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розроблення програмного забезпечення для організації та проведення офлайн-заходів. Development of Software for Organizing and Running Offline Events

керівник проєкту (роботи) к.т.н., доцент, СКРУПСЬКИЙ Степан Юрійович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139

2. Строк подання студентом проєкту (роботи) 10 червня 2026 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Реалізація програми. 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	СКРУПСЬКИЙ С.Ю., доцент		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання « 20 » квітня 2026 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.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій розробки.	2 тиждень	Розділ 2
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження програмного забезпечення.	5 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	6 тиждень	Додатки
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студент(ка)

_____ Михайло ГРИГОРОВ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Степан СКРУПСЬКИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 100 с., 5 табл., 41 рис., 3 дод., 25 джерел.

ОРГАНІЗАЦІЯ ЗАХОДІВ, ЗАХІД, УЧАСНИК, РЕЄСТРАЦІЯ, НАЛАШТУВАННЯ ЗАХОДІВ, ВЕБЗАСТОСУНОК, ВЕБФРЕЙМВОРК.

Об'єкт роботи - процес інженерії програмного забезпечення для організації офлайн-заходів.

Предмет роботи - програмне забезпечення для керування офлайн-заходами, що базуватиметься на автоматизованих модулях реєстрації учасників, планування та модерування заходів.

Мета роботи - зменшення часу організації офлайн-заходів та зменшення витрат ресурсів організаторів шляхом розроблення програмного забезпечення що автоматизує реєстрацію учасників та планування заходів.

Матеріали, методи та технічні засоби: мова програмування JavaScript, система керування базами даних PostgreSQL, вебфреймворк React, REST API.

Результати. Результатами виконання роботи є створення програмного забезпечення для організації офлайн-заходів. Воно охопить реєстрацію, контроль списками учасників, створення та модерування заходів. Система зменшить ручну обробку даних, спростить координацію заходів та забезпечить керування всією інформацією про заходи в одному місці.

Висновки. Розроблене програмне забезпечення скоротить час підготовки офлайн-заходів, зменшить роботу організаторів та покращить взаємодію між учасниками через структуровану інформацію та засоби комунікації за рахунок автоматизації ключових процесів керування заходами.

Галузь використання. Організація зустрічей, навчальних заходів, клубів за інтересами та інших офлайн-заходів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor:
100 pages, 5 tables, 41 figures, 3 appendixes, 25 sources.

EVENT MANAGEMENT, EVENT, PARTICIPANT, REGISTRATION,
EVENT SETTINGS, WEB APPLICATION, WEB FRAMEWORK.

Object of study is a process of software engineering for offline event organizations.

Subject of study is a software for managing offline events that are based on automation modules for participant registration, planning, and event moderation.

The purpose of the work is to reduce the time of offline event organization and reduce the cost of organizers' resources by developing software with automated means of participant registration and event planning.

Materials, methods, and tools: programming language JavaScript, database management system PostgreSQL, web framework React, REST API.

Results. The result of this work is the development of software for organizing offline events. It covers registration, participant list management, event creation and moderation. The system will reduce manual data processing, simplify event coordination, and ensure the management of all information related to the event in a single place.

Conclusions. The developed software will reduce the preparation time for offline events, decrease the work for organizers, and improve interaction among participants through structured information and communication tools by automating key event management processes.

The field of use is an organization of meetings, educational events, interest clubs, and other offline events.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	7
Вступ.....	8
1 Аналіз предметної області.....	10
1.1 Аналіз застосунків для організації та пошуку офлайн-заходів.....	10
1.2 Порівняльний аналіз розробки та аналогів.....	19
1.3 Висновки за розділом.....	24
2 Розробка архітектури програми.....	26
2.1 Вибір засобів проектування.....	26
2.2 Сценарії використання програми.....	27
2.3 Структура бази даних.....	30
2.4 Прототипи системи.....	37
2.5 Схеми UML роботи системи.....	39
2.6 Висновки за розділом 2.....	44
3 Реалізація програми.....	46
3.1 Визначення порядку роботи програми.....	46
3.2 Результати реалізації програмних одиниць застосунку.....	47
3.3 Висновки за розділом 3.....	58
4 Експлуатація та тестування програми.....	60
4.1 Призначення програми.....	60
4.2 Умови виконання програми.....	60
4.3 Виконання і тестування програми.....	61
4.4 Висновки за розділом 4.....	66
Висновки.....	67
Перелік джерел посилання.....	68

Додаток А Технічне завдання.....	73
Додаток Б Текст програми.....	79
Додаток В Слайди презентації.....	93

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

- БД – база даних;
ВФ – вебфреймворк.

ВСТУП

Організація офлайн-заходів є актуальною та важливою частиною соціальної взаємодії суспільства в наші часи. З розвитком цифрових технологій, потреба в офлайн-активностях та різного роду заходів не зменшується, оскільки вони забезпечують пряму комунікацію між людьми, створюючи умови для формування спільнот та обміну досвідом.

Беручи в приклад настільні ігри як один із типів офлайн-активності бачимо, що у багатьох країнах індустрія настільних ігор активно розвивається. Це підтверджується зростанням кількості спеціалізованих магазинів, клубів та тематичних заходів. Значна увага приділяється виробництву і розповсюдженню настільних ігор, а також створенню контенту, пов'язаного з ними. Таким чином можна прослідкувати високий рівень зацікавленості користувачів у подібних активностях й заходах.

Підтримка офлайн-заходів здійснюється на різних рівнях: через соціальні мережі для інформування про захід, через спеціалізовані платформи для продажу квитків, а також через окремі вебресурси для планування заходів. Однак ці рішення часто є фрагментованими та не забезпечують комплексного підходу до організації заходів, особливо в контексті пошуку учасників за інтересами, керування реєстрацією та координації заходів.

Особливо актуальною є проблема пошуку однодумців для участі в офлайн-заходах. Користувачі можуть мати бажання відвідати захід, але не завжди мають відповідну компанію або інформацію про доступні заходи. Крім того, організатори витрачають значний час на координацію учасників, обробку заявок та керування списками відвідувачів. Це створює потребу у створенні програмного забезпечення, яке робило б автоматизацію цих процесів та забезпечило зручну взаємодію між учасниками та організаторами.

Таким чином, актуальним є розроблення програмного забезпечення для організації та проведення офлайн-заходів, яке враховує специфіку заходів за інтересами та забезпечує підтримку ключових процесів організації та

керування заходами.

Метою роботи є зменшення часу організації офлайн-заходів та скорочення витрат ресурсів організаторів шляхом розроблення програмного забезпечення з автоматизованими засобами пошуку учасників за інтересами, реєстрації, планування та керування заходами.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз застосунків для організації та пошуку офлайн-заходів

Під час аналізу було розглянуто ряд програмних застосунків, які так чи інакше стосуються організації офлайн-заходів та пошуку учасників за інтересами, однак не вирішують поставлену задачу комплексно.

Існує велика кількість вебзастосунків, які дозволяють переглядати інформацію про захід, проте не забезпечують повного функціоналу для організації заходів або пошуку однодумців. Як приклад такого рішення можна навести платформу Eventbrite (рис. 1.1) [1]. Даний сервіс дозволяє створювати захід, переглядати їх опис та реєструватися на них. Проте функціональність взаємодії між учасниками є обмеженою, а пошук однодумців за інтересами реалізований недостатньо ефективно.

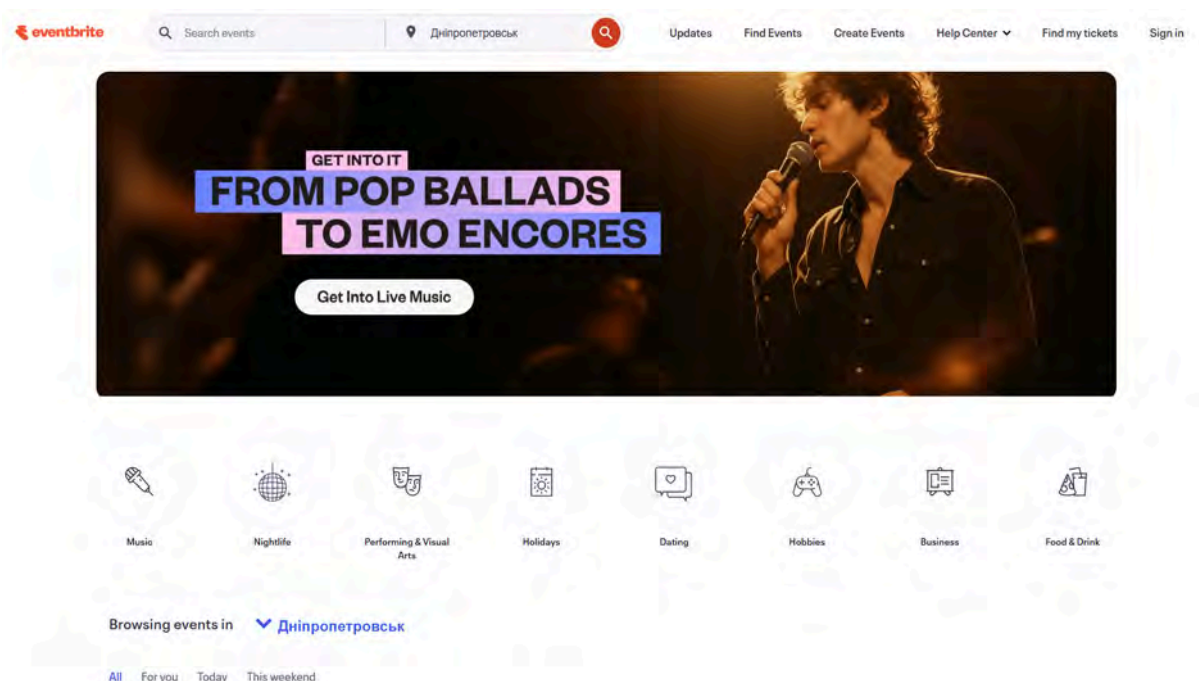


Рисунок 1.1 - Платформа Eventbrite [1]

Окремо варто виділити платформу Meetup (рис. 1.2) [2], яка є одним із найпоширеніших застосунків для пошуку заходів за інтересами. Користувачі можуть приєднуватися до груп, брати участь у заходах, а також створювати

власні зустрічі. Проте цей сервіс не спеціалізується на конкретних типах заходів, зокрема настільних іграх. Пошук заходів є загальним, а деталізація (наприклад, рівень складності гри, кількість учасників, формат взаємодії) обмежена.

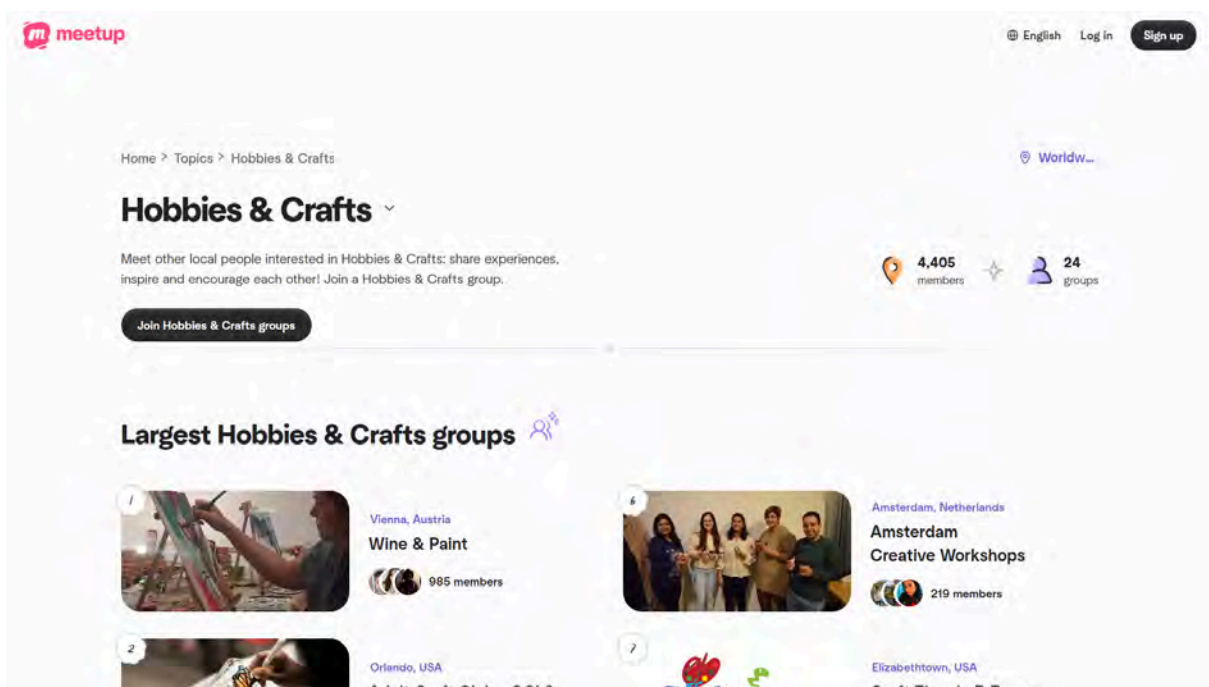


Рисунок 1.2 - Платформа Meetup [2]

Для організації заходів також використовуються соціальні мережі, зокрема Facebook (рис. 1.3) [3]. У рамках платформи доступний функціонал створення заходів, запрошення учасників та комунікації. Однак такі рішення не є спеціалізованими: інформація про захід подається у вигляді тексту, відсутні структуровані механізми керування реєстрацією, а також немає зручних інструментів підбору учасників за інтересами.

У сфері настільних ігор популярним ресурсом соціальна мережа є BoardGameGeek (рис. 1.4) [4], який містить велику базу ігор, рейтинги та обговорення. Проте цей ресурс орієнтований на інформаційну підтримку користувачів, а не на організацію офлайн-зустрічей. Відсутні інструменти для створення заходів, керування учасниками та координації заходів.

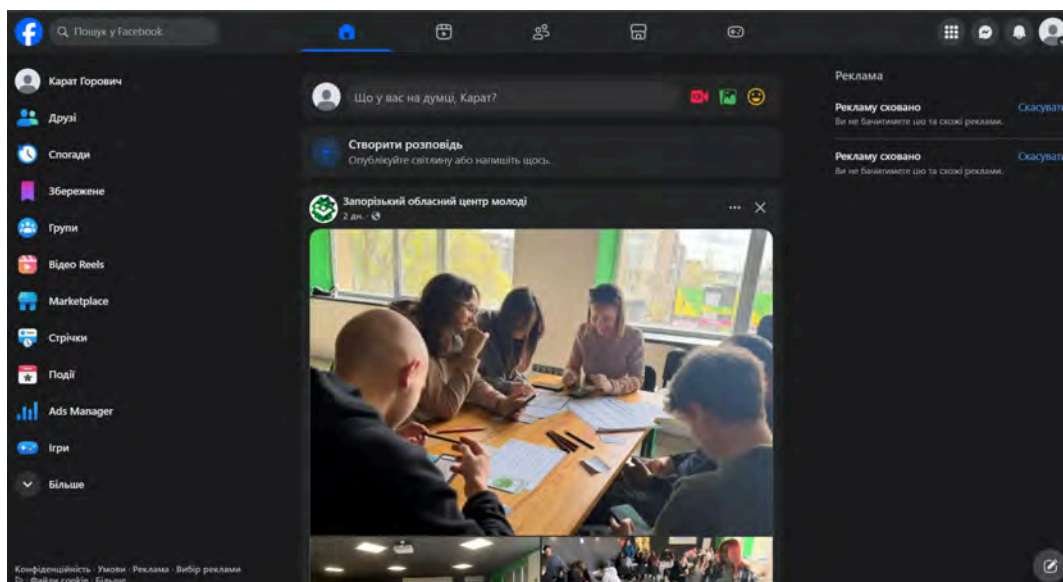


Рисунок 1.3 - соціальна мережа Facebook [3]

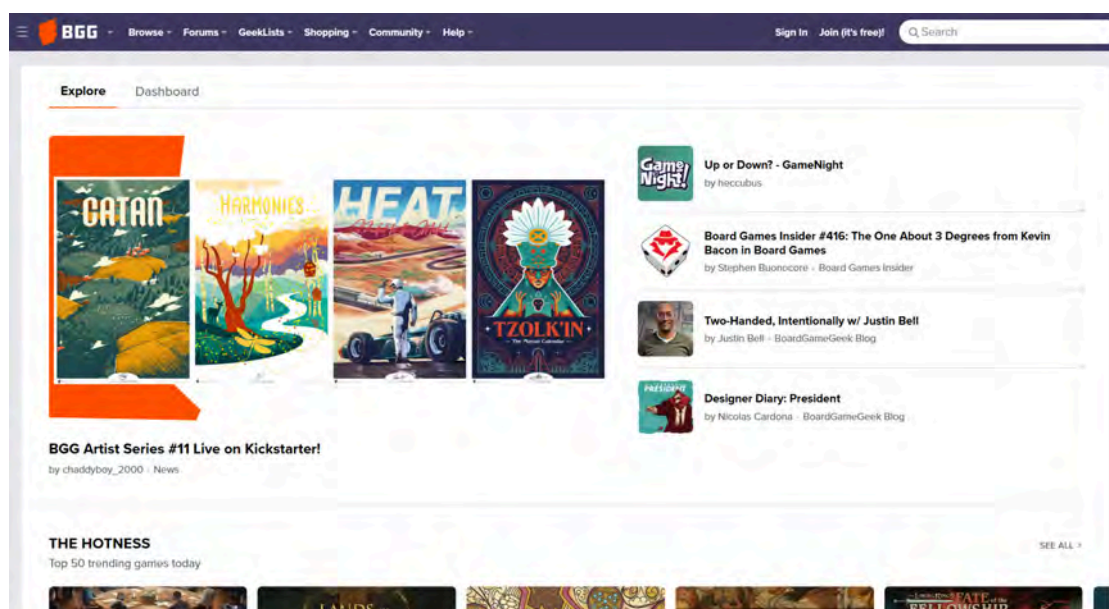


Рисунок 1.4 - ресурс BoardGameGeek [4]

Подальше розширення аналізу охопило нішеві мобільні застосунки та сервіси, орієнтовані на пошук соціальних зв'язків та автоматизацію. Зокрема, такі рішення як Bumble For Friends (BFF) [5] (рис. 1.5) використовує механіку гортання для пошуку однодумців. Однак ключовим недоліком платформи є її орієнтація на індивідуальні зустрічі, а не на координацію та організацію групових офлайн-заходів.

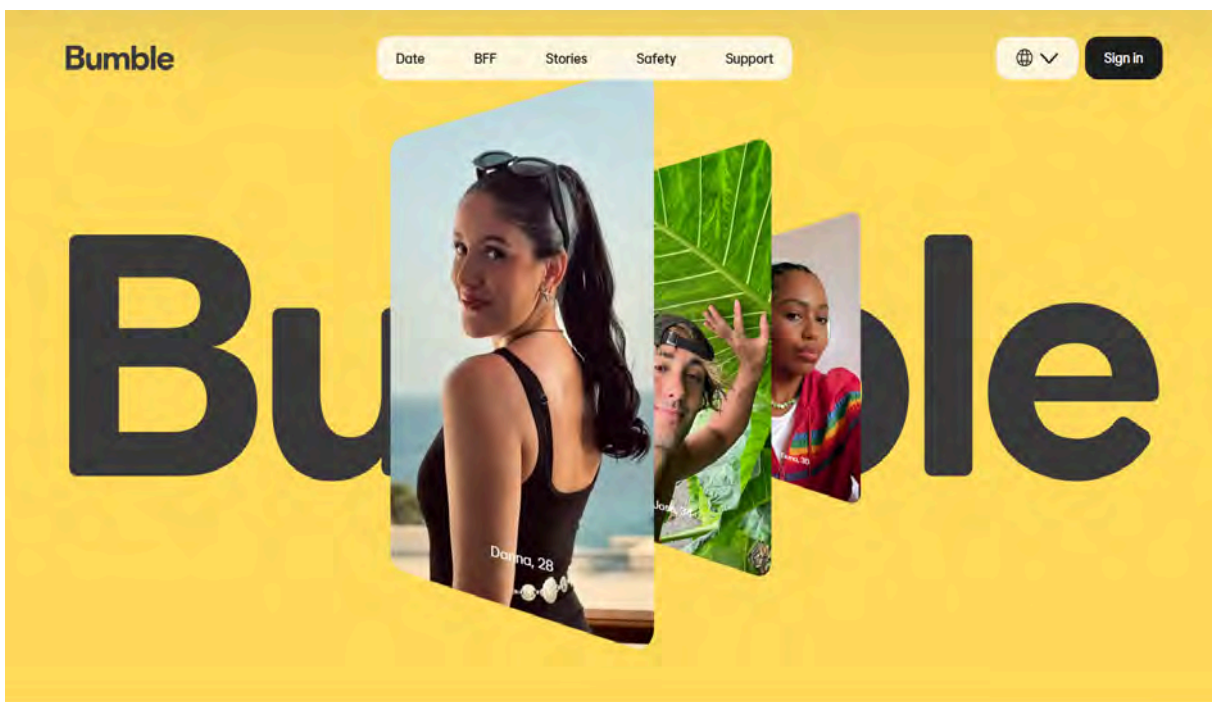


Рисунок 1.5 - Сайт Bumble For Friends (BFF) [5]

Схожою проблемою географічної обмеженості та вузької спрямованості володіє й соціальна мережа Nextdoor [6] (рис. 1.6), яка ефективна лише для комунікації сусідів у межах одного району, що робить складним пошук специфічних захоплень.

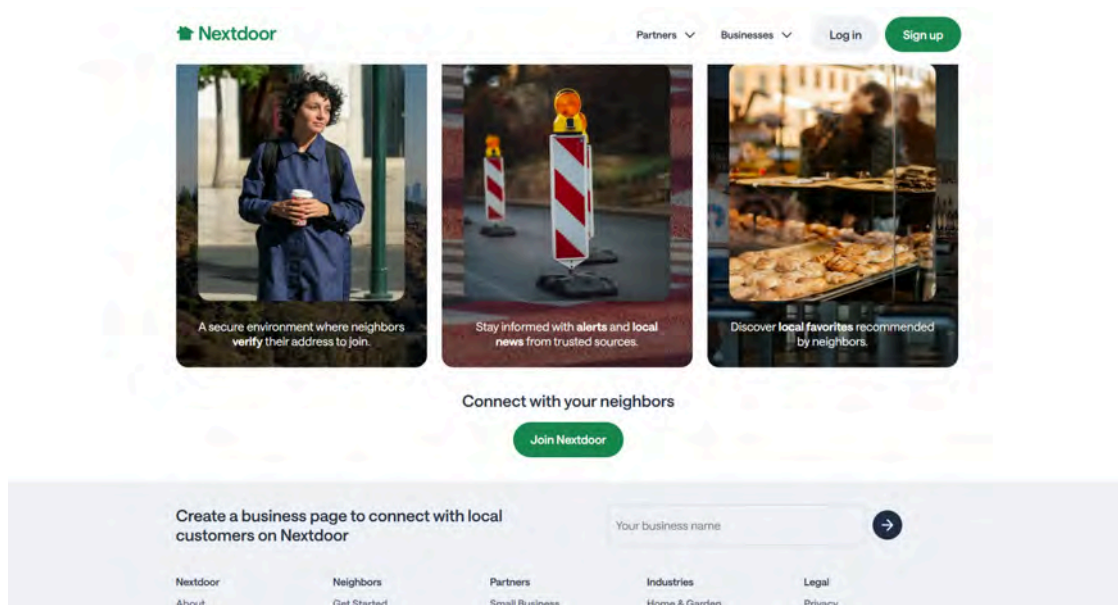


Рисунок 1.6 - Соціальна мережа Nextdoor [6]

Для координації великих спільнот часто застосовують Discord [7] (рис. 1.7). Цей сервіс надає потужні інструменти для комунікації та створення структурованих серверів. Проте його використання для організації саме офлайн-заходів ускладнюється хаотичністю інформації, або відсутності чи складності налаштування певних елементів інформаційних полів. Крім того, вагомим недоліком є повна відсутність зручної географічної прив'язки до локацій у форматі інтерактивної мапи, що робить просторове планування та візуальний пошук місця проведення заходу вкрай незручним для учасників.

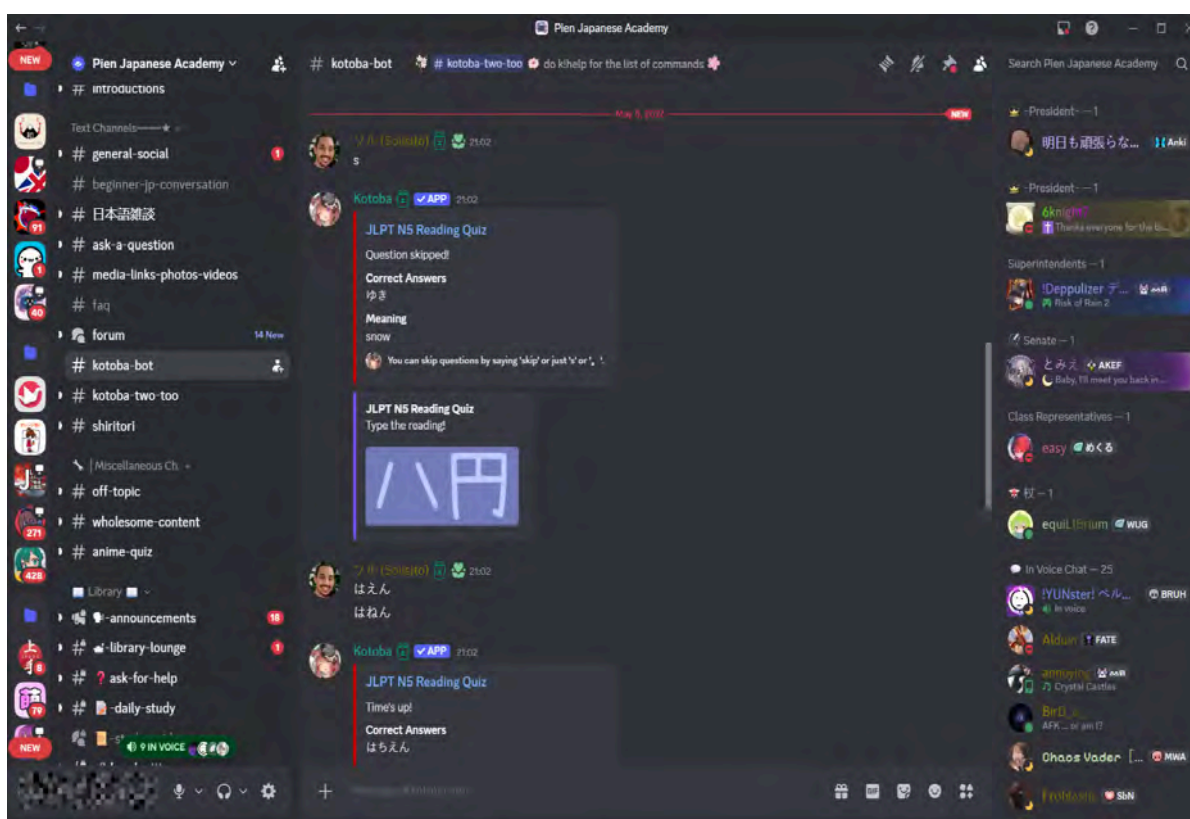


Рисунок 1.7 - Застосунок Discord [7]

Крім того, існують дуже спеціалізовані платформи. До них належать такі ресурси, як Tabletop Together [8] (рис. 1.8) або Tabletop Events [9] (рис. 1.9), створені спеціально для настільних ігор. Проте ці сервіси часто мають складний інтерфейс і фокусуються переважно на великих заходах та фестивалях, а не на локальних, невеликих посиденьках.

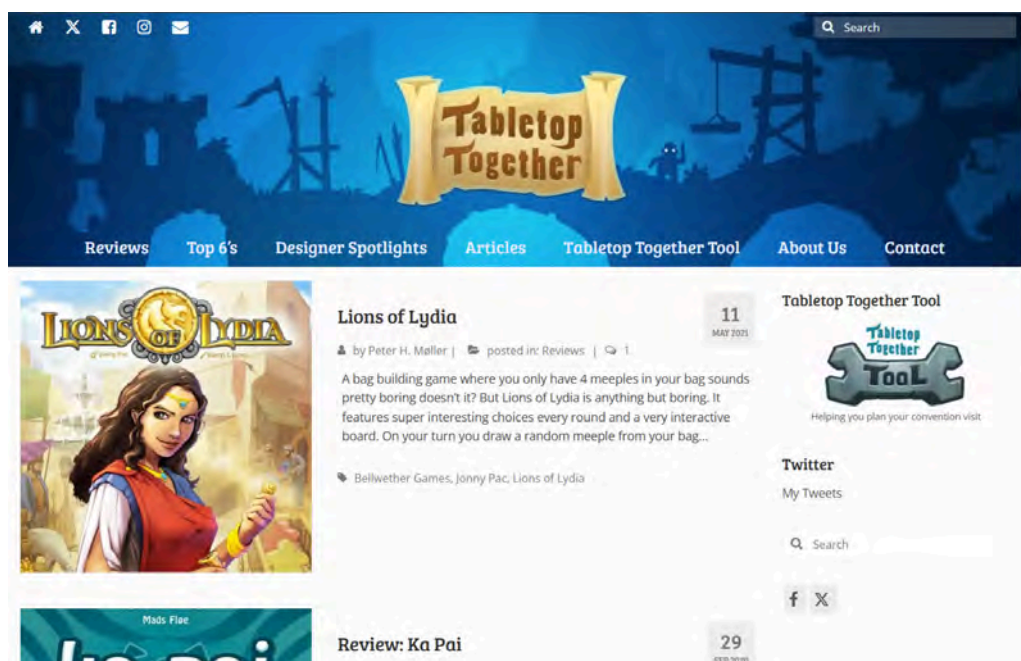


Рисунок 1.8 - Платформа Tabletop Together [8]

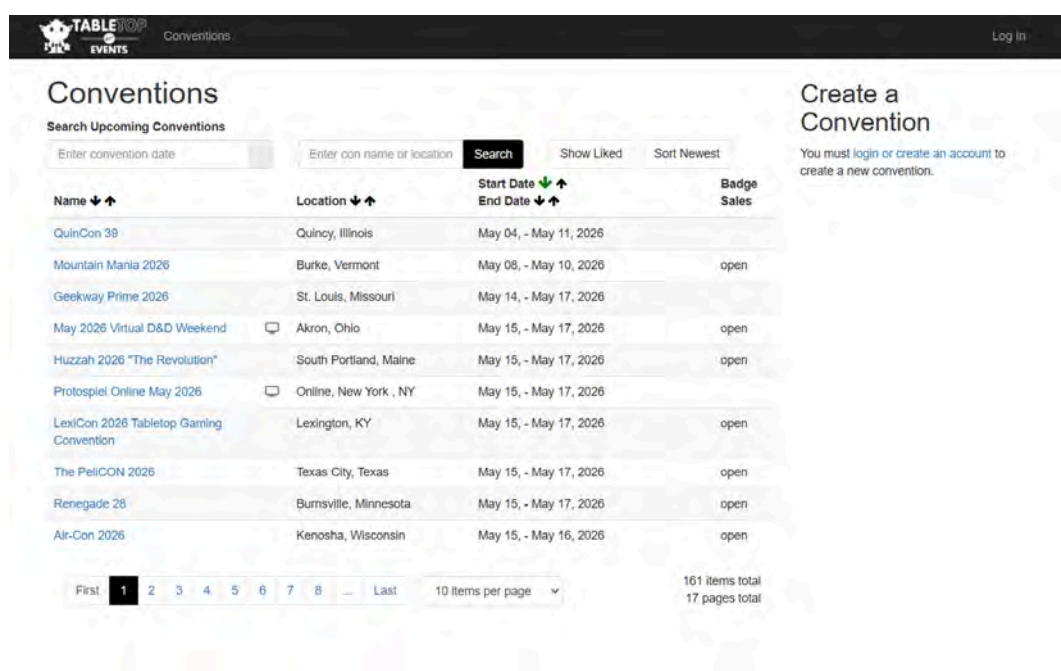


Рисунок 1.9 - Платформа Tabletop Events [9]

Ще вужчу спеціалізацію має застосунок Playgroup [10] (рис. 1.10), який сфокусований виключно на підборі гравців для колекційних карткових ігор, зокрема Magic: The Gathering. Хоча в межах своєї ніші він пропонує зручний інструменти для гравців, така жорстка прив'язка до одного

специфічного жанру критично обмежує його використання. Таким чином платформа залишає поза увагою величезний сегмент аудиторії які є прихильниками класичних настільних ігор читацьких клубів та інших типів офлайн-заходів.

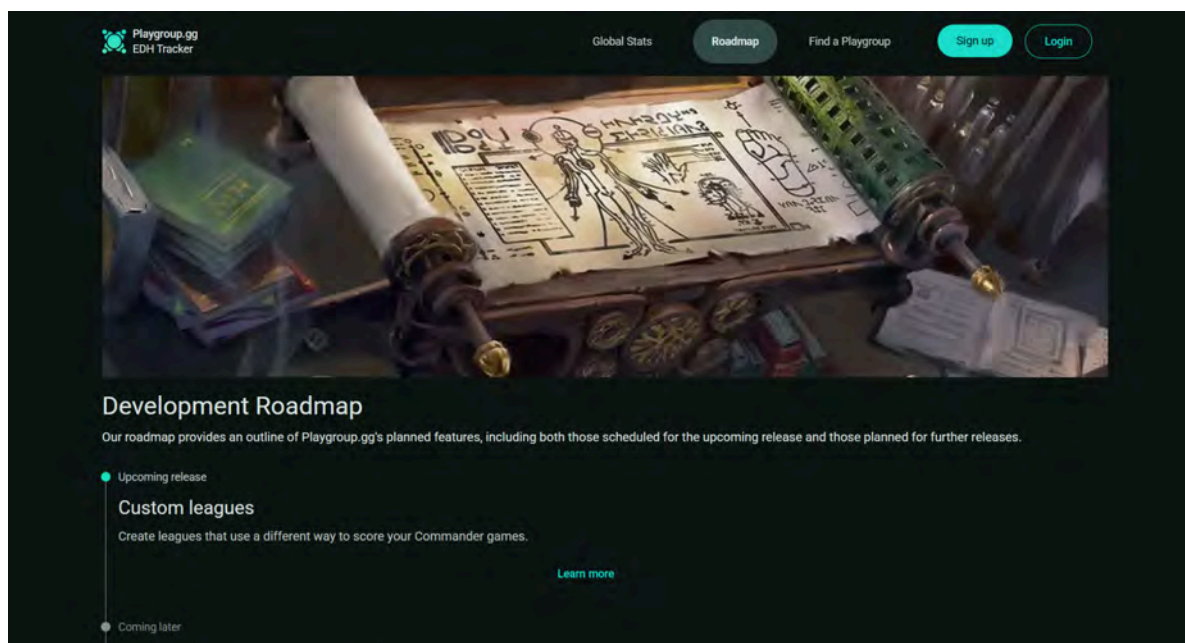


Рисунок 1.10 - Застосунок Playgroup [10]

Для прикладу, у сегменті освітніх та культурних заходів існують спеціалізовані рішення, такі як застосунки для пошуку читацьких клубів, що фокусуються на зустрічах за однією конкретною тематикою. Проте, незважаючи на їхню вузьку спрямованість, такі платформи часто не інтегрують гнучких механізмів геопошуку, що ускладнює організацію зустрічей у нових локаціях, та обмежують функціонал лише однією тематикою, не покриваючи інші потенційні інтереси користувачів.

Нішеві платформи для настільних ігор, такі як інформаційна платформа Baduk Club [11] (рис. 1.11), спеціалізуються на об'єднанні гравців у Го (Вейці/Бадук). Цей ресурс вже сформував значну спільноту, що налічує понад 6 800 гравців та 842 клуби по всьому світу. Серед його безперечних переваг є наявність продуманих інструментів для організаторів (керування членством, цільові сповіщення про заходи, інструменти для проведення

турнірів), а також вдала реалізація інтерактивної цифрової мапи, яка дозволяє користувачам позначати себе для швидкого локального пошуку однодумців.

Проте, незважаючи на ефективне розв'язання важливих проблем, архітектура ресурсу залишається жорстко обмеженою рамками лише однієї гри і не може бути застосованим для широкого спектра офлайн-заходів, що є його ключовим обмеженням.

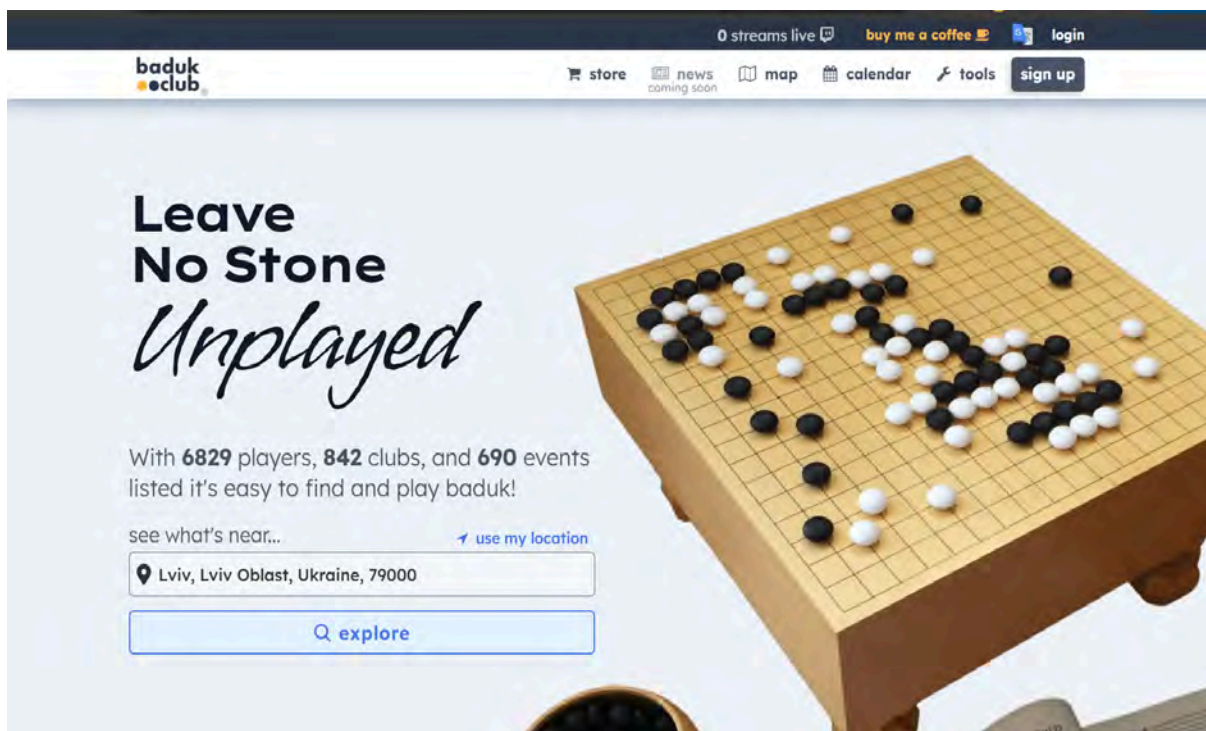


Рисунок 1.11 - Платформа Baduk Club [11]

Водночас, інший сегмент соціальної взаємодії охоплюють платформи для мовного обміну, такі як Tandem [12] (рис. 1.12). Такі платформи зусередженні на забезпеченні комунікації між носіями різних мов у реальному часі. Незважаючи на високу ефективність у сприянні спілкуванню людей, ці застосунки орієнтовані виключно на віртуальну взаємодію.

Зазвичай вони зосереджені на пошуку мовного партнера та не містять інтегрованих інструментів для планування спільних офлайн-заходів, створення локальних спільнот чи організації регулярних групових розмовних клубів офлайн й не містять можливостей для групової координації,

визначення локацій. Тому такі сервіси не можуть використовуватися як зручний інструмент для перенесення спілкування у реальний світ та організації офлайн-зустрічей.

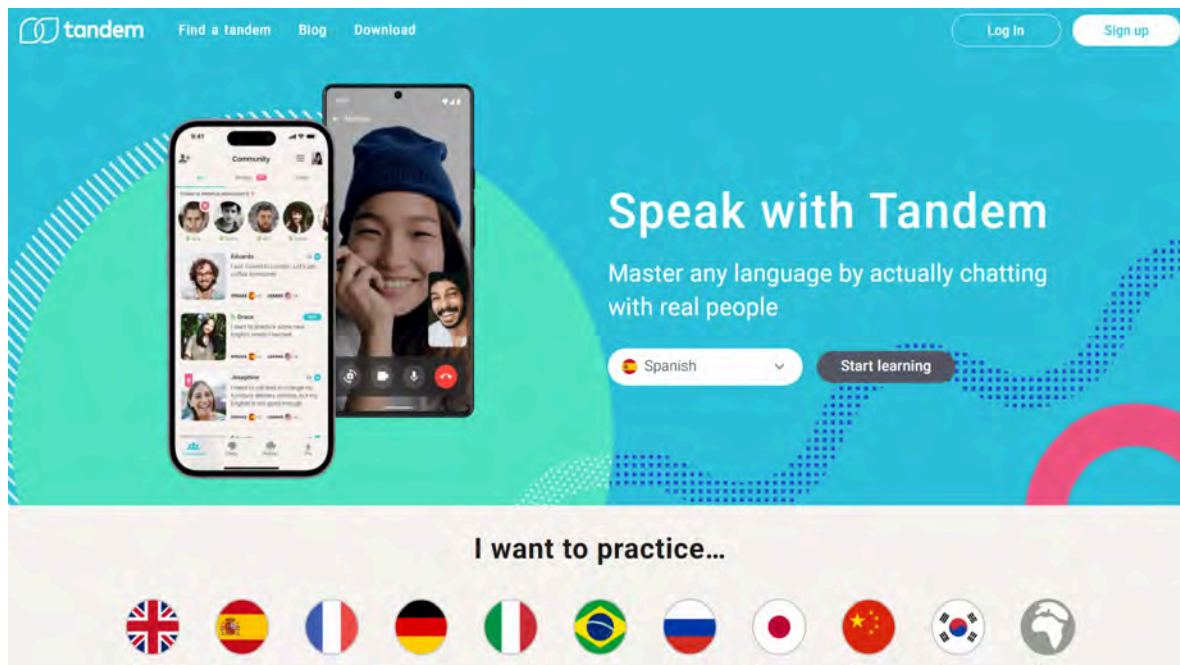


Рисунок 1.12 - Платформа Tandem [12]

Таким чином, аналіз існуючих рішень показав, що:

- існуючі платформи або орієнтовані на загальні заходи без урахування специфіки інтересів;
- або надають лише інформаційний контент без можливості організації заходів;
- або зусередженні виключно на одній тематиці;
- або не забезпечують зручного пошуку однодумців та автоматизації процесів керування заходами.

Це означає що розроблення спеціалізованого програмного забезпечення для організації офлайн-заходів, яке б поєднувало функції планування, реєстрації учасників та пошуку однодумців в межах єдиної системи є достатньо доцільним.

1.2 Порівняльний аналіз розробки та аналогів

До фінального порівняння були внесені застосунки Meetup та Eventbrite, які були порівняні з розробкою (табл. 1.1).

Таблиця 1.1 – Порівняння програм для організації офлайн-заходів та пошуку учасників

Критерій	Meetup	Eventbrite	Розробка
Визначення заходів	Як окремі заходи з текстовим описом	Як заходи з описом та квитками	Як заходи, що створюються користувачами з деталізацією
Фільтрація за інтересами	Загальні категорії	Обмежена	Назва, місце, дата, категорії, учасники
Керування учасниками	Є	Є (з оплатою)	Є
Відображення деталей заходу	Текстовий опис	Текстовий опис	Структуровані дані в окремих полях
Можливість створення власних заходів	Так	Так	Так
Комунікація між учасниками	Обмежена	Обмежена	Чат заходу та особистий
Автоматизація організації	Часткова	Часткова	Реєстрація, керування учасниками, планування

Порівняння показало, що існуючі аналоги не враховують специфіку заходів за інтересами. У застосунках Meetup та Eventbrite заходи представлені переважно у вигляді загальних описів без глибокої структуризації.

Фільтрація за інтересами є обмеженою, що ускладнює пошук відповідних заходів для користувачів. Крім того, комунікація між учасниками реалізована частково, а автоматизація процесів організації заходів не є повною.

На відміну від аналогів, запропонована розробка передбачає спеціалізацію на офлайн-заходах за інтересами, що дозволяє більш точно враховувати потреби користувачів. Реалізація структурованого підходу до опису заходів, розширеної фільтрації та автоматизації процесів керування дозволяє покращити організацію заходів та взаємодію між учасниками.

Однак порівняння лише двох сайтів не буде об'єктивною перевіркою, тому вважаю за необхідним провести додаткові дослідження використовуючи ширші критерії для вибору аналогів.

В якості критерії оцінювання та порівняння в ширших аналогах вважаю за необхідність взяти їхню основну специфікацію та функціонал й наскільки вони дозволяють створювати групові офлайн-заходи.

Окремими пунктами вважаю за необхідне винести те наскільки інформація цих застосунків є структурованою й дозволяє в одному стилі швидко орієнтуватися в декількох заходах основну інформацію про них, та наявність в цих програмах способів географічного відображення місця проведення заходу, що є дуже важливою частиною для швидкої орієнтації на місцевості.

Для продовження цього порівняльного аналізу розглянемо наведені нижче таблиці. На них відображають, яким чином нова розробка буде виправляти ключові недоліки інших типів аналогів, орієнтованих на загальну соціальну взаємодію та нішеві захоплення користувачів. Почнемо з порівняння програм загального призначення (табл. 1.2).

Таблиця 1.2 – Порівняння програм загального призначення

Критерій	Facebook	Discord	Nextdoor	Розробка
Основна спеціалізація та комунікація	Загальні заходи	Лише онлайн	Лише у межах району	Організація офлайн-заходів по інтересах на будь якій відстань
Організація групових заходів	Базові функції	Лише онлайн спілкування	Вузький фокус, що ускладнює пошук	Автоматизований процес створення заходу з підтримкою комунікації
Структура даних	Просто текст	Повна відсутність структури	Нема структури даних про захід	Деталізація заходів та структуровані дані у полях
Географічна прив'язка	Без пошуку на карті	Повна відсутність карти	Обмежена лише межами одного району	Гнучкий механізм геопошуку та інтеграція інтерактивної карти для точної локалізації

Далі розглянемо порівняння програм пошуку соціальних зв'язків (табл. 1.3).

Таблиця 1.3 – Порівняння програм пошуку соціальних зв'язків

Критерій	Bumble BFF	Tandem	Розробка
Основна спеціалізація	Пошук особистих контактів	Мовний обмін онлайн	Спеціалізована організація групових офлайн-заходів за інтересами
Організація групових заходів	Орієнтація на особисті зустрічі	Інструменти планування спільних офлайн-заходів відсутні	Спеціалізований функціонал для координації та організації групових офлайн-заходів
Структура даних	Дані про заходи відсутні	Дані про заходи відсутні	Деталізація заходів та структуровані дані у спеціальних полях
Географічна прив'язка	Відсутність інструментів планування дати/місця всередині застосунку	Відсутність інструментів геопошуку для локальних зустрічей	Гнучкий механізм геопошуку та інтеграція карти для точної локалізації та автоматичне створення чату заходу

Далі розглянемо порівняння нішевих та дуже спеціалізованих платформ (табл. 1.4).

Таблиця 1.4 – Порівняння нішевих та дуже спеціалізованих платформ

Критерій	Board Game Geek	Tabletop Together / Tabletop Events	Playgroup / Baduk Club	Розробка
Тип платформи та фокус	База даних та форум	Великі з'їзди та фестивалі	Жорстка вузька спеціалізація на одній грі/тематичі	Універсальне програмне забезпечення для локальних, офлайн заходів
Функціонал організації заходів	Орієнтація на контент, відсутні інструменти для створення офлайн заходів	Часто мають складний інтерфейс	Лише пошук, без координації	Планування, автоматична реєстрація, керування учасниками та чатами заходів
Обмеження гнучкості	Не орієнтоване на соціальну взаємодію та пошук однодумців	Обмеження лише тематикою настільних ігор	Обмеження лише однією тематикою	Підтримка широкого спектра інтересів

1.3 Висновки за розділом

Висновками цього розділу є сформовані завдання роботи та опис проведеного аналізу.

Завдання роботи:

- проаналізувати програмне забезпечення для організації офлайн-заходів та пошуку учасників;
- визначити методи та засоби проєктування програмного забезпечення;
- сформулювати вимоги до системи організації офлайн-заходів;
- виконати реалізацію програмного забезпечення для керування заходами та взаємодії користувачів.

Аналіз, що було проведено серед існуючих програмних рішень показав, що вони не є достатніми поодиночі, оскільки або орієнтовані на загальні заходи не враховуючи інтересів користувачів, або не забезпечують достатньої автоматизації процесів організації заходів.

Було встановлено, що застосунки загального призначення пропонують обмежену функціональність, подають інформацію про заходи переважно у вигляді текстових описів без належної структуризації даних, а їхня фільтрація за вузькими інтересами є недостатньою.

Платформи для соціальних зв'язків орієнтовані на індивідуальні зустрічі або онлайн-комунікацію і не мають спеціалізованих інструментів для координації та планування групових офлайн-заходів, а також не забезпечують зручної географічної прив'язки.

Нішеві та дуже спеціалізовані ресурси або сфокусовані лише на інформаційному контенті, або жорстко обмежені однією тематикою, що унеможлиблює їх універсальне застосування для широкого спектра захоплень.

Таким чином, виявлені недоліки основними з яких є обмежена фільтрація, недостатня підтримка комунікації між учасниками, відсутність

спеціалізації на групових офлайн-заходах та гнучкого механізму геопошуку обґрунтовують доцільність розроблення спеціалізованого програмного забезпечення, яке зменшить час організації заходів та скорочення витрат ресурсів організаторів за рахунок централізованої системи, структуроване та керування даними заходів, та автоматизації ключових процесів, що і буде реалізовано в наступних розділах роботи.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір засобів проєктування

Вебзастосунок було розроблено на основі вебфреймворку, використовуючи його як каркас для подальшої розробки. Мовою програмування була обрана зручна та звична для вебзастосунків мова JavaScript [14]. В якості ВФ до неї було проведено порівняльну роботу між React [15] та Vue [16]. На основі порівняння можливих рішень (табл. 2.1) було обрано React, як інструмент для створення динамічного та адаптивного користувацького інтерфейсу.

Таблиця 2.1 – Вибір засобів для розробки програми

Критерій	React	Vue
Екосистема	Найбільша зріла спільнота, багато інструментів.	Менша, але зростаюча спільнота, "монолітні" рішення.
Гнучкість	Максимальна гнучкість, кастомна архітектура.	Більш стандартизований підхід.
Синтаксис	Використовує JSX (JS + розмітка).	Використовує окремі HTML-шаблони.

Була використана клієнт-серверна архітектура, що забезпечує розділення логіки та підвищує гнучкість, масштабованість та швидкість взаємодії користувача з інтерфейсом.

Для швидкої розробки та збирання проєкту використаємо Vite [17]. Для інтерактивного визначення локації заходів застосовано бібліотеку українського виробництва Leaflet у зв'язці з React-Leaflet [18], що дозволяє безкоштовно й зручно відображати місце проведення та при необхідності підтягувати назву цієї місцевості.

Серверну частину логіки було вирішено реалізовувати на Node.js [19], що дає змогу писати серверну частину тією ж мовою JS яку ми обрали для клієнтської частини застосунку. Як мінімалістичний вебфреймворк для створення REST API обрано Express [20] що в поєднанні з Axios [21] робить роботу зі шляхами дуже зручною та уникати повторень в адресі шляхів.

Для забезпечення комунікації в реальному часі, такої як чати, миттєві сповіщення чи важливі дії використаємо Socket.IO [22], що використовує екосистему Node.js та ідеально інтегрується в JavaScript.

Для автентифікації та забезпечення безпеки акаунтів в системі застосовуються JWT-токени (jsonwebtoken) та хешування паролів за допомогою bcryptjs.

Як система керування базами даних (СКБД) обрано PostgreSQL [23] через його надійність та підтримку складних SQL-запитів які можуть знадобитись під час розробки. PostgreSQL також надає механізми CHECK-обмежень [24] для додаткової валідації даних на рівні бази.

2.2 Сценарії використання програми

У програмі є дві головні ролі: Звичайний користувач і Модератор. Така структура потрібна, щоб контролювати й захищати контент.

Звичайний користувач має повний доступ до соціальних функцій, керування профілем та своїми заходами:

- реєстрація користувача;
- авторизація в існуючий акаунт;
- редагування профілю користувача;
- створення редагування та видалення власних заходів;
- приєднання та вихід з всіх доступних заходів;
- відправлення повідомлень у групові чати заходів та особисті повідомлення;
- відправка/прийняття запитів на дружбу та видалення з друзів;

- блокування/розблокування користувачів;
- фільтрація, пошук та перегляд списку заходів з пагінацією.

Модератор має права прирівняні до власників івентів але не може створювати свої та виконує фільтраційні, валідаційні та роз'яснювальні функції з можливістю повного бану користувача, а саме:

- бан будь-якого користувача;
- кік учасника з будь-якого заходу;
- редагування та видалення будь-яких заходів;
- видалення будь-яких повідомлень у чатах.

Діаграма, що відображає функціональні вимоги від програми керування офлайн-заходами для обох ролей, представлена на рис. 2.1.

На цій діаграмі видно всі функціональні вимоги системи для обох ролей. Усередині прямокутника «Offline events hub / Система» зображені обидва актори й їхні функції: зліва звичайний користувач, що є основним ініціатором дій, справа модератор із окремими лініями, які вказують на особливі привілеї контролю контенту.

Вгорі діаграми знаходяться загальні функції, такі як реєстрація, вхід і редагування профілю. Вони доступні для всіх типів користувачів, як єдиний механізм доступу в систему.

Посередині розміщені основні сценарії: приєднання до заходів, спілкування у чатах, пошук та фільтрація. Всі ці дії пов'язані зі звичайним користувачем. Окремо виділено створення власного заходу, де звичайний користувач отримує права модератора в межах свого заходу.

У нижній частині діаграми показані функції для модератора: бан користувачів, керування чужими заходами і видалення повідомлень.

Ця схема добре показує, що більшість функцій відкриті обом ролям. Модератор теж є користувачем. Модератори просто мають ще додаткові повноваження, недоступні звичайному користувачеві, для контролю та підтримки порядку. Таким чином базовий функціонал є доступним для всіх в

той час як більш спеціалізовані дії доступний лише для ролей яким він потрібен.

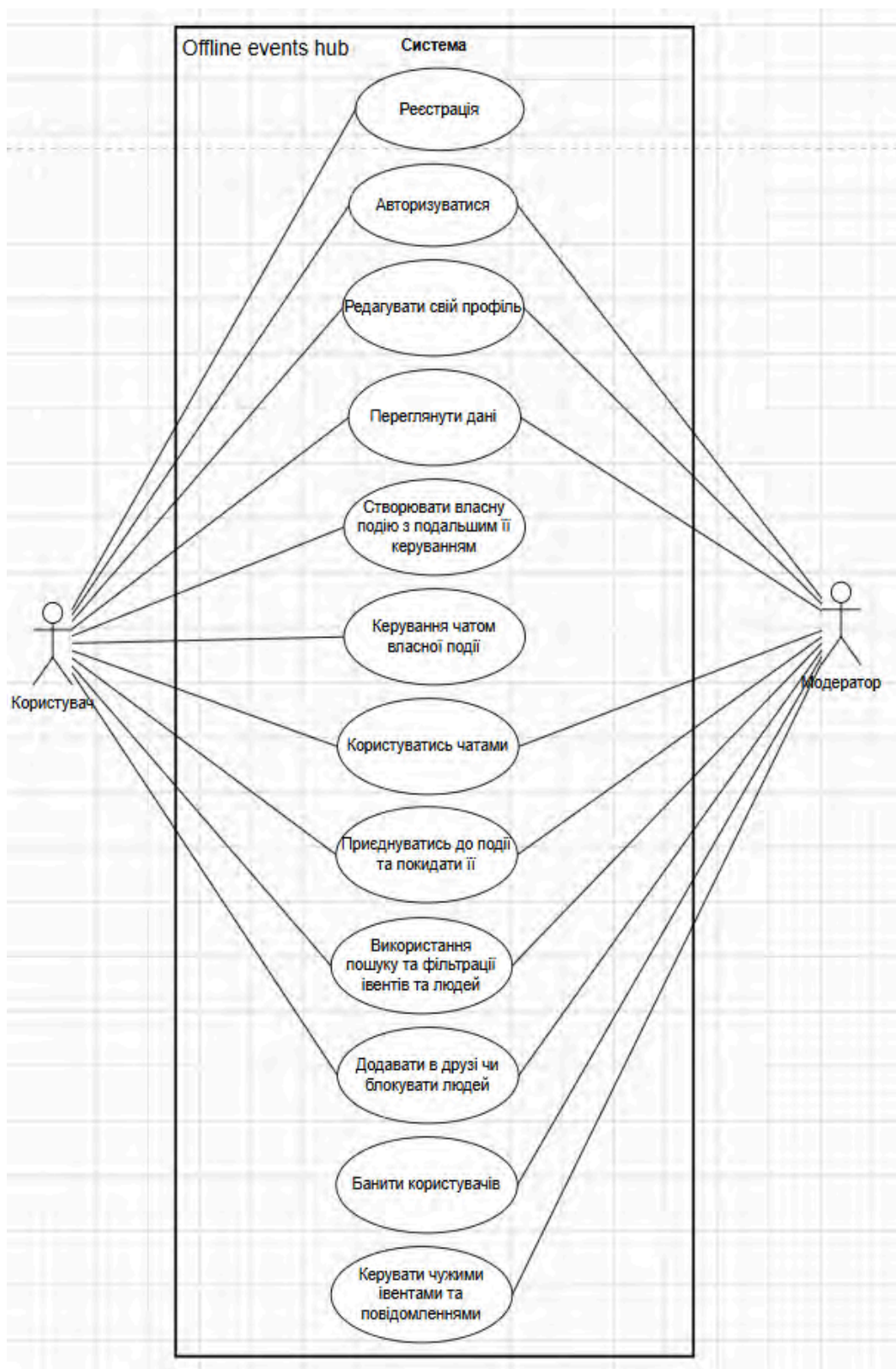


Рисунок 2.1 – Use-case діаграма

2.3 Структура бази даних

Обрана PostgreSQL має об'єктно-реляційну структуру. Це означає, що дані в ній організовані у вигляді таблиць зі строго визначеними типами стовпців, а зв'язки між ними будуються за допомогою зовнішніх ключів, але водночас вона підтримує масиви, деякі складні типи даних тощо. Структура БД складатиметься з п'яти основних таблиць, які охоплюють усі функціональні вимоги, що нам необхідні:

Таблиця БД Users визначає користувача програми зі стовпцями:

- id – ідентифікатор користувача програми;
- phone_number – номер телефону користувача у форматі +380XXXXXXXXX, який є унікальним для кожного користувача;
- full_name – повне ім'я користувача програми;
- nickname – нікнейм (псевдонім), за яким відомий цей користувач у програмі, який є унікальним серед усіх користувачів;
- age – вік користувача програми;
- photo_url – посилання на фотографію (аватар) користувача програми;
- bio – текстовий опис профілю користувача програми, що презентує його іншим користувачам;
- email – адреса електронної пошти користувача, яка є унікальною серед усіх користувачів і використовуватиметься для автентифікації;
- password_hash – хешований пароль користувача для забезпечення безпечної автентифікації;
- role – роль користувача в програмі, що визначає рівень його привілеїв та може набувати значень 'USER' або 'MODERATOR';
- is_active – позначення того, чи знаходиться користувач в даний момент у програмі;
- is_banned – позначення того, чи заблокований цей обліковий запис модератором, чи ні;

- created_at – дата та час створення облікового запису користувача;
- updated_at – дата та час останнього оновлення профілю користувача.

Поле role має обмеження CHECK, тож можна вибрати лише дві ролі: USER чи MODERATOR. Це дає змогу чітко розділити права доступу – звичайний користувач або модератор, який може адміністративно керувати іншими та заходами.

Таблиця БД Events визначає захід, створений користувачем програми, зі стовпцями:

- id – ідентифікатор заходу;
- creator_id – користувач, який створив цей захід у програмі;
- title – назва заходу;
- description – детальний опис заходу, що розкриває його суть та деталі;
- location – текстова адреса місця проведення заходу;
- latitude – географічна широта місця проведення заходу на карті;
- longitude – географічна довгота місця проведення заходу на карті;
- photo_url – посилання на зображення, яке ілюструє цей захід;
- event_date – дата та час проведення заходу;
- max_participants – максимальна кількість учасників, яку може прийняти цей захід;
- status – поточний стан заходу, який може набувати значень: upcoming (запланований), ongoing (відбувається), completed (завершений) або cancelled (скасований);
- is_private – позначення того, чи є цей захід приватним (видимим лише для друзів організатора), чи публічним;
- category – категорія заходу, що визначає його тематичну спрямованість;
- created_at – дата та час створення заходу в програмі;
- updated_at – дата та час останнього оновлення інформації.

Поле `creator_id` зв'язує кожен захід із конкретним користувачем із таблиці `Users`. Це означає: один захід має одного автора. А ось один користувач може створити скільки завгодно заходів чи не створювати жодного це типовий “один до багатьох”. Коли користувача видаляють, всі його заходи автоматично зникають разом із ним.

Поле `status` має обмеження типу `CHECK` яке забезпечує що статус заходу може бути тільки одним із чотирьох варіантів: `upcoming`, `ongoing`, `completed`, або `cancelled`.

Реєстрація участі користувачів у заходах визначена таблицею `Event_participants` зі стовпцями:

- `id` – ідентифікатор реєстрації участі користувача в заході;
- `event_id` – захід, на який зареєструвався користувач;
- `user_id` – користувач, який зареєструвався на участь у заході;
- `status` – поточний стан заявки на участь, який може набувати значень `pending` (на розгляді), `approved` (підтверджено організатором), `rejected` (відхилено організатором) або `left` (користувач самотійно покинув захід);
- `registered_at` – дата та час реєстрації участі користувача.

Поле `event_id` є зовнішнім ключем до `Events`, тому кожна реєстрація стосується конкретного заходу, хоча один захід може збирати багатьох учасників, або взагалі не зібрати нікого. Якщо сам захід видалили, всі пов'язані з ним записи про учасників також видаляються автоматично.

Поле `user_id` є зовнішнім ключем до `Users`. Реєстрація прив'язана до конкретного користувача, але той користувач може брати участь у багатьох заходах, або жодному, тобто знову “один до багатьох”. Якщо видалити то зникнуть всі записи його участі.

У парі `event_id` та `user_id` є обмеження унікальності: той самий користувач не зможе зареєструватись у тому самому заході двічі.

Ця таблиця також має обмеження `CHECK` для статусу участі тому він може бути тільки одним із чотирьох: `pending`, `approved`, `rejected`, або `left`. Будь-які інші значення не пройдуть.

Повідомлення в програмі визначені таблицею Messages зі стовпцями:

- id – ідентифікатор повідомлення;
- sender_id – користувач, який надіслав це повідомлення;
- event_id – захід, до чату якого надіслано це повідомлення (якщо повідомлення є частиною групового чату заходу);
- receiver_id – користувач, якому адресовано це персональне повідомлення (якщо повідомлення є частиною особистого листування);
- text – текстовий вміст повідомлення;
- created_at – дата та час створення (відправлення) повідомлення.

Таблиця Messages реалізує двоцільову модель повідомлень: якщо заповнено поле event_id, то повідомлення є частиною групового чату конкретного заходу, а якщо заповнено поле receiver_id, то повідомлення є персональним (прямим) повідомленням між двома користувачами.

Поле sender_id є зовнішнім ключем до таблиці БД Users, таким чином закріплюючи те, що кожне повідомлення може бути надіслане тільки одним користувачем, при цьому один користувач може створити від нуля і більше повідомлень, тобто використовуючи тип один-до-багатьох. При видаленні користувача-відправника всі його повідомлення також видаляються каскадно.

Поле event_id є зовнішнім ключем до таблиці БД Events, таким чином закріплюючи те, що кожне повідомлення в груповому чаті стосуватиметься тільки одного заходу, при цьому один захід може мати від нуля і більше повідомлень у своєму чаті, тобто використовуючи тип один-до-багатьох. При видаленні заходу всі повідомлення його чату також видаляються каскадно.

Поле receiver_id є зовнішнім ключем до таблиці БД Users, таким чином закріплюючи те, що кожне персональне повідомлення може бути адресовано тільки одному користувачу, при цьому одному користувачу може бути надіслано від нуля і більше персональних повідомлень, тобто використовуючи тип один-до-багатьох. При видаленні користувача-отримувача всі адресовані йому повідомлення також видаляються каскадно.

Відносини дружби та блокування між користувачами програми визначені таблицею `User_friends` зі стовпцями:

- `id` – ідентифікатор відношення між двома користувачами програми;
- `user_id` – користувач, який ініціював створення цього відношення (надіслав запит на дружбу або заблокував іншого користувача);
- `friend_id` – користувач, до якого направлено це відношення (отримувач запиту на дружбу або заблокований користувач);
- `status` – поточний стан цього відношення, який може набувати значень 'pending' (запит на дружбу очікує відповіді), 'accepted' (дружбу підтверджено обома сторонами) або 'blocked' (користувач заблокований ініціатором);
- `created_at` – дата та час створення цього відношення.

Поле `user_id` є зовнішнім ключем до таблиці БД `Users`, таким чином закріплюючи те, що кожне відношення між користувачами ініціюватиметься тільки одним користувачем, при цьому один користувач може ініціювати від нуля і більше відношень з іншими користувачами, тобто використовуючи тип один-до-багатьох. При видаленні користувача-ініціатора всі створені ним відношення також видаляються каскадно.

Поле `friend_id` є зовнішнім ключем до таблиці БД `Users`, таким чином закріплюючи те, що кожне відношення направлено тільки на одного користувача, при цьому на одного користувача може бути направлено від нуля і більше відношень від інших користувачів, тобто використовуючи тип один-до-багатьох. При видаленні цільового користувача всі направлені на нього відношення також видаляються каскадно.

Комбінація полів `user_id` та `friend_id` має обмеження унікальності (UNIQUE), що гарантує неможливість створення дублікатів відношень між одними й тими самими двома користувачами в одному напрямку.

Обмеження CHECK на поле status забезпечує те, що стан відношення може набувати лише одного з трьох визначених значень: pending, accepted або blocked.

Для підвищення продуктивності виконання запитів у базі даних створені наступні індекси:

- idx_events_creator_id – індекс на поле creator_id таблиці Events для прискорення пошуку заходів за їх творцем;
- idx_events_event_date – індекс на поле event_date таблиці Events для прискорення пошуку та сортування заходів за датою проведення;
- idx_events_category – індекс на поле category таблиці Events для прискорення фільтрації заходів за категорією;
- idx_events_is_private – індекс на поле is_private таблиці Events для прискорення фільтрації заходів за ознакою приватності;
- idx_event_participants_event_id – індекс на поле event_id таблиці Event_participants для прискорення пошуку учасників конкретного заходу;
- idx_event_participants_user_id – індекс на поле user_id таблиці Event_participants для прискорення пошуку заходів, на які зареєстровано конкретного користувача;
- idx_users_nickname – індекс на поле nickname таблиці Users для прискорення пошуку користувачів за нікнеймом;
- idx_messages_sender_id – індекс на поле sender_id таблиці Messages для прискорення пошуку повідомлень за відправником;
- idx_messages_event_id – індекс на поле event_id таблиці Messages для прискорення пошуку повідомлень конкретного чату заходу;
- idx_messages_receiver_id – індекс на поле receiver_id таблиці Messages для прискорення пошуку персональних повідомлень за отримувачем;
- idx_user_friends_user_id – індекс на поле user_id таблиці User_friends для прискорення пошуку відношень, ініційованих конкретним користувачем;

- `idx_user_friends_friend_id` – індекс на поле `friend_id` таблиці `User_friends` для прискорення пошуку відношень, направлених на конкретного користувача.

Схема БД, яка відображає реляційні зв'язки між цими таблицями (ER-модель), представлена на рис. 2.2.

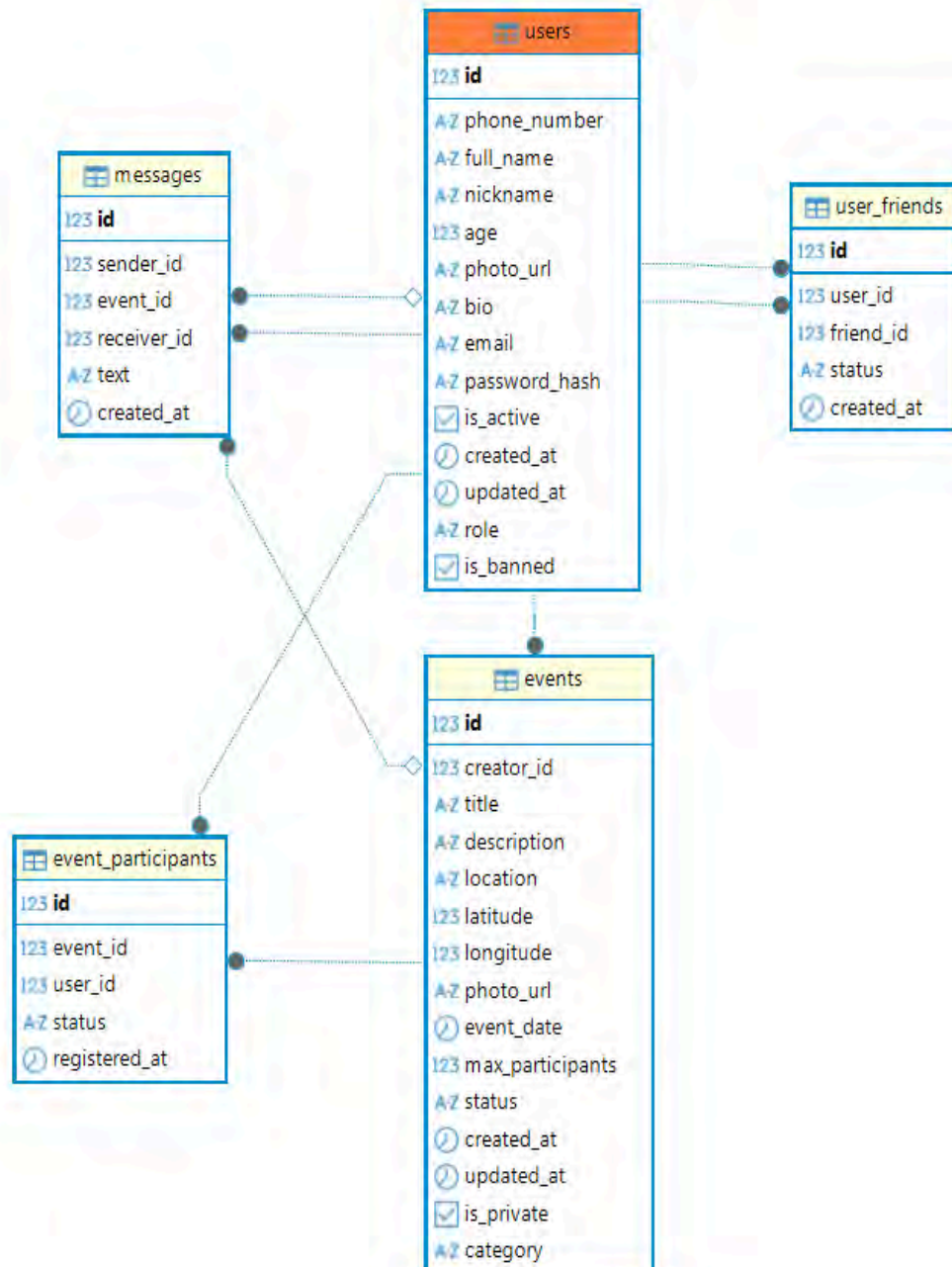


Рисунок 2.2 – ER-модель БД програми

2.4 Прототипи системи

Під час розробки інтерфейсу системи керування офлайн-заходами було створено базові прототипи wireframe головної сторінки. Вони чітко показують основні функціональні блоки та всю структуру інформації, яку містить продукт. Початковий стан інтерфейсу відображено на рис. 2.3.

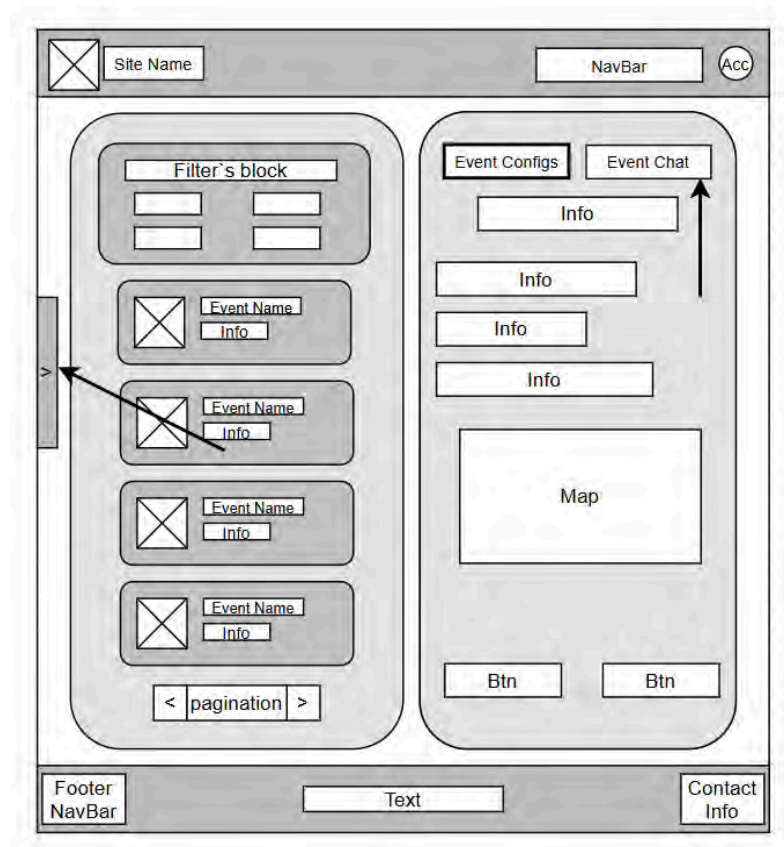


Рисунок 2.3 – Прототипування сторінки основного функціоналу

Тут зображено головну сторінку системи як вона є в самому початку. В контенті використали зручне двоколонкове розміщення.

Ліва частина робочої області відведена під інструменти пошуку та навігації. Вона містить комплексний блок фільтрів для точного налаштування параметрів вибірки та структурований вертикальний список карток заходів, кожна з яких виводить базову інформацію про захід. Для зручної навігації великими масивами даних передбачено механізм пагінації у знизу сторінки.

Права колонка екрана виступає областю детального перегляду обраного заходу. Тут можна побачити всі тонкощі заходу: текстові блоки, розширені дані, і найголовніше інтегровану карту. Для локалізації офлайн-заходів карта просто необхідна як зручне і зрозуміле відображення місцезнаходження заходу. Також у цьому блоці розміщені головні функціональні кнопки, що дозволяють користувачу безпосередньо взаємодіяти із заходом.

Глобальна навігація ресурсом забезпечується класичною верхньою зоною хедера із назвою сайту та доступом до профілю, а також інформаційним блоком у футері сторінки.

Слідуючи стрілкам, можемо побачити як змінюється інтерфейс, що відображено на рис. 2.4.

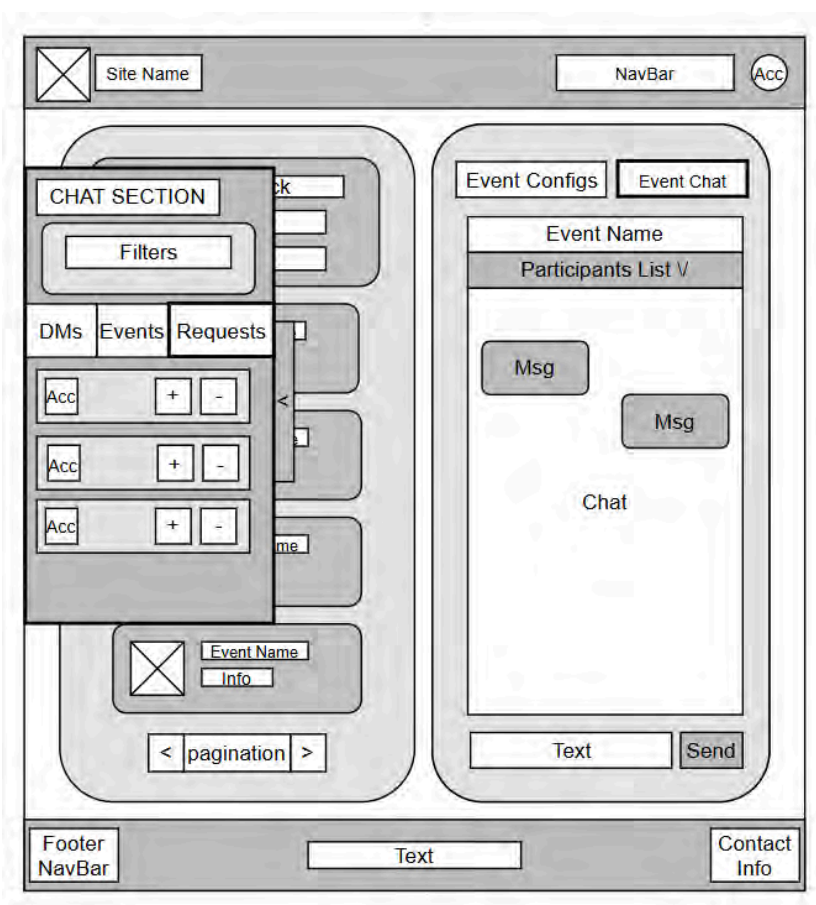


Рисунок 2.4 – Прототипування сторінки після відкриття чатів

Тут вже видно детальну реалізацію соціальної складової платформи: чати та список користувачів.

Оновлена робоча область включає назву поточного івенту, кнопку відображення списку учасників, простір для відображення історії повідомлень та фіксований блок введення тексту з кнопкою відправки.

З іншого боку після відкриття закріплена панель секції чатів частково перекриває лівий навігаційний блок, забезпечуючи швидкий доступ до комунікацій без необхідності повного переходу на іншу сторінку. Панель містить власні фільтри та систему вкладок для зручного розділення типів спілкування та взаємодії користувачів. Особисті повідомлення відповідають за комунікацію користувачів, чати конкретних заходів дублюють чати івентів у яких користувач може писати повідомлення, та запити на додавання, де передбачено зручний механізм керування контактами.

Такий підхід добре оптимізує робочий простір і дає змогу користувачам спілкуватися постійно, навіть під час пошуку чи координації заходів.

2.5 Схеми UML роботи системи

Процес функціонування інформаційної системи базуватиметься на чіткій взаємодії між клієнтською частиною, серверною логікою та базою даних, що дозволяє забезпечити стабільну роботу всіх сервісів у реальному часі.

Під час реєстрації нового користувача (рис. 2.5) система виконує багатоступеневу перевірку введених даних безпосередньо на стороні клієнта для мінімізації зайвих запитів до сервера. Після успішної валідації форми клієнтська частина відправляє POST-запит на сервер, де відбуватиметься фінальна перевірка унікальності електронної пошти, номера телефону та нікнейму. Якщо дані є коректними, пароль безпечно хешуватиметься за допомогою алгоритму bcrypt, а інформація про користувача записуватиметься

до бази даних PostgreSQL. У відповідь сервер генерує унікальний токен доступу JWT, який зберігатиметься у локальному сховищі браузера для подальшої автентифікації, а користувача перенаправляє на головну сторінку для початку роботи.

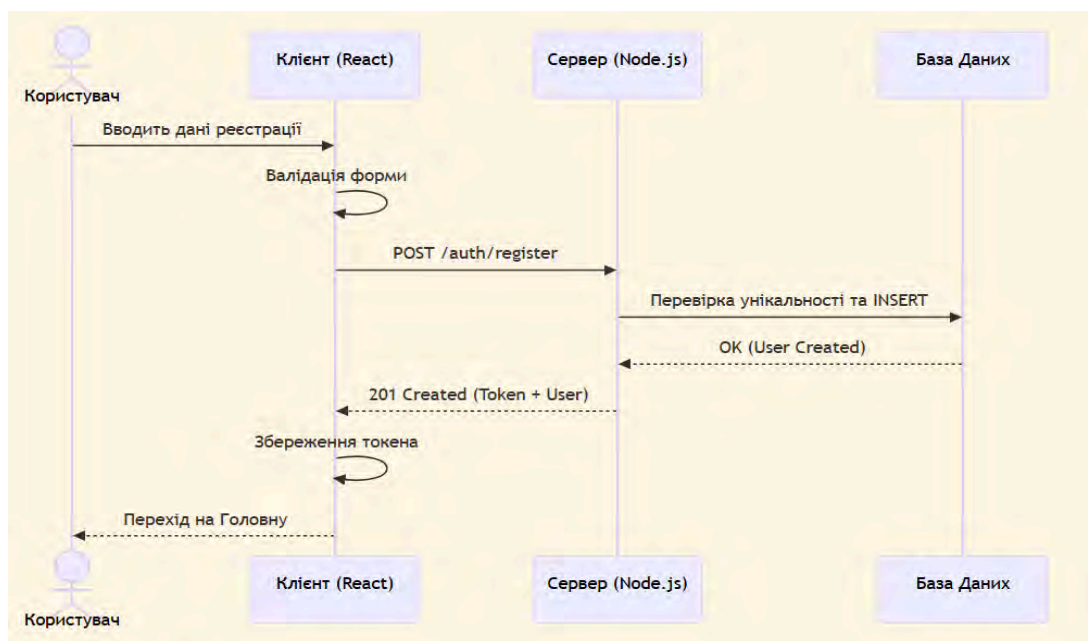


Рисунок 2.5 – Реєстрація користувача

Створення нового заходу (рис. 2.6) в системі організовано через спеціалізовану панель, де автор вказує всі необхідні деталі майбутнього заходу. Якщо користувач бажає додати ілюстрацію, система спочатку обробляє завантаження зображення через окремий шлях, де сервер перевіряє тип та розмір файлу за допомогою бібліотеки Multer [25], зберігає його на диску та повертає посилання на ресурс. При виборі локації на карті додаток фіксує точні координати, а після натискання кнопки збереження надсилає повний об'єкт даних на сервер. Сервер перевіряє права доступу користувача через JWT-токен та виконує операцію внесення запису в таблицю заходів. Після успішного завершення операції база даних повертає ідентифікатор нового заходу, а клієнтська частина оновлює інтерфейс, після чого івент стає доступним для пошуку та перегляду іншими учасниками спільноти.

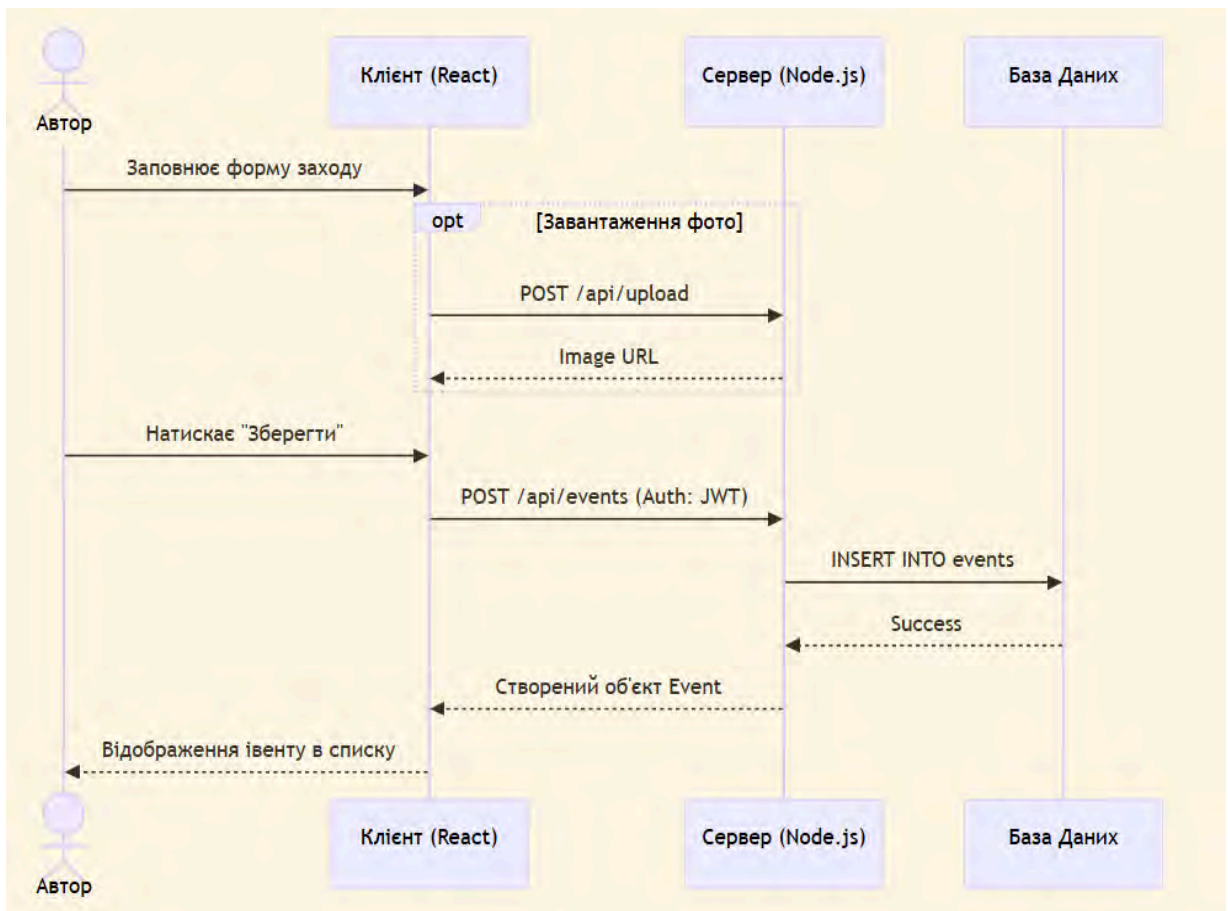


Рисунок 2.6 – Створення заходу

Особливе місце в архітектурі системи займає механізм обміну повідомленнями (рис. 2.7), який реалізовано за допомогою технології WebSockets [26] через бібліотеку Socket.IO. Коли користувач ініціює відправлення персонального повідомлення, клієнт не чекає стандартної відповіді від сервера, а надсилає подію безпосередньо через встановлене WebSocket-з'єднання. Серверний компонент отримує це повідомлення, паралельно записує його в базу даних для збереження історії та миттєво ретранслює об'єкт повідомлення одержувачу, якщо той перебуває онлайн. Отримувач бачить нове повідомлення в інтерфейсі без необхідності оновлювати сторінку, що створює ефект живого спілкування. Одночасно з цим система оновлює списки чатів у обох сторін, піднімаючи активний діалог вгору та відображаючи фрагмент останнього надісланого тексту.

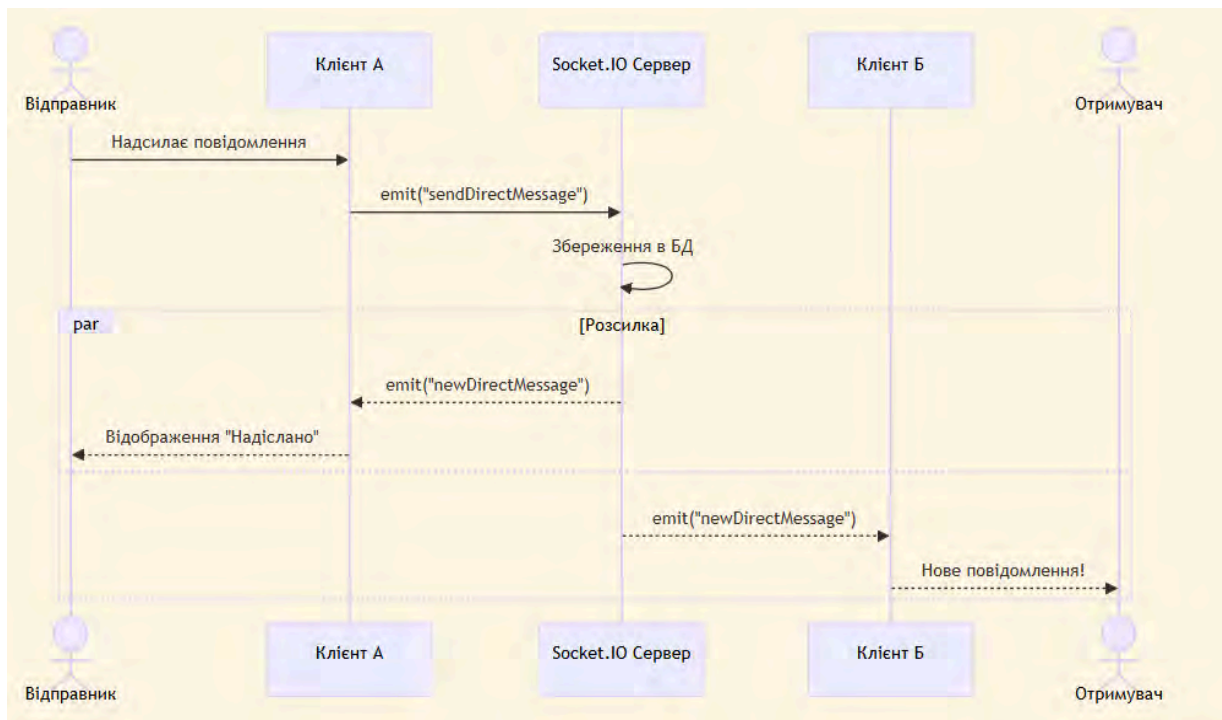


Рисунок 2.7 – Персональний чат

Процес приєднання користувача до обраного заходу (рис. 2.8) реалізовано за допомогою гібридного підходу, який поєднує класичні HTTP-запити для фіксації стану системи та механізми двосторонньої синхронізації в реальному часі через вебсокети. При відповідній натисканні кнопки участі система реєструє зв'язок між користувачем та івентом у базі даних, автоматично створюючи відповідний запис у таблиці учасників. Після цього сервер сповіщає всіх існуючих учасників про нового користувача, а сам клієнт автоматично підписуватиметься на спеціальну «кімнату» заходу в Socket.IO. Це дозволяє користувачу отримати доступ до групового чату заходу. Після цього система завантажує історію попереднього листування та в подальшому отримувати всі повідомлення та оновлення, що стосуються цього конкретного заходу. Такий підхід забезпечує розподіл відповідальності між протоколами, що забезпечує цілісність та надійність збереження даних та високу швидкість взаємодії між великою кількістю користувачів, у межах одного офлайн-заходу.

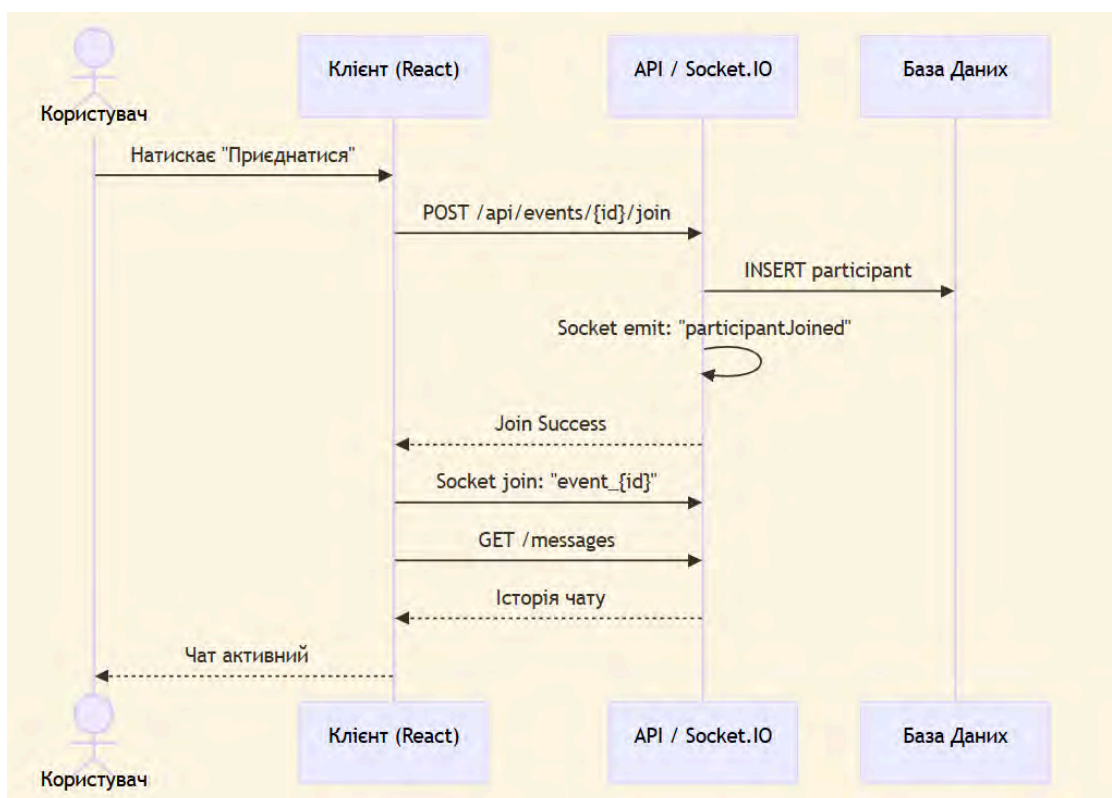


Рисунок 2.8 – Приєднання до заходу

Процес надсилання повідомлення в межах заходу (рис. 2.9) реалізовуватиметься через інтеграцію клієнтської частини, сокет-з'єднання та серверної логіки. Користувач вводить текст повідомлення у веб застосунку та натискає кнопку відправки, після чого React ініціює подію `sendMessage`, передаючи ідентифікатор заходу та сам текст через Socket.IO клієнт.

Отримавши цю подію, сервер на базі Express і Socket.IO обробляє запит і зберігає повідомлення в базі даних PostgreSQL, створюючи новий запис у таблиці повідомлень. Після успішного збереження сервер додатково отримує інформацію про відправника, щоб сформувати повноцінний об'єкт повідомлення.

Далі сервер виконує розсилку через Socket.IO, надсилаючи подію `newMessage` всім клієнтам, підключеним до відповідної «кімнати» заходу. На стороні клієнта ця подія оброблятиметься колбеком, який оновлює локальний стан повідомлень, додаючи нове повідомлення до списку.

У результаті повідомлення миттєво відображатиметься в чаті у всіх учасників заходу. Така архітектура забезпечує синхронізацію в реальному часі, мінімізує затримки та дозволяє ефективно обслуговувати одночасну взаємодію великої кількості користувачів.

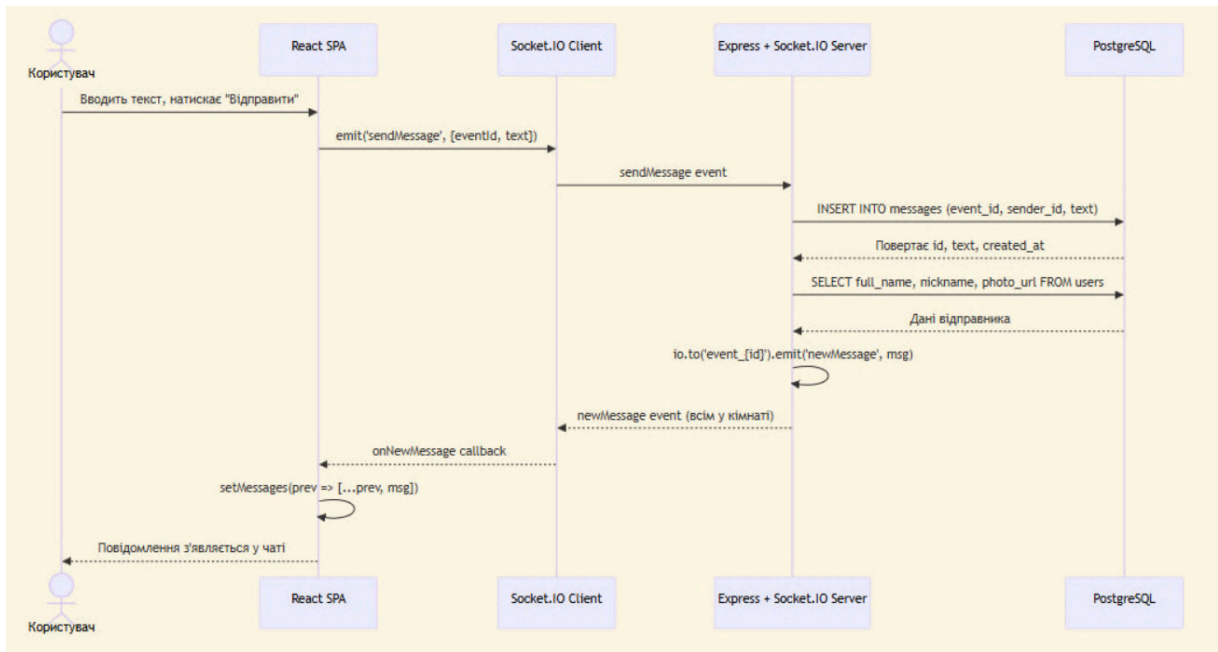


Рисунок 2.9 – Відправлення повідомлення в заході

2.6 Висновки за розділом 2

Обґрунтовано прийняття архітектурного рішення, що базуватиметься на використанні підходу з двоканальною комунікацією (REST API + WebSocket). Визначено набір технологій, включаючи React для клієнтської частини, Express.js для серверної частини та PostgreSQL як надійну реляційну СКБД.

Було детально описано та графічно представлено сценарії використання програми для двох основних ролей у вигляді Use-case, що чітко розмежовує їхні функціональні вимоги та можливості.

Сформовано структуру бази даних, адаптовану для керування учасниками, заходами, комунікацією та соціальними зв'язками. Це включає

п'ять основних таблиць із детальним описом їхніх стовпців, перевірок та зовнішніх ключів, а також набір індексів для оптимізації продуктивності запитів. Реляційні зв'язки між таблицями візуалізовані на ER-моделі.

Крім того, на етапі проєктування інтерфейсу розроблено базові wireframe-прототипи головної сторінки та комунікаційного модуля. Для забезпечення чіткого розуміння роботи системи в реальному часі, описані та проілюстровані схемами взаємодії ключові процеси функціонування системи, зокрема реєстрація користувачів, створення заходу, приєднання до заходу та механізми чатів та взаємодії користувачів.

Таким чином, сформовано етап проєктування, що закладає необхідну, деталізовану основу для реалізації всіх функціональних модулів системи.

3 РЕАЛІЗАЦІЯ ПРОГРАМИ

3.1 Визначення порядку роботи програми

Функціональна схема програми організації та керування офлайн-заходами базується на системі інтерактивного меню, яке пропонується користувачеві для роботи, починаючи з перегляду стрічки актуальних заходів та подальшим вибором відповідного функціонального розділу як керування заходами, повідомленнями та соціальна взаємодія з поверненням до головного вікна після завершення кожного сценарію використання (рис. 3.1).

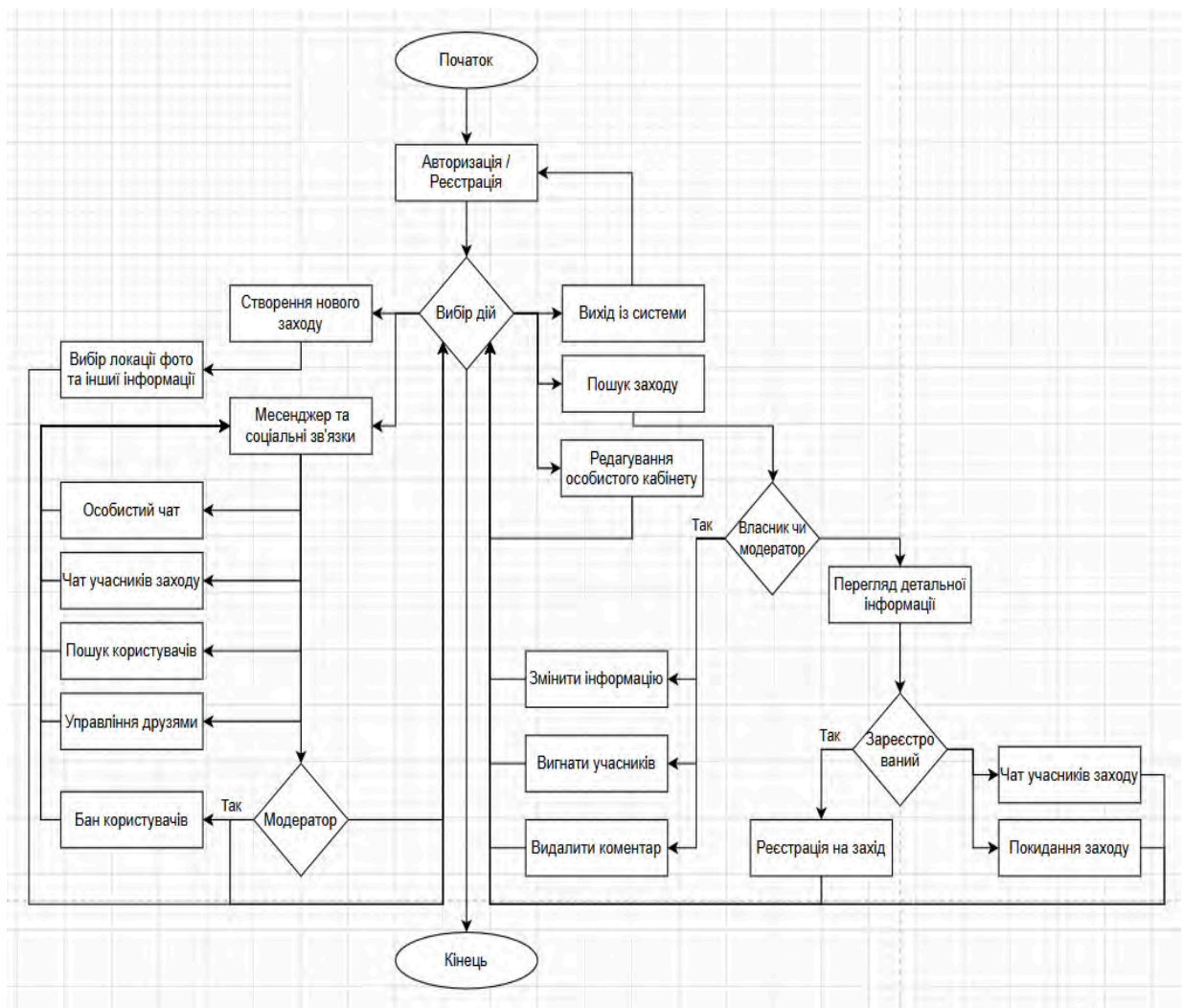


Рисунок 3.1 – Функціональна схема програми

3.2 Результати реалізації програмних одиниць застосунку

У лістингу 3.1 представлено основу налаштувань серверної частини. Наведений фрагмент коду базується на використанні середовища Node.js та фреймворку Express. Основною особливістю даного блоку є створення сервера, що одночасно обробляє стандартні HTTP-запити та підтримує постійні з'єднання через протокол WebSocket.

Важливим аспектом безпеки є наведений фрагмент коду механізму `io.use`. Це функція-посередник, яка виконується на рівні протоколу рукописання. Вона перехоплює JWT-токен користувача, переданий з фронтенду, та верифікує його за допомогою секретного ключа. Якщо токен неправильний, з'єднання обривається ще до моменту виділення системних ресурсів під сокет, що захищає сервер. При успішній автентифікації дані користувача прикріплюються до об'єкта сокета, що дозволяє іншим функціям точно ідентифікувати відправника без повторних запитів до бази даних.

Лістинг 3.1 - Каркас серверної частини

```
const app = express();
const PORT = process.env.SERVER_PORT || 3000;
const httpServer = createServer(app);
const io = new Server(httpServer, {
  cors: { origin: '*',
    methods: ['GET', 'POST', 'PUT', 'DELETE']
  }
});
app.use(cors());
app.use(express.json());
app.use('/uploads',
express.static(path.join(path.resolve(), 'uploads')));
io.use((socket, next) => {
  const token = socket.handshake.auth.token;
  if (!token) {
    return next(new Error('Authentication error: Token
missing'));
  }
  try {
    const decoded = verifyToken(token);
    socket.user = decoded;
    next();
  } catch (err) {
```

```

        return next(new Error('Authentication error: Invalid
token'));
    }));
    httpServer.listen(PORT, async () => {
console.log(`✓ Server running on http://localhost:${PORT}`);
    try {
        const res = await pool.query('SELECT NOW()');
console.log(`✓ Database connected: ${res.rows[0].now}`);
    } catch (err) {
console.error(`✗ Database connection failed!`, err.message);
    }));

```

Фрагмент коду, наведена у лістингу 3.2, відповідає за забезпечення інтерактивної взаємодії користувачів. Наведений фрагмент коду базується на концепції кімнат. Це дозволяє уникнути передачі надлишкових даних користувачам, які не є учасниками даного заходу.

Обробник `sendMessage` демонструє транзакційний підхід до обробки даних. Спочатку виконується асинхронний запит до бази даних через пул з'єднань. Це забезпечує надійність збереження історії листування. Після успішного запису сервер формує розширений об'єкт `newMessage`, куди додаються метадані відправника, які беруться з кешу або через швидкий `SELECT`-запит. Використання методу `io.to().emit()` гарантує, що повідомлення буде доставлено всім активним клієнтам миттєво. Весь блок обгорнутий у конструкцію `try-catch`, що запобігає падінню всього сервера при поодиноких помилках у базі даних або при спробі відправити некоректні дані.

Лістинг 3.2 – Реалізація логіки чатів у реальному часі

```

io.on('connection', (socket) => {
    socket.on('joinEvent', (eventId) => {
        socket.join(`event_${eventId}`);
console.log(`User ${socket.user.id} joined event_${eventId}`);
    });
    socket.on('sendMessage', async (data) => {
        try {
            const { eventId, text } = data;
            const senderId = socket.user.id;
            if (!text || !text.trim()) return;
            const result = await pool.query(
                `INSERT INTO messages (event_id, sender_id, text)
VALUES ($1, $2, $3) RETURNING id, text, created_at`,
                [eventId, senderId, text.trim()]
            );

```

```

    );
    const userResult = await pool.query(
      'SELECT id, full_name, nickname, photo_url FROM
users WHERE id = $1', [senderId]
    );
    const sender = userResult.rows[0];
    const newMessage = {
      id: result.rows[0].id,
      text: result.rows[0].text,
      created_at: result.rows[0].created_at,
      sender_id: sender.id,
      sender_name: sender.full_name,
      sender_photo: sender.photo_url
    };
    io.to(`event_${eventId}`).emit('newMessage', newMessage);
  }
}

```

У лістингу 3.3 наведено програмну реалізацію системи автентифікації на основі стандартів JSON Web Token (JWT). На відміну від застарілих сесій на базі cookies, JWT дозволяє створити архітектуру Stateless (без стану), що є ідеальним для сучасних веб-додатків та мобільних клієнтів.

Ключовим елементом безпеки є використання секретного ключа JWT_SECRET, який зберігається у змінних оточення сервера (.env) і ніколи не передається клієнту. Функція generateToken підписує дані користувача цим ключем, створюючи унікальний рядок, який неможливо підробити без знання секрету. Встановлений термін дії є компромісом між безпекою та зручністю користувача. Метод verifyToken використовується при кожному захищеному запиті для перевірки цілісності даних. Якщо зломисник спробує змінити хоча б один біт у токени, перевірка підпису не пройде, і доступ буде заблоковано.

Лістинг 3.3 – Реалізація механізму генерації та верифікації JWT-токенів

```

import jwt from 'jsonwebtoken';
const JWT_SECRET = process.env.JWT_SECRET ||
'your_super_secret_key_123';
const JWT_EXPIRES_IN = '1h';

export const generateToken = (payload) => {
  return jwt.sign(payload, JWT_SECRET, {

```

```

    expiresIn: JWT_EXPIRES_IN,
  });
};
export const verifyToken = (token) => {
  try {
    return jwt.verify(token, JWT_SECRET);
  } catch (error) {
    throw new Error('Invalid or expired token');
  }
};

```

Лістинг 3.4 описує структуру даних, що забезпечує соціальну інтерактивність додатку. Таблиця `messages` має універсальну структуру: вона підтримує як групові чати заходів (через `event_id`), так і прямі повідомлення між користувачами (через `receiver_id`). На рівні бізнес-логіки сервер перевіряє заповненість одного з цих полів, що дозволяє використовувати єдину таблицю для всієї переписки, спрощуючи запити на отримання історії.

Таблиця `user_friends` реалізує складні стани відносин між користувачами: від запиту на дружбу до блокування. Це дозволяє робити заходи виключно для друзів.

Окрему увагу в даному лістингу приділено продуктивності. Оскільки база даних очікує велику кількість запитів на читання (пошук заходу за категоріями, фільтрація за датою, завантаження чатів), було створено систему індексів (`CREATE INDEX`). Індеси за полями `event_date` та `category` дозволяють СКБД швидко знаходити потрібні записи без повного сканування таблиці, що особливо важливо при масштабуванні проєкту. Індксація зовнішніх ключів у таблиці повідомлень прискорює завантаження діалогів, що безпосередньо впливає на швидкість відображення інтерфейсу.

Лістинг 3.4 - Опис частини БД

```

CREATE TABLE IF NOT EXISTS messages (
  id SERIAL PRIMARY KEY,
  sender_id INT NOT NULL REFERENCES users(id) ON DELETE
CASCADE,
  event_id INT REFERENCES events(id) ON DELETE CASCADE,
  receiver_id INT REFERENCES users(id) ON DELETE CASCADE,
  text TEXT NOT NULL,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP

```

```

);
CREATE TABLE IF NOT EXISTS user_friends (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    friend_id INT NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    status VARCHAR(50) DEFAULT 'pending',
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, friend_id)
);
CREATE INDEX IF NOT EXISTS idx_messages_event_id ON
messages(event_id);
CREATE INDEX IF NOT EXISTS idx_messages_receiver_id ON
messages(receiver_id);
CREATE INDEX IF NOT EXISTS idx_events_category ON
events(category);
CREATE INDEX IF NOT EXISTS idx_events_event_date ON
events(event_date);
CREATE INDEX IF NOT EXISTS idx_user_friends_status ON
user_friends(status);

```

Лістинг 3.5 демонструє логіку підключення сервера до бази даних з використанням технології Connection Pooling. У високонавантажених системах створення нового з'єднання з БД для кожного окремого запиту є надто "дорогою" операцією, яка споживає багато процесорного часу та пам'яті.

Використання об'єкта Pool дозволяє тримати певну кількість готових з'єднань (max: 20) у "гарячому" резерві. Коли надходить запит, він миттєво отримує вільне з'єднання з пулу, а після завершення роботи повертає його назад. Це значно (у десятки разів) підвищує швидкість відповіді сервера. Налаштування connectionTimeoutMillis гарантує, що додаток не буде нескінченно чекати на відповідь від бази даних, якщо вона перевантажена, а видасть помилку за 2 секунди, що дозволяє зберегти стабільність всього сервера. Такий підхід свідчить про професійну підготовку архітектури додатка до реальних умов експлуатації.

Лістинг 3.5 – Конфігурація пулу з'єднань з базою даних PostgreSQL

```

import pg from 'pg';
import dotenv from 'dotenv';
dotenv.config();
const { Pool } = pg;

```

```

const pool = new Pool({
  host: process.env.POSTGRES_HOST || 'localhost',
  port: process.env.POSTGRES_PORT || 5432,
  database: process.env.POSTGRES_DB || 'kmg_events_db',
  user: process.env.POSTGRES_USER || 'postgres',
  password: process.env.POSTGRES_PASSWORD || 'postgres',
});
pool.on('error', (err) => {
  console.error('Unexpected error on idle client', err);
});
export default pool;

```

Лістинг 3.6 описує програмний механізм роботи з медіа-контентом заходу. На відміну від звичайних текстових полів, завантаження зображень потребує використання об'єкта `FormData`, що дозволяє передавати файли у форматі `multipart/form-data` через протокол HTTP.

Ця логіка реалізована за принципом асинхронної взаємодії. Після того, як користувач вибирає файл, він одразу відправляється на сервер. Сервер обробляє зображення (перевіряє формат, розмір, зберігає на диску) та повертає унікальний шлях до нього. Фронтенд-частина оновлює стан форми, додаючи повний URL зображення. Такий підхід дозволяє реалізувати попередній перегляд фото (preview) ще до того, як сама форма створення заходу буде відправлена. Також передбачено обробку помилок, що забезпечує стабільність роботи інтерфейсу при спробі завантаження некоректних файлів.

Лістинг 3.6 – Обробка завантаження зображень через Multipart

```

const handlePhotoUpload = async (e) => {
  const file = e.target.files[0];
  if (!file) return;
  const formDataUpload = new FormData();
  formDataUpload.append('image', file);
  try {
    const uploadRes = await API.post('/upload',
      formDataUpload, {headers: { 'Content-Type':
        'multipart/form-data' }});
    const imageUrl = uploadRes.data.imageUrl;
    const fullUrl = `${API.defaults.baseURL.replace('/api',
      '')}${imageUrl}`;
    handleFormChange({ target: { name: 'photo_url', value:
      fullUrl } });
  } catch (err) {

```

```

        console.error('Event photo upload error:', err);
        alert('Помилка при завантаженні фото. Перевірте формат
        файлу.'));}});
        const handleRemovePhoto = () => {
        handleFormChange({ target: { name: 'photo_url', value: null } });
        };

```

Лістинг 3.7 ілюструє інтеграцію фронтенду з сервісом повідомлень у реальному часі. Програмний код вирішує критичну задачу підтримки актуальності даних без необхідності вручну оновлювати сторінку. Використання хука `useEffect` тут виконує роль менеджера підписок.

Важливою деталлю є функція очистки (`return` у `useEffect`). Вона гарантує, що при закритті компонента або зміні користувача всі слухачі подій сокета будуть видалені. Це запобігає витокам пам'яті (`memory leaks`) та ситуаціям, коли одна і та сама подія обробляється кілька разів. Також реалізована логіка автоматичного закриття вікна чату, якщо адміністратор видалив захід, до якої цей чат належить (`onEventDeleted`). Це забезпечує консистентність інтерфейсу та запобігає помилкам доступу до неіснуючих даних.

Лістинг 3.7 - Функціональна частина фронтенду чату

```

useEffect(() => {
    if (!currentUser) return;
    socketService.connect();
    socketService.joinPersonalRoom(currentUser.id);
    const refresh = () => loadTabData(true);

    socketService.onFriendRequestReceived(refresh);
    socketService.onFriendRequestAccepted(refresh);
    socketService.onChatListUpdate(refresh);
    socketService.onKickedFromEvent(refresh);

    const handleEventDeleted = (data) => {
        if (activeChat?.type === 'event' && activeChat?.id ===
data.eventId) {
            setActiveChat(null);
        }
        refresh();
    };
    socketService.onEventDeleted(handleEventDeleted);
    return () => {

```

```

        socketService.offFriendRequestReceived(refresh);
        socketService.offFriendRequestAccepted(refresh);
        socketService.offChatListUpdate(refresh);
        socketService.offEventDeleted(handleEventDeleted);
    };
}, [currentUser?.id, loadTabData, activeChat]);

```

Лістинг 3.8 ілюструє реалізацію одного з найскладніших модулів системи універсального API для роботи з великими списками даних. Основна програмна задача тут забезпечити гнучкість вибірки при високій швидкодії.

Застосовано метод динамічної генерації SQL-запиту. Замість написання десятків окремих функцій для різних комбінацій фільтрів, код аналізує параметри URL-запиту та покроково формує масив умов фільтрації.

Також у коді реалізовано механізм пагінації. Розрахунок зсуву дозволяє серверу не завантажувати всю базу даних в пам'ять, а вибирати лише необхідну кількість записів для поточної сторінки. Використання оператора LEFT JOIN дозволяє в одному запиті отримати не лише дані про захід, а й інформацію про її автора та кількість зареєстрованих учасників, що суттєво зменшує кількість звернень до бази даних та прискорює рендеринг інтерфейсу на стороні користувача.

Для підвищення якості пошуку в коді використано оператор ILIKE, що дозволяє здійснювати гнучкий пошук без урахування регістру символів, щоб зробити зручнішим пошук для користувача на платформі. Застосування віконної функції COUNT(*) OVER() безпосередньо в основному запиті оптимізує навантаження на систему, оскільки позбавляє потреби виконувати окремий ресурсомісткий запит для підрахунку загальної кількості рядків у базі даних.

Варто також зауважити, що архітектура цього модуля враховує необхідність валідації вхідних параметрів та обробки виняткових ситуацій. Використання блоку try-catch гарантує стабільність роботи всього сервісу. Окрім основного списку об'єктів, система повертає клієнту обчислений мета-об'єкт із загальною кількістю сторінок та результатів. Це дозволяє фронтенд-частині адаптувати інтерфейс пагінації, без потреба прораховувати

це окремо навіть за умови значного зростання обсягів інформації в базі даних.

Лістинг 3.8 - Реалізація API пагінації

```
app.get('/api/events', async (req, res) => {
  try {
    const { page = 1, limit = 10 } = req.query;
    const parsedPage = parseInt(page, 10);
    const parsedLimit = parseInt(limit, 10);
    const offset = (parsedPage - 1) * parsedLimit;
    let queryText = `
      SELECT e.*, COUNT(*) OVER() as total_count
      FROM events e
      ORDER BY e.event_date ASC
      LIMIT $1 OFFSET $2
    `;
    const result = await pool.query(queryText, [parsedLimit,
offset]);
    const totalCount = result.rows.length > 0 ?
parseInt(result.rows[0].total_count, 10) : 0;
    res.json({
      events: result.rows,
      totalCount,
      totalPages: Math.ceil(totalCount / parsedLimit),
      currentPage: parsedPage
    });
  } catch (error) {
    res.status(500).json({ error: error.message });
  }
});
```

У лістингу 3.9 продемонстровано професійний підхід до організації мережевої взаємодії. Замість того, щоб вручну додавати токени безпеки в кожному окремому запиті по всьому додатку, використано механізм інтерцепторів (interceptors).

Перший інтерцептор (request) працює як фільтр перед виходом запиту: він перевіряє наявність токена у локальному сховищі браузера і, якщо він є, автоматично додає його в заголовки HTTP за стандартом Bearer. Це критично важливо для безпеки, оскільки сервер приймає запити лише з валідним токеном.

Другий інтерцептор (response) обробляє відповіді від сервера. Якщо сервер повертає помилку 401 (Unauthorized), це означає, що термін дії токена закінчився або він був відкликаний. У такому випадку додаток автоматично "скидає" стан користувача і перенаправляє його на сторінку логіна. Це забезпечує надійність системи та захищає від спроб використання застарілих сесій, роблячи роботу додатка більш стабільною та передбачуваною.

Лістинг 3.9 – Налаштування HTTP-клієнта та інтерцепторів безпеки

```
import axios from 'axios';
const API = axios.create({
  baseURL: process.env.REACT_APP_API_URL ||
'http://localhost:3000/api',
});
API.interceptors.request.use((config) => {
  const token = localStorage.getItem('token');
  if (token) {
    config.headers.Authorization = `Bearer ${token}`;
  }
  return config;
}, (error) => {
  return Promise.reject(error);
});
API.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response && error.response.status === 401) {
      localStorage.removeItem('token');
      localStorage.removeItem('user');
      window.location.href = '/login';
    }
    return Promise.reject(error);
  }
);
export default API;
```

У лістингу 3.10 наведено приклад реалізації декларативної маршрутизації в односторінковому застосунку. Використано бібліотеку React Router. Ключовим елементом тут є логіка захищених маршрутів. Завдяки використанню тернарного оператора в описі шляху профіля, система автоматично перевіряє стан currentUser. Якщо користувач не авторизований, його примусово перенаправить на сторінку входу. Це гарантує, що

конфіденційні частини інтерфейсу будуть доступні тільки після проходження автентифікації.

Важливою архітектурною особливістю є механізм збереження сесії користувача. Ініціалізація стану `currentUser` безпосередньо з `localStorage` дозволяє застосунку «пам'ятати» авторизованого користувача навіть після повного оновлення сторінки браузером. Це позбавляє необхідності повторного введення облікових даних при кожному візиті, що є зручним та важливим для приємного використання сайту користувачем. Такий підхід у поєднанні з функцією `setCurrentUser` створює замкнений цикл керування автентифікацією: від моменту входу в систему до повного виходу з видаленням даних зі сховища.

Також у коді продемонстровано принцип умовного рендерингу функціональних модулів. Зокрема, компонент `ChatPanel` монтується в DOM-дерево лише за умови наявності активного об'єкта користувача. Це не лише захищає месенджер від несанкціонованого доступу, а й оптимізує продуктивність застосунку, оскільки важкі компоненти що використовують систему комунікацій в реальному часі не завантажуються без потреби. Централізований розподіл даних через прокидування інформації від головного модуля до дочірніх компонентів забезпечує цілісність інтерфейсу: будь-яка зміна в стані користувача миттєво синхронізується між усіма компонентами які її використовують без використання складних зовнішніх менеджерів стану.

Лістинг 3.10 – Глобальна маршрутизація та авторизаційний захист сторінок

```
function App() {
  const [currentUser, setCurrentUser] =
  useState(JSON.parse(localStorage.getItem('user')));
  return (
    <Router>
      <div className="app-container">
        <Navbar user={currentUser} setUser={setCurrentUser} />
        <main className="main-content">
          <Routes>
```

```

        <Route path="/" element={<HomePage
        currentUser={currentUser} />} />
        <Route path="/login" element={<LoginPage
        setUser={setCurrentUser} />} />
        <Route path="/register" element={<RegisterPage />}
    />

    <Route
        path="/profile"
        element={
            currentUser ? <ProfilePage user={currentUser}
    /> : <Navigate to="/login" />
        }
    />
    </Routes>
    </main>
    {currentUser && <ChatPanel currentUser={currentUser} />}
    </div>
    </Router>
  );
}

```

3.3 Висновки за розділом 3

У ході виконання третього розділу було здійснено повну програмну реалізацію вебзастосунку, призначеного для пошуку однодумців та спільної організації заходів. Представлено детальну функціональну схему програми, що базується на інтерактивному меню та компонентній архітектурі, яка забезпечує логічний розподіл обов'язків між клієнтською та серверною частинами. У процесі розробки було успішно впроваджено всі заплановані функції, включаючи систему автентифікації на основі токенів JWT, механізми створення та фільтрації заходів за категоріями, а також інтеграцію з картографічними сервісами для точного позиціонування місць зустрічей. Описано ключові програмні одиниці, сервіси та контролери, що складають ядро системи, з акцентом на чистоту коду та можливість його подальшого масштабування. Особливу увагу в розділі приділено реалізації інтерактивної взаємодії користувачів у реальному часі за допомогою протоколу WebSocket, що дозволило створити повноцінну систему миттєвого обміну повідомленнями. Проєктування бази даних у СКБД PostgreSQL із застосуванням індексів та обмежень цілісності гарантує високу

продуктивність системи та захищеність конфіденційної інформації користувачів. Таким чином, створений програмний продукт повністю відповідає поставленим технічним вимогам, забезпечуючи стабільну роботу всіх модулів та високий рівень зручності користувацького інтерфейсу, що є необхідною умовою для успішної експлуатації застосунку в реальних умовах.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма призначена для організації та керування офлайн-заходами. Платформа слугує ефективним інструментом для об'єднання людей та пошуку однодумців на основі спільних інтересів (настільні ігри, читацькі зустрічі, дворовий спорт, локальні зібрання), визначення місцезнаходження майбутніх заходів та зручної орієнтації на інтерактивній карті, взаємодії з іншими користувачами через систему миттєвих повідомлень як в особистих так і групових чатах, керування соціальними зв'язками, що включають запити у друзі та блокування користувачів, що дозволяє уникати приєднання небажаних користувачів, а також модерації контенту та користувачів для забезпечення безпечного середовища спілкування.

4.2 Умови виконання програми

Обов'язковою умовою експлуатації програми організації та керування локальними заходами є наявність адміністратора програми або користувача з роллю модератора, здатного виконувати функції моніторингу та контролю. В обов'язки адміністратора входить розгортання серверної частини застосунку та бази даних PostgreSQL, налаштування змінних середовища, створення модераторських облікових записів, а також модерація некоректного контенту (видалення повідомлень, заходів) та обмеження доступу порушникам правил (блокування користувачів).

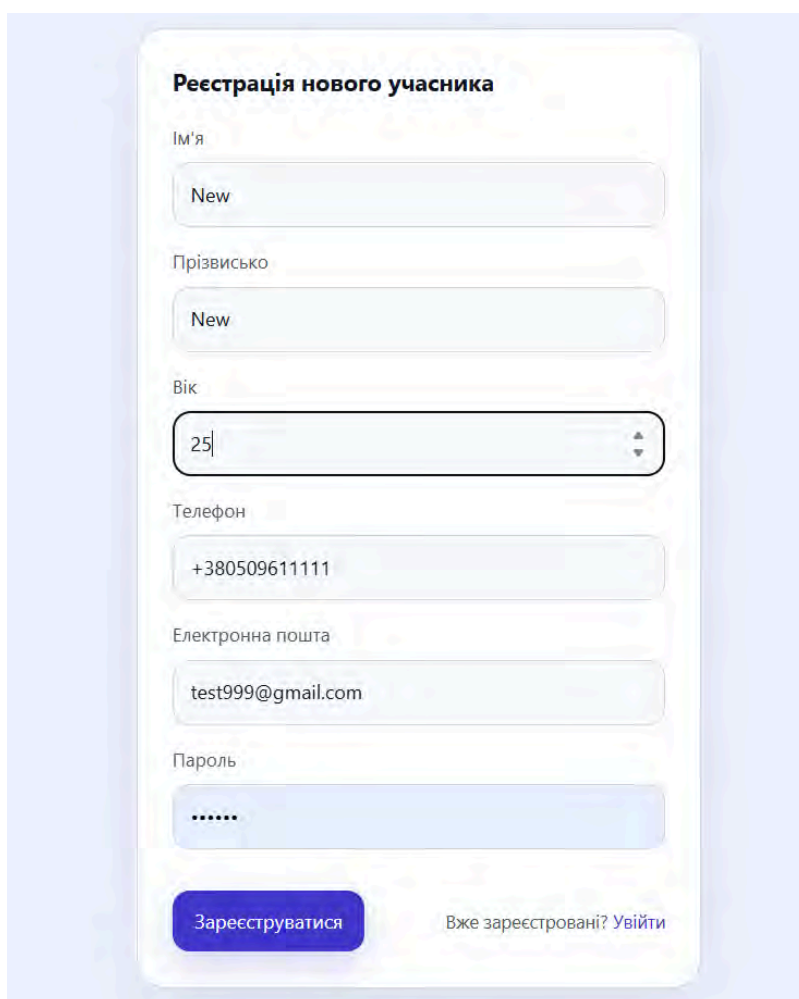
Технічне забезпечення сервера повинно включати мінімально:

- процесор: 2 ядра (частота від 2.0 ГГц);
- жорсткий диск – 20 Гб (з можливістю розширення для зберігання фотографій);
- оперативна пам'ять – 2 Гб.

4.3 Виконання і тестування програми

Задля перевірки працездатності розробленої системи було проведено декілька тестів, що відображають одні з найважливіших її функцій.

Для початку слід створити нового користувача для цього коли нас перекидає на сторінку логіна слід натиснути на посилання “Зареєструватися” після чого користувача перекине на сторінку реєстрації. Заповнивши правильно всі необхідні дані натискаємо на кнопку “Зареєструватися” знизу, якщо все пройшло успішно користувача перекине на головну сторінку вже під новоствореним акаунтом. Відображення заповнення форми можемо бачити на рис. 4.1.



The image shows a mobile application interface for user registration. The form is titled "Реєстрація нового учасника" (Registration of a new participant). It contains several input fields: "Ім'я" (Name) with the placeholder "New", "Прізвище" (Surname) with the placeholder "New", "Вік" (Age) with a numeric input field containing "25", "Телефон" (Phone) with the placeholder "+380509611111", "Електронна пошта" (Email) with the placeholder "test999@gmail.com", and "Пароль" (Password) with a masked input field showing ".....". At the bottom, there is a blue button labeled "Зареєструватися" (Register) and a link that says "Вже зареєстровані? Увійти" (Already registered? Log in).

Рисунок 4.1 - Створення нового користувача

Тепер коли нового користувача було створено можемо зайти в аккаунт, дозаповнити та оновити його. Для цього слід натиснути на посилання справа зверху “Аккаунт” що відкриє відповідну форму в якій ми можемо змінювати наповнення. Після того як всі бажані зміни в аккаунт внесено слід не забути їх зберегти. За це відповідає відповідна кнопка знизу форми. Рис. 4.2 відображає стан форми одразу після збереження змін.

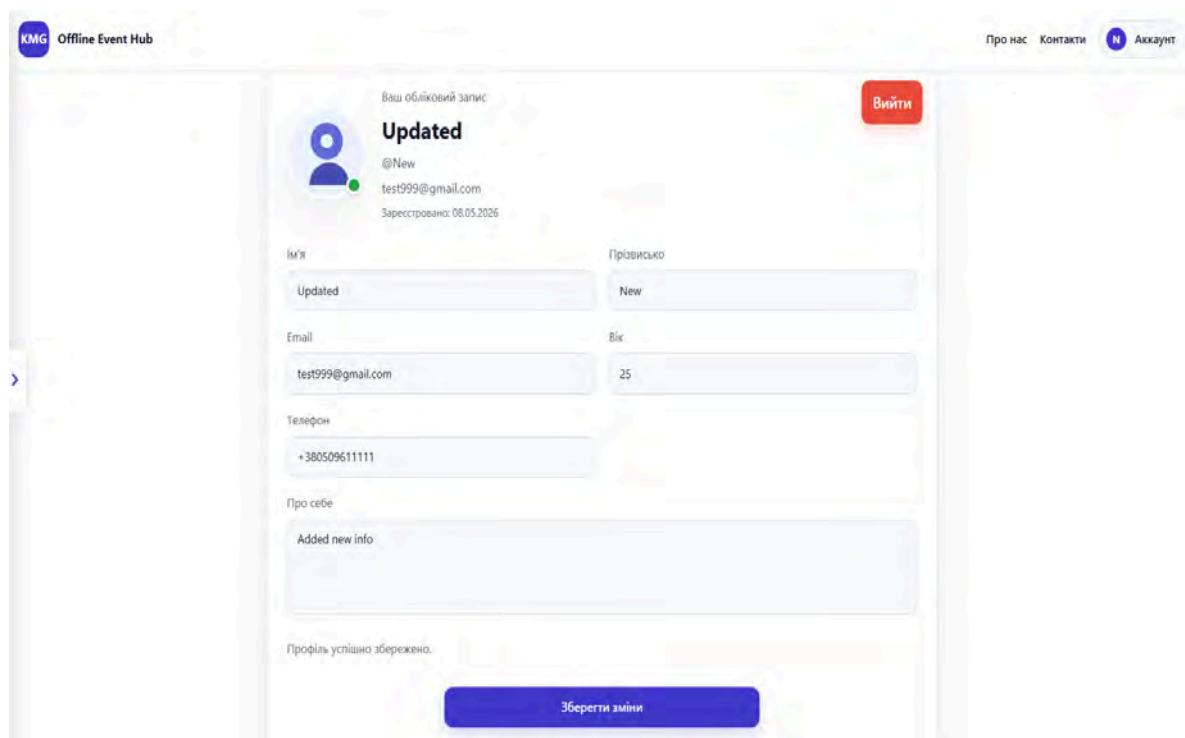


Рисунок 4.2 - Оновлення даних користувача

Після того як ми задоволені інформацією нашого аккаунта ми можемо створити захід. Щоб він був доступним іншим користувачам не будемо робити його приватним. Додамо фото для заходу, оберемо тип заходу, бажану кількість учасників та час проведення й звісно не забудемо про місце проведення. Щоб не писати вручну натиснемо на мапу, що дозволяє їй підтягнути місце самостійно. Після заповнення всієї бажаної інформації нам слід натиснути кнопку “Створити захід”. Рис. 4.3 відображає стан форми створення заходу після заповнення даних.

Створити новий захід

×

Фото заходу

Організатор

Ви

Приватний

Назва заходу *

New Event

Місце проведення *

Гімназія №838, 7-А, вулиця Парамонова, Південне селище, Космічний рай

Взяти з карти

Категорія заходу

Настільні ігри

Виберіть локацію на мапі

Широта: 47.7834 Довгота: 35.2137

Дата та час *

10.05.2026 00:13

Максимальна кількість учасників

4

Скасувати

Створити захід

Рисунок 4.3 - Створення нового івенту

Після того як ми створили захід він має стати доступним для певного кола осіб. Оскільки ми не робили його приватним то його можуть бачити всі крім заблокованих нами осіб. Зайдемо з іншого аккаунта та приєднаємось до заходу. Приєднуючись до заходу нам стає доступний її чат і ми стаємо його учасником, що відображається у випадяючому списку учасників зверху чату. Модератори та власники заходу завжди можуть виграти вас з івента, що відображається в їхньому інтерфейсі червоними хрестиками. Рис. 4.4 відображає наш захід в двох різних вікнах, одне з яких належить гістю, і показує, що ми можемо покинути захід після того як приєднались, а інше автору заходу, яке відображає гістя в переліку чату де ми можемо його вигнати при бажанні.

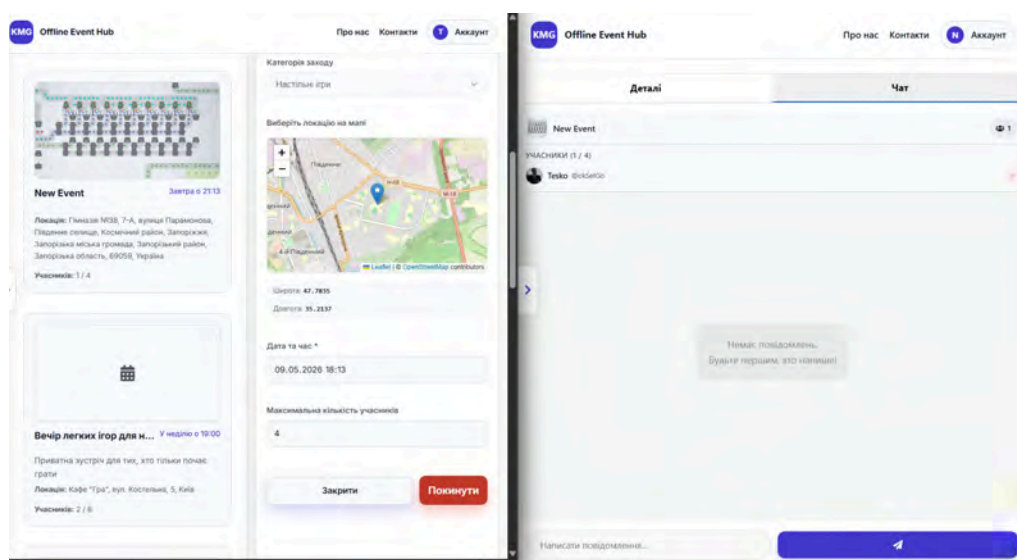


Рисунок 4.4 - Приєднання до заходу та чату

Оскільки ми приєднались з іншого аккаунту до нашого івенту то тепер цей аккаунт має змогу написати повідомлення, написавши повідомлення вони одночасно з'являються в усіх учасниках без необхідності перезавантажувати сторінку. Також наглядно видно що автор заходу може видаляти не тільки свої але й чужі коментарі в його заході. Модератори мають право видаляти будь чий коментарі в усіх групових чатах. Рис. 4.5 знов відображає гістя та автора

івенту та їхню можливість комунікувати й те що автор може видаляти повідомлення гістя.

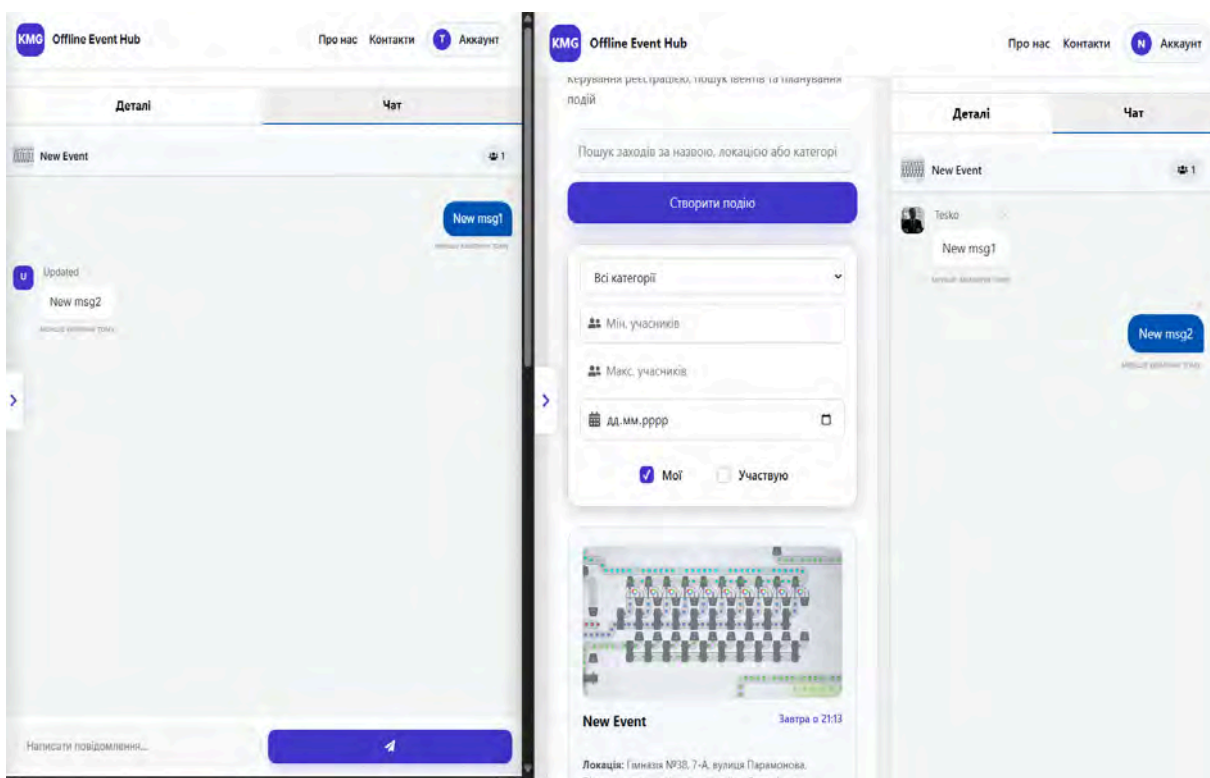


Рисунок 4.5 - Написати в чаті івенту

Після того як користувачі поговорили в чаті в них може виникнути бажання додаться в друзі щоб поспілкуватись особисто або щоб не загубити один одного або в майбутньому провести приватний захід. Для цього їм слід відкрити панель чатів що знаходиться зліва на екрані. Коли вона відкриється хоча б одному з них слід знайти іншого серед людей у відповідній вкладці та надіслати запит дружби. якщо інший користувач приймає запит дружби то вони стають друзями то будуть трохи відрізнятись від інших людей у вкладці Люди. Також наявний фільтр що відображатиме у цій вкладці виключно друзів поточного користувача. Рис. 4.6 відображає інтерфейси двох користувачів які стали друзями, і бачать один одного у вкладці Люди використовують різні типи фільтрів цієї вкладки.

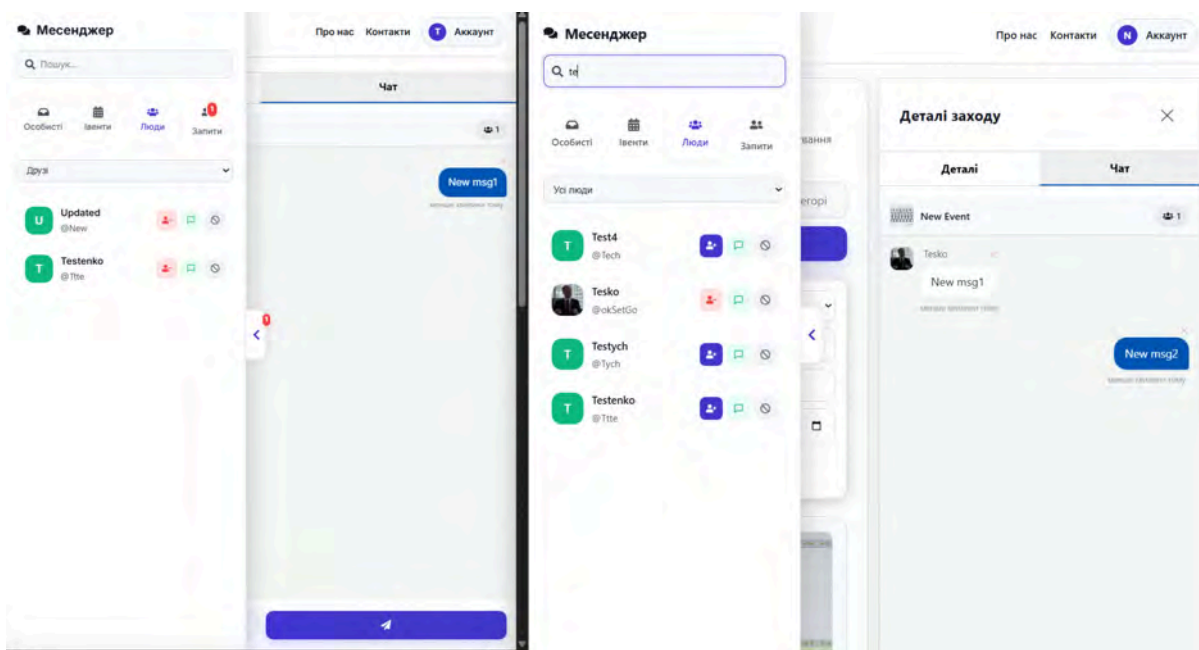


Рисунок 4.6 - Відображення в друзях після приймання запрошення

4.4 Висновки за розділом 4

У результаті виконання четвертого розділу було проведено тестування та аналіз ефективності роботи розробленого вебзастосунку. Програму для пошуку однодумців та організації офлайн-заходів було успішно реалізовано із повним забезпеченням усіх визначених функціональних вимог, що підтверджується результатами тестувань.

Показано, що навіть під час відображення декількох вікон одночасно, не зважаючи на зменшену широту екрану, системою було зручно користуватись, що підтвердило адаптивність розробленої системи.

Розроблена система навігації та меню керування дозволяють користувачам зручно взаємодіяти з усіма основними частинами програми, та забезпечують чітку послідовність дій від реєстрації профілю до участі у заходах та комунікаціях. Перевірка роботи системи в описаних умовах не викликає жодних помилок серверної частини та справно справилась із передачею інформації в реальному часі що підтверджує стійким до помилкових дій користувача та готовність до практичного використання.

ВИСНОВКИ

Підбиваючи підсумки виконаної роботи, можна сказати, що поставленої мети розробки системи для організації офлайн-заходів та пошуку однодумців було повністю досягнуто. Цей проєкт став відповіддю на актуальний запит сучасної спільноти: мати зручний і швидкий інструмент для координації зустрічей, який поєднує в собі комунікацію учасників, і менеджер заходів. Під час аналізу існуючих рішень я виявив, що більшість платформ або занадто сконцентровані на одному типі заходів, або мають обмежений функціонал чатів та заходів, який не дозволяє зручно орієнтуватись на карті чи керувати списком учасників у реальному часі. Саме на подолання цих розривів і була спрямована моя розробка.

Для реалізації проєкту я обрав сучасний технологічний стек, який включає Node.js та Express для побудови серверної частини, React для створення динамічного інтерфейсу та СКБД PostgreSQL для роботи з базою даними. Особливе місце в архітектурі посідає технологія Socket.io, яка дозволила впровадити повноцінний месенджер для миттєвого обміну повідомленнями між учасниками.

Результатом роботи став програмний продукт, що охоплює всі визначені вимоги та надає базові інструменти для планування: від зручного та зрозумілого вибору локації на інтерактивній карті до автоматизації процесу реєстрації та модерації учасників.

У підсумку розроблений вебзастосунок є повноцінною системою для керування заходами. Програма дозволяє користувачам легко знаходити цікаві заходи, ділитися враженнями, створювати власні заходи та будувати соціальні зв'язки через систему друзів та приватних чатів. Ця система спрощує організаційні моменти та створює основу для формування спільнот за інтересами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Eventbrite vs Meetup Pro: conditional analysis. *G2*. URL: <https://www.g2.com/compare/eventbrite-vs-meetup-pro> (date of access: 10.04.2026).
2. Ricken S., Barkhuus L., Jones Q. Going Online to Meet Offline: Organizational Practices of Social Activities through Meetup. *ResearchGate*. URL: https://www.researchgate.net/publication/317595581_Going_Online_to_Meet_Offline_Organizational_Practices_of_Social_Activities_through_Meetup (date of access: 11.04.2026).
3. Fostering Social Capital Through Community Events. *Extension University of New Hampshire*. URL: <https://extension.unh.edu/resource/fostering-social-capital-through-community-events> (date of access: 11.04.2026).
4. Proschinger J. Navigating the Risks of Censorship: The Urgent Need for Transparency and Accountability in Online Spaces. *Medium*. 2025. URL: <https://medium.com/@jproschinger/when-boardgamegeek-silenced-me-a-warning-about-the-dangers-of-opaque-moderation-and-cancel-culture-c6c5de90a3fe> (date of access: 12.04.2026).
5. Satyaningrum N. S. Social penetration in digital spaces: Developing interpersonal relationships through the Bumble dating app. *International Journal of Research and Innovation in Social Science (IJRISS)*. *RSIS International*. URL: <https://rsisinternational.org/journals/ijriss/articles/social-penetration-in-digital-spaces-developing-interpersonal-relationships-through-the-bumble-dating-app/> (date of access: 13.04.2026).
6. Devitt J. Where Do Nextdoor Communities Exist—And What Do These Communities Talk About? *NYU News*. 2024. URL: <https://www.nyu.edu/about/news-publications/news/2024/may/where-do-nextdoor-communities-exist-and-what-do-these-communities.html> (date of access: 13.04.2026).

7. Côté-Massicotte S. Discord: An Efficient Communication Tool for Pedagogical Purposes. *Eductive*. 2020. URL: <https://eductive.ca/en/resource/discord-an-efficient-communication-tool-for-pedagogical-purposes/> (date of access: 14.04.2026).
8. Tabletop Together Tool: Helping you plan your convention visit. *Tabletop Together*. URL: <https://tabletoptogogether.com/> (date of access: 14.04.2026).
9. Tabletop.Events: Total Event Management. *Tabletop Events*. URL: <https://tabletop.events/> (date of access: 15.04.2026).
10. Morgan M. Playgroup.gg: A New Way to Track Your Games. *Commander's Herald*. URL: <https://commandersherald.com/playgroup-gg-a-new-way-to-track-your-games/> (date of access: 15.04.2026).
11. Baduk Club: Find Players and Places for offline games. *Baduk Club*. URL: <https://baduk.club/welcome> (date of access: 16.04.2026).
12. Tandem. The Ultimate Guide to Language Exchange. *Tandem*. 2026. URL: <https://tandem.net/pages/ultimate-guide-language-exchange> (date of access: 16.04.2026).
13. Lawson L. Trends That Defined JavaScript in 2025. *The New Stack*. 2025. URL: <https://thenewstack.io/trends-that-defined-javascript-in-2025/> (date of access: 17.04.2026).
14. Harikrishna K. Vue.js vs React: Which is the ideal JavaScript framework for 2024? *Developer Nation*. URL: <https://www.developernation.net/blog/vue-js-vs-react-which-is-the-ideal-javascript-framework-for-2024/> (date of access: 17.04.2026).
15. Vue vs React: A Detailed Comparison for 2024. *Medium*. URL: <https://medium.com/@secret.point2020/vue-vs-react-a-detailed-comparison-for-2024-b722afba9bd5> (date of access: 18.04.2026).
16. Aideloje P. Vite vs Webpack for React apps in 2025: A senior engineer's perspective. *LogRocket Blog*. URL:

<https://blog.logrocket.com/vite-vs-webpack-react-apps-2025-senior-engineer/> (date of access: 18.04.2026).

17. Leaflet vs OpenLayers: Choosing the right mapping library. *Geoapify*. URL: <https://www.geoapify.com/leaflet-vs-openlayers/> (date of access: 19.04.2026).

18. Kalik A. Node.js vs Golang: Performance Benchmark. *ITNEXT*. URL: <https://itnext.io/performance-benchmark-node-js-vs-go-9dbad158c3b0> (date of access: 19.04.2026).

19. Itsekson A. NestJS vs Express.js: Choosing the right backend framework. *Jelvix*. URL: <https://jelvix.com/blog/nestjs-vs-express> (date of access: 20.04.2026).

20. Kelhini F. Axios vs Fetch API for modern JavaScript development. *LogRocket Blog*. URL: <https://blog.logrocket.com/axios-vs-fetch-2025/> (дата звернення: 21.04.2026).

21. Booker A. Socket.IO vs WebSocket: Comparing two popular realtime technologies. *Ably*. URL: <https://dev.to/ably/socketio-vs-websocket-comparing-two-popular-realtime-technologies-3ff3> (date of access: 21.04.2026).

22. Liquibase. PostgreSQL vs MySQL 2025: Complete Database Comparison Guide. *Liquibase*. 2025. URL: <https://www.liquibase.com/resources/guides/postgresql-vs-mysql> (date of access: 23.04.2026).

23. Celko J. The CHECK() Constraint. *Simple Talk*. 2024. URL: <https://www.red-gate.com/simple-talk/databases/theory-and-design/the-check-constant/> (date of access: 24.04.2026).

24. CVE-2025-7338: Multer DoS Vulnerability. *SentinelOne*. URL: <https://www.sentinelone.com/vulnerability-database/cve-2025-7338/> (date of access: 25.04.2026).

25. Building Real-time Applications with Node.js and WebSockets. *DEV Community*. 2024. URL:

<https://dev.to/allenchrios/building-real-time-applications-with-nodejs-and-websockets-7cc> (date of access: 26.04.2026).

ДОДАТОК А
ТЕХНІЧНЕ ЗАВДАННЯ

Вступ

Організація локальних заходів, від настільних ігор та читацьких зустрічей до дворових спортивних активностей, є важливою складовою соціального життя міських спільнот. Попри те, що сучасні соціальні мережі надають загальні засоби для комунікації, вони не пропонують спеціалізованих інструментів для створення, пошуку та керування невеликими тематичними заходами. Існуючі рішення, або не призначені для створення заходів напряду, або не мають інструменту інтерактивного вибору місця проведення на карті, або орієнтовані переважно на масштабні комерційні заходи та конференції, що створює надлишкову складність для організації невеликих неформальних зустрічей.

Водночас попит на об'єднання людей за спільними інтересами залишається високим. Настільні ігри продовжують бути популярними, дворовий спорт стає альтернативою дорогим фітнес-клубам, а книжкові клуби формують інтелектуальні спільноти. Усі ці активності потребують зручних інструментів для координації: визначення місця та часу, визначення кількості учасників, спілкування перед та під час заходу. Необхідний вебзастосунок, який поєднує функціональність організатора заходу із вбудованою системою комунікації в реальному часі та соціальними механізмами такими як дружба, чата та модерація. Саме на створенні такого спеціалізованого інструменту зосереджена ця розробка.

A.1 Підстави для розробки

Дипломна кваліфікаційна робота виконувалась у відповідності та на підставі Наказу № 139 від 07 квітня 2026 р. за Національним університетом «Запорізька політехніка», яким визначено тему роботи – «Програмне забезпечення організації та керування локальними заходами».

A.2 Призначення розробки

Програма призначена для організації та керування локальними заходами (настільні ігри, читацькі зустрічі, дворові спортивні активності, локальні зібрання), забезпечення комунікації між учасниками через групові та особисті чати в реальному часі, керування соціальними зв'язками між користувачами (дружба, блокування), а також модерації контенту та користувачів адміністратором системи.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Функціональні характеристики програми організації та керування локальними заходами повинні включати:

- реєстрація користувача із зазначенням імені, прізвища, віку, номера телефона (+380XXXXXXXXXX), електронної пошти та пароля;
- авторизація користувача за електронною поштою та паролем;
- автоматичне відновлення сесії при повторному відкритті програми (за наявності чинного токена);
- перегляд власного профілю з відображенням усіх своїх даних;
- редагування профілю (ім'я, прізвище, вік, телефон, електронна пошта, біографія);
- завантаження та зміна фотографії профілю;
- автоматичне відстеження онлайн/офлайн статусу користувача;
- вихід із системи;
- створення нового заходу із зазначенням назви, опису, категорії, дати та часу проведення, максимальної кількості учасників, ознаки приватності;
- визначення локації заходу шляхом вибору точки на інтерактивній карті з автоматичним визначенням адреси;

- завантаження фотографії до заходу;
- редагування власного заходу;
- видалення власного заходу з підтвердженням;
- приєднання до заходу іншого користувача;
- вихід зі складу учасників заходу;
- перегляд списку заходів з пагінацією;
- фільтрація заходів за категорією;
- фільтрація заходів за датою проведення;
- фільтрація заходів за кількістю учасників;
- перегляд лише власних створених заходів;
- перегляд заходів, в яких користувач бере участь;
- текстовий пошук заходів за назвою, описом або локацією;
- автоматичне приховування минулих заходів у загальному списку;
- відображення усіх заходів за певними фільтрами;
- приховування заходів заблокованих користувачів;
- обмеження видимості приватних заходів;
- відправлення повідомлення у груповий чат заходу;
- перегляд повідомлень групового чату заходу;
- відправлення особистого повідомлення іншому користувачу;
- перегляд особистих повідомлень з конкретним користувачем;
- перегляд списку особистих діалогів;
- видалення власного повідомлення;
- миттєва доставка повідомлень через WebSocket без перезавантаження сторінки;
- перегляд списку всіх користувачів системи зі статусом дружби;
- відправлення запиту на дружбу;
- прийняття запиту на дружбу;
- відхилення запиту на дружбу;
- видалення користувача зі списку друзів;
- блокування користувача з видаленням спільного чату;

- розблокування користувача;
- перегляд списку вхідних запитів на дружбу з відображенням бейджу кількості;
- фільтрація списку користувачів (усі, друзі, заблоковані);
- миттєве оновлення статусів дружби через WebSocket;
- бан користувача з примусовим миттєвим виходом із системи;
- розбан користувача;
- видалення учасника з будь-якого заходу;
- видалення будь-якого повідомлення у чатах;
- редагування та видалення будь-якого заходу.

А.3.2 Умови експлуатації

Обов'язковою умовою експлуатації програми є наявність адміністратора системи або користувача, здатного виконувати такі функції. В обов'язки адміністратора входить:

- початкове розгортання серверної частини застосунку та бази даних;
- конфігурація змінних середовища (параметри підключення до БД, порт сервера, JWT-секрет);
- виконання міграції бази даних та початкового наповнення (категорії, тестові дані);
- створення облікового запису з роллю MODERATOR для модерації контенту;
- моніторинг роботи сервера та обсягу завантажених зображень.

Програма розрахована на роботу через браузер, що не потребує встановлення додаткового програмного забезпечення на стороні користувача. Програма має адаптивний дизайн з мінімальна підтримуваною шириною екрану в 320 пікселів.

A.3.3 Вимоги до складу та параметрів технічних засобів

Технічне забезпечення сервера повинно бути принаймні:

- процесор 2 ядра, частота від 2.0 ГГц;
- оперативна пам'ять мінімум 2 ГБ (для одночасної роботи Node.js та PostgreSQL);
- жорсткий диск мінімум 20 ГБ (наявність фотографій заходів та профілів може значно збільшувати вимоги);
- стабільне мережеве з'єднання.

Програмне забезпечення сервера повинно бути принаймні:

- версія Node.js від 20 або новіше;
- версія PostgreSQL від 14 або новіше;
- для керування залежностями необхідний npm.

Клієнт повинен мати доступ до:

- вебоглядач з підтримкою ECMAScript 2020, WebSocket API;
- стабільне інтернет-з'єднання для комунікації в реальному часі.

A.4 Порядок контролю та приймання

Починаючи з дати видачі завдання – 7 квітня, протягом наступних 5 тижнів має бути забезпечено виконання роботи за календарним планом, що міститься в завданні на роботу. Календарний план визначає послідовність етапів виконання роботи та за кожним таким етапом строк виконання і основний очікуваний результат. Після забезпечення цих очікуваних результатів формується пояснювальна записка до дипломної кваліфікаційної роботи. Остаточним кроком приймання роботи є її публічний захист.

ДОДАТОК Б
ТЕКСТ ПРОГРАМИ

Б.1 Текст файла server\config\jwt.js

```
const JWT_SECRET = process.env.JWT_SECRET || 'MikeBlAbLaCoDe';
const JWT_EXPIRES_IN = process.env.JWT_EXPIRES_IN || '1h';

export const generateToken = (payload) => {
  return jwt.sign(payload, JWT_SECRET, { expiresIn:
JWT_EXPIRES_IN });
};

export const verifyToken = (token) => {
  try {
    return jwt.verify(token, JWT_SECRET);
  } catch (error) {
    throw new Error('Invalid or expired token');
  }
};

export { JWT_SECRET, JWT_EXPIRES_IN };
```

Б.2 Текст файла server/index.js

```
const io = new Server(httpServer, {
  cors: { origin: '*', methods: ['GET', 'POST'] }
});

io.use((socket, next) => {
  const token = socket.handshake.auth.token;
  if (!token) return next(new Error('Authentication error: Token
missing'));
  try {
    const decoded = verifyToken(token);
    socket.user = decoded;
    next();
  } catch (err) {
    next(new Error('Authentication error: Invalid token'));
  }
});

io.on('connection', (socket) => {
  socket.on('sendMessage', async (data) => {
    try {
      const { eventId, text } = data;
      const senderId = socket.user.id;

      const result = await pool.query(
        `INSERT INTO messages (event_id, sender_id, text) VALUES
($1, $2, $3) RETURNING id, text, created_at`,
        [eventId, senderId, text.trim()]
      );
    }
  });
});
```

```

    const newMessage = { ...result.rows[0], sender_id: senderId
  };
    io.to(`event_${eventId}`).emit('newMessage', newMessage);
  } catch (err) {
    console.error('Socket error:', err);
  }
});
});
const authenticateToken = async (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];
  if (!token) return res.status(401).json({ error: 'Access token
required' });
  try {
    const decoded = verifyToken(token);
    const userResult = await pool.query('SELECT id, role,
is_banned FROM users WHERE id = $1', [decoded.id]);

    if (userResult.rows.length === 0 ||
userResult.rows[0].is_banned) {
      return res.status(403).json({ error: 'Access denied or
account banned' });
    }

    req.user = userResult.rows[0];
    next();
  } catch (error) {
    res.status(401).json({ error: 'Invalid token' });
  }
};

app.get('/api/events', async (req, res) => {
  try {
    const { category, search, page = 1, limit = 10 } = req.query;
    const offset = (page - 1) * limit;

    let queryText = `
      SELECT e.*, u.full_name as creator_name, COUNT(ep.id) as
participant_count, COUNT(*) OVER() as total_count
      FROM events e
      LEFT JOIN users u ON e.creator_id = u.id
      LEFT JOIN event_participants ep ON e.id = ep.event_id
    `;

    const whereClauses = [];
    const queryParams = [];

    if (category) {
      whereClauses.push(`e.category = ${queryParams.length +
1}`);
      queryParams.push(category);
    }
    if (search) {

```

```

        whereClauses.push(`(e.title ILIKE ${queryParams.length +
1} OR e.description ILIKE ${queryParams.length + 1})`);
        queryParams.push(`%${search}%`);
    }
    if (whereClauses.length > 0) queryText += ' WHERE ' +
whereClauses.join(' AND ');
    queryText += ` GROUP BY e.id, u.id ORDER BY e.event_date ASC
LIMIT ${queryParams.length + 1} OFFSET ${queryParams.length +
2}`;
    queryParams.push(limit, offset);

    const result = await pool.query(queryText, queryParams);
    const totalCount = result.rows.length > 0 ?
parseInt(result.rows[0].total_count) : 0;
    res.json({ events: result.rows, totalPages:
Math.ceil(totalCount / limit), currentPage: page });
    } catch (error) {
        res.status(500).json({ error: error.message });
    }
});

app.put('/api/users/:id/ban', authenticateToken, async (req, res)
=> {
    if (req.user.role !== 'MODERATOR') return
res.status(403).json({ error: 'Forbidden' });
    const { id } = req.params;
    await pool.query('UPDATE users SET is_banned = true WHERE id =
$1', [id]);
    io.to(`user_${id}`).emit('accountBanned', {});
    res.json({ success: true, message: 'User banned' });
});

app.put('/api/users/:id/unban', authenticateToken, async (req,
res) => {
    if (req.user.role !== 'MODERATOR') return
res.status(403).json({ error: 'Forbidden' });
    const { id } = req.params;
    await pool.query('UPDATE users SET is_banned = false WHERE id =
$1', [id]);
    res.json({ success: true, message: 'User unbanned' });
});

```

Б.3 Текст файла src\App.jsx

```

function App() {
    const [loading, setLoading] = useState(true)
    const isModerator = user?.role === 'MODERATOR'
    if (loading) {return <div className="app-shell"><div
className="page-card">Загрузка...</div></div>}
    return (
        <BrowserRouter>

```

```

    <Routes>
      <Route path="/" element={<Navigate to="/home" replace />}
    />
      <Route path="/login" element={<LoginPage
setUser={setUser} />} />
      <Route path="/register" element={<RegisterPage
setUser={setUser} />} />
      <Route path="/home" element={<HomePage currentUser={user}
setUser={setUser} />} />
      <Route path="/account" element={<AccountPage
currentUser={user} setUser={setUser} />} />
      {!isModerator && (
        <>
          <Route path="/contacts" element={<ContactPage />} />
          <Route path="/about" element={<AboutPage />} />
        </>
      )}
      <Route path="*" element={<Navigate to="/home" replace />}
    />
  </Routes>
  <ScrollToTop />
  {user && <ChatPanel currentUser={user} />}
</BrowserRouter>
)
}

```

Б.4 Текст файла src\services\authService.js

```

import API from '../utils/api';
export const authService = {
  login: async (credentials) => {
    const response = await API.post('/auth/login', credentials);
    if (response.data.token) {
      localStorage.setItem('authToken', response.data.token);
      try {
        await API.put('/users/status', { is_active: true });
      } catch (err) {}
    }
    return response.data;
  },

  register: async (userData) => {
    const response = await API.post('/auth/register', userData);
    if (response.data.token) {
      localStorage.setItem('authToken', response.data.token);
      try {
        await API.put('/users/status', { is_active: true });
      } catch (err) {}
    }
    return response.data;
  },

```

```

getCurrentUser: async () => {
  const response = await API.get('/auth/me');
  return response.data;
}
};

```

Б.5 Текст файла server\db\schema.sql

```

CREATE TABLE IF NOT EXISTS users (
  id SERIAL PRIMARY KEY,
  phone_number VARCHAR(20) UNIQUE,
  full_name VARCHAR(255) NOT NULL,
  nickname VARCHAR(100) NOT NULL UNIQUE,
  age INT,
  photo_url TEXT,
  bio TEXT,
  email VARCHAR(255) NOT NULL UNIQUE,
  password_hash VARCHAR(255),
  is_active BOOLEAN DEFAULT TRUE,
  is_banned BOOLEAN DEFAULT FALSE,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS events (
  id SERIAL PRIMARY KEY,
  creator_id INT NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  title VARCHAR(255) NOT NULL,
  description TEXT,
  location VARCHAR(255) NOT NULL,
  latitude DECIMAL(10, 8),
  longitude DECIMAL(11, 8),
  photo_url TEXT,
  event_date TIMESTAMP NOT NULL,
  max_participants INT,
  status VARCHAR(50) DEFAULT 'upcoming', -- upcoming, ongoing,
  completed, cancelled
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS event_participants (
  id SERIAL PRIMARY KEY,
  event_id INT NOT NULL REFERENCES events(id) ON DELETE CASCADE,
  user_id INT NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  status VARCHAR(50) DEFAULT 'registered', -- registered,
  confirmed, cancelled
  registered_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  UNIQUE(event_id, user_id)
);

```

```

CREATE INDEX IF NOT EXISTS idx_events_creator_id ON
events(creator_id);
CREATE INDEX IF NOT EXISTS idx_events_event_date ON
events(event_date);
CREATE INDEX IF NOT EXISTS idx_event_participants_event_id ON
event_participants(event_id);
CREATE INDEX IF NOT EXISTS idx_event_participants_user_id ON
event_participants(user_id);
CREATE INDEX IF NOT EXISTS idx_users_nickname ON users(nickname);

```

```

ALTER TABLE users
ADD COLUMN IF NOT EXISTS role VARCHAR(20) DEFAULT 'USER';

```

```

ALTER TABLE events
ADD COLUMN IF NOT EXISTS is_private BOOLEAN DEFAULT FALSE,
ADD COLUMN IF NOT EXISTS category VARCHAR(100);

```

```

ALTER TABLE event_participants
ALTER COLUMN status SET DEFAULT 'pending';

```

```

CREATE TABLE IF NOT EXISTS messages (
    id SERIAL PRIMARY KEY,
    sender_id INT NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    event_id INT REFERENCES events(id) ON DELETE CASCADE,
    receiver_id INT REFERENCES users(id) ON DELETE CASCADE,
    text TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

```

```

CREATE TABLE IF NOT EXISTS user_friends (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    friend_id INT NOT NULL REFERENCES users(id) ON DELETE CASCADE,
    status VARCHAR(50) DEFAULT 'pending', -- pending, accepted,
blocked
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, friend_id)
);

```

```

CREATE INDEX IF NOT EXISTS idx_messages_sender_id ON
messages(sender_id);
CREATE INDEX IF NOT EXISTS idx_messages_event_id ON
messages(event_id);
CREATE INDEX IF NOT EXISTS idx_messages_receiver_id ON
messages(receiver_id);

```

```

CREATE INDEX IF NOT EXISTS idx_user_friends_user_id ON
user_friends(user_id);
CREATE INDEX IF NOT EXISTS idx_user_friends_friend_id ON
user_friends(friend_id);

```

```

CREATE INDEX IF NOT EXISTS idx_events_category ON
events(category);
CREATE INDEX IF NOT EXISTS idx_events_is_private ON
events(is_private);

ALTER TABLE users
ADD CONSTRAINT check_user_role
CHECK (role IN ('USER', 'MODERATOR'));

ALTER TABLE events
ADD CONSTRAINT check_event_status
CHECK (status IN ('upcoming', 'ongoing', 'completed',
'cancelled'));

ALTER TABLE event_participants
ADD CONSTRAINT check_participant_status
CHECK (status IN ('pending', 'approved', 'rejected', 'left'));

ALTER TABLE user_friends
ADD CONSTRAINT check_friend_status
CHECK (status IN ('pending', 'accepted', 'blocked'));

```

Б.6 Текст файла src\utils\api.js

```

const API = axios.create({
  baseURL: '/api'
})
API.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('authToken')
    if (token) {
      config.headers.Authorization = `Bearer ${token}`
    }
    return config
  },
  (error) => {
    return Promise.reject(error)
  }
)
API.interceptors.response.use(
  (response) => response,
  (error) => {
    if (error.response?.status === 401) {
      localStorage.removeItem('authToken')
      window.location.href = '/login'
    }
    return Promise.reject(error)
  }
)

```

Б.7 Текст файла src\components\ChatPanel.jsx

```
function LocationMarker ({setPosition, onAddressFound, disabled}) {
  const t=useRef();

  useMapEvents ({
    click: ({latlng: {lat, lng}}) => {
      if (disabled) return;

      setPosition(lat, lng);
      clearTimeout(t.current);

      t.current=setTimeout(() => {

fetch(`https://nominatim.openstreetmap.org/reverse?format=jsonv2&
lat=${lat}&lon=${lng}`, {
  headers: { 'Accept-Language': 'uk, en' }
})
  .then(r => r.ok ? r.json() : Promise.reject())
  .then(d => d?.display_name && onAddressFound(d.display_name))
  .catch(() => onAddressFound(null));
}, 500);
    }
  });

  return null;
}

{
const [activeTab, setActiveTab] = useState('details');
const [messages, setMessages] = useState([]);
const [newMessage, setNewMessage] = useState('');
const [localParticipants, setLocalParticipants] = useState([]);
const messagesEndRef = useRef(null);

useEffect(() => {
  setActiveTab('details');
  setLocalParticipants(selectedEvent?.participants || []);
}, [selectedEvent?.id]);

useEffect(() => {
  if (!selectedEvent?.id) return;

  socketService.connect();
  socketService.joinEvent(selectedEvent.id);

  const joined = d =>
    d.eventId === selectedEvent.id && d.user &&
    setLocalParticipants(p =>
      p.some(x => x.id === d.user.id) ? p : [...p, d.user]
    );
}
```

```

const left=d=>
  d.eventId===selectedEvent.id&&
  setLocalParticipants(p=>p.filter(x=>x.id!==d.userId));

socketService.onParticipantJoined(joined);
socketService.onParticipantLeft(left);

return()=>{
  socketService.offParticipantJoined(joined);
  socketService.offParticipantLeft(left);
};
},[selectedEvent?.id]);

useEffect(()=>{
  if(activeTab!=='chat'&&!selectedEvent?.id)return;

  socketService.connect();
  socketService.joinEvent(selectedEvent.id);

  const newMsg=m=>setMessages(p=>[...p,m]);

  const
  delMsg=d=>setMessages(p=>p.filter(m=>m.id!==d.messageId));

  socketService.onNewMessage(newMsg);
  socketService.onMessageDeleted(delMsg);

  eventService.getEventMessages(selectedEvent.id)
    .then(setMessages)
    .catch(console.error);

  return()=>{
    socketService.leaveEvent(selectedEvent.id);
    socketService.offNewMessage(newMsg);
    socketService.offMessageDeleted(delMsg);
  };
},[activeTab,selectedEvent?.id]);

useEffect(()=>{

  activeTab==='chat'&&messagesEndRef.current?.scrollIntoView({behavior:'smooth'});
},[messages]);

const onSendMessage=e=>{
  e.preventDefault();
  if(!newMessage.trim())return;
  socketService.sendMessage(selectedEvent.id,newMessage);
  setNewMessage('');
};

const
canJoin=isEditMode&&!hasEventAccess&&!localParticipants.some(p=>p
.id===currentUser?.id)&&currentUser;

```

```

const handlePhotoUpload=async e=>{
  const file=e.target.files[0];
  if(!file)return;

  const fd=new FormData();
  fd.append('image',file);

  try{
    const{data}=await API.post('/upload',fd,{
      headers:{'Content-Type':'multipart/form-data'}
    });

    handleFormChange({
      target:{
        name:'photo_url',

value:`${API.defaults.baseURL.replace('/api','')}${data.imageUrl}
      },
    });
  }catch(err){console.error(err);}
};

return<>
  <aside className="create-event-panel">
    <div className="panel-header">
      <h2>{selectedEvent?'Деталі':'Створити'}</h2>
      <button onClick={onClose}>X</button>
    </div>

    {isEditMode&&
      <div className="event-tabs">
        {['details','chat'].map(x=>
          <button
            key={x}
            className={clsx(activeTab===x&&'active')}
            onClick={()=>setActiveTab(x)}
          >
            {x==='details'?'Деталі':'Чат'}
          </button>
        )}
      </div>
    }

    {activeTab==='details'
      ?<form onSubmit={handleCreateEvent}>
        <div>
          <label>Photo</label>

          <div className="event-photo-preview">
            {formData.photo_url
              ?<img src={formData.photo_url} alt="" />

```

```

        :<FaCalendarAlt size={40} />
      }

      {(!isEditMode||hasEventAccess)&&
        <label htmlFor="event-photo-upload">
          <FaPencilAlt />
        </label>
      }
    </div>

    <input
      id="event-photo-upload"
      type="file"
      accept="image/*"
      hidden
      onChange={handlePhotoUpload}
      disabled={isEditMode&&!hasEventAccess}
    />
  </div>

  {[
    ['title', 'Назва *', 'text'],
    ['location', 'Місце *', 'text'],
    ['event_date', 'Дата *', 'datetime-local']
  ].map(([name, label, type])=>
    <div key={name}>
      <label>{label}</label>

      <input
        type={type}
        name={name}
        value={formData[name]}
        onChange={handleFormChange}
        required
        disabled={!hasEventAccess&&isEditMode}
      />
    </div>
  )}

  <div>
    <label>Мапа</label>

    <MapContainer
      center={[formData.latitude||50.45, formData.longitude||30.52]}
      zoom={13}
    >
      <TileLayer
        url="https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png"/>

      <LocationMarker
        disabled={!hasEventAccess&&isEditMode}
        setPosition={(lat, lng)=>{

```

```

handleFormChange ({target: {name: 'latitude', value: lat}});

handleFormChange ({target: {name: 'longitude', value: lng}});
    }}
  />
</MapContainer>
</div>

<div>
  <button type="button" onClick={onClose}>Закрити</button>

  {canJoin&&
    <button
      type="button"
      onClick={() => handleJoinEvent (selectedEvent.id) }
    >
      Приєднатися
    </button>
  }

  { (!isEditMode || hasEventAccess) &&
    <button type="submit" disabled={submitting}>
      {selectedEvent ? 'Зберегти' : 'Створити'}
    </button>
  }
</div>
</form>

:<div className="chat-container">
  <div className="chat-messages-area">
    {messages.map (msg=>
      <div
        key={msg.id}
        className={clsx(
          'chat-message-wrapper',
          msg.sender_id === currentUser?.id
            ? 'chat-message-mine'
            : 'chat-message-theirs'
        )}
      >
        <div>{msg.text}</div>
      </div>
    )}

    <div ref={messagesEndRef}/>
  </div>

  <form onSubmit={onSendMessage}>
    <input
      value={newMessage}
      onChange={e=> setNewMessage (e.target.value) }
      placeholder="Повідомлення..."
    >
  </form>
</div>

```

```

    />

    <button type="submit">
      <FaPaperPlane/>
    </button>
  </form>
</div>}
</aside>

{showDeleteConfirm&&
  <div className="delete-confirm-modal">
    <div className="delete-confirm-content">
      <h3>Видалити?</h3>

      <div>
        <button
onClick={ ()=>setShowDeleteConfirm(false) }>Hi</button>
        <button onClick={handleDeleteEvent}>Так</button>
      </div>
    </div>
  </div>
}
</>;
}

```

ДОДАТОК В
СЛАЙДИ ПРЕЗЕНТАЦІЇ

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЗАПОРІЗЬКА ПОЛІТЕХНІКА»
КАФЕДРА ПРОГРАМНИХ ЗАСОБІВ

ТЕМА ДИПЛОМНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ БАКАЛАВРА:
**РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ
ОРГАНІЗАЦІЇ ТА ПРОВЕДЕННЯ ОФЛАЙН-ЗАХОДІВ.
DEVELOPMENT OF SOFTWARE FOR ORGANIZING AND
RUNNING OFFLINE EVENTS**

Виконавець: Михайло ГРИГОРОВ
Керівник: к.т.н., доцент Степан СКРУПСЬКИЙ

2026

Рисунок В.1 – Слайд 1

Об'єкт, предмет, мета роботи

- об'єкт роботи - процес інженерії програмного забезпечення для організації офлайн-заходів;
- предмет роботи - програмне забезпечення для керування офлайн-заходами, що базуватиметься на автоматизованих модулях реєстрації учасників, планування та модерування заходів;
- мета роботи - зменшення часу організації офлайн-заходів та зменшення витрат ресурсів організаторів шляхом розроблення програмного забезпечення що автоматизує реєстрацію учасників та планування заходів.

Рисунок В.2 – Слайд 2

Завдання роботи

- проаналізувати програмне забезпечення для організації офлайн-заходів та пошуку учасників;
- визначити методи та засоби проектування програмного забезпечення;
- сформулювати вимоги до системи організації офлайн-заходів;
- виконати реалізацію програмного забезпечення для керування заходами та взаємодії користувачів.

3

Рисунок В.3 – Слайд 3

Порівняння деяких програм організації заходів

Критерій	Meetup	Eventbrite	Розробка
Визначення заходів	Як окремі заходи з текстовим описом	Як заходи з описом та квитками	Як заходи, що створюються користувачами з деталізацією
Фільтрація за інтересами	Загальні категорії	Обмежена	Назва, місце, дата, категорії, кількістю людей
Керування учасниками	Є	Є (з оплатою)	Є
Відображення деталей заходу	Текстовий опис	Текстовий опис	Структуровані дані в окремих полях
Можливість створення власних заходів	Так	Так	Так
Комунікація між учасниками	Обмежена	Обмежена	Чат заходу та особистий
Автоматизація організації	Часткова	Часткова	Реєстрація, керування учасниками, планування

4

Рисунок В.4 – Слайд 4

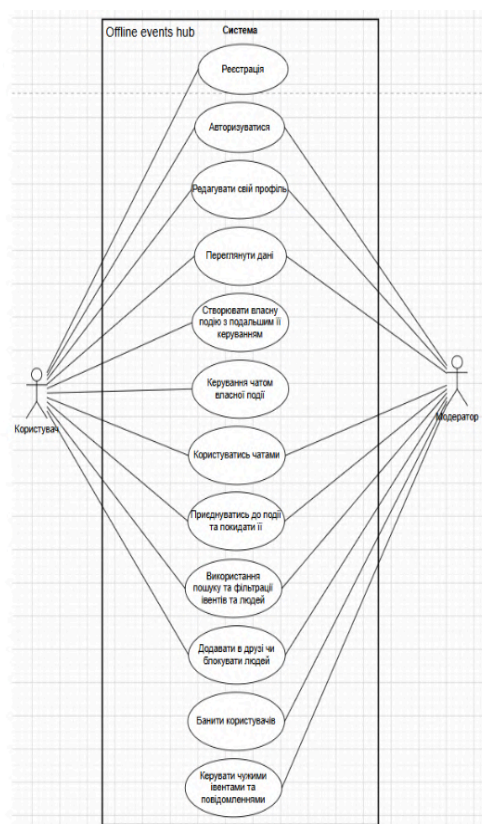
Порівняння засобів для розробки програми

Критерій	React	Vue
Екосистема	Найбільша зріла спільнота, багато інструментів.	Менша, але зростаюча спільнота, "монолітні" рішення.
Гнучкість	Максимальна гнучкість, кастомна архітектура.	Більш стандартизований підхід.
Синтаксис	Використовує JSX (JS + розмітка).	Використовує окремі HTML-шаблони.

5

Рисунок В.5 – Слайд 5

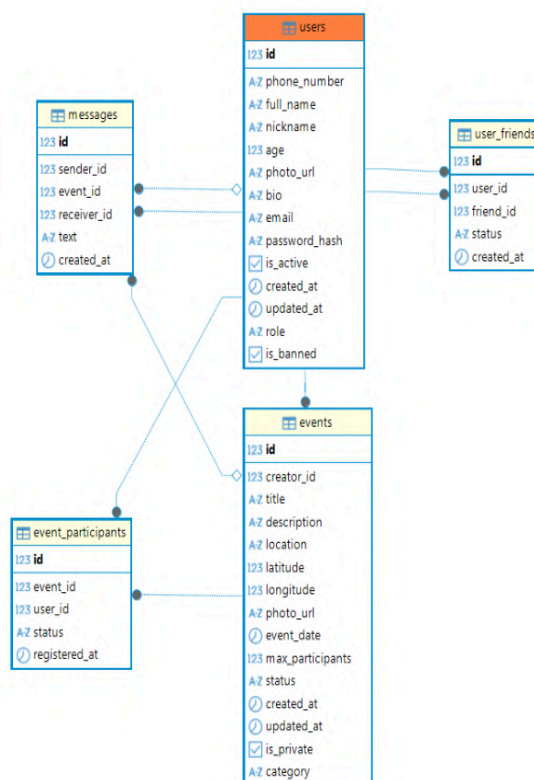
Use-case діаграма



6

Рисунок В.6 – Слайд 6

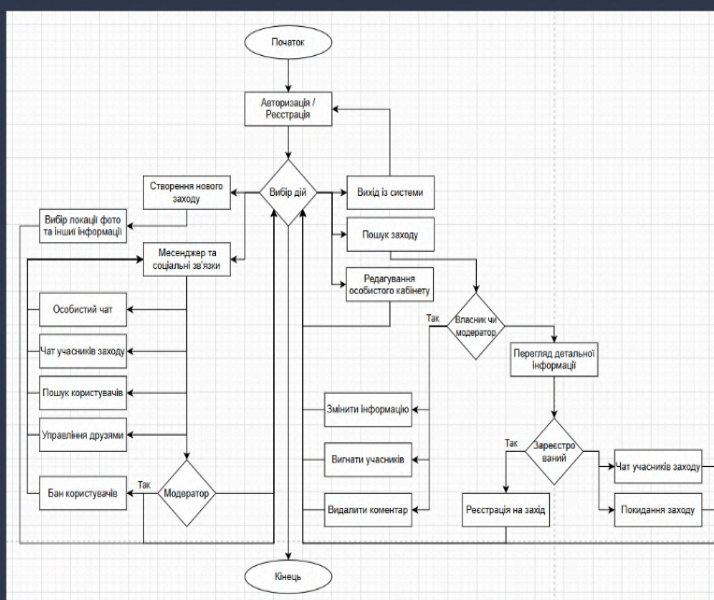
ER-модель БД



7

Рисунок В.7 – Слайд 7

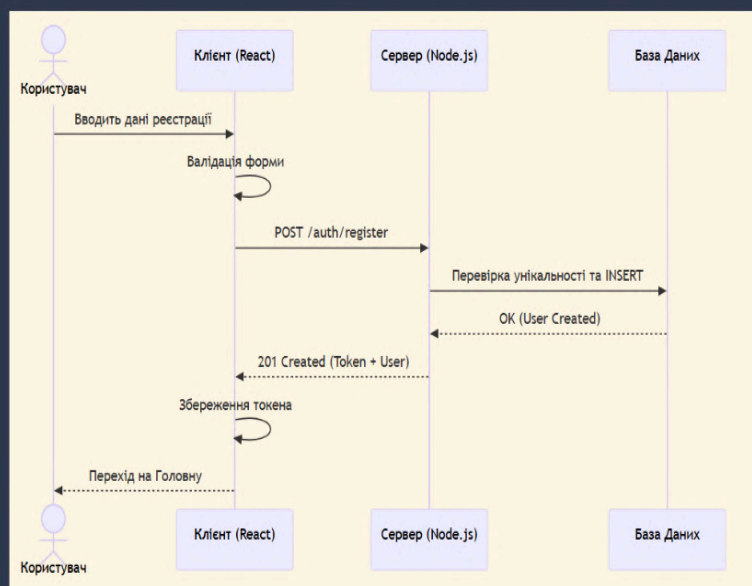
Функціональна схема програми



8

Рисунок В.8 – Слайд 8

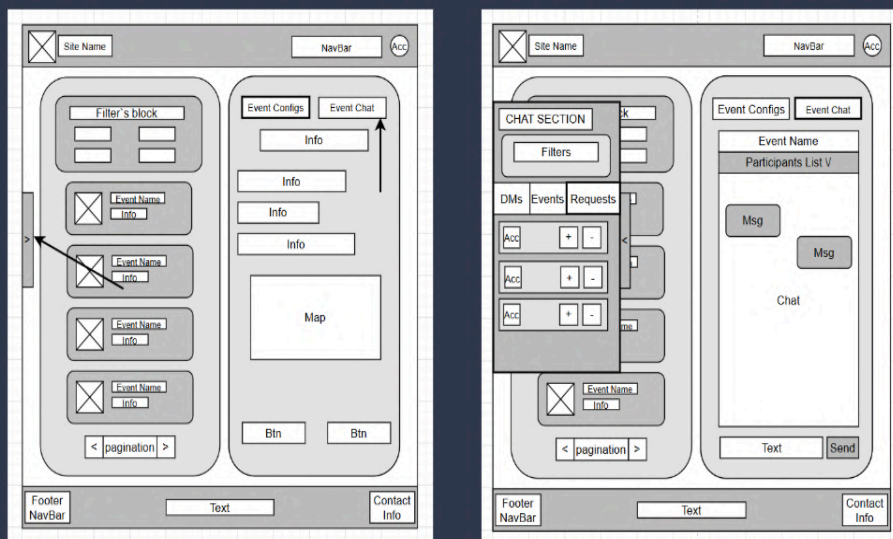
UML схема реєстрації користувача



9

Рисунок В.9 – Слайд 9

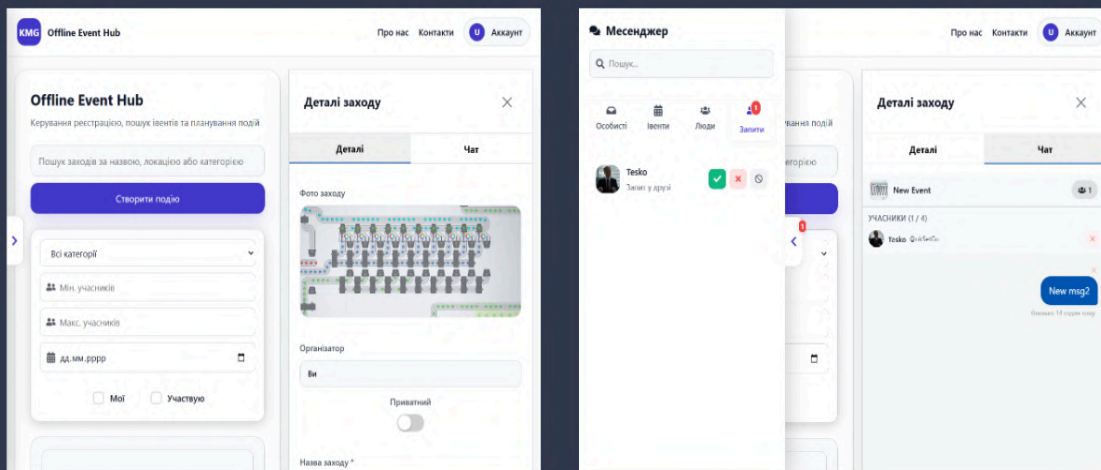
Розроблені прототипи



10

Рисунок В.10 – Слайд 10

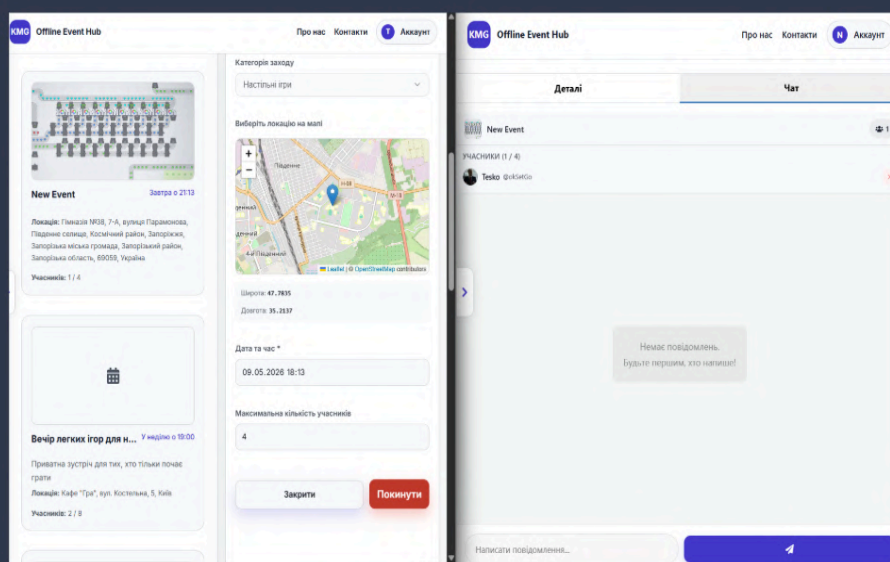
Вікна застосунку, що відповідають прототипам



11

Рисунок В.11 – Слайд 11

Інтерфейси приєднання до заходу та чату



12

Рисунок В.12 – Слайд 12

Висновки

- було проаналізовано існуючі програмні забезпечення для організації офлайн-заходів та пошуку учасників. Було виявлено нестачу необхідного функціоналу та обмеження. Тому було вирішено створити нову систему, спеціально для офлайн-подій;
- було визначено та пояснено, використання методів та засобів проєктування програмного забезпечення. Весь обраний набір інструментів та технологій є сучасним, що гарантує швидку роботу, масштабування;
- було сформульовано вимоги до системи, що стали основою структури бази даних. До цих вимог були включені: реєстрація учасників, модерація, повне керування заходами, переписками та акаунтами;
- зрештою, було розроблено програму для керування офлайн-заходами та взаємодії користувачів. Вона відповідає поставленим задачам, швидко працює та має зручний інтерфейс.

Рисунок В.13 – Слайд 13