

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ  
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет інформаційних технологій**

**ПОГОДЖЕНО**  
**Декан факультету**

\_\_\_\_\_ **Ігор БОЛБОТ**  
(підпис)

«\_\_\_» \_\_\_\_\_ 2026 р.

**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**  
**Завідувач кафедри комп'ютерних наук**

\_\_\_\_\_ **Белла ГОЛУБ**  
(підпис)

«\_\_\_» \_\_\_\_\_ 2026 р.

**БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА**

**на тему: «Програмне забезпечення вебзастосунку для планування  
та керування зустрічами»**

**Спеціальність «121 Інженерія програмного забезпечення»**

**Освітньо-професійна програма «Інженерія програмного забезпечення»**

**Гарант освітньої програми**

к.т.н., доцент

\_\_\_\_\_ **Ганна ВАЙГАНГ**  
(підпис)

**Керівник бакалаврської  
кваліфікаційної роботи**

\_\_\_\_\_ **Максим НЕДЬОШЕВ**  
(підпис)

**Консультант бакалаврської  
кваліфікаційної роботи**

д.е.н., професор

\_\_\_\_\_ **Роман РУДЕНСЬКИЙ**  
(підпис)

**Виконав**

\_\_\_\_\_ **Тимофій СВИРИДЕНКО**  
(підпис)

**Київ-2026**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ  
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ  
Факультет інформаційних технологій**

**ЗАТВЕРДЖУЮ**

Завідувач кафедри комп'ютерних наук

к.т.н., доцент \_\_\_\_\_ Белла ГОЛУБ  
(підпис)

«\_\_\_» \_\_\_\_\_ 2025 р.

**ЗАВДАННЯ  
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ  
ЗДОБУВАЧУ**

Свириденко Тимофію Андрійовичу  
(прізвище, ім'я, по батькові)

Спеціальність «121 Інженерія програмного забезпечення»

Освітньо-професійна програма «Інженерія програмного забезпечення»

Тема бакалаврської кваліфікаційної роботи «Програмне забезпечення вебзастосунку для планування та керування зустрічами»

затверджена наказом від «19» грудня 2025 р. № 3196 «С»

Термін подання завершеної роботи на кафедру «22» травня 2026 р.

Вихідні дані до бакалаврської кваліфікаційної роботи: документація Laravel, вимоги до застосунку для організації зустрічей

Перелік питань, що підлягають дослідженню: Аналіз предметної області, проектування моделі даних та реалізація бази даних, проектування архітектури та розробка застосунку, тестування, розгортання та експлуатація системи.

\_\_\_\_\_  
Перелік графічних документів (за необхідності).

\_\_\_\_\_  
Дата видачі завдання «19» грудня 2025 р.

**Керівник бакалаврської  
кваліфікаційної роботи**

\_\_\_\_\_  
(підпис)

**Максим НЕДЬОШЕВ**

**Консультант бакалаврської  
кваліфікаційної роботи**

д.е.н., професор

\_\_\_\_\_  
(підпис)

**Роман РУДЕНСЬКИЙ**

**Завдання взяв до виконання**

\_\_\_\_\_  
(підпис)

**Тимофій СВИРИДЕНКО**

## ЗМІСТ

|                                                                                     |    |
|-------------------------------------------------------------------------------------|----|
| ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ .....                                                     | 5  |
| ВСТУП.....                                                                          | 6  |
| 1 ВЕБ-ЗАСТОСУНОК ДЛЯ ПЛАНУВАННЯ ГРУПОВИХ ЗУСТРІЧЕЙ ЯК<br>ОБ'ЄКТ ДОСЛІДЖЕННЯ.....    | 8  |
| 1.1 Огляд інформаційних джерел та існуючих рішень.....                              | 9  |
| 1.2 Моделювання предметної області .....                                            | 16 |
| 1.3 Постановка завдання.....                                                        | 18 |
| 2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ .....                                    | 23 |
| 2.1 Логічна модель даних у вигляді ER-діаграми .....                                | 23 |
| 2.2 Вибір системи управління базою даних.....                                       | 26 |
| 2.3 Розробка структури інформаційної бази.....                                      | 28 |
| 3 РОЗРОБКА ПРИКЛАДНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ.....                                | 35 |
| 3.1 Організаційна структура програмного забезпечення .....                          | 35 |
| 3.2 Моделювання структури та взаємодії об'єктів .....                               | 36 |
| 3.3 Компонентна структура архітектури системи.....                                  | 42 |
| 3.4 Вибір інструментарію для створення прикладного програмного<br>забезпечення..... | 44 |
| 3.3 Алгоритмізація та програмування програмних модулів .....                        | 47 |
| 4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ<br>СИСТЕМИ.....                    | 59 |
| 4.1 Тестування системи.....                                                         | 59 |
| 4.2 Вимоги до апаратного та програмного забезпечення .....                          | 61 |
| 4.3 Склад інсталяційного пакету.....                                                | 62 |
| ВИСНОВКИ .....                                                                      | 66 |
| СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....                                                     | 68 |
| ДОДАТОК А .....                                                                     | 70 |
| ДОДАТОК Б.....                                                                      | 81 |
| ДОДАТОК В .....                                                                     | 90 |
| ДОДАТОК Д .....                                                                     | 93 |
| ДОДАТОК Е.....                                                                      | 94 |
| ДОДАТОК Ж .....                                                                     | 95 |

|                 |     |
|-----------------|-----|
| ДОДАТОК И ..... | 96  |
| ДОДАТОК К ..... | 97  |
| ДОДАТОК Л ..... | 100 |
| ДОДАТОК М ..... | 109 |

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

VPS (Virtual Private Server) - віртуальний приватний сервер

MVC (Model-View-Controller) - архітектурний патерн «Модель-Вигляд-Контролер»

ORM (Object-Relational Mapping) - об'єктно-реляційне відображення

UML (Unified Modeling Language) - уніфікована мова моделювання

ER (Entity-Relationship) - сутність-зв'язок (модель даних)

API (Application Programming Interface) - інтерфейс програмування застосунків

REST (Representational State Transfer) - архітектурний стиль побудови API

OAuth - відкритий протокол авторизації

HTTPS (HyperText Transfer Protocol Secure) - захищений протокол передачі даних

SSL/TLS (Secure Sockets Layer / Transport Layer Security) - протоколи криптографічного захисту з'єднання

CSRF (Cross-Site Request Forgery) - міжсайтова підробка запиту

XSS (Cross-Site Scripting) - міжсайтовий скриптинг

СУБД - система управління базами даних

SQL (Structured Query Language) - мова структурованих запитів

JSON (JavaScript Object Notation) - текстовий формат обміну даними

JSONB - бінарне представлення JSON у PostgreSQL

UUID (Universally Unique Identifier) - універсально унікальний ідентифікатор

ENUM (Enumeration) - перелічуваний тип даних

PHP (PHP: Hypertext Preprocessor) - мова програмування для веб-розробки

TSX (TypeScript XML) - розширення TypeScript із підтримкою JSX-синтаксису

DOM (Document Object Model) - об'єктна модель документа

DNS (Domain Name System) - система доменних імен

DDoS (Distributed Denial of Service) - атака на відмову в обслуговуванні

ППЗ - прикладне програмне забезпечення

ЗНФ - третя нормальна форма бази даних

## ВСТУП

В умовах сучасного темпу життя управління часом стає одним із ключових факторів продуктивності як для окремих осіб, так і для команд. Використання традиційних засобів комунікації - електронної пошти або месенджерів - для узгодження часу зустрічей є неефективним, а координація групових заходів за участю великої кількості учасників із різними графіками перетворюється на складну організаційну задачу.

Існуючі програмні рішення переважно орієнтовані на корпоративний сектор або на планування зустрічей у форматі «один-на-один». Водночас на ринку спостерігається дефіцит інструментів, які б поєднували автоматизований пошук оптимального часу для групи, можливість пошуку спільнот за інтересами та організацію публічних подій в єдиній платформі.

Актуальність теми зумовлена необхідністю розробки веб-застосунку, який автоматизує процеси узгодження розкладів і водночас забезпечує готову платформу для формування та управління спільнотами. Такий підхід дозволяє вирішити практичні задачі організації заходів для студентських груп, клубів за інтересами та професійних об'єднань.

Метою роботи є проектування та реалізація програмного забезпечення веб-застосунку для планування і керування груповими зустрічами, орієнтованого на широке коло користувачів - від приватних осіб до організаторів публічних подій.

У роботі застосовано методи системного аналізу для формування функціональних вимог та методи об'єктно-орієнтованого проектування для побудови архітектури системи. Реалізацію виконано з використанням сучасного стека веб-технологій. Систему розгорнуто на віддаленому VPS-сервері для забезпечення публічного доступу. Вибір зазначених технологій обумовлений їхньою зрілістю, широкою екосистемою та придатністю для швидкої розробки масштабованих застосунків.

Основні положення роботи було представлено на конференції під назвою ТЕОРЕТИЧНІ ТА ПРИКЛАДНІ АСПЕКТИ РОЗРОБКИ КОМП'ЮТЕРНИХ

СИСТЕМ '2026.

Пояснювальна записка містить 4 розділи, висновки та список використаних джерел. У першому розділі проведено системний аналіз предметної області: сформульовано постановку завдання, визначено функціональні та нефункціональні вимоги, виконано огляд існуючих аналогів і побудовано UML-моделі системи. У другому розділі описано інформаційне забезпечення: спроектовано логічну модель даних, обґрунтовано вибір PostgreSQL як СУБД та описано фізичну реалізацію бази даних. У третьому розділі розглянуто прикладне програмне забезпечення: архітектуру системи, технологічний стек та реалізацію семи функціональних модулів застосунку. Четвертий розділ містить рекомендації щодо впровадження та експлуатації системи: результати тестування, вимоги до апаратного й програмного забезпечення та інструкцію з розгортання.

# **1 ВЕБ-ЗАСТОСУНОК ДЛЯ ПЛАНУВАННЯ ГРУПОВИХ ЗУСТРІЧЕЙ ЯК ОБ'ЄКТ ДОСЛІДЖЕННЯ**

Питання ефективного управління часом та координації групових комунікацій широко обговорюється як у сучасному корпоративному середовищі, так і у сфері громадської та студентської взаємодії. Дана проблема перебуває на перетині соціальної взаємодії та технічної оптимізації, що зумовлює різноманітність підходів до її вирішення з боку фахівців у галузі менеджменту, соціології, інформаційних технологій та психології управління.

Програмні рішення, представлені на сучасному ринку, вирішують окремі аспекти організації групових заходів, проте кожне з них має суттєві обмеження. Calendly та Doodle орієнтовані переважно на узгодження розкладів у форматі «один-на-один» або для невеликих груп і не пропонують інструментів для побудови спільнот. Clockwise зосереджений на автоматизації особистого та командного календаря виключно в корпоративному контексті. Meetup, натомість, успішно вирішує задачу формування тематичних спільнот та організації публічних подій, однак позбавлений функціоналу для автоматизованого узгодження оптимального часу зустрічі серед учасників групи. Таким чином, жодне з існуючих рішень не поєднує алгоритми пошуку спільного вільного часу з інструментами управління спільнотами - саме цю нішу може заповнює розроблюваний застосунок.

Процес планування зустрічей та координації подій є важливим як для організаторів - лідерів команд, модераторів клубів - так і для учасників. Організатори прагнуть знайти оптимальний час для залучення максимальної кількості осіб з мінімальними адміністративними витратами. Учасники, своєю чергою, шукають події та спільноти, що відповідають їхнім інтересам, і потребують зручних інструментів для інтеграції цих подій у власний графік.

## 1.1 Огляд інформаційних джерел та існуючих рішень

При виборі аналогів програмного забезпечення для планування та керування груповими зустрічами були розглянуті додатки, які займають лідируючі позиції на ринку: Calendly, Meetup Doodle та Clockwise. Кожне з цих рішень було проаналізовано з точки зору їх основних функцій, переваг та недоліків, щоб визначити їх релевантність для розроблюваної системи та виявити функціональні ніші, які залишаються незаповненими.

### 1.1.1. Аналог: Calendly

Calendly є хмарним сервісом для автоматизації процесу узгодження зустрічей. Користувач отримує персональне посилання, яке відображає його доступний час, - запрошені самостійно обирають зручний слот без жодного листування. Платформа підтримує двосторонню синхронізацію з популярними календарними сервісами. Переваги та недоліки Calendarly наведені в табл. 1.3.

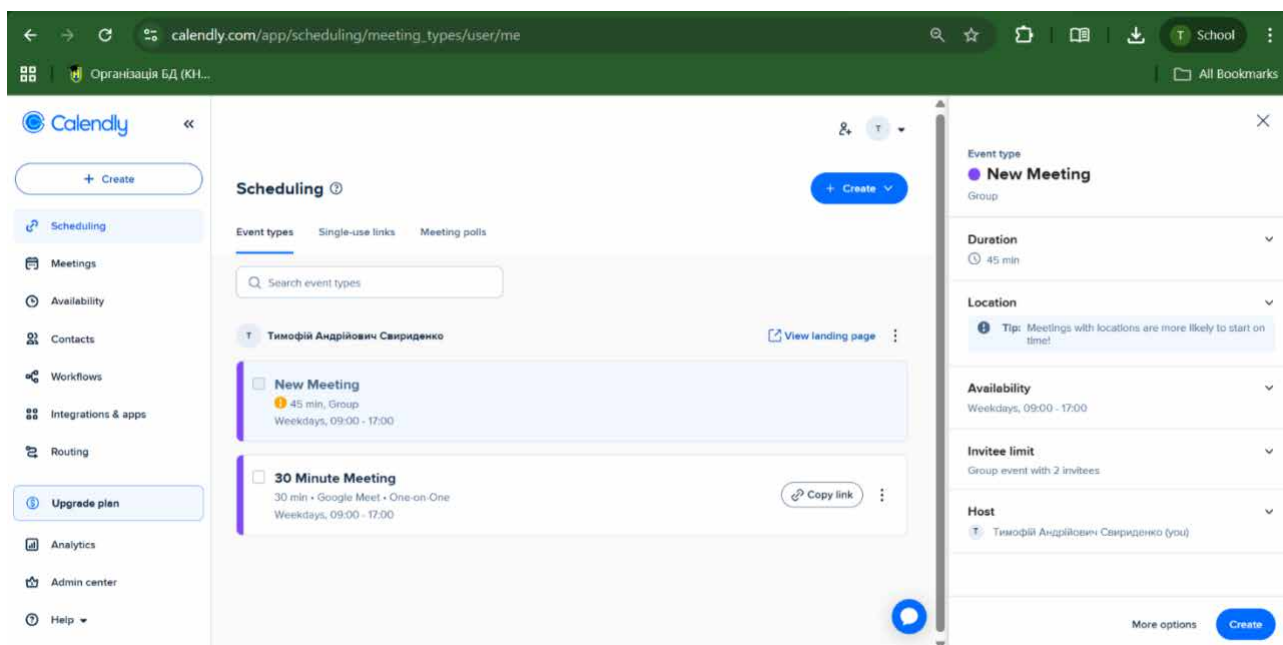


Рис. 1.1 Сторінка з зустрічами користувача

Рис. 1.2 Форма створення нової зустрічі

Таблиця 1.3

## Переваги та недоліки Calendly

| Програма        | Переваги                                                                                                                                                                                                                                                                                                           | Недоліки                                                                                                                                                                                                                                                                           |
|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Calendly</b> | Зрозумілий та зручний інтерфейс;<br>широкі можливості інтеграції з CRM-системами, відеоконференціями та платіжними сервісами;<br>підтримка як індивідуального, так і командного режимів роботи;<br>цілодобова підтримка користувачів у платних тарифах;<br>наявність мобільного додатку та розширення для браузера | Обмежені інструменти для проведення групових опитувань;<br>переважна орієнтація на формат зустрічей один-на-один;<br>відсутність публічного каталогу для пошуку подій; висока вартість платних тарифів для невеликих команд; суттєво обмежена функціональність безкоштовного плану |

**1.1.2. Аналог: Doodle**

Doodle - це спеціалізований сервіс для колективного погодження часу

зустрічей. Організатор формує опитування з переліком можливих часових слотів, а учасники позначають зручні для них варіанти. На основі зібраних відповідей система визначає оптимальний час, який влаштовує найбільшу кількість людей. Переваги та недоліки Doodle наведені в табл. 1.6.

Add your times

Duration

60 min

90 min

120 min

All day

+ Custom duration

Week

Month

←

→

Today

October 5 - 11

Ukraine, Kyiv (GMT+3) ▾

|          | SUN<br>5 | MON<br>6 | TUE<br>7 | WED<br>8 | THU<br>9 | FRI<br>10 | SAT<br>11 |
|----------|----------|----------|----------|----------|----------|-----------|-----------|
| All day  |          |          |          |          |          |           |           |
| 12:00 AM |          |          |          |          |          |           |           |
| 1:00 AM  |          |          |          |          |          |           |           |
| 2:00 AM  |          |          |          |          |          |           |           |
| 3:00 AM  |          |          |          |          |          |           |           |
| 4:00 AM  |          |          |          |          |          |           |           |
| 5:00 AM  |          |          |          |          |          |           |           |
| 6:00 AM  |          |          |          |          |          |           |           |

0 of 60 times selected

Рис. 1.4 Елемент створення нової зустрічі з календарем

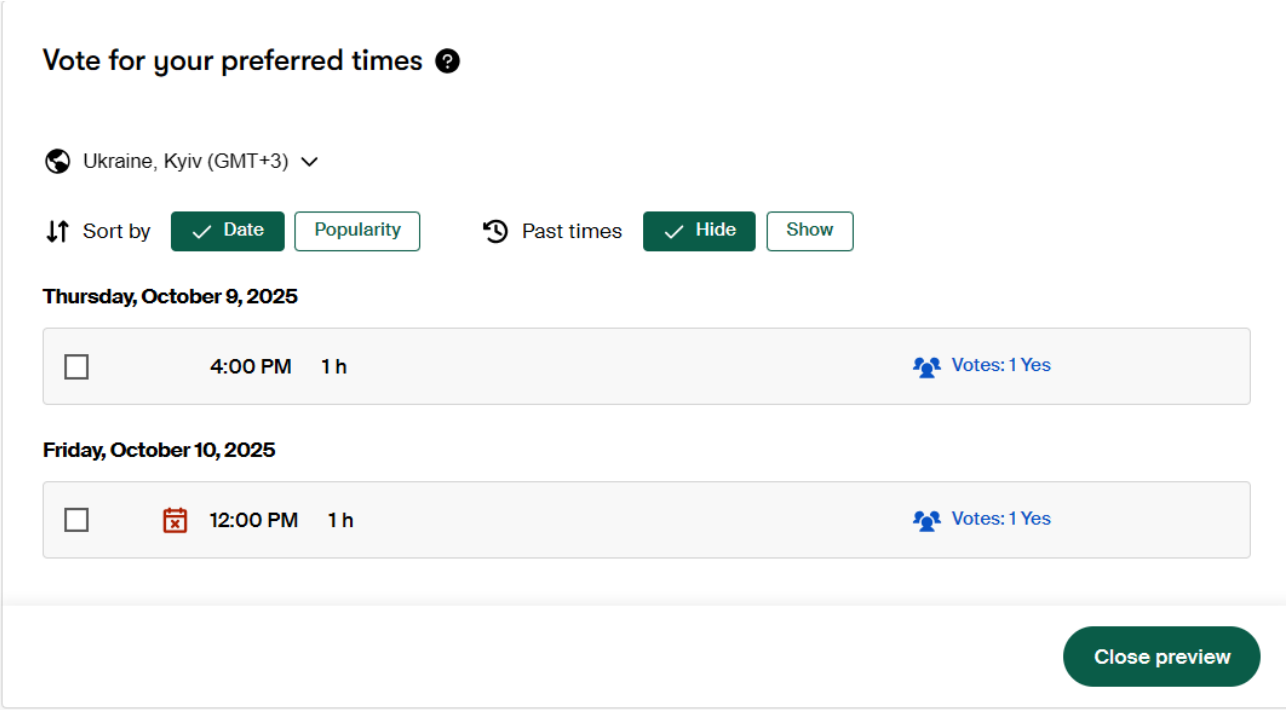


Рис. 1.5 Сторінка голосування за за час проведення зустрічі

Таблиця 1.6

Переваги та недоліки Doodle

| Програма | Переваги                                                                                                                                                                                                                                 | Недоліки                                                                                                                                                               |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Doodle   | Зручна координація великих груп учасників;<br>швидке налаштування опитування;<br>простий механізм голосування без обов'язкової реєстрації;<br>наочна візуалізація результатів;<br>можливість пропонувати кілька варіантів часу одночасно | наявність реклами у безкоштовній версії;<br>відсутність AI-алгоритмів оптимізації;<br>немає функції пошуку груп за тематикою;<br>слабкий модуль аналітики та звітності |

1.1.3. Аналог: Meetup

Meetup - це платформа, орієнтована на створення спільнот, організацію та пошук групових заходів (як онлайн, так і офлайн) за спільними інтересами. На відміну від інструментів для точкового узгодження часу, Meetup дозволяє

організаторам створювати публічні чи приватні групи, анонсувати заплановані зустрічі, керувати списком учасників та взаємодіяти з аудиторією. Основний акцент системи зроблено на соціальній взаємодії, побудові ком'юніті та можливості публічного пошуку подій для людей, які шукають нові знайомства або професійний нетворкінг. Переваги та недоліки Meetup наведені в табл. 1.8.

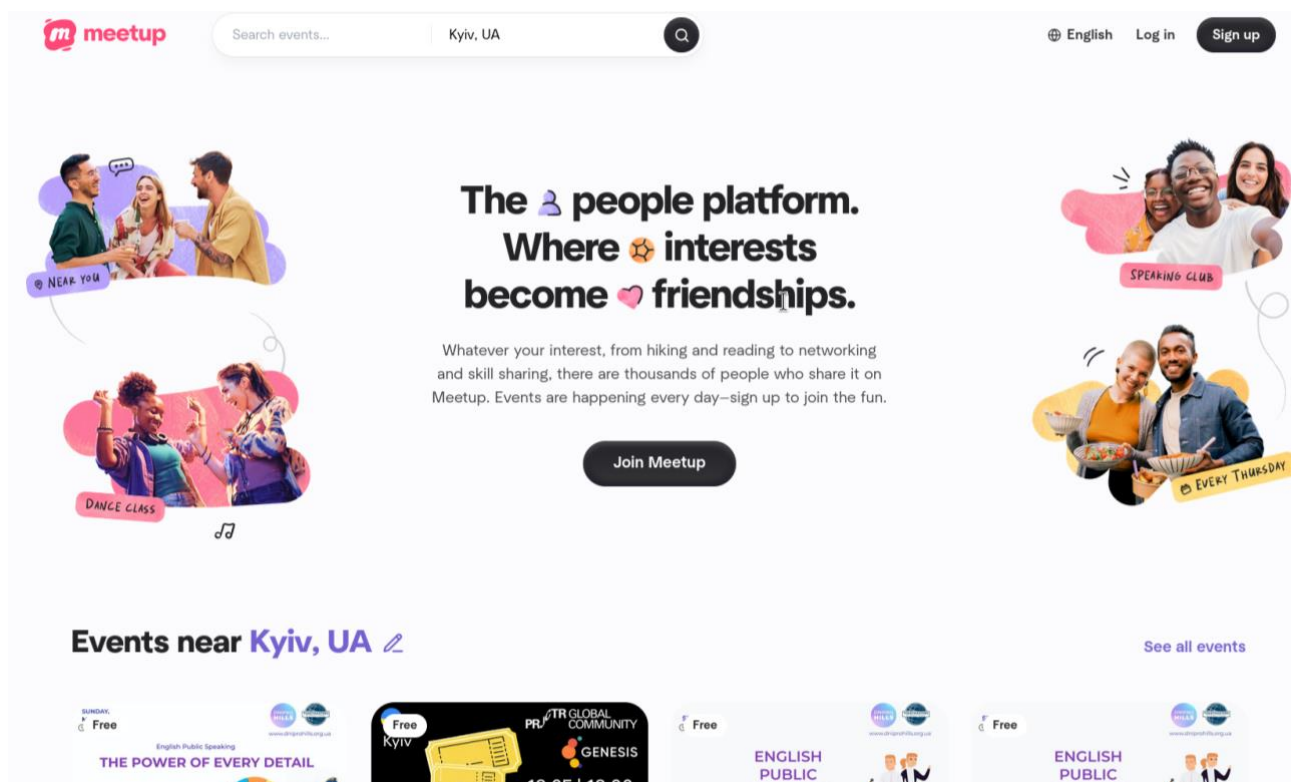


Рис. 1.7 Головна сторінка з пошуком подій за локацією

Таблиця 1.8

## Переваги та недоліки Meetup

| Програма      | Переваги                                                                                                                                                                                                                                                                                                                        | Недоліки                                                                                                                                                                 |
|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Meetup</b> | Потужні інструменти для побудови та управління спільнотами;<br>наявність великого публічного каталогу подій та зручний пошук за геолокацією/темами;<br>вбудовані інструменти комунікації (форуми, чат);<br>можливість монетизації зустрічей (продаж квитків, збір внесків);<br>мобільний додаток для залучення нових учасників; | Відсутність функціоналу для опитувань щодо вибору оптимального часу зустрічі для всіх;<br>висока вартість підписки для організаторів (платне створення та ведення груп); |

**1.1.4. Аналог: Clockwise**

Clockwise - це інструмент інтелектуального управління розкладом, орієнтованим на підвищення продуктивності команд. Завдяки вбудованому алгоритму штучного інтелекту платформа самостійно аналізує завантаженість учасників і перерозподіляє зустрічі таким чином, щоб зберегти максимальні блоки часу для зосередженої роботи. При формуванні розкладу система враховує як корпоративні пріоритети, так і особисті налаштування кожного співробітника. Переваги та недоліки Clockwise наведені в табл. 1.10.

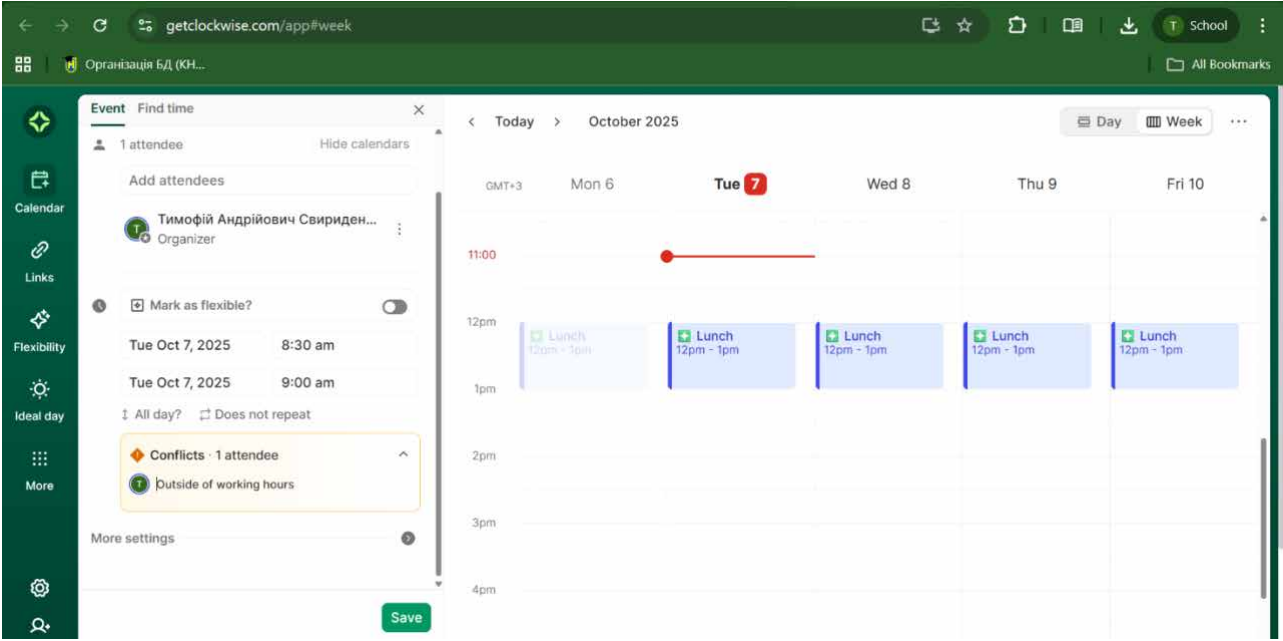


Рис. 1.9 Сторінка з профілем та календарем користувача

Таблиця 1.10

Переваги та недоліки Clockwise

| Програма         | Переваги                                                                                                                                                                                                                                                                                         | Недоліки                                                                                                                                                                                                                       |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Clockwise</b> | Розвинена AI-оптимізація календаря;<br>автоматичне резервування блоків часу для глибокої роботи;<br>аналітика командної продуктивності;<br>автоматичне виявлення та вирішення конфліктів у розкладі;<br>підтримка балансу між роботою та відпочинком (перерви, обідній час); інтеграція зі Slack | Складне початкове налаштування;<br>сумісність виключно з Google Calendar;<br>висока вартість ліцензії для невеликих команд;<br>відсутність публічного каталогу груп;<br>надмірна функціональність для базових задач планування |

## 1.2 Моделювання предметної області

Для візуалізації та моделювання процесів та взаємодій у веб-застосунку для планування та керування груповими зустрічами використано два типи діаграм з нотації UML (Unified Modeling Language) [12].

### 1.2.1 Діаграма прецедентів.

Діаграма випадків використання (прецедентів) представляє функціональні вимоги до системи та ілюструє взаємодію користувачів із нею. На діаграмі прецедентів (див. Рис. 2.1) зображено двох основних дійових осіб (акторів): «Член групи» (User) та «Організатор» (Org).

Член групи може виконувати наступні дії (прецеденти):

- Пошук подій: користувач здійснює пошук доступних заходів у системі;
- Перегляд деталей події: ознайомлення з детальним описом, часом та умовами проведення заходу;
- Записатися на подію: реєстрація користувача для участі в заході (цей прецедент розширено логікою, яка за необхідності включає процес «Приєднатися до групи»).

Організатор наділений ширшими повноваженнями і може виконувати такі дії:

- Створити групу: ініціалізація та налаштування нової тематичної спільноти;
- Редагувати назву/опис групи: зміна параметрів існуючої спільноти (включає обов'язкову наявність попередньо створеної групи);
- Керувати членами групи: адміністрування учасників спільноти, опрацювання заявок на вступ;
- Створити подію: формування нового заходу в межах існуючої групи;
- Редагувати або Скасувати подію: управління життєвим циклом створеного заходу (ці дії можливі лише за умови попереднього створення події).

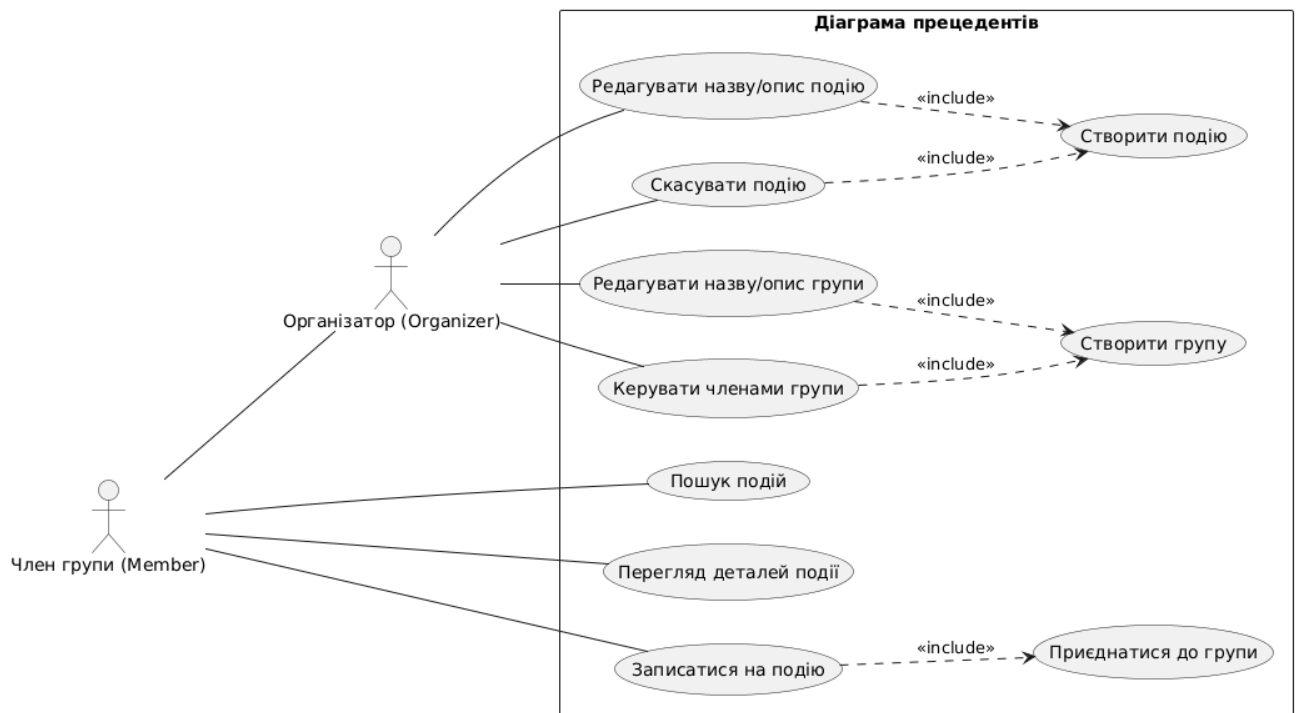


Рис. 2.1 Діаграма прецедентів

### 1.2.2 Діаграма активності.

Було побудовано діаграму активності (графічне представлення наведено в Додатку Е). Вона відображає алгоритм управління «Життєвим циклом події» розділений між чотирма учасниками процесу: Організатором, Користувачем, Групою та Подією.

Процес ініціюється Організатором, який розпочинає зі створення нової тематичної групи (якщо такої ще не існує в системі). Після цього він безпосередньо генерує захід. Наступним етапом є взаємодія з Користувачем, який за допомогою пошуку або рекомендацій знаходить цю подію.

Алгоритм передбачає строгу умову участі - обов'язкове членство у відповідній групі. Якщо Користувач ще не долучився до неї, система ініціює процес валідації: заявка відправляється на розгляд адміністратора групи. У разі відхилення запиту алгоритм для цього користувача завершується. Якщо ж заявку схвалено (або користувач заздалегідь був учасником спільноти), управління переходить до етапу реєстрації на подію, де здійснюється автоматична перевірка

наявності вільних слотів (місць).

У випадку, коли ліміт учасників вичерпано, користувач отримує повідомлення про брак місць і процес зупиняється. За умови ж наявності вільних слотів система успішно фіксує бронювання.

На фінальному етапі логіка процесу повертається до Організатора, який ухвалює остаточне рішення щодо проведення заходу з огляду на актуальні обставини. У разі виникнення перешкод (наприклад, несприятливих погодних умов для активностей на відкритому повітрі), організатор скасовує подію. Якщо ж жодних проблем не виникає, захід проводиться за планом, після чого організатор переводить його у фінальний статус - закритої події.

### **1.3 Постановка завдання**

Розроблювана система належить до класу веб-орієнтованих інформаційних систем. За масштабом - це багатокористувацька платформа з підтримкою одночасної роботи понад 1000 активних користувачів. За характером використання інформації система поєднує оперативну обробку даних (реєстрація на події, голосування, сповіщення) зі збереженням історичних записів (архів зустрічей, статистика участі). За ступенем автоматизації система відноситься до автоматизованих - частина процесів (визначення оптимального часу, надсилання сповіщень, перевірка доступності місць) виконується без участі користувача. За підтримуваними технологіями комунікації система забезпечує взаємодію через веб-інтерфейс з підтримкою email-сповіщень та інтеграції з зовнішніми календарними сервісами.

Система призначена для автоматизації таких процесів: узгодження розкладів групових зустрічей шляхом проведення опитувань; управління складом учасників спільнот; реєстрації на публічні та приватні події; надсилання сповіщень учасникам на різних етапах організації заходу.

Основними об'єктами зберігання та обробки є: облікові записи

користувачів, профілі груп (спільнот), події, опитування з варіантами часу та результатами голосування, журнали активності та сповіщень.

Результатами роботи системи є: сформований оптимальний розклад зустрічі на основі голосування учасників; актуальний список зареєстрованих учасників події; персоналізована стрічка подій та груп для кожного користувача; статистика участі в групах та архів проведених заходів.

### **1.3.1 Функціональні вимоги**

#### **1. Модуль ідентифікації та управління профілем**

Система повинна забезпечувати реєстрацію та авторизацію користувачів через email або обліковий запис Google. Користувач має можливість керувати особистим профілем: редагувати ім'я та псевдонім, завантажувати аватар, налаштовувати часовий пояс та вказувати інтереси для персоналізованого пошуку публічних подій. У профілі відображається історія активності: перелік груп користувача та архів його записів на події.

#### **2. Модуль управління групами (спільнотами)**

Система повинна підтримувати створення публічних і приватних груп із зазначенням назви, опису та тегів. Редагування та архівування групи доступне лише організатору. Передбачено механізм приєднання до групи та виходу з неї, а також управління складом учасників: перегляд списку членів, примусове виключення, надання або скасування прав адміністратора. Рольова модель розмежовує права між Учасником, Адміністратором та Організатором.

#### **3. Модуль управління подіями**

Система повинна дозволяти створювати події в межах групи із зазначенням назви, опису, часу, формату проведення (офлайн або онлайн) та максимальної кількості місць. Організатор має можливість редагувати або скасовувати подію. Учасник може зареєструватися на подію або скасувати реєстрацію; система автоматично перевіряє наявність вільних місць та членство у відповідній групі.

#### **4. Модуль опитувань для узгодження часу**

Система повинна забезпечувати створення опитувань із кількома варіантами дати та часу, встановлення дедлайну голосування та автоматичне визначення оптимального часу зустрічі на основі отриманих голосів. Після завершення опитування учасники отримують запит на підтвердження участі у фінальній зустрічі.

#### 5. Модуль пошуку та навігації

Система повинна надавати каталог публічних груп і подій з можливістю пошуку за ключовими словами, тегами, а також фільтрації за кількістю учасників і активністю. Детальні сторінки групи та події відображають усю необхідну інформацію, включно зі статусом і кількістю вільних місць.

#### 6. Система сповіщень

Система повинна надсилати сповіщення про нові опитування, нагадування про незавершені голосування, повідомлення про результати опитування та нагадування про майбутні зустрічі. Підтримуються канали: електронна пошта, push-сповіщення та внутрішні (in-app) повідомлення.

#### 7. Інтеграція з календарями

Система повинна забезпечувати експорт підтверджених зустрічей до Google Calendar та автоматичне їх додавання після підтвердження участі.

### **1.3.2 Нефункціональні вимоги**

Були визначені нефункціональні вимоги, які регламентують якісні характеристики роботи система: продуктивність, надійність, безпеку, масштабованість та зручність використання. Нефункціональні вимоги наведено у табл. 2.10.

Таблиця 2.10

## Нефункціональні вимоги

| Категорія              | Вимога                                                                                             |
|------------------------|----------------------------------------------------------------------------------------------------|
| Продуктивність         | Підтримка 1000+ одночасних користувачів; час відгуку інтерфейсу < 2 с; завантаження сторінок < 3 с |
| Надійність             | Щоденне автоматичне резервне копіювання бази даних                                                 |
| Безпека                | Протокол HTTPS; шифрування паролів; захист від CSRF та XSS атак                                    |
| Масштабованість        | Підтримка зростання до 10 000+ користувачів                                                        |
| Зручність використання | Мінімальна кількість кроків для виконання основних задач;                                          |
| Адаптивність           | Responsive-дизайн для смартфонів, планшетів і десктопів                                            |
| Сумісність             | Коректна робота у браузерях Chrome, Firefox, Brave, Edge                                           |

**Висновок до розділу 1**

У першому розділі проведено системний аналіз предметної області веб-застосунку для планування та керування груповими зустрічами. Розглянуто чотири існуючі аналоги - Calendly, Doodle, Meetup і Clockwise - та встановлено, що жодне з них не поєднує в єдиній платформі інструменти управління спільнотами з механізмом автоматизованого узгодження часу зустрічей. Для

моделювання предметної області побудовано діаграму прецедентів, що відображає функціональні можливості акторів системи, та діаграму активності, що ілюструє життєвий цикл події. Сформульовано функціональні та нефункціональні вимоги до розроблюваного застосунку, що стали основою для проектування інформаційної бази та програмного забезпечення в наступних розділах.

## 2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОГО ЗАБЕЗПЕЧЕННЯ

Розділ присвячено проектуванню та реалізації інформаційної бази веб-застосунку для планування та керування груповими зустрічами. У розділі визначено логічну структуру даних предметної області, обґрунтовано вибір системи управління базою даних та описано фізичну реалізацію інформаційної бази засобами обраного інструментарію.

### 2.1 Логічна модель даних у вигляді ER-діаграми

Для розробки структури зберігання даних системи було побудовано ER-модель, яка відображає основні об'єкти предметної області та зв'язки між ними. Графічне представлення моделі наведено в Додатку Ж.

#### 2.1.1 Сутності ER-модель:

USER (Користувач) - центральна сутність системи, що представляє зареєстрованого учасника платформи. Зберігає ідентифікаційні та аутентифікаційні дані: ім'я користувача (name), електронну адресу (email), ідентифікатор облікового запису Google для OAuth-авторизації (google\_id), хеш пароля (password), мітку часу підтвердження пошти (email\_verified\_at), токен сесії (remember\_token) та секрет двофакторної автентифікації (two\_factor\_secret). Користувач може виступати як звичайним учасником, так і організатором груп - роль визначається через зв'язок із сутністю ROLE.

GROUP (Група) - сутність, що представляє спільноту користувачів. Містить атрибути: унікальний ідентифікатор (group\_id), ознаку приватності (private), короткий опис (short\_description) та назву (title). Група належить одному власнику (USER) та може мати довільну кількість учасників.

Приватність групи визначає необхідність подачі заявки на вступ.

EVENT (Подія) - сутність, що описує захід у межах групи. Атрибути: унікальний ідентифікатор (`event_id`), короткий опис (`short_description`), тривалість у хвилинах (`duration_minutes`), максимальна кількість учасників (`max_participance`), ознака скасування (`canceled`), вартість участі (`price`), тип заходу (`type`) та місце проведення або посилання для онлайн-формату (`location_or_link`). Подія обов'язково прив'язана до конкретної групи.

TAG (Тег) - сутність для категоризації груп та формування персоналізованих рекомендацій. Містить назву тегу (`tag`) та його категорію (`tag_category`). Використовується у двох зв'язках: теги групи (для класифікації спільнот) та теги уподобань користувача (для підбору релевантного контенту).

ROLE (Роль) та PERMISSION (Дозвіл) - сутності, що реалізують рольову модель доступу. Роль об'єднує набір дозволів і призначається користувачеві. Це дозволяє розмежовувати права доступу між організаторами та звичайними учасниками.

POLL (Голосування) - сутність для узгодження часу проведення події. Атрибути: унікальний ідентифікатор (`poll_id`), дедлайн голосування (`vote_deadline`), ознака завершення (`finished`), опція відсутності за замовчуванням (`default_absence_option`) та ознака анонімності (`anonymous`). Кожне голосування прив'язане до конкретної події.

POLL\_OPTION (Варіант голосування) - сутність, що представляє один із запропонованих часових варіантів у голосуванні. Містить зовнішній ключ голосування (`poll_id`) та пропонуванний час (`proposed_time`)

Зв'язки між сутностями

Між виділеними сутностями встановлено такі зв'язки:

- USER → GROUP: один користувач може бути власником багатьох груп (один-до-багатьох); користувач може бути учасником багатьох груп і водночас група може мати багатьох учасників (багато-до-багатьох).
- USER → EVENT: користувач може брати участь у багатьох подіях, подія може мати багатьох учасників (багато-до-багатьох).

- USER → TAG: зберігаються теги уподобань користувача для персоналізації (багато-до-багатьох).
- USER → ROLE → PERMISSION: користувачу призначаються ролі, кожна роль об'єднує набір дозволів (багато-до-багатьох на обох рівнях).
- GROUP → TAG: група позначається тематичними тегами (багато-до-багатьох).
- GROUP → EVENT: одна група може проводити багато подій (один-до-багатьох).
- EVENT → POLL → POLL\_OPTION: до події прив'язується голосування, яке складається з варіантів часу (один-до-багатьох на кожному рівні).
- USER → POLL\_OPTION: користувач голосує за варіанти (багато-до-багатьох).

Забезпечено виконання першої нормальної форми шляхом приведення всіх атрибутів до атомарного вигляду та визначення первинного ключа для кожної таблиці. Оптимізація до рівня другої нормальної форми дозволила усунути часткові залежності, гарантуючи, що кожен неключовий атрибут має функціональну залежність саме від первинного ключа. Фінальний перехід до третьої нормальної форми полягав у повному відключенні транзитивних залежностей. Це було реалізовано через винесення специфічних даних у самостійні структури: зокрема, права доступу більше не зберігаються безпосередньо у профілі користувача, а виділені в окремі сутності ROLE та PERMISSION. Аналогічний підхід застосовано і для тегів уподобань користувачів чи груп, які тепер організовані за допомогою зв'язкових таблиць замість їхнього прямого дублювання в основних об'єктах системи.

Розроблена модель даних відповідає вимогам третьої нормальної форми (3НФ), що забезпечує високу якість структури бази даних. Завдяки дотриманню принципів нормалізації вдалося мінімізувати дублювання інформації, спростити механізми підтримки цілісності даних та зменшити ризики появи логічних помилок у процесі експлуатації системи.

## **2.2 Вибір системи управління базою даних**

Вибір системи управління базою даних (СУБД) є одним із ключових архітектурних рішень при розробці веб-застосунку, оскільки безпосередньо впливає на продуктивність, масштабованість та надійність системи. При виборі СУБД для розроблюваного застосунку розглядалися три поширені рішення: PostgreSQL, MySQL та MongoDB.

### **2.2.1 Порівняльний аналіз СУБД**

MySQL - реляційна СУБД з відкритим вихідним кодом, широко застосовується у веб-розробці. Забезпечує високу швидкість читання даних та просту початкову конфігурацію. Проте має обмежену підтримку складних типів даних, а також поступається PostgreSQL у реалізації стандарту SQL та підтримці складних транзакцій.

MongoDB - документно-орієнтована NoSQL СУБД, що зберігає дані у форматі, подібному до JSON. Добре підходить для неструктурованих або слабо структурованих даних та забезпечує горизонтальне масштабування. Однак для даної предметної області, де між сутностями існують чіткі реляційні зв'язки (користувачі, групи, події, голосування), використання документно-орієнтованої моделі ускладнює підтримку цілісності даних та виконання складних запитів із об'єднанням таблиць.

PostgreSQL - об'єктно-реляційна СУБД з відкритим вихідним кодом, що відповідає стандарту SQL і забезпечує розширену підтримку типів даних, транзакцій та складних запитів. Підтримує зовнішні ключі, каскадні операції, повнотекстовий пошук, а також власні типи даних, такі як boolean, timestamp та text, які активно використовуються у розробленій логічній моделі.

### **2.2.2 Обґрунтування вибору PostgreSQL**

За результатами порівняльного аналізу для реалізації інформаційної бази

застосунку обрано PostgreSQL.

Відповідність реляційній моделі даних. Розроблена логічна модель містить численні зв'язки між сутностями (один-до-багатьох, багато-до-багатьох), що потребує підтримки зовнішніх ключів, каскадного видалення та складних JOIN-запитів. PostgreSQL забезпечує це в повному обсязі.

Розширена підтримка типів даних. PostgreSQL надає нативну підтримку типів даних, які відсутні або обмежено реалізовані в інших реляційних СУБД. Зокрема, у розробленій схемі використовується тип ENUM для поля type таблиці events, що дозволяє обмежити допустимі значення на рівні бази даних без додаткових перевірок у прикладному коді. Тип UUID застосовано як первинний ключ таблиці notifications - PostgreSQL підтримує його нативно та має вбудовані функції генерації (gen\_random\_uuid()), на відміну від MySQL, де UUID зберігається як рядок. Поле data таблиці notifications зберігає дані у форматі JSON: PostgreSQL підтримує тип JSONB із можливістю індексування та виконання запитів безпосередньо до вкладених JSON-полів, що є суттєвою перевагою перед збереженням JSON як звичайного тексту.

Відкрита ліцензія. PostgreSQL [9] розповсюджується під ліцензією PostgreSQL License, що є дозвоільною ліцензією відкритого програмного забезпечення та не накладає обмежень на використання у комерційних проектах.

### 2.2.3 Метод взаємодії з базою даних - Eloquent ORM

Для взаємодії застосунку з базою даних використовується Eloquent ORM [3] - вбудований об'єктно-реляційна модель фреймворку Laravel, де кожна таблиця бази даних відповідає окремому класу-моделі, а рядок таблиці - екземпляру цього класу.

Основні переваги використання Eloquent ORM у поєднанні з PostgreSQL:

Абстракція від SQL. Розробник описує запити до бази даних засобами PHP, що прискорює розробку та знижує ймовірність помилок у SQL-коді. Наприклад, отримання всіх активних подій групи виражається як `$group->events()->where('canceled', false)->get()` замість написання SQL-запиту

вручну.

Декларативний опис зв'язків. Eloquent підтримує всі типи реляційних зв'язків, реалізованих у логічній моделі: `hasMany`, `belongsTo`, `belongsToMany`. Зв'язки багато-до-багатьох (наприклад, учасники групи, теги уподобань, голоси за варіанти) реалізуються через проміжні таблиці з мінімальним обсягом коду.

Міграції. Laravel надає механізм міграцій, що дозволяє визначати та змінювати структуру бази даних у вигляді PHP-коду, зберігати її у системі контролю версій і відтворювати на будь-якому середовищі. Це забезпечує узгодженість схеми між середовищами розробки, тестування та продакшену.

Повна сумісність з PostgreSQL. Laravel офіційно підтримує PostgreSQL як один із основних драйверів бази даних. Усі можливості Eloquent, включно з повнотекстовим пошуком, JSON-стовпцями та складними підзапитами, доступні при використанні PostgreSQL.

Таким чином, поєднання PostgreSQL як СУБД та Eloquent ORM модель взаємодії з нею в рамках фреймворку Laravel було обрано для реалізації інформаційної бази розроблюваного застосунку.

Код Eloquent-моделей застосунку наведено в Додатку А.

## 2.3 Розробка структури інформаційної бази

Структура бази даних визначається та версіонується через механізм міграцій фреймворку Laravel, що забезпечує відтворюваність схеми на будь-якому середовищі (розробка, тестування, продакшн) та дозволяє відстежувати зміни структури у системі контролю версій. Кожна міграція є окремим PHP-класом із методами `up()` (застосування змін) та `down()` (відкат змін). Код міграцій наведено в Додатку Б.

Загалом інформаційна база містить 19 таблиць, які поділяються на три групи: основні таблиці предметної області, зв'язкові (проміжні) таблиці та службові таблиці Laravel. Повну структуру бази даних представлено на Рис. 3.6.

### 2.3.1 Основні таблиці предметної області

Таблиця `users` зберігає дані зареєстрованих користувачів системи (Додаток К табл. К.1). Поле `google_id` забезпечує інтеграцію з OAuth-авторизацією через Google. Поля `two_factor_secret`, `two_factor_recovery_codes` та `two_factor_confirmed_at` додано окремою міграцією для підтримки двофакторної автентифікації.

Таблиця `tags` зберігає тематичні теги для категоризації груп та формування уподобань користувачів (Додаток К табл. К.2). Таблиця заповнюється при міграції через `GroupTagSeeder`, що забезпечує наявність базового набору тегів одразу після розгортання системи.

Таблиця `groups` зберігає дані тематичних спільнот користувачів (табл. 3.1).

Таблиця 3.1

Структура таблиці `groups`

| Стовпець                       | Тип даних | Обмеження                             | Опис                                |
|--------------------------------|-----------|---------------------------------------|-------------------------------------|
| <code>id</code>                | BIGINT    | PK                                    | Унікальний ідентифікатор            |
| <code>user_id</code>           | BIGINT    | FK → <code>users.id</code> , NOT NULL | Власник групи                       |
| <code>title</code>             | CHAR(255) | NOT NULL                              | Назва групи                         |
| <code>short_description</code> | TEXT      | NOT NULL                              | Опис групи                          |
| <code>private</code>           | BOOLEAN   | NOT NULL                              | Ознака приватності                  |
| <code>deleted_at</code>        | TIMESTAMP | NULLABLE                              | Час м'якого видалення (Soft Delete) |

Для таблиці реалізовано м'яке видалення (Soft Delete) через поле `deleted_at`:

видалені групи не видаляються фізично з бази даних, що дозволяє зберегти цілісність пов'язаних даних та за потреби відновити групу.

Таблиця events зберігає дані заходів, що проводяться в межах груп (Додаток К табл. К.3). Поле type реалізовано як перелічуваний тип (ENUM) з двома значеннями: online та in\_person. Залежно від значення цього поля, поле location\_or\_link містить або фізичну адресу проведення, або посилання на онлайн-зустріч.

Таблиці polls та poll\_options реалізують механізм голосування за зручний час проведення події (табл. 3.2).

Таблиця 3.2

## Структура таблиці polls

| Стовпець               | Тип даних | Обмеження                | Опис                                     |
|------------------------|-----------|--------------------------|------------------------------------------|
| id                     | BIGINT    | PK                       | Унікальний ідентифікатор                 |
| event_id               | BIGINT    | FK → events.id, NOT NULL | Подія, до якої належить голосування      |
| vote_deadline          | DATETIME  | NOT NULL                 | Дедлайн голосування                      |
| finished               | BOOLEAN   | NOT NULL, DEFAULT false  | Ознака завершення                        |
| default_absence_option | BOOLEAN   | NOT NULL, DEFAULT true   | Опція «не зможу прийти» за замовчуванням |

|          |         |                            |                                  |
|----------|---------|----------------------------|----------------------------------|
| anonymus | BOOLEAN | NOT NULL,<br>DEFAULT false | Ознака анонімного<br>голосування |
|----------|---------|----------------------------|----------------------------------|

Кожен варіант голосування зберігається в таблиці `poll_options` із зовнішнім ключем `poll_id` та полем `proposed_time` типу `DATETIME`, що містить пропонуванй час проведення події.

### 2.3.2 Зв'язкові (проміжні) таблиці

Для реалізації зв'язків типу багато-до-багатьох у базі даних створено окремі проміжні таблиці, перелік яких наведено у табл. 3.6.

Таблиця 3.6

Зв'язкові таблиці бази даних

| Таблиця                           | Зв'язок                           | Додаткові поля                             |
|-----------------------------------|-----------------------------------|--------------------------------------------|
| <code>user_preference_tags</code> | <code>users ↔ tags</code>         | -                                          |
| <code>group_members</code>        | <code>groups ↔ users</code>       | <code>status</code> , <code>role_id</code> |
| <code>group_join_requests</code>  | <code>groups ↔ users</code>       | <code>invited_by</code>                    |
| <code>group_to_tag</code>         | <code>groups ↔ tags</code>        | -                                          |
| <code>event_participants</code>   | <code>events ↔ users</code>       | <code>will_attend</code>                   |
| <code>poll_votes</code>           | <code>poll_options ↔ users</code> | -                                          |
| <code>role_has_permissions</code> | <code>roles ↔ permissions</code>  | -                                          |

Таблиця `group_members` містить додаткові поля: `status` (статус учасника у групі) та `role_id` (роль учасника, зовнішній ключ до таблиці `roles`), що реалізує рольову модель доступу на рівні групи. Таблиця `group_join_requests` зберігає заявки на вступ до приватних груп з можливістю відстеження, хто саме надіслав запрошення (`invited_by`). Таблиця `event_participants` містить поле `will_attend` типу

BOOLEAN, що фіксує підтвердження участі в події.

### 2.3.3 Службові таблиці Laravel

Окрім таблиць предметної області, у базі даних присутні службові таблиці фреймворку Laravel, які забезпечують роботу вбудованих підсистем:

- sessions - зберігання серверних сесій користувачів із фіксацією IP-адреси та User-Agent;
- password\_reset\_tokens - токени для скидання паролю з прив'язкою до email;
- cache та cache\_locks - кешування даних застосунку засобами Laravel Cache;
- jobs, job\_batches, failed\_jobs - черга фонових завдань (Laravel Queue) для асинхронної обробки операцій, зокрема надсилення email-сповіщень.

### 2.3.4 Індокси та цілісність даних

Для всіх зовнішніх ключів визначено поведінку при видаленні батьківського запису. Каскадне видалення (CASCADE) застосовано для зв'язку events → groups, що автоматично депрекує всі події при видаленні групи. Для зв'язкових таблиць user\_preference\_tags та group\_to\_tag також застосовано CASCADE, щоб автоматично прибирати теги при видаленні користувача або групи.

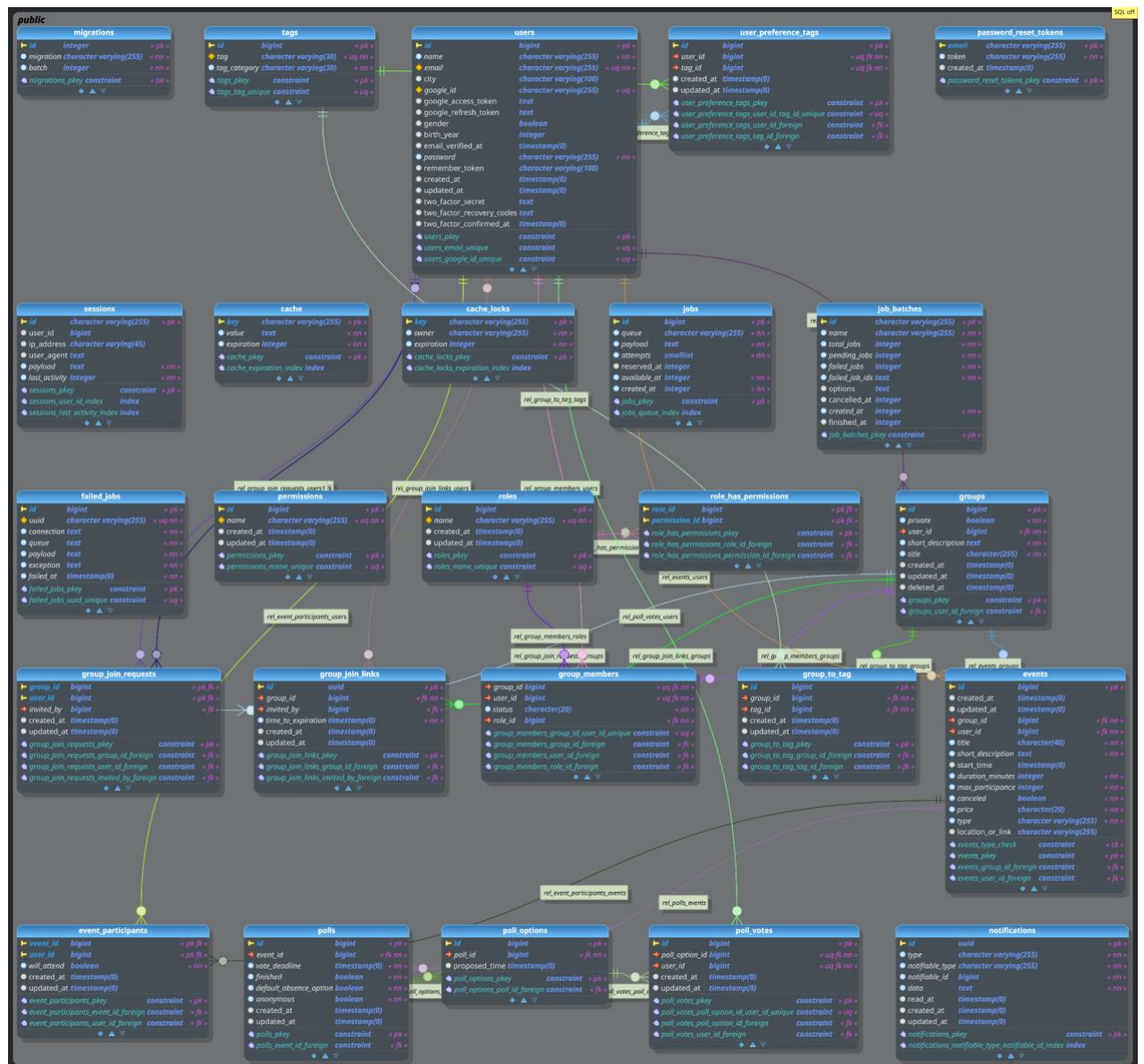


Рис. 3.6 Фізична модель бази даних

## Висновок до розділу 2

Спроектовано та реалізовано інформаційну базу застосунку. Побудовано ER-модель предметної області, що охоплює вісім основних сутностей із чітко визначеними зв'язками між ними. Модель даних приведено до третьої нормальної форми, що забезпечує мінімізацію надлишковості та цілісність даних. За результатами порівняльного аналізу MySQL, MongoDB та PostgreSQL обрано PostgreSQL як СУБД, що найповніше відповідає реляційній структурі даних предметної області та забезпечує нативну підтримку типів ENUM, UUID

та JSONB. Інформаційну базу реалізовано у вигляді 19 таблиць, розподілених на основні, зв'язкові та службові, з використанням механізму міграцій Laravel для версіонування схеми.

## **3 РОЗРОБКА ПРИКЛАДНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

### **3.1 Організаційна структура програмного забезпечення**

Застосунок побудовано на основі архітектурного патерну MVC (Model-View-Controller), який забезпечує чіткий поділ відповідальності між рівнями системи. Laravel реалізує цей патерн у поєднанні з додатковим сервісним шаром (Services), що виокремлює бізнес-логіку з контролерів та робить код більш підтримуваним.

#### **3.1.1 Організація пакетів**

Внутрішня структура серверної частини організована за принципом багаторівневої архітектури з чітким розмежуванням відповідальності між пакетами. Діаграму пакетів наведено в Додатку Е.

Presentation Layer - верхній рівень, що відповідає за отримання та первинну обробку HTTP-запитів. Містить компоненти Routes (визначення маршрутів та їх прив'язка до контролерів; код файлу маршрутизації наведено в Додатку В), Middleware (перевірка автентифікації, авторизації та валідація запитів до їх передачі контролеру) та React Components(компоненти інтерфейсу які бачить користувач).

Application Layer - рівень контролерів, координує обробку HTTP/HTTPS запитів.

Business Logic Layer - сервісний шар, що інкапсулює бізнес-логіку та ізолює її від контролерів. Кожному контролеру відповідає окремий сервіс: AuthService, GroupService, PollService, MeetingService. Окремо виділено NotificationService, який відповідає за формування та надсилання сповіщень користувачам незалежно від контролерів.

Data Access Layer - рівень доступу до даних, реалізований засобами Eloquent ORM. Містить моделі (Models), що відображають таблиці бази даних на PHP-класи (код моделей наведено в Додатку А), та міграції (Migrations), що описують схему бази даних у вигляді версіонованого коду (код міграції наведено в Додатку Б).

Infrastructure - рівень інфраструктурних залежностей: PostgreSQL як основне сховище даних, Redis для кешування часто запитуваних даних та черги завдань, а також зовнішні сервіси - поштовий сервер і Google Calendar.

Cross-Cutting Concerns - наскрізний компонент Logger, що використовується всіма рівнями системи для фіксації подій, помилок та діагностичної інформації незалежно від поточного рівня виконання.

Потік даних між рівнями є строго однонаправленим згори донизу: запит проходить від Presentation Layer через Application Layer до Business Logic Layer, далі до Data Access Layer і врешті зберігається в Infrastructure. Такий підхід унеможливорює пряму залежність нижніх рівнів від верхніх та спрощує тестування кожного рівня незалежно.

## **3.2 Моделювання структури та взаємодії об'єктів**

### **3.2.1 Діаграма послідовності**

Діаграма послідовності відображає взаємодію між об'єктами системи з плином часу та показує порядок повідомлень, що відправляються між ними. Для даної предметної області виділено два ключових сценарії.

На Рис. 4.2 показана діаграма послідовності сценарію «Створення та управління подією». Процес починається з того, що Організатор перевіряє наявність відповідної групи. Якщо група відсутня, він ініціює її створення та отримує підтвердження від системи. Після цього Організатор створює безпосередньо подію. В процесі управління (до початку заходу) він має можливість редагувати або скасовувати її, отримуючи відповідні оновлення

статусу від об'єкта Події. Завершальним етапом є проведення заходу та закриття події в системі.

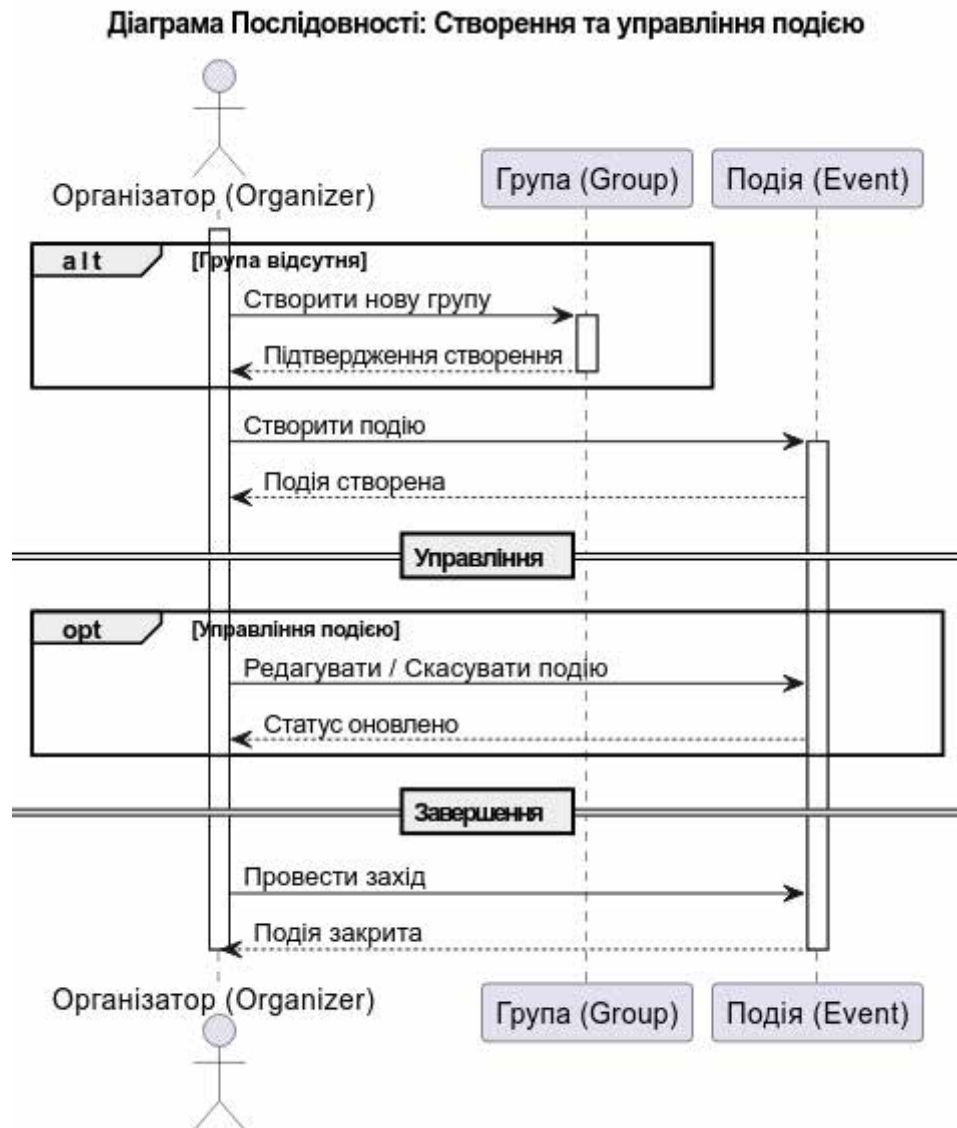


Рис. 4.2 Діаграма послідовності «Створення та управління подією»

На Рис. 4.3 зображено сценарій «Записатися на подію», який деталізує взаємодію між Користувачем, Подією, Групою та Організатором (Адміністратором). Користувач надсилає системі запит на участь у події. Якщо він ще не є членом відповідної групи, система автоматично перенаправляє запит на вступ до об'єкта Групи, який передає його Організатору для опрацювання. У разі відхилення заявки процес припиняється, і користувач отримує відмову. Якщо заявку схвалено (або користувач вже перебуває у групі), система ініціює процес бронювання: перевіряє наявність вільних місць і, залежно від результату,

або закріплює місце за користувачем з підтвердженням запису, або видає повідомлення про брак вільних місць.

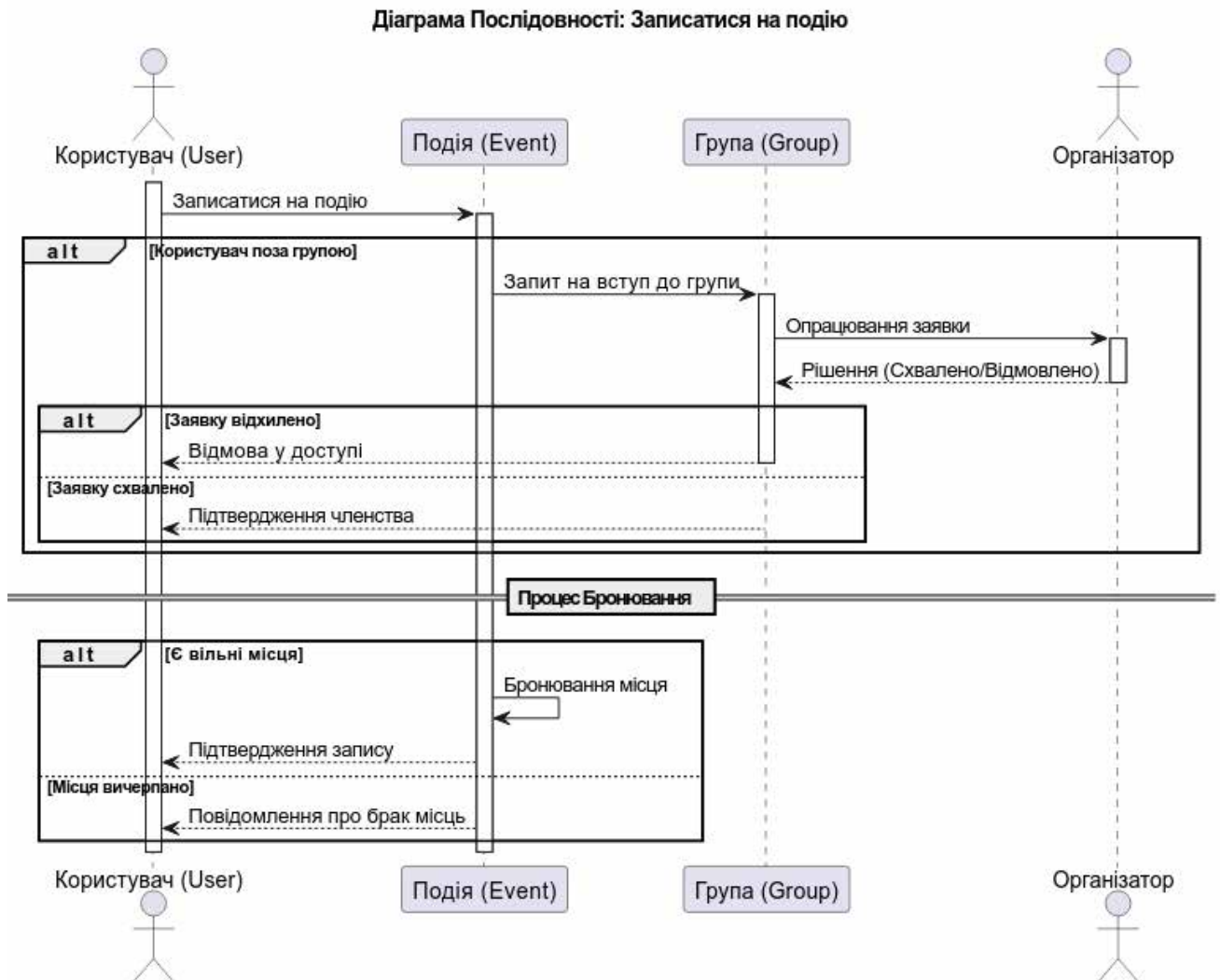


Рис. 4.3 Діаграма послідовності «Записатися на подію»

### 3.2.2 Діаграма класів

На Рис. 4.4 виділено чотири основні класи: «Користувач», «Організатор», «Група» та «Подія». Клас «Користувач» інкапсулює базові ідентифікаційні атрибути (ім'я, контактна інформація) та методи взаємодії з системою - пошук і перегляд подій, реєстрацію на захід та перевірку наявності вільних місць. Клас «Організатор» розширює можливості користувача та зосереджує в собі операції керування: створення та редагування груп і подій, управління учасниками, прийняття заявок на вступ. Клас «Група» агрегує інформацію про спільноту -

назву, опис, тематичні теги, власника, список учасників та пов'язані події. Клас «Подія» містить атрибути конкретного заходу: назву, опис, дату і час проведення, місце, максимальну кількість місць та перелік зареєстрованих учасників.

Між класами визначено такі зв'язки: один «Організатор» може володіти однією або кількома «Групами»; одна «Група» може містити довільну кількість «Подій»; «Організатор» керує подіями безпосередньо; «Користувач» може бути учасником багатьох груп і зареєстрованим на багато подій, що реалізує зв'язки типу багато-до-багатьох на обох рівнях.

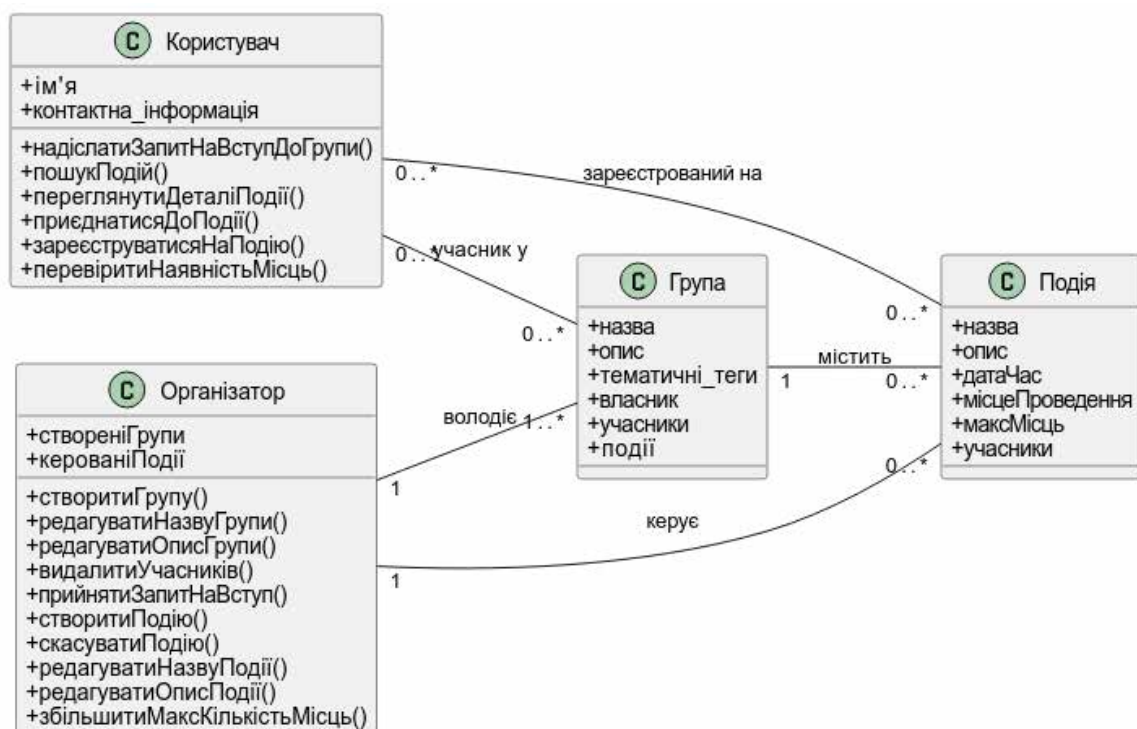


Рис. 4.4 Діаграма класів, із предметної області

### 3.2.3 Діаграми кооперації

Діаграми кооперації є структурними моделями, що деталізують взаємодію між класами в межах конкретного функціонального сценарію. На відміну від загальної діаграми класів, кожна кооперація виокремлює лише ті класи та зв'язки, які безпосередньо задіяні у відповідній функції.

«Управління групами» (Рис. 4.5) описує механізм створення спільнот та

контролю членства. Центральним елементом виступає «Організатор», який через агрегацію володіє однією або кількома групами та має методи для їх адміністрування. «Користувач» взаємодіє з «Групою» через залежність - він може подати заявку на вступ, яку «Організатор» приймає або відхиляє.

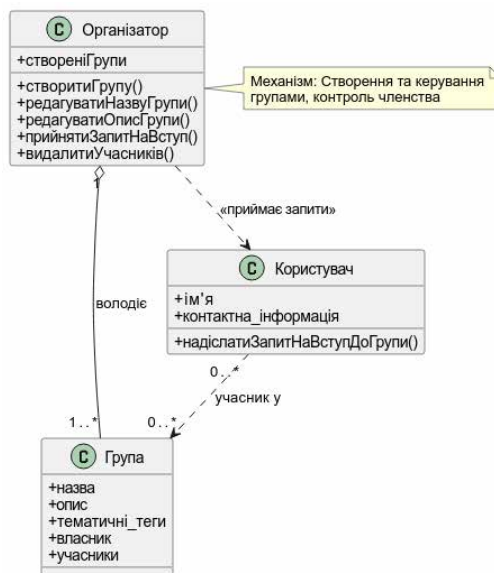


Рис. 4.5 Прості кооперації: Управління групами

«Організація подій» (Рис. 4.6) зосереджується на процесі створення та управління заходами. «Організатор» через агрегацію керує подіями, а «Група» через композицію містить їх у собі - це підкреслює, що подія не може існувати поза групою і знищується разом із нею. Кооперація охоплює повний цикл роботи з подією: від створення та редагування назви й опису до збільшення максимальної кількості місць і скасування заходу.

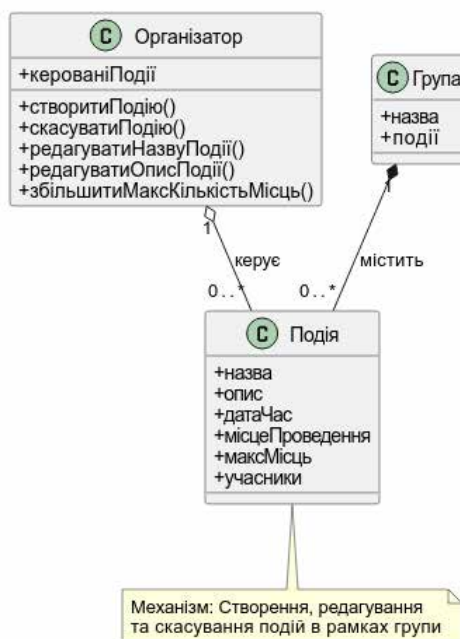


Рис. 4.6 Прості кооперації: Організація події

«Участь користувачів у подіях» (Рис. 4.7) описує сценарій взаємодії «Користувача» із системою при пошуку та реєстрації на захід. Користувач шукає події через групи, після чого може переглянути деталі, перевірити наявність вільних місць та зареєструватися. «Група» через композицію містить події, що забезпечує контекст для пошуку.

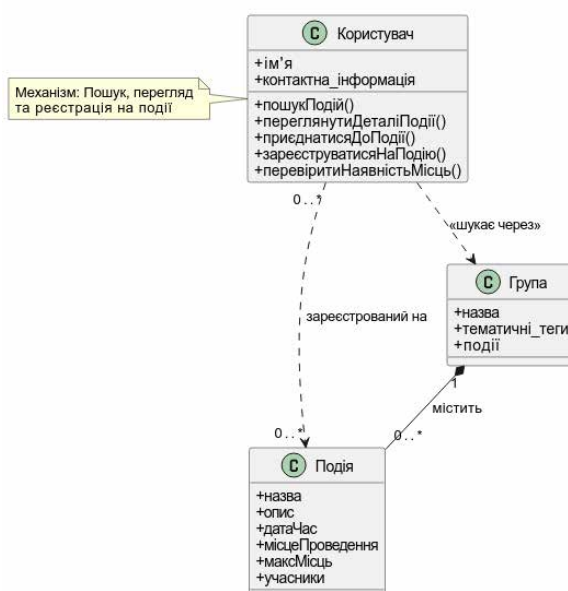


Рис. 4.7 Прості кооперації: Участь користувача у події

«Членство в групах» (Рис. 4.8) моделює повний цикл управління

членством із урахуванням ієрархії ролей. Принципова відмінність цієї кооперації від першої полягає у явному відображенні відношення успадкування: «Організатор» є спеціалізацією «Користувача», що відображає наявність усіх базових можливостей користувача з розширеними адміністративними правами. «Група» є центральним об'єктом взаємодії: «Користувач» подає заявку на вступ, «Організатор» приймає або відхиляє її та за потреби видаляє учасників.

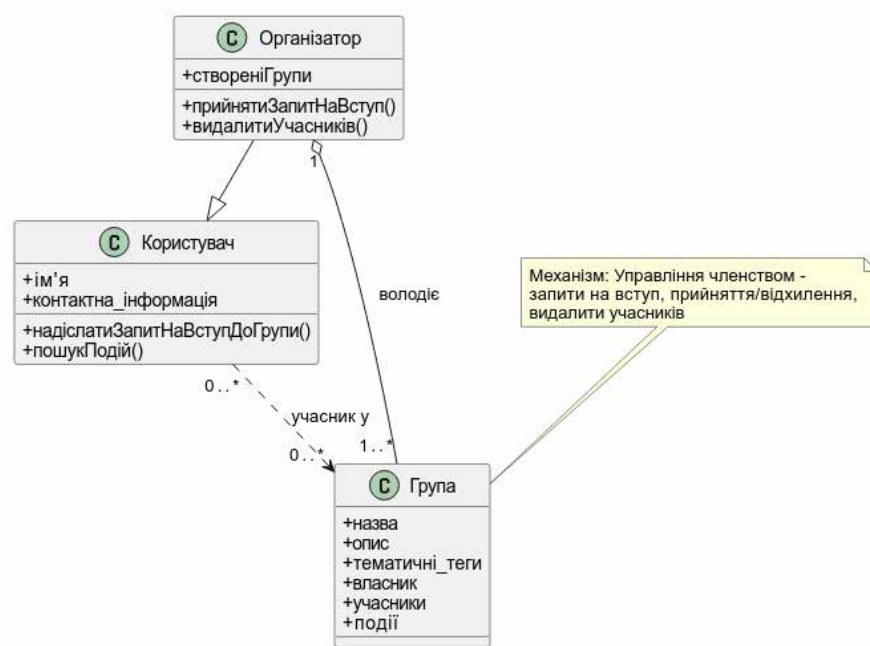


Рис. 4.8 Прості кооперації: Членство в групах

### 3.3 Компонентна структура архітектури системи

На Рис. 4.9 зображено три основні рівні системи: Frontend, Backend та Infrastructure.

Рівень Frontend реалізовано засобами React і Inertia.js. React-сторінки отримують запити через HTTP/HTTPS та взаємодіють із серверною частиною через Inertia Protocol - спеціалізований протокол, що замінює традиційні API-запити та передає дані безпосередньо з контролерів у компоненти як props. Для відображення географічних даних Frontend звертається до Google Maps API.

Рівень Backend побудовано на Laravel і складається з чотирьох внутрішніх компонентів: маршрутизатора з проміжним програмним забезпеченням (Routes + Middleware), контролерів (Controllers), сервісного шару (Services) та Eloquent-моделей (Models). Вхідний запит проходить через маршрутизатор, потрапляє до відповідного контролера, який делегує виконання бізнес-логіки сервісу. Сервіси взаємодіють із базою даних, кешем та поштовим сервером. Моделі Eloquent забезпечують об'єктно-реляційне відображення даних із PostgreSQL.

Рівень Infrastructure об'єднує зовнішні служби: базу даних PostgreSQL, Redis для кешування, поштовий сервер для надсилання сповіщень, а також Google Calendar та Google Maps як зовнішні інтеграції.

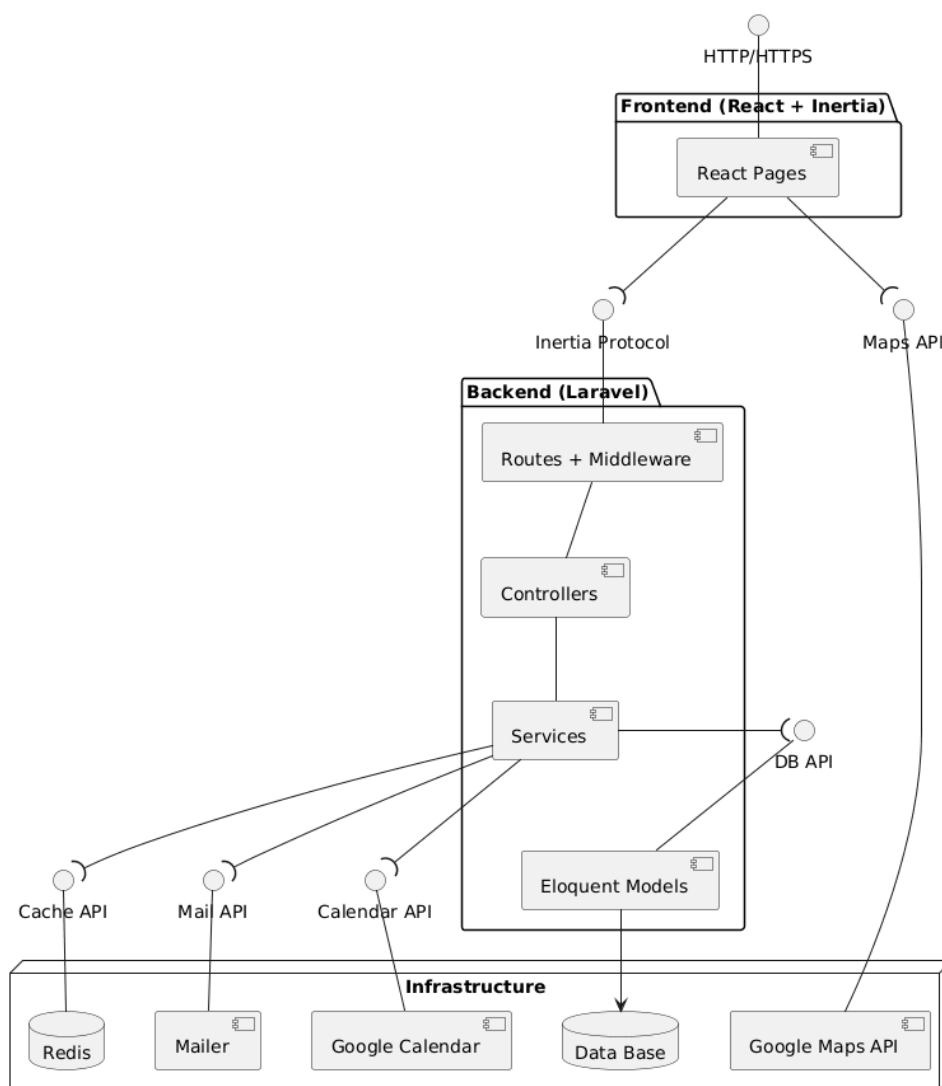


Рис. 4.9 Діаграма компонентів: архітектура системи

### 3.4 Вибір інструментарію для створення прикладного програмного забезпечення

Вибір технологічного стеку для розробки веб-застосунку було визначено з урахуванням вимог до продуктивності, зручності розробки та довгострокової підтримки інструментарію. Застосунок побудовано на основі архітектури, де серверна частина реалізована за допомогою фреймворку Laravel, а клієнтська - бібліотекою React із використанням Inertia.js як міст між ними.

PHP [1] у поєднанні з фреймворком Laravel [2] обрано як основу серверної частини застосунку. Серед альтернатив розглядались Python (Django/FastAPI) та Node.js (Express/NextJS), однак PHP має значні переваги у для даного застосунку. PHP має найбільшу кількість готових рішень, бібліотек та хостинг-провайдерів із вбудованою підтримкою мови. Laravel, побудований на PHP, надає вбудовану підтримку черг, сповіщень, OAuth-авторизації, міграцій бази даних та планувальника завдань - функціональність, яка в Node.js-екосистемі потребує підбору та інтеграції окремих сторонніх бібліотек. Завдяки цьому Laravel значно скорочує час розробки типових веб-функцій: замість написання інфраструктурного коду розробник може зосереджуватися на бізнес-логіці. PHP 8.4 офіційно підтримується до кінця 2028 року, що гарантує довгострокову безпеку та актуальність обраного стеку. Детальніше переваги Eloquent ORM розглянуто у підрозділі 2.2.

Inertia.js [4] виконує роль архітектурного містка між Laravel і React, це дозволяє будувати односторінковий застосунок без необхідності розробки окремого REST API або GraphQL API.

TypeScript [5] використовується як основна мова програмування клієнтської частини замість JavaScript. Статична типізація дозволяє виявляти помилки на етапі компіляції, а не під час виконання, що є особливо важливим при роботі з даними, що передаються з серверу через Inertia.js - типи props компонентів явно описуються інтерфейсами, що унеможливорює звернення до неіснуючих полів. Крім того, TypeScript суттєво покращує автодоповнення та

навігацію по коду у середовищі розробки.

React [6] обрано як бібліотеку для побудови інтерфейсу завдяки компонентному підходу та механізму Virtual DOM. Компоненти реалізовано у форматі TSX - розширення TypeScript, що підтримує JSX-синтаксис для опису розмітки безпосередньо у коді компонента. Це дозволяє поєднувати переваги статичної типізації TypeScript із декларативним описом інтерфейсу, характерним для React. Компонентна архітектура дозволяє розбивати інтерфейс на незалежні частини, що повторно використовуються у різних частинах застосунку - картки подій, форми, елементи навігації. Virtual DOM забезпечує реакцію інтерфейсу на зміни стану без зайвих операцій з реальним DOM-деревом.

Tailwind CSS [7] використовується для стилізації інтерфейсу як utility-first фреймворк. На відміну від компонентних CSS-фреймворків на зразок Bootstrap, Tailwind не нав'язує готовий візуальний стиль, а надає набір атомарних службових класів, які комбінуються безпосередньо у TSX-розмітці. Це дозволяє створювати унікальний інтерфейс без написання окремих CSS-файлів і без ризику конфліктів між стилями.

Bun [8] обрано як середовище виконання JavaScript, пакетний менеджер та інструмент збірки замість традиційної комбінації Node.js та npm. Bun нативно підтримує TypeScript та TSX без додаткового налаштування - файли .ts та .tsx транспілюються вбудованим компілятором, що виключає необхідність окремого налаштування Babel або tsc у процесі збірки. Ключовою перевагою є значно вища швидкість встановлення залежностей та виконання скриптів порівняно з Node.js, що скорочує час збірки як у процесі розробки. Також Bun забезпечує повну сумісність із пакетами екосистеми npm.

Зведену характеристику обраного технологічного стеку наведено у табл. 5.1.

Таблиця 5.1

## Технологічний стек застосунку

| Технологія   | Роль у застосунку                           | Версія |
|--------------|---------------------------------------------|--------|
| PHP          | Мова програмування серверної частини        | 8.4.1  |
| Laravel      | Серверний фреймворк, роутинг, бізнес-логіка | 12.0   |
| Inertia.js   | Міст між Laravel та React                   | 2.3.7  |
| TypeScript   | Мова програмування клієнтської частини      | 5.7.2  |
| React + TSX  | Побудова клієнтського інтерфейсу            | 19.2   |
| Tailwind CSS | Стилізація інтерфейсу                       | 4.0.0  |
| Bun          | Runtime, пакетний менеджер, збірка          | 1.3.11 |
| PostgreSQL   | Система управління базою даних              | 16.13  |

Обраний стек забезпечує продуктивну модель розробки: серверна логіка, маршрутизація, робота з базою даних та інтерактивність інтерфейсу. Inertia.js усуває необхідність дублювання логіки між сервером і клієнтом, що спрощує підтримку застосунку та зменшує кількість точок можливих помилок.

Обраний стек забезпечує продуктивну та строгу типізацію під час розробки: серверна логіка, маршрутизація та робота з базою даних зосереджені у Laravel, тоді як TypeScript і React відповідають за інтерактивність клієнтської частини. Inertia.js усуває необхідність дублювання логіки між сервером і клієнтом, що спрощує підтримку застосунку та зменшує кількість можливих помилок.

### **3.3 Алгоритмізація та програмування програмних модулів**

Розроблений застосунок складається із семи функціональних модулів, кожен з яких реалізує окрему групу вимог до системи. У цьому підрозділі описано алгоритм основного процесу системи та реалізовані модулі.

#### **3.3.1 Алгоритм життєвого циклу події**

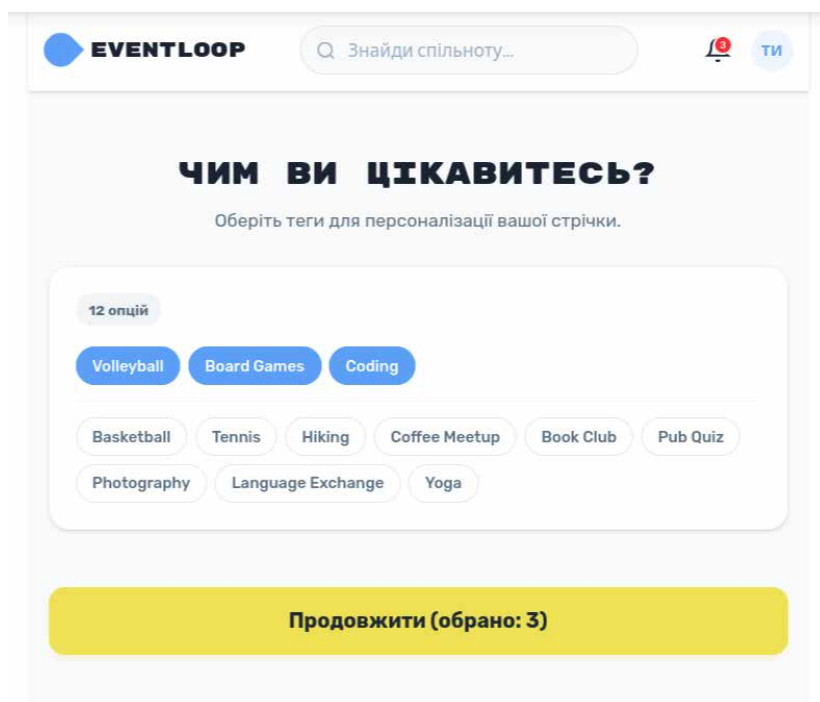
Центральним алгоритмом системи є життєвий цикл події, блок-схему якого наведено в Додатку II. Алгоритм охоплює повний процес - від створення події організатором до її завершення або скасування.

Після створення події система розгалужується залежно від обраного режиму визначення часу. У разі вибору фіксованого часу організатор одразу встановлює дату проведення, і процес переходить до етапу підтвердження участі. Якщо обрано режим голосування, організатор додає кілька варіантів часу та встановлює дедлайн голосування. Учасники голосують за зручні варіанти, і система повторює цикл збору голосів до настання дедлайну або до моменту, коли організатор вручну фіналізує результат. Організатор може завершити голосування будь-коли; інакше після закінчення дедлайну система автоматично надсилає нагадування. Після визначення часу учасники підтверджують або відхиляють участь - при підтвердженні подія автоматично додається до Google Calendar учасника. Якщо кількість підтверджених учасників виявляється недостатньою, організатор скасовує подію. В іншому випадку захід проводиться.

#### **3.3.2 Модуль ідентифікації та управління профілем**

Модуль забезпечує реєстрацію, авторизацію та керування особистим профілем користувача. Підтримується два способи входу: через email з паролем та через OAuth-авторизацію Google [10]. Після першої реєстрації система пропонує користувачеві пройти опитування інтересів для персоналізованого підбору публічних груп і подій - цей крок реалізовано як окремий екран із вибором тегів уподобань (рис. 6.1). У профілі користувач може редагувати

ім'я, вік, місто проживання (рис. 6.2). Форми реєстрації та авторизації наведено Додаток Л рис. Д.1 та рис. Д.2.



The screenshot shows the 'EVENTLOOP' app interface. At the top, there is a navigation bar with the app logo, a search bar containing the text 'Знайди спільноту...', and a user profile icon labeled 'ТИ'. Below the navigation bar, the main heading reads 'ЧИМ ВИ ЦІКАВИТЕСЬ?' (What are you interested in?). Underneath this heading is a subtext: 'Оберіть теги для персоналізації вашої стрічки.' (Select tags for personalizing your feed.). A box labeled '12 опцій' (12 options) contains twelve tags: 'Volleyball', 'Board Games', 'Coding', 'Basketball', 'Tennis', 'Hiking', 'Coffee Meetup', 'Book Club', 'Pub Quiz', 'Photography', 'Language Exchange', and 'Yoga'. The first three tags are highlighted with blue borders, indicating they have been selected. At the bottom of the screen, there is a large yellow button with the text 'Продовжити (обрано: 3)' (Continue (selected: 3)).

Рис. 6.1 Опитування інтересів нового користувача


**EVENTLOOP**




## НАЛАШТУВАННЯ

Керуйте налаштуваннями вашого профілю та акаунту

### Профіль

#### МОЇ ІНТЕРЕСИ

Тут ви можете змінити теги своїх інтересів

**Редагувати інтереси**

#### ІНФОРМАЦІЯ ПРОФІЛЮ

Тут ви можете оновити своє ім'я, електронну адресу та інші дані.

Ім'я

Електронна адреса

Місто

Стать

Рік народження

**Зберегти**

**Видалити акаунт**

Видалення вашого акаунту та всіх пов'язаних з ним даних

Рис. 6.2 Редагування профілю

### 3.3.3 Модуль управління групами

Модуль реалізує цикл роботи зі спільнотами. Організатор може створити публічну або приватну групу (Додаток Л рис. Д.3) із зазначенням назви, опису та

тематичних тегів. Для приватних груп реалізовано механізм заявок на вступрис. (рис. 6.3, рис. 6.4): користувач надсилає запит, який організатор приймає або відхиляє. Окрім цього, організатор може згенерувати унікальне посилання для автоматичного приєднання до групи без подачі заявки, що зручно для запрошення учасників напряду. Рольова модель розмежовує права між трьома ролями: Учасник, Адміністратор та Організатор - роль кожного члена групи можна змінити безпосередньо зі сторінки управління учасниками. Також організатор може додати будь-якого зареєстрованого користувача системи до групи вручну через пошук зі списку або надіславши посилання на вступ.

Рольову модель та управління учасниками зображено на рис. 6.5, Додаток Л рис. Д.4. Редагування групи наведено Додаток Л на рис. Д.5, генерацію посилання - у Додатку Л рис. Д.6. Сторінку групи з точки зору учасника та організатора наведено Додаток Л рис. Д.7 та рис. 6.6.

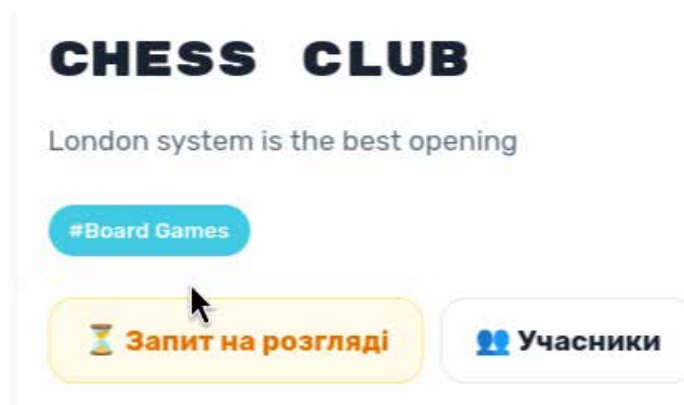


Рис. 6.3 Запит на вступ до групи

# CHESSE CLUB

London system is the best opening

<

ЗАПИТИ НА ВСТУП

Chess club • 1 запит

ТА

Тимофій Андрійович Свириденко

ipz23-t.svyrydenko@nubip.edu.ua

Роль після прийняття

Member

Статус

New member

Прийняти

Відхилити

Рис. 6.4 Організатор прийняв або відхилив запит на вступ

<

2 учасників

ВЛАСНИК

ТС

Тимофей Свириденко

sviridenkotimofei@gmail.com

Point master

УЧАСНИКИ

ТА

Тимофій Андрійович Свириденко

ipz23-t.svyrydenko@nubip.edu.ua

Member

Рис. 6.5 Організатор має можливість керувати членами групи, змінюючи їх роль



Рис. 6.6 Сторінка групи очима організатора

### 3.3.4 Модуль управління подіями

Модуль дозволяє організатору створювати події в межах групи із зазначенням назви, опису, формату проведення (офлайн або онлайн), максимальної кількості місць та часу початку. При створенні події доступна опція голосування - у такому випадку поле часу початку залишається незаповненим до завершення опитування, а процес визначення часу відбувається

за алгоритмом, описаним у блок-схемі Додатку И. Організатор може редагувати назву, опис та збільшувати максимальну кількість учасників вже після публікації події. Учасник групи може зареєструватися на подію, скасувати реєстрацію та переглядати учасників які підтвердили участь; система автоматично перевіряє наявність вільних місць перед підтвердженням запису та додає подію до календаря учасника.

Інтерфейс створення події наведено у Додатку Л рис. Д.9, редагування - на рис. 6.7. Підтвердження участі учасником зображено на рис. 6.8, відображення скасованої події - Додатку Л рис. Д.8.

**РЕДАГУВАТИ ПОДІЮ**

**НЕЗМІННІ ДАНІ**

⌚ 120 хв 🇺🇦 free

**Назва**

**Тренування еншпілів**

**Опис події**

**B** / **S** | **H1** **H2** **H3** | **≡** **1≡** | **"** **</>** | **↶** **↷**

Пройдемося по типовим помилкам гравців в еншпілі

**Адреса проведення**

м. Київ, НУБіП, 15 корпус

**Максимум учасників**

10

Поточне значення: 10. Можна лише збільшити.

**Зберегти зміни**

Рис. 6.7 Організатор може редагує назву, опис та максимальну кількість

учасників події

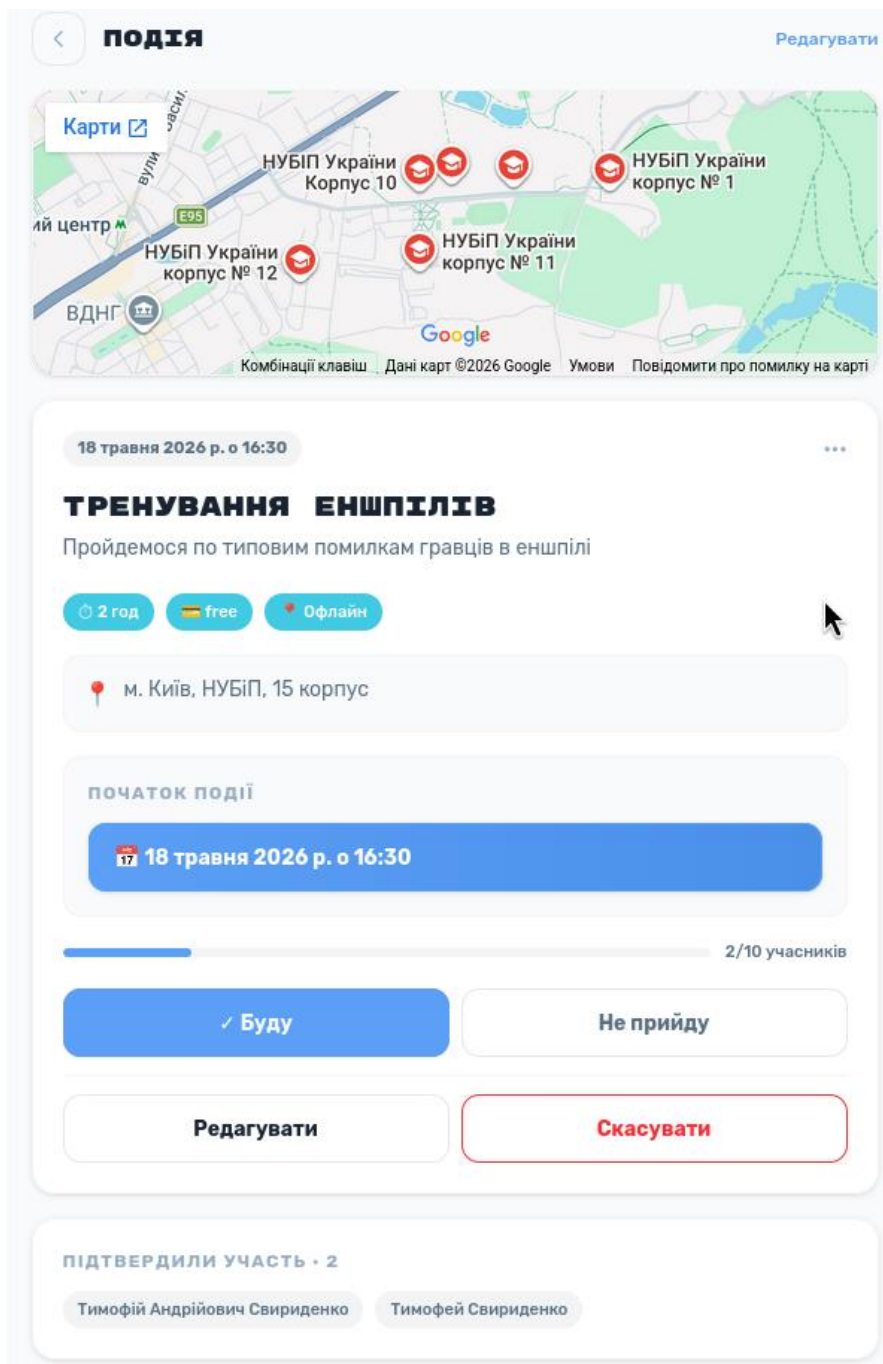


Рис. 6.8 Учасник групи приймає або відхиляє участь у події

### 3.3.5 Модуль опитувань для узгодження часу

Модуль реалізує механізм колективного вибору зручного часу проведення події. Організатор створює опитування з кількома варіантами дати та часу та встановлює дедлайн голосування. Учасники голосують за один або кілька

варіантів, що їм підходять. Після закінчення дедлайну або після ручної фіналізації організатором система визначає оптимальний варіант на основі кількості голосів і встановлює його як час проведення події. Реалізацію цього процесу відображено на блок-схемі Додатку І.

Форму створення події з увімкненою опцією опитування наведено на рис. 6.9. Підтвердження часу організатором зображено на рис. 6.10, вигляд сторінки під час голосування - на рис. 6.11.

Час проведення

**17** Фіксований час

Опитування

Дедлайн голосування

05 / 17 / 2026, 12:00 PM

Варіанти часу

05 / 18 / 2026, 10:10 AM

05 / 18 / 2026, 01:30 PM

05 / 19 / 2026, 01:30 PM

+ Додати варіант

Рис. 6.9 Форми створення подію, обрана опція опитувань

← Назад до спільноти

ПІДТВЕРДЖЕННЯ ЧАСУ

**ТЕРНУВАННЯ: ДЕБЮТИ**

Голосування завершено. Оберіть фінальний час проведення події.

РЕЗУЛЬТАТИ ГОЛОСУВАННЯ

|           |                           |           |
|-----------|---------------------------|-----------|
| <b>17</b> | 18 травня 2026 р. о 10:10 | 0 голосів |
| <b>17</b> | 18 травня 2026 р. о 13:30 | 0 голосів |
| <b>17</b> | 19 травня 2026 р. о 13:30 | 0 голосів |

✓ Підтвердити час події

Після підтвердження час буде встановлено та учасники побачать дату події.

Рис. 6.10 Форми підтвердження часу організатором, після голосування

**ТЕРНУВАННЯ: ДЕБЮТИ**

Як обрати ефективний дебют для твого рівня та стилю гри

3 год free Офлайн

NUBiP, корпус 9

**ОБЕРІТЬ ЗРУЧНИЙ ЧАС** до 17 травня 2026 р. о 12:00

Всього голосів: 1

|                           |                |
|---------------------------|----------------|
| 18 травня 2026 р. о 10:10 | 1 голос • 100% |
| 18 травня 2026 р. о 13:30 | 0 голосів • 0% |
| 19 травня 2026 р. о 13:30 | 0 голосів • 0% |
| Не можу прийти            | 0 голосів • 0% |

Підтвердити час Редагувати Скасувати

**РЕЗУЛЬТАТИ ГОЛОСУВАННЯ**

|                                                                       |           |
|-----------------------------------------------------------------------|-----------|
| Не зможу бути                                                         | 0 голосів |
| понеділок, 18 травня 2026 р. о 10:10<br>Тимофій Андрійович Свириденко | 1 голос   |
| понеділок, 18 травня 2026 р. о 13:30                                  | 0 голосів |
| вівторок, 19 травня 2026 р. о 13:30                                   | 0 голосів |

Рис. 6.11 Сторінка події з точки зору організатора, під час голосування

### 3.3.6 Модуль сповіщень

Модуль забезпечує інформування учасників про важливі події в системі через три канали: внутрішні (in-app) повідомлення, електронну пошту та push-сповіщення. Сповіщення реалізовано засобами Laravel Notifications із використанням черги завдань (Laravel Queue) для асинхронного надсилання, що унеможливорює затримки в роботі інтерфейсу. Учасники отримують сповіщення про нові опитування в групах, нагадування про незавершені голосування перед дедлайном, результати опитування після його завершення та нагадування про майбутні заходи. In-app повідомлення зберігаються в таблиці notifications та відображаються у відповідному розділі інтерфейсу.

Приклад in-аpp повідомлення наведено на рис. 6.12, історію повідомлень - на рис. 6.13.

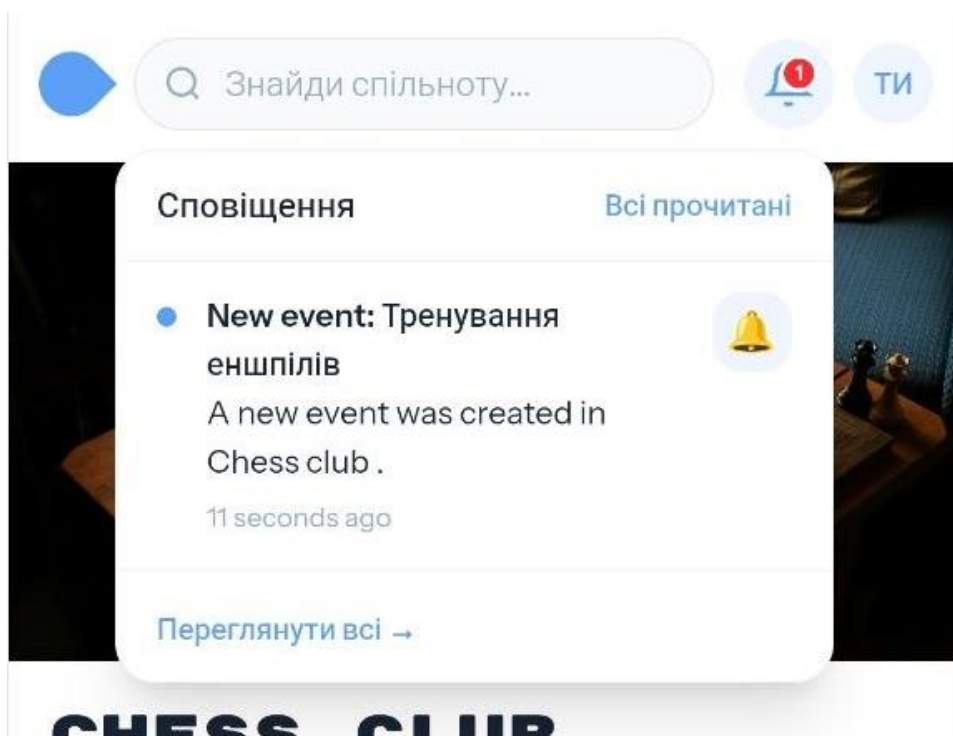


Рис. 6.12 Учасник групи отримує in-аpp повідомлення стосовно подій

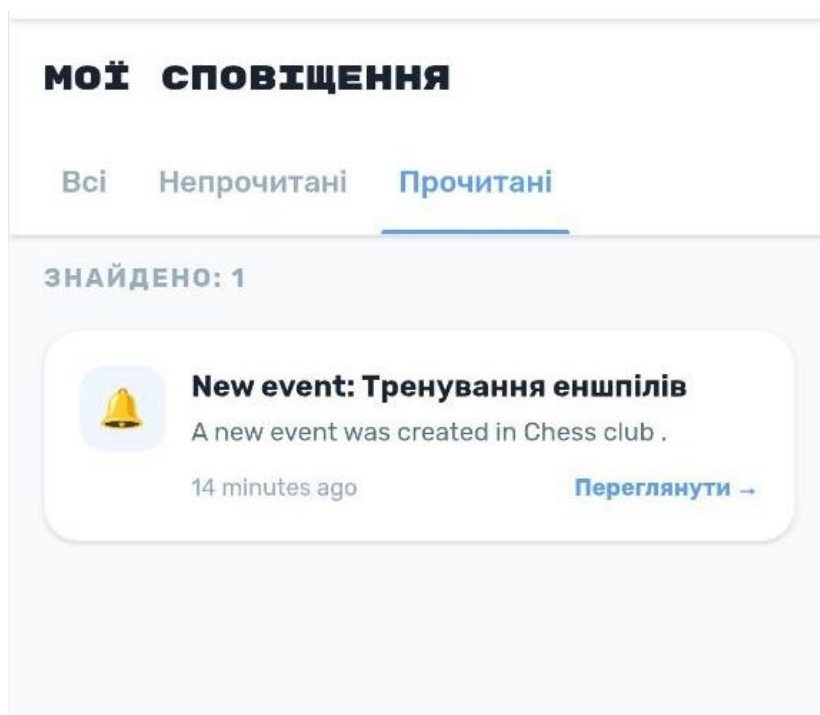


Рис. 6.13 Користувач може переглянути історію всіх повідомлень

### **Висновок до розділу 3**

У третьому розділі описано архітектуру та програмну реалізацію застосунку. Визначено організаційну структуру системи на основі патерну MVC із виокремленим сервісним шаром. Засобами UML змодельовано структуру та взаємодію об'єктів: побудовано діаграму класів із чотирма основними класами та діаграми послідовності й кооперацій для ключових сценаріїв. Діаграма компонентів відображає три рівні системи - Frontend, Backend та Infrastructure - і взаємодію між ними. Обґрунтовано вибір технологічного стеку: PHP 8.4 та Laravel 12 для серверної частини, React із TypeScript та Tailwind CSS для клієнтської, Inertia.js як містка між ними, Bun як інструменту збірки.

Реалізовані модулі в сукупності забезпечують виконання всіх функціональних вимог, визначених у технічному завданні.

## **4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ**

Розділ містить практичні відомості, необхідні для розгортання та експлуатації застосунку. Наведено результати функціонального та навантажувального тестування, визначено апаратні й програмні вимоги до серверного та клієнтського середовища, а також описано склад інсталяційного пакету і процедуру розгортання системи на продуктивному сервері.

### **4.1 Тестування системи**

Тестування застосунку проводилось у двох форматах: ручне функціональне тестування та навантажувальне тестування продуктивності. Автоматизовані тести у межах даної роботи не розроблялись - усі функціональні сценарії перевірялись вручну в браузері на розгорнутому робочому сервері .

#### **4.1.1 Функціональне тестування**

Усі сім функціональних модулів системи пройшли ручну перевірку. Тестувались основні користувацькі сценарії: реєстрація та авторизація через email і Google OAuth, створення груп та управління учасниками, створення подій із фіксованим часом та через механізм голосування, надсилання сповіщень і їх відображення в інтерфейсі, експорт події до Google Calendar. Критичних помилок у ході тестування виявлено не було.

#### **4.1.2 Навантажувальне тестування продуктивності**

Навантажувальне тестування проводилось за допомогою утиліти ApacheBench [13].

Команда для тесту:

ab -n 1000 -c 100 -t 60 https://event-loop.org/group

Параметри тесту: 100 одночасних з'єднань, тривалість 60 секунд. За цей час було оброблено 1927 запитів без жодної помилки (Failed requests: 0).

Таблиця 7.1

Результати навантажувального тестування

| Показник                          | Значення    |
|-----------------------------------|-------------|
| Запитів на секунду                | 32,11 req/s |
| Середній час відповіді (на запит) | 3114 мс     |
| Медіанний час відповіді           | 2799 мс     |
| Максимальний час відповіді        | 6118 мс     |
| Провалених запитів                | 0           |

Нефункціональні вимоги системи передбачали час завантаження сторінок до 3 секунд та підтримку 1000+ одночасних користувачів із можливістю масштабування до 10 000+. За результатами тестування ці цілі в поточній конфігурації не досягнуто: середній час відповіді становить 3114 мс, що перевищує цільовий показник, а пропускна здатність у 32 req/s є недостатньою для обслуговування тисяч одночасних користувачів.

Причина полягає не в архітектурі застосунку, а в характеристиках поточного VPS - сервер із мінімальною конфігурацією (1–2 vCPU, 4 ГБ RAM) не розрахований на таке навантаження. Архітектура системи при цьому залишається масштабованою: використання Redis для кешування та черги завдань, асинхронне надсилання сповіщень і відокремлення фонових процесів через Supervisor надають основу для горизонтального та вертикального масштабування.

Для досягнення цільових показників продуктивності рекомендується перехід на VPS із потужнішими характеристиками (4–8 vCPU, 16 ГБ RAM), що за оцінкою дозволить скоротити середній час відповіді до прийнятних значень та забезпечити стабільну роботу при значно більшому навантаженні.

## **4.2 Вимоги до апаратного та програмного забезпечення**

### **4.2.1 Апаратні вимоги до VPS (Сервера)**

Для розгортання веб-застосунку необхідний сучасний хмарний віртуальний сервер (VPS) з актуальним обладнанням. Використання застарілих накопичувачів типу HDD є недоцільним через низьку швидкість введення-виведення, що негативно позначиться на продуктивності бази даних та часі відповіді системи.

Мінімальні вимоги:

- Процесор (vCPU): 1–2 ядра (сучасні серверні процесори від AMD/Intel).
- ОЗУ (RAM): 4 ГБ.
- Накопичувач: 20 ГБ+ SSD або NVMe.

### **4.2.2 Програмні вимоги до сервера**

Як операційну систему рекомендується використовувати Ubuntu 22.04 LTS або 24.04 LTS [11] — ці дистрибутиви мають розширену підтримку та найширшу екосистему для розгортання Laravel-застосунків. Вхідні HTTP-запити обробляє Nginx [14], що виступає зворотним проксі-сервером та роздає статичні файли. PHP-запити передаються застосунку через PHP-FPM, для управління PHP-залежностями використовується Composer [17]. Збірку frontend-активів виконує Bun — сучасний JavaScript runtime та бандлер, що замінює традиційну комбінацію Node.js і npm. Як основну базу даних використовується PostgreSQL [9] версії 14 або вище, розгорнутий на окремому порту з налаштованою

автентифікацією. Фонові процеси — воркер черги та планувальник завдань — контролюються через Supervisor [16], що забезпечує їх автоматичний перезапуск у разі збою. Для кешування даних, зберігання сесій та черги завдань рекомендується Redis [15], що суттєво знижує навантаження на базу даних.

#### **4.2.3 Вимоги до системи (з точки зору користувача)**

Для роботи з системою користувачеві достатньо мати сучасний веб-браузер з підтримкою JavaScript. Підтримувані браузери та їх мінімальні версії:

- Google Chrome: версія 110 і вище (Windows, macOS, Linux).
- Apple Safari: версія 16.0 і вище (macOS, iOS).
- Mozilla Firefox: версія 110 і вище (Windows, macOS, Linux).
- Microsoft Edge: версія 110 і вище (Windows, macOS).

Використання застарілих або нестандартних браузерів може призвести до некоректного відображення інтерфейсу або неповної підтримки функціональності системи. Додаткове встановлення програмного забезпечення на пристрій користувача не вимагається.

### **4.3 Склад інсталяційного пакету**

Інсталяційний пакет застосунку охоплює вихідний код репозиторію, скрипт автоматичного розгортання та конфігураційні файли. Вихідний код розділено на дві частини: серверну - PHP-файли Laravel разом із міграціями та сідерами бази даних, і клієнтську - JSX/CSS-файли React-компонентів. Налаштування середовища виконується через файл `.env`, який містить всі параметри конфігурацій Laravel: параметри підключення до бази даних, ключ застосунку та облікові дані сторонніх сервісів. Управління залежностями здійснюється через `composer.json` для PHP-пакетів та `package.json` / `bun.lock` для JavaScript/TypeScript. Конфігурація Nginx та Supervisor генерується автоматично під час виконання `bash`-скрипту розгортання.

### 4.3.1 Серверна архітектура

Застосунок розгорнуто на VPS Ubuntu. Діаграму інфраструктури наведено на Рис. 8.1.

Вхідний HTTPS-трафік спочатку проходить через Cloudflare, який забезпечує DNS-резольову, видачу SSL/TLS-сертифіката та фільтрацію DDoS-атак. Після цього запит надходить до Nginx, що виступає зворотним проксі: статичні файли з `public/build/` він роздає безпосередньо, а PHP-запити передає процесу PHP-FPM 8.4 через Unix-сокети. PHP-FPM виконує Laravel-застосунок, який звертається до PostgreSQL через SQL для зберігання основних даних та до Redis через TCP для кешування, сесій і черги завдань. Supervisor підтримує роботу двох фонових процесів - воркера черги (`artisan queue:work`) для асинхронного надсилання сповіщень та планувальника завдань (`artisan schedule:run`).

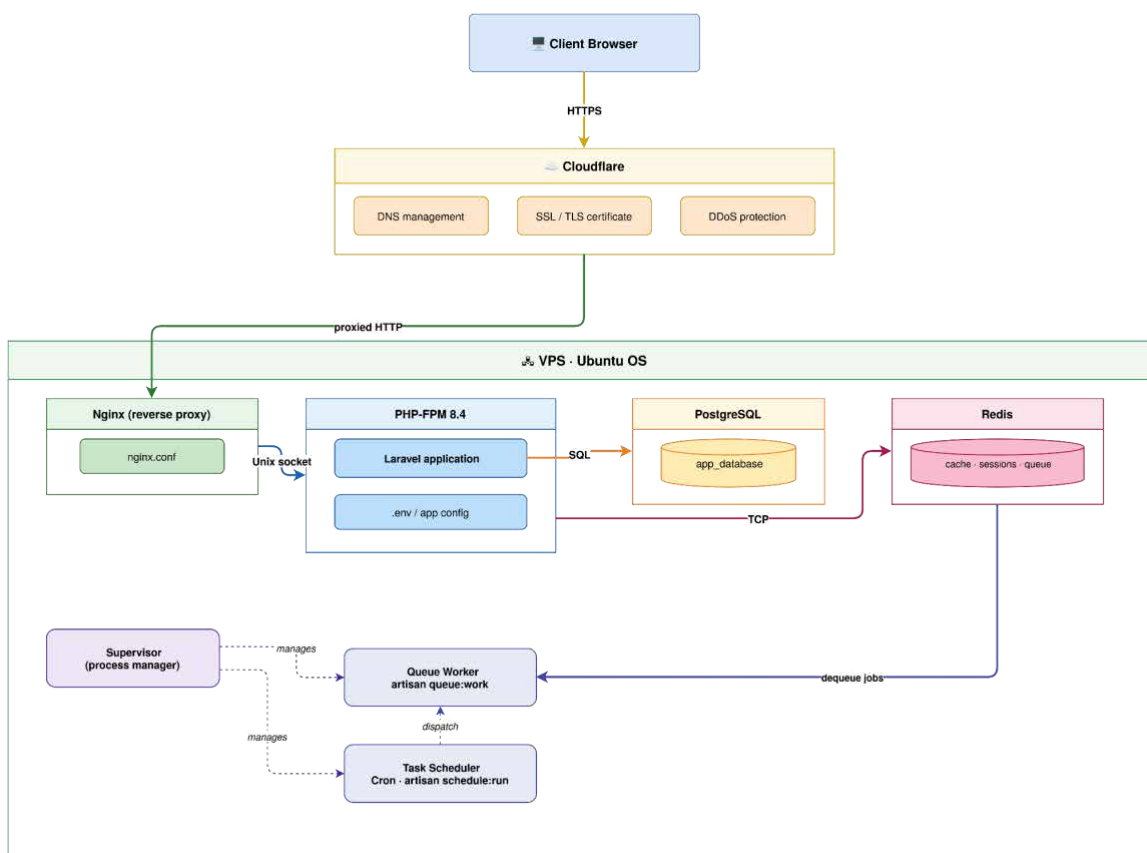


Рис. 8.1 Діаграма серверної інфраструктури

### 4.3.2 Процес розгортання

Первинне розгортання виконується запуском `bash`-скрипту на чистому VPS. Скрипт послідовно виконує такі кроки:

1. Оновлення системи та встановлення залежностей - Nginx, PHP 8.4 з необхідними розширеннями, Composer, PostgreSQL, Supervisor, Bun.
2. Клонування репозиторію до `/var/www/EventLoop` та встановлення PHP-залежностей через `composer install`.
3. Автоматичне налаштування файлу `.env` - підстановка облікових даних бази даних за допомогою `sed`.
4. Генерація ключа застосунку (`artisan key:generate`) та виконання міграцій бази даних (`artisan migrate`).
5. Налаштування прав доступу до файлової системи - директорії `storage/` та `bootstrap/cache/` отримують права на запис для користувача `www-data`.
6. Створення та активація конфігурації Nginx server block.
7. Налаштування Supervisor для запуску воркера черги у фоновому режимі.
8. Збірка frontend-активів командою `bun run build`, яка компілює JSX та Tailwind CSS у статичні файли у директорії `public/build/`.
9. Кешування конфігурації, маршрутів та шаблонів (`config:cache`, `route:cache`, `view:cache`) для оптимізації продуктивності.

### 4.3.3 Оновлення застосунку

Після первинного розгортання оновлення розгорнутого застосунку на сервері виконується однією командою:

```
cd /var/www/EventLoop && git pull && bun run build
```

### Висновок до розділу 4

Були наведені практичні рекомендації щодо розгортання та експлуатації системи. Проведено ручне функціональне тестування усіх семи модулів - критичних помилок не виявлено. Навантажувальне тестування засобами ApacheBench показало, що на поточній конфігурації VPS (1–2 vCPU, 4 ГБ RAM)

цільові показники продуктивності не досягнуто: середній час відповіді становить 3114 мс при цільовому значенні до 3000 мс. Встановлено, що причиною є апаратні обмеження сервера, а не архітектура застосунку; перехід на конфігурацію 4–8 vCPU та 16 ГБ RAM дозволить вийти на цільові показники. Визначено апаратні та програмні вимоги до серверного середовища, описано склад інсталяційного пакету та процедуру повністю автоматизованого розгортання через bash-скрипт.

## ВИСНОВКИ

У бакалаврській кваліфікаційній роботі розроблено програмне забезпечення веб-застосунку для планування та керування груповими зустрічами - EventLoop. Застосунок розгорнуто у відкритому доступі за адресою [event-loop.org](http://event-loop.org).

У ході виконання роботи проведено системний аналіз предметної області та розглянуто існуючі рішення - Calendly, Doodle, Meetup і Clockwise. Аналіз показав, що жодне з них не поєднує в інструменти управління спільнотами з механізмом автоматизованого узгодження часу зустрічей - саме цю нішу заповнює розроблений застосунок.

На основі виявлених вимог спроектовано логічну модель даних із 19 таблицями, що відповідає вимогам третьої нормальної форми, та реалізовано інформаційну базу засобами PostgreSQL. Для взаємодії з базою даних використано Eloquent ORM у складі фреймворку Laravel.

Архітектуру застосунку побудовано на основі патерну MVC. Серверну частину реалізовано на Laravel 12, клієнтську - на React із TypeScript та Tailwind CSS, зв'язок між ними забезпечено через Inertia.js. Реалізовано сім функціональних модулів: ідентифікації та управління профілем, управління групами, управління подіями, опитувань для узгодження часу, пошуку та навігації, сповіщень та інтеграції з Google Calendar. Усі модулі пройшли ручне функціональне тестування без виявлення критичних помилок.

Застосунок розгорнуто на VPS під керуванням Ubuntu з використанням Nginx, PHP-FPM, PostgreSQL, Redis та Supervisor. Розгортання повністю автоматизовано через bash-скрипт.

Навантажувальне тестування засобами ApacheBench виявило, що на поточній конфігурації сервера (1–2 vCPU, 4 ГБ RAM) цільові показники продуктивності не досягнуто. Водночас архітектура системи є масштабованою - перехід на більш потужніший VPS (4–8 vCPU, 16 ГБ RAM) дозволить вийти на цільові показники без змін у коді застосунку.

Таким чином, поставлену мету досягнуто: спроектовано та реалізовано повнофункціональний веб-застосунок, що автоматизує процеси організації групових зустрічей і може бути використаний студентськими об'єднаннями, клубами за інтересами та іншими спільнотами.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. PHP. Documentation. URL: <https://www.php.net/docs.php> (дата звернення: 17.05.2026).
2. Laravel. Documentation. URL: <https://laravel.com/docs/12.x/releases>. (дата звернення: 17.05.2026).
3. Laravel. Eloquent ORM Documentation. URL: <https://laravel.com/docs/12.x/eloquent>. (дата звернення: 17.05.2026).
4. Inertia.js Documentation. How It Works. URL: <https://inertiajs.com/docs/v2/core-concepts/how-it-works>. (дата звернення: 17.05.2026).
5. TypeScript Documentation. URL: <https://www.typescriptlang.org/docs/>. (дата звернення: 17.05.2026).
6. React. URL: <https://react.dev/>. (дата звернення: 17.05.2026).
7. Tailwind CSS Documentation. URL: <https://tailwindcss.com/>. (дата звернення: 17.05.2026).
8. Bun Documentation. URL: <https://bun.sh/docs>. (дата звернення: 17.05.2026).
9. PostgreSQL Documentation. URL: <https://www.postgresql.org/docs/>. (дата звернення: 17.05.2026).
10. Google. Using OAuth 2.0 to Access Google APIs. URL: <https://developers.google.com/identity/protocols/oauth2>. (дата звернення: 17.05.2026).
11. Ubuntu Server documentation. URL: <https://ubuntu.com/server/docs/>. (дата звернення: 17.05.2026).
12. Visual Paradigm. UML Diagram Documentation. URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/>. (дата звернення: 17.05.2026).
13. Apache HTTP Server Project. ApacheBench Documentation. URL: <https://httpd.apache.org/docs/2.4/programs/ab.html>. (дата звернення: 17.05.2026).
14. Nginx Documentation. URL: <https://nginx.org/en/docs/>. (дата звернення: 17.05.2026).

- 17.05.2026).
15. Redis Documentation. URL: <https://redis.io/docs/>. (дата звернення: 17.05.2026).
16. Supervisor Documentation. URL: <http://supervisord.org/introduction.html>. (дата звернення: 17.05.2026).
17. Composer Documentation. URL: <https://getcomposer.org/doc/>. (дата звернення: 17.05.2026).

## ДОДАТОК А

### Код Eloquent ORM моделей програми

```
<?php

namespace App\Models;
use Illuminate\Database\Eloquent\Factories\HasFactory;

use Illuminate\Foundation\Auth\User as Authenticatable;
use Illuminate\Notifications\Notifiable;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Laravel\Fortify\TwoFactorAuthenticatable;

class User extends Authenticatable
{
    /** @use HasFactory<\Database\Factories\UserFactory> */
    use HasFactory, Notifiable, TwoFactorAuthenticatable;

    /**
     * The attributes that are mass assignable.
     *
     * @var list<string>
     */
    protected $fillable = [
        'name',
        'email',
        'password',
        'gender',
        'google_id',
        'birth_year',
        'city',
    ];

    /**
     * The attributes that should be hidden for serialization.
     *

```

```

* @var list<string>
*/

protected $hidden = [
    'password',
    'two_factor_secret',
    'two_factor_recovery_codes',
    'remember_token',
    'google_access_token',
    'google_refresh_token',
];

/**
 * Get the attributes that should be cast.
 *
 * @return array<string, string>
 */
protected function casts(): array
{
    return [
        'email_verified_at' => 'datetime',
        'password' => 'hashed',
        'two_factor_confirmed_at' => 'datetime',
    ];
}

public function tags()
{
    return $this->belongsToMany(Tag::class, 'user_preference_tags');
}

public function group(): HasMany
{
    return $this->hasMany(Group::class);
}

public function userCreatedEvents(): HasMany
{
    return $this->hasMany(Event::class);
}

```

```

    }

    public function events(): BelongsToMany
    {
        return $this->belongsToMany(Event::class, 'event_participants', 'user_id', 'event_id')
            ->using(Event::class);
    }

    public function userGroupMembers(): BelongsToMany
    {
        return $this->belongsToMany(Group::class, 'group_members', 'user_id', 'group_id')
            ->withPivot(['role_id', 'status'])
            ->using(GroupMember::class);
    }

    public function userJoinRequest(): BelongsToMany
    {
        return $this->belongsToMany(Group::class, 'group_join_requests', 'user_id', 'group_id')
            ->using(GroupJoinRequest::class);
    }
}

```

```
<?php
```

```
namespace App\Models;
```

```

use Illuminate\Database\Eloquent\Relations\HasOne;
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;

```

```
class Event extends Model
```

```

{
    use HasFactory;

    protected $fillable = [
        'short_description',

```

```

        'user_id',
        'start_time',
        'duration_minutes',
        'max_participance',
        'title',
        'use_pull',
        'canceled',
        'price',
        'type',
        'location_or_link',
    ];

```

```

protected $casts = [
    'start_time' => 'datetime',
    'canceled'   => 'boolean',
    'use_pull'   => 'boolean',
];

```

```

public function group(): BelongsTo
{
    return $this->belongsTo(Group::class);
}

```

```

public function orginizer(): BelongsTo
{
    return $this->belongsTo(User::class, 'user_id');
}

```

```

public function user(): BelongsTo
{
    return $this->belongsTo(User::class, 'user_id');
}

```

```

public function participants(): BelongsToMany
{
    return $this->belongsToMany(User::class, 'event_participants', 'event_id', 'user_id')
        ->withPivot(['will_attend'])
        ->withTimestamps()
        ->using(EventParticipant::class);
}

```

```

    }

    public function poll(): HasOne
    {
        return $this->hasOne(Poll::class);
    }
}

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\Pivot;

class EventParticipant extends Pivot
{
    use HasFactory;

    protected $table = 'event_participants';

    public function event(): BelongsTo
    {
        return $this->belongsTo(Event::class);
    }

    public function user(): BelongsTo
    {
        return $this->belongsTo(User::class);
    }
}

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;

```

```

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;
use Illuminate\Database\Eloquent\Relations\HasMany;
use Illuminate\Database\Eloquent\SoftDeletes;
use App\Models\User;
use App\Models\GroupMember;
use App\Models\Tag;
use App\Models\GroupJoinRequest;

class Group extends Model
{
    use SoftDeletes;
    use HasFactory;
    protected $fillable = ['title', 'private', 'short_description', 'cover_image'];

    public function user(): BelongsTo
    {
        return $this->belongsTo(User::class);
    }

    public function members(): BelongsToMany
    {
        return $this->belongsToMany(User::class, 'group_members', 'group_id', 'user_id')
            ->withPivot(['role_id', 'status'])
            ->using(GroupMember::class);
    }

    public function events(): HasMany
    {
        return $this->hasMany(Event::class);
    }

    public function tags(): BelongsToMany
    {
        return $this->belongsToMany(Tag::class, 'group_to_tag', 'group_id', 'tag_id')
            ->withTimestamps();
    }

```

```

    }

    public function join_group_request(): HasMany
    {
        return $this->HasMany(GroupJoinRequest::class);
    }

    public function join_requests(): BelongsToMany
    {
        return $this->belongsToMany(User::class, 'group_join_requests', 'group_id', 'user_id')
            ->using(GroupJoinRequest::class);
    }

    public function get_member_role(User $user): ?int
    {
        $user_status = $this->members()->firstWhere("user_id", "=", $user->id);
        if ($user_status === null) {
            return null;
        }
        return $user_status->pivot->role_id;
    }

    public function join_group_link(): HasMany
    {
        return $this->HasMany(GroupJoinLink::class);
    }
}

```

```
<?php
```

```

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Concerns\HasUuids;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class GroupJoinLink extends Model
{
    use HasFactory, HasUuids;

    protected $fillable = [

```

```

        'group_id',
        'invited_by',

];
protected $casts = [
    'time_to_expiration' => 'datetime:Y-m-d H:i:s',
];
/**
 * The "booted" method of the model.
 */
protected static function booted(): void
{
    static::creating(function (GroupJoinLink $invite) {
        // If the time hasn't been manually set, default to 24 hours from now
        if (!$invite->time_to_expiration) {
            $invite->time_to_expiration = now()->addHours(24);
        }
    });
}
public function created_by(): BelongsTo
{
    return $this->belongsTo(User::class, 'invited_by');
}

public function group(): BelongsTo
{
    return $this->belongsTo(Group::class);
}
}

```

```
<?php
```

```
namespace App\Models;
```

```
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Relations\Pivot;
```

```
class GroupJoinRequest extends Pivot
{
    use HasFactory;
}

```

```
<?php
```

```
namespace App\Models;
```

```
use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\Pivot;
```

```
class GroupMember extends Pivot
{
    use HasFactory;

    protected $table = 'group_members';

    protected $fillable = ['role_id', 'status'];
}

```

```

protected $with = ['role'];

public function role(): BelongsTo
{
    return $this->belongsTo(Role::class, 'role_id');
}
public function getRouteKeyName()
{
    return 'user_id';
}
}

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;

class Permission extends Model
{
    use HasFactory;
    protected $fillable = [
        'name',
    ];
    public function roles()
    {
        return $this->belongsToMany(Role::class, 'role_has_permissions', 'permissions_id', 'roles_id');
    }
}

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;

class Poll extends Model
{
    protected $fillable = [
        'vote_deadline',
        'finished',
    ];
    protected $attributes = [
        'anonymous' => false,
        'default_absence_option' => true,
    ];

    public function event(): BelongsTo
    {
        return $this->belongsTo(Event::class);
    }
    public function poll_options(): HasMany
    {

```

```

        return $this->hasMany(PollOption::class);
    }
}

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsToMany;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;

class PollOption extends Model
{
    protected $fillable = [
        'proposed_time',
    ];

    public $timestamps = false;

    public function poll(): BelongsTo
    {
        return $this->belongsTo(Poll::class);
    }

    public function poll_option_votes(): HasMany
    {
        return $this->hasMany(PollVote::class);
    }
    public function users(): BelongsToMany
    {
        return $this->belongsToMany(User::class, 'poll_votes', 'poll_option_id', 'user_id')
            ->withTimestamps();
    }
}

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Relations\BelongsToMany;
use Illuminate\Database\Eloquent\Relations\Pivot;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class PollVote extends Pivot
{
    protected $table = 'poll_votes';
    public function user(): BelongsTo
    {
        return $this->belongsTo(User::class, 'user_id');
    }

    // Changed to 'BelongsTo'
    public function poll_option(): BelongsTo
    {
        return $this->belongsTo(PollOption::class, 'poll_option_id');
    }
}

```

```

    }
}

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Factories\HasFactory;
use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;
use Illuminate\Database\Eloquent\Relations\HasMany;

class Role extends Model
{
    use HasFactory;

    public const IS_CREATOR = 1;
    public const IS_ADMIN = 2;
    public const IS_MEMBER = 3;
    public const IS_BANED = 4;

    protected $fillable = ['name'];

    public function permissions()
    {
        return $this->belongsToMany(Permission::class, 'role_has_permissions', 'role_id', 'permission_id');
    }

    public function role_members(): HasMany
    {
        return $this->hasMany(GroupMember::class);
    }
}

<?php

namespace App\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\Relations\BelongsTo;

class Tag extends Model
{
    //
    protected $fillable = [
        'tag',
        'tag_category',
    ];

    public function groups(): BelongsTo
    {
        return $this->belongsTo(Group::class, 'group_to_tag', 'tag_id', 'group_id')
            ->withTimestamps();
    }
}

```

## ДОДАТОК Б

### Міграцій, код генерації структури бази даних

```
<?php

use Database\Seeders\GroupTagSeeder;
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('tags', function (Blueprint $table) {
            $table->id();
            $table->string('tag',30)->unique();
            $table->string('tag_category',30);
        });

        new GroupTagSeeder()->run();
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('tags');
    }
};
```

```
<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('users', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->string('email')->unique();
            $table->string('city', 100)->nullable();
            $table->string('google_id')->nullable()->unique();
```

```

        $table->text('google_access_token')->nullable();
        $table->text('google_refresh_token')->nullable();
        $table->boolean('gender')->nullable();
        $table->year('birth_year')->nullable();
        $table->timestamp('email_verified_at')->nullable();
        $table->string('password');
        $table->rememberToken();
        $table->timestamps();
    });

    Schema::create('user_preference_tags', function (Blueprint $table) {
        $table->id();

        $table->foreignId('user_id')->constrained()->onDelete('cascade');
        $table->foreignId('tag_id')->constrained()->onDelete('cascade');

        $table->unique(['user_id', 'tag_id']);

        $table->timestamps();
    });

    Schema::create('password_reset_tokens', function (Blueprint $table) {
        $table->string('email')->primary();
        $table->string('token');
        $table->timestamp('created_at')->nullable();
    });

    Schema::create('sessions', function (Blueprint $table) {
        $table->string('id')->primary();
        $table->foreignId('user_id')->nullable()->index();
        $table->string('ip_address', 45)->nullable();
        $table->text('user_agent')->nullable();
        $table->longText('payload');
        $table->integer('last_activity')->index();
    });
}

/**
 * Reverse the migrations.
 */
public function down(): void
{
    Schema::dropIfExists('users');
    Schema::dropIfExists('password_reset_tokens');
    Schema::dropIfExists('sessions');
}
};

<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**

```

```

* Run the migrations.
*/
public function up(): void
{
    Schema::create('cache', function (Blueprint $table) {
        $table->string('key')->primary();
        $table->mediumText('value');
        $table->integer('expiration')->index();
    });

    Schema::create('cache_locks', function (Blueprint $table) {
        $table->string('key')->primary();
        $table->string('owner');
        $table->integer('expiration')->index();
    });
}

/**
 * Reverse the migrations.
 */
public function down(): void
{
    Schema::dropIfExists('cache');
    Schema::dropIfExists('cache_locks');
}
};

```

<?php

```

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

```

```

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('jobs', function (Blueprint $table) {
            $table->id();
            $table->string('queue')->index();
            $table->longText('payload');
            $table->unsignedTinyInteger('attempts');
            $table->unsignedInteger('reserved_at')->nullable();
            $table->unsignedInteger('available_at');
            $table->unsignedInteger('created_at');
        });

        Schema::create('job_batches', function (Blueprint $table) {
            $table->string('id')->primary();
            $table->string('name');
            $table->integer('total_jobs');
            $table->integer('pending_jobs');
            $table->integer('failed_jobs');
            $table->longText('failed_job_ids');
        });
    }
}

```

```

        $table->mediumText('options')->nullable();
        $table->integer('cancelled_at')->nullable();
        $table->integer('created_at');
        $table->integer('finished_at')->nullable();
    });

    Schema::create('failed_jobs', function (Blueprint $table) {
        $table->id();
        $table->string('uuid')->unique();
        $table->text('connection');
        $table->text('queue');
        $table->longText('payload');
        $table->longText('exception');
        $table->timestamp('failed_at')->useCurrent();
    });
}

/**
 * Reverse the migrations.
 */
public function down(): void
{
    Schema::dropIfExists('jobs');
    Schema::dropIfExists('job_batches');
    Schema::dropIfExists('failed_jobs');
}
};

<?php

use Illuminate\Support\Facades\Schema;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Database\Migrations\Migration;
use Database\Seeders\PermissionSeeder;

return new class extends Migration {
    /**
     * Run the migrations.
     */
    public function up(): void
    {

        Schema::create('permissions', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->timestamps();

            $table->unique('name');
        });

        Schema::create('roles', function (Blueprint $table) {
            $table->id();
            $table->string('name');
            $table->timestamps();

            $table->unique('name');
        });
    }
};

```

```

        Schema::create('role_has_permissions', function (Blueprint $table) {
            $table->foreignId('role_id')->constrained()->cascadeOnDelete();
            $table->foreignId('permission_id')->constrained()->cascadeOnDelete();
            $table->primary(['role_id', 'permission_id']);
        });
        (new PermissionSeeder)->run();
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::drop('role_has_permissions');
        Schema::drop('roles');
        Schema::drop('permissions');
    }
};

<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::table('users', function (Blueprint $table) {
            $table->text('two_factor_secret')->after('password')->nullable();
            $table->text('two_factor_recovery_codes')->after('two_factor_secret')->nullable();
            $table->timestamp('two_factor_confirmed_at')->after('two_factor_recovery_codes')->nullable();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::table('users', function (Blueprint $table) {
            $table->dropColumn([
                'two_factor_secret',
                'two_factor_recovery_codes',
                'two_factor_confirmed_at',
            ]);
        });
    }
};

<?php

```

```

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create("groups", function (Blueprint $table) {
            $table->id();
            $table->string('cover_image')->nullable();
            $table->boolean("private");
            $table->foreignId("user_id")->constrained();
            $table->text("short_description");
            $table->char("title", 255);
            $table->timestamps();
            $table->softDeletes();
        });

        Schema::create("group_join_requests", function (Blueprint $table) {
            $table->foreignId("group_id")->constrained();
            $table->foreignId("user_id")->constrained();
            $table->foreignId("invited_by")->nullable()->constrained("users");
            $table->timestamps();

            $table->primary(["group_id", "user_id"]);
        });

        Schema::create("group_join_links", function (Blueprint $table) {
            $table->uuid("id")->primary();
            $table->foreignId("group_id")->constrained();
            $table->foreignId("invited_by")->nullable()->constrained("users");
            $table->dateTime("time_to_expiration");
            $table->timestamps();
        });
        // new GroupTagSeeder()->run();

        Schema::create("group_members", function (Blueprint $table) {
            $table->foreignId("group_id")->constrained();
            $table->foreignId("user_id")->constrained();
            $table->char("status", 20);
            $table->foreignId("role_id")->references("id")->on("roles");
            $table->unique(["group_id", "user_id"]);
        });

        Schema::create("group_to_tag", function (Blueprint $table) {
            $table->id();
            $table->foreignId("group_id")->constrained()->onDelete("cascade");
            $table->foreignId("tag_id")->constrained()->onDelete("cascade");
            $table->timestamps();
        });
    }
}

```

```

/**
 * Reverse the migrations.
 */
public function down(): void
{
    Schema::dropIfExists("group_to_tag");
    Schema::dropIfExists("group_join_requests");
    Schema::dropIfExists("group_invite_link");

    Schema::dropIfExists("group_members");
    Schema::dropIfExists("group");
}
};

<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
use Database\Seeders\PermissionSeeder;

return new class extends Migration {
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('events', function (Blueprint $table) {
            $table->id();
            $table->timestamps();

            $table->foreignId('group_id')->constrained()->cascadeOnDelete();
            $table->foreignId('user_id')->constrained();

            $table->char("title", 40);
            $table->text('short_description');
            $table->dateTime('start_time')->nullable();
            $table->unsignedInteger('duration_minutes');
            $table->integer('max_participance');
            $table->boolean('canceled')->default(false);
            $table->char('price', 20)->default('free');

            $table->enum('type', ['online', 'in_person'])->default('in_person');
            $table->string('location_or_link')->nullable(); // Address OR Meeting Link
        });

        Schema::create('event_participants', function (Blueprint $table) {
            $table->foreignId('event_id')->constrained();
            $table->foreignId('user_id')->constrained();
            $table->boolean('will_attend');
            $table->timestamps();
            $table->primary(['event_id', 'user_id']);
        });
    }
}

/**
 * Reverse the migrations.

```

```

    */
    public function down(): void
    {
        Schema::dropIfExists('event_participants');
        Schema::dropIfExists('events');
    }
};

<?php

use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;

return new class extends Migration
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('polls', function (Blueprint $table) {
            $table->id();
            $table->foreignId('event_id')->constrained();
            $table->dateTime('vote_deadline');
            $table->boolean('finished')->default(false);
            $table->boolean('default_absence_option')->default(true);
            $table->boolean('anonymous')->default(false);
            $table->timestamps();
        });

        Schema::create('poll_options', function (Blueprint $table) {
            $table->id();
            $table->foreignId('poll_id')->constrained();
            $table->dateTime('proposed_time')->nullable();
        });

        Schema::create('poll_votes', function (Blueprint $table) {
            $table->id();
            $table->foreignId('poll_option_id')->constrained();
            $table->foreignId('user_id')->constrained();
            $table->unique(['poll_option_id', 'user_id']);
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('polls');
        Schema::dropIfExists('poll_options');
        Schema::dropIfExists('poll_votes');
    }
};

```

```
<?php
```

```
use Illuminate\Database\Migrations\Migration;
use Illuminate\Database\Schema\Blueprint;
use Illuminate\Support\Facades\Schema;
```

```
return new class extends Migration
```

```
{
    /**
     * Run the migrations.
     */
    public function up(): void
    {
        Schema::create('notifications', function (Blueprint $table) {
            $table->uuid('id')->primary();
            $table->string('type');
            $table->morphs('notifiable');
            $table->text('data');
            $table->timestamp('read_at')->nullable();
            $table->timestamps();
        });
    }

    /**
     * Reverse the migrations.
     */
    public function down(): void
    {
        Schema::dropIfExists('notifications');
    }
};
```

## ДОДАТОК В

### Файл маршрутизації web.php

```
<?php
```

```
use App\Http\Controllers\GoogleAuthController;
use App\Http\Controllers\MainController;
use App\Http\Controllers\EventController;
use App\Http\Controllers\GroupMembersController;
use App\Http\Controllers\JoinGroupController;
use App\Http\Controllers\OnboardingController;
use Illuminate\Support\Facades\Route;
use Inertia\Inertia;
use Laravel\Fortify\Features;
use App\Http\Controllers\GroupController;
use App\Http\Controllers\JoinGroupLinkController;
use App\Http\Controllers\UserDashboardController;

Route::get('/', [MainController::class, 'index'])->name('home');

Route::get('/auth/google/redirect', [GoogleAuthController::class, 'redirect'])->name('auth.google.redirect');
Route::get('/auth/google/callback', [GoogleAuthController::class, 'callback'])->name('auth.google.callback');

Route::middleware(['auth', 'verified'])->group(function () {
    Route::post('/notifications/{id}/read', function ($id) {
        auth()->user()->unreadNotifications->where('id', $id)->markAsRead();
        return back();
    });

    // Mark all as read
    Route::post('/notifications/read-all', function () {
        auth()->user()->unreadNotifications->markAsRead();
        return back();
    });

    Route::get('/onboarding', [OnboardingController::class, 'create'])->name('onboarding.create');
    Route::get('/onboarding/edit', [OnboardingController::class, 'edit'])->name('onboarding.edit');
    Route::post('/onboarding', [OnboardingController::class, 'store'])->name('onboarding.store');

    Route::resource('/group', GroupController::class)
        ->only(['create', 'store', 'update', 'destroy']);

    Route::get('/group/{group}/edit', [GroupController::class, 'edit'])->name('group.edit');

    Route::prefix('dashboard')->name('dashboard.')->group(function () {
        Route::get('/', [UserDashboardController::class, 'index'])->name('index');
        Route::get('/notifications', [UserDashboardController::class, 'notifications'])->name('notifications');
    });

    Route::prefix('group/{group}/members')->name('group.members.')->group(function () {

        // List all members (tab on group page)
        Route::get('/', [GroupMembersController::class, 'index'])->name('index');
```

```

// Admin: search & add a new member
Route::get('/add', [GroupMembersController::class, 'create'])->name('add');
Route::post('/{new_member}', [GroupMembersController::class, 'store'])->name('store');

// Admin: remove a member (actor = auth user, target = member)
Route::delete(
    '{member_that_deletes}/{member_to_delete}',
    [GroupMembersController::class, 'delete_member']
)->name('delete');

// Admin: change role
Route::patch(
    '{member_to_change_role}/role',
    [GroupMembersController::class, 'edit_member_role']
)->name('role');

// Admin: change status
Route::patch(
    '{member_to_change_status}/status',
    [GroupMembersController::class, 'edit_member_status']
)->name('status');
});
Route::prefix('group/{group}/join_group_link')->name('group.join_group_link.')->group(function () {
    Route::get('/show', [JoinGroupLinkController::class, 'create'])->name('create');
    Route::post('/show', [JoinGroupLinkController::class, 'store'])->name('store');
});

Route::prefix('group/{group}/events')->name('event.')->group(function () {

    Route::post('/', [EventController::class, 'store'])->name('store');

    Route::get('/create', [EventController::class, 'create'])
        ->name('create');

    Route::get('/{event}', [EventController::class, 'show'])->name('show');
    Route::patch('/{event}', [EventController::class, 'update'])
        ->name('update');
    Route::get('/{event}/edit', [EventController::class, 'edit'])
        ->name('edit');

    Route::post('/{event}/toggle_cancel', [EventController::class, 'toggle_cancel'])
        ->name('cancel')
        ->withTrashed()
        ->withoutScopedBindings();

    Route::post('{event}/poll/vote', [EventController::class, 'vote'])
        ->name('poll.vote');
    Route::get('{event}/confirm_time', [EventController::class,
'createConfirmTime'])->name('events.confirm_time');
    Route::post('{event}/confirm_time', [EventController::class,
'storeConfirmedTime'])->name('events.set_time');
    Route::post('{event}/participate', [EventController::class,
'storeFinalParticipation'])->name('events.set_time');

```

```

});

// — Join requests


---


Route::prefix('group/{group}/requests')->name('group.requests.')->group(function () {

    // Admin: list all pending requests
    Route::get('/', [JoinGroupController::class, 'index'])->name('index');

    // Any authenticated non-member: send join request
    Route::post('/', [JoinGroupController::class, 'store'])->name('store');

    // Admin: approve request
    Route::post('/{member_request}/approve', [JoinGroupController::class, 'approve'])->name('approve');

    // Admin: decline request
    Route::delete('/{member_request}/decline', [JoinGroupController::class, 'decline'])->name('decline');
});

Route::resource('/group', GroupController::class)
    ->only(['index', 'show']);

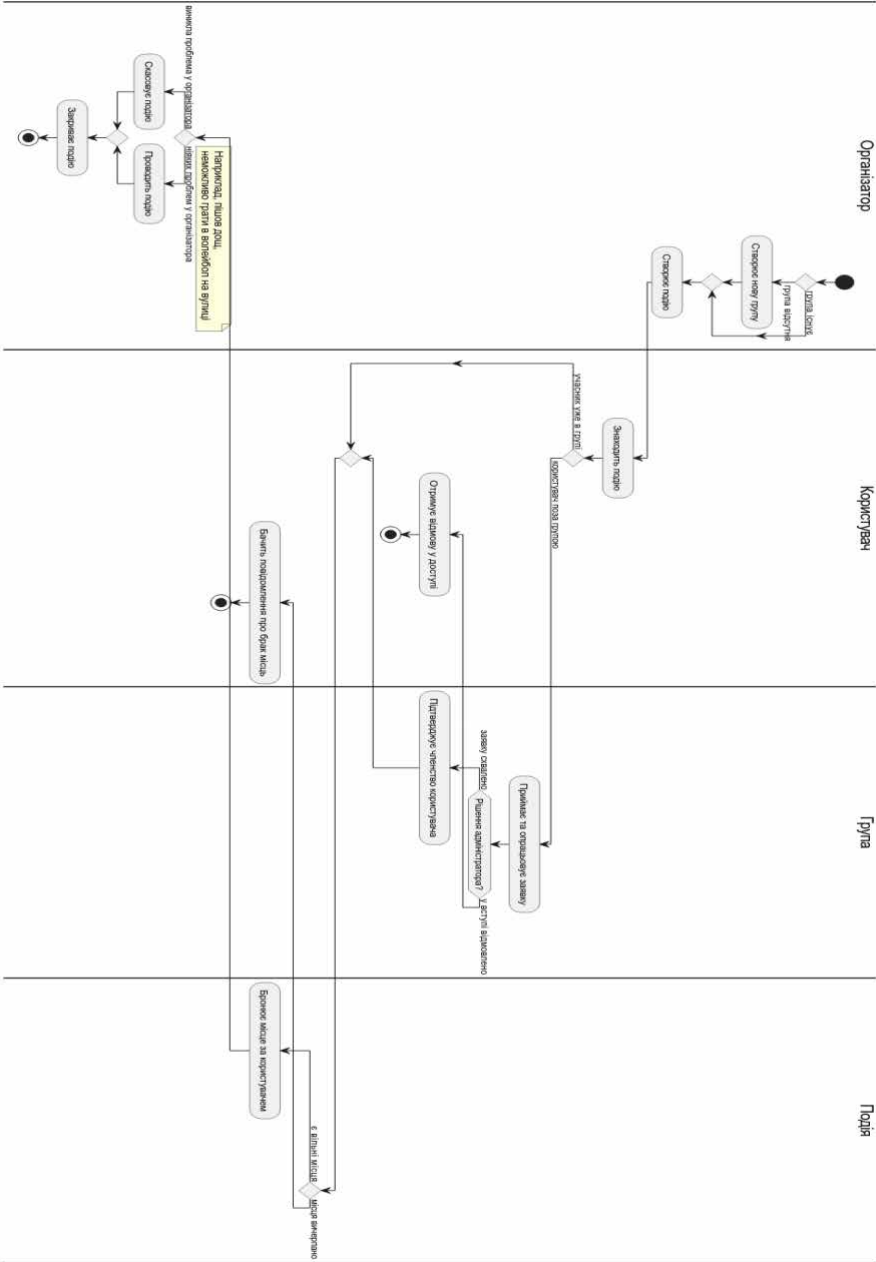
Route::get('/group/{group}/join_group_link/{join_group_link}', [JoinGroupLinkController::class,
'join_group'])
    ->name('group.join_group_link.join_group');

require __DIR__ . '/settings.php';

```

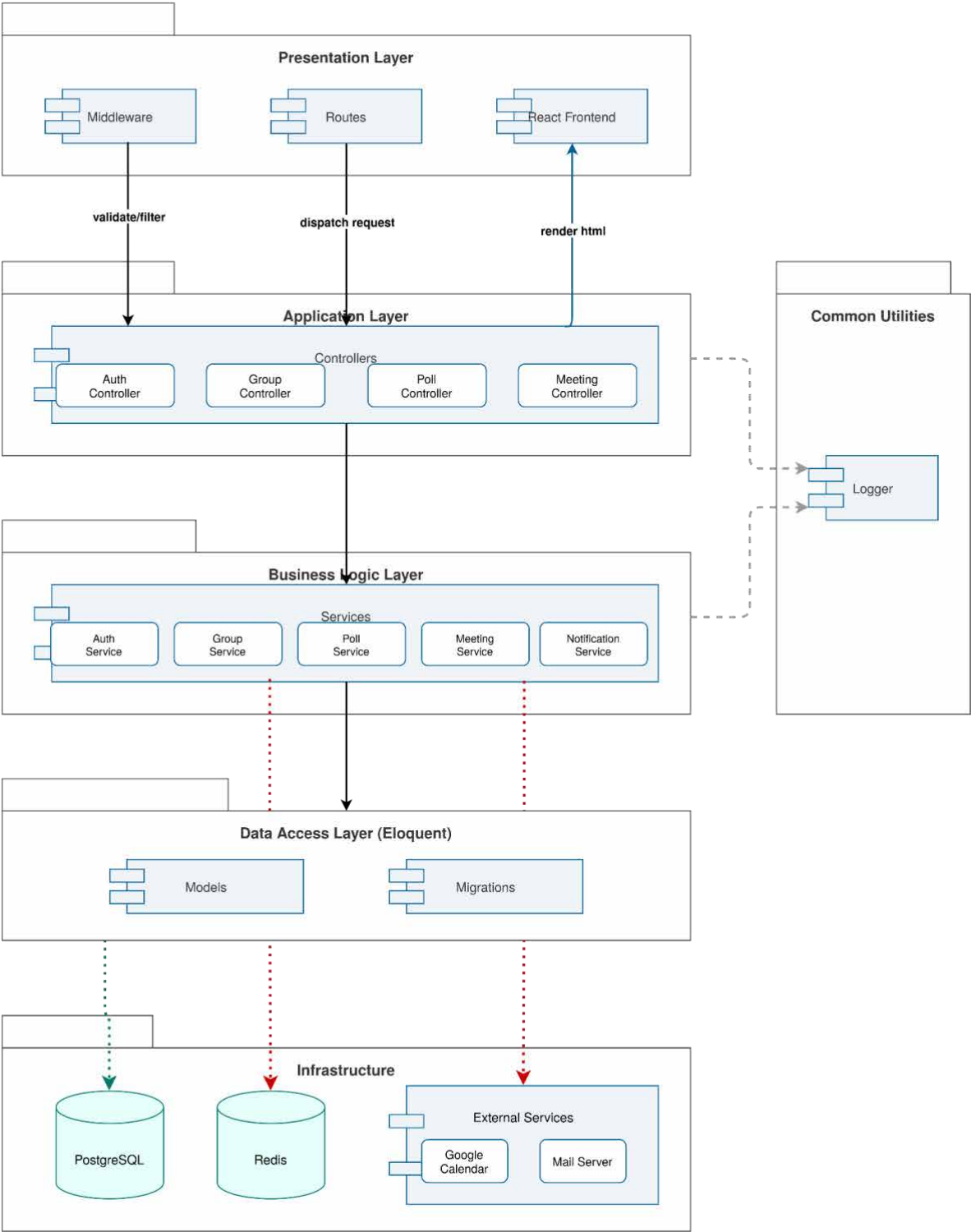
# ДОДАТОК Д

## Діаграма активності «Життєвий цикл події»



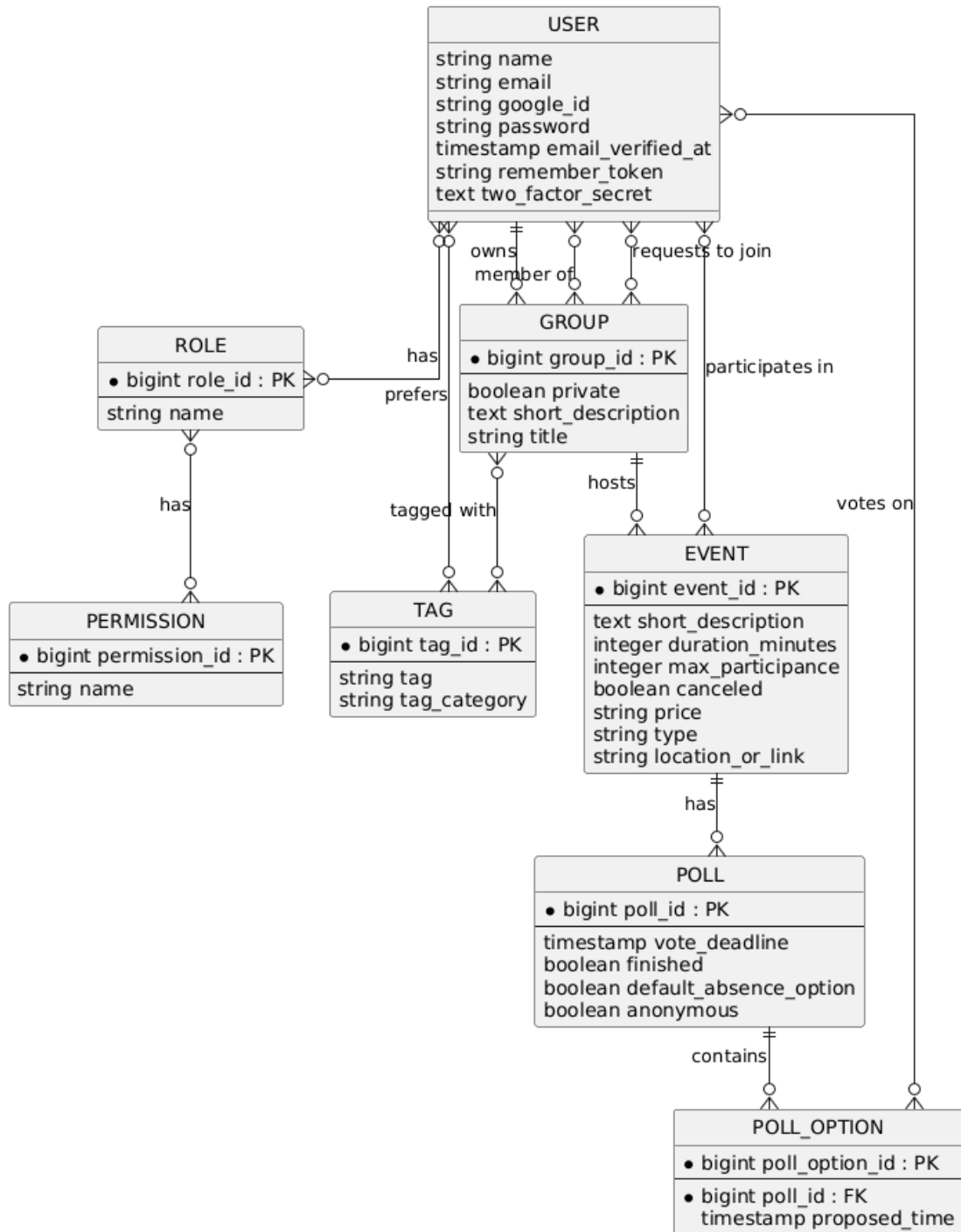
# ДОДАТОК Е

## Діаграма пакетів



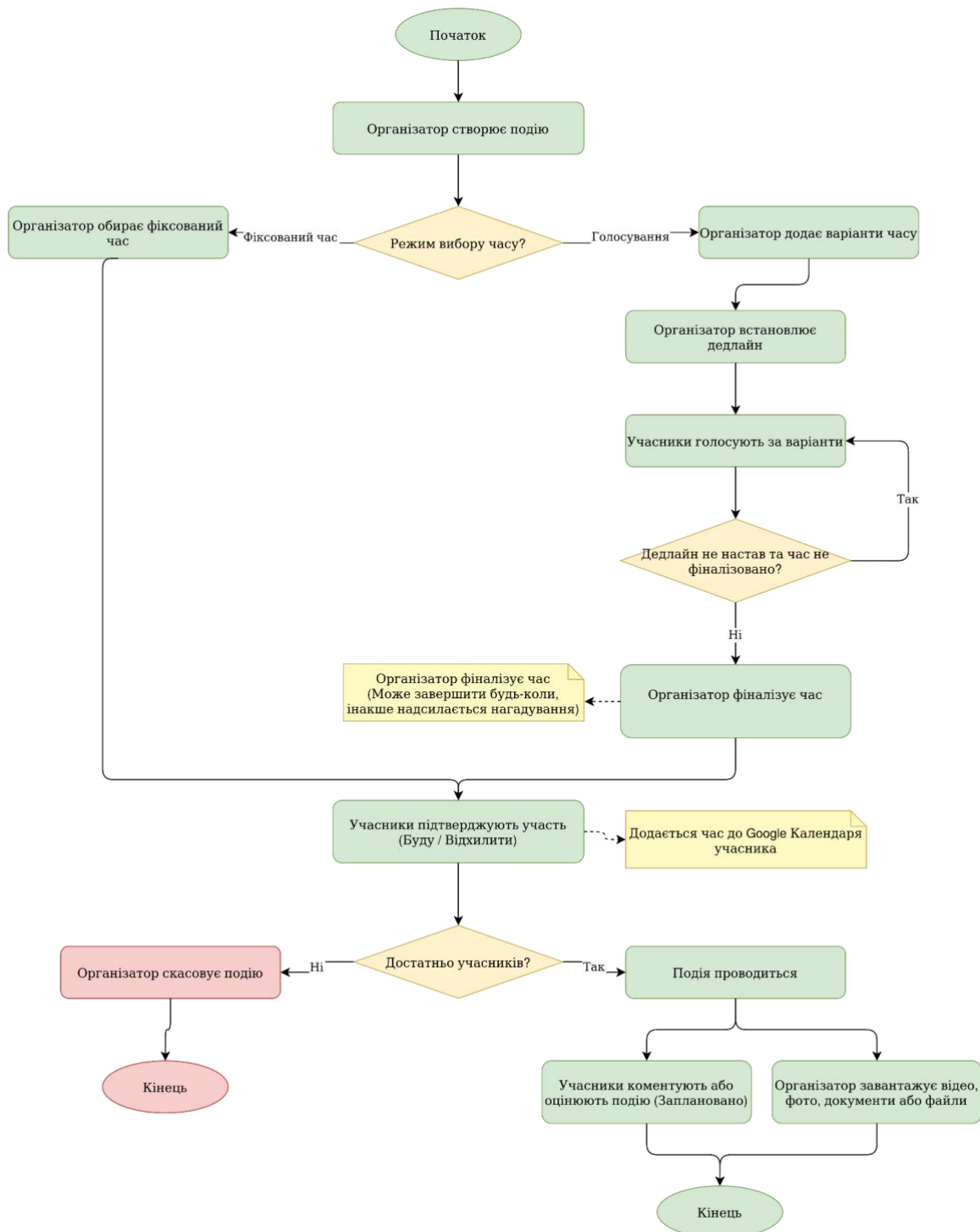
## ДОДАТОК Ж

### ER-діаграма логічної моделі даних



## ДОДАТОК II

### Блок-схема «Життєвий цикл події»



## ДОДАТОК К

### Основні таблиці предметної області з бази даних

Таблиця К.1

Структура таблиці users

| Стовпець          | Тип даних    | Обмеження           | Опис                               |
|-------------------|--------------|---------------------|------------------------------------|
| id                | BIGINT       | PK                  | Унікальний ідентифікатор           |
| name              | VARCHAR(255) | NOT NULL            | Ім'я користувача                   |
| email             | VARCHAR(255) | NOT NULL,<br>UNIQUE | Електронна адреса                  |
| city              | VARCHAR(100) | NULLABLE            | Місто проживання                   |
| google_id         | VARCHAR(255) | NULLABLE,<br>UNIQUE | ID облікового запису Google        |
| gender            | BOOLEAN      | NULLABLE            | Стать користувача                  |
| birth_year        | YEAR         | NULLABLE            | Рік народження                     |
| email_verified_at | TIMESTAMP    | NULLABLE            | Час підтвердження email            |
| password          | VARCHAR(255) | NOT NULL            | Хеш пароля                         |
| remember_token    | VARCHAR(100) | NULLABLE            | Токен сесії «запам'ятати мене»     |
| two_factor_secret | TEXT         | NULLABLE            | Секрет двофакторної автентифікації |

|                            |           |          |                       |
|----------------------------|-----------|----------|-----------------------|
| two_factor_recover_y_codes | TEXT      | NULLABLE | Коди відновлення 2FA  |
| two_factor_confirmed_at    | TIMESTAMP | NULLABLE | Час підтвердження 2FA |

Таблиця К.2

## Структура таблиці tags

| Стовпець     | Тип даних   | Обмеження        | Опис                     |
|--------------|-------------|------------------|--------------------------|
| id           | BIGINT      | PK               | Унікальний ідентифікатор |
| tag          | VARCHAR(30) | NOT NULL, UNIQUE | Назва тегу               |
| tag_category | VARCHAR(30) | NOT NULL         | Категорія тегу           |

Таблиця К.3

## Структура таблиці events

| Стовпець          | Тип даних | Обмеження               | Опис                          |
|-------------------|-----------|-------------------------|-------------------------------|
| id                | BIGINT    | PK                      | Унікальний ідентифікатор      |
| group_id          | BIGINT    | FK → groups.id, DELETE  | Група, до якої належить подія |
| user_id           | BIGINT    | FK → users.id, NOT NULL | Організатор події             |
| title             | CHAR(40)  | NOT NULL                | Назва події                   |
| short_description | TEXT      | NOT NULL                | Опис події                    |
| start_time        | DATETIME  | NULLABLE                | Дата та час початку           |

|                  |                                |                                  |                                 |
|------------------|--------------------------------|----------------------------------|---------------------------------|
| duration_minutes | INTEGER<br>UNSIGNED            | NOT NULL                         | Тривалість у хвилинах           |
| max_participance | INTEGER                        | NOT NULL                         | Максимальна кількість учасників |
| canceled         | BOOLEAN                        | NOT NULL,<br>DEFAULT false       | Ознака скасування               |
| price            | CHAR(20)                       | NOT NULL,<br>DEFAULT 'free'      | Вартість участі                 |
| type             | ENUM('online','in_per<br>son') | NOT NULL,<br>DEFAULT 'in_person' | Формат проведення               |
| location_or_link | VARCHAR(255)                   | NULLABLE                         | Адреса або посилання            |

## ДОДАТОК Л

### Програмування програмних модулів

**ПРИЄДНУЙСЯ ДО СПІЛЬНОТИ**

**Створіть обліковий запис**  
Введіть дані нижче, щоб створити свій обліковий запис

 Зареєструватися через Google

або за електронною поштою

Ім'я

Повне ім'я

Електронна адреса

email@example.com

Пароль

\*\*\*\*\*

Підтвердження пароля

\*\*\*\*\*

**Створити обліковий запис**

Вже є обліковий запис? [Увійти](#)

Рис. Д.1 Модуль ідентифікації та управління профілем:  
Форма реєстрації нового користувача

## ЗНАЙДІТЬ СВОЮ СПІЛЬНОТУ

Введіть електронну адресу та пароль, щоб увійти

### Ласкаво просимо назад 🙌

Увійдіть, щоб продовжити

Увійти через Google

або за електронною поштою

Електронна адреса

Пароль

☐ Пам'ятати мене

[Забув пароль?](#)

Увійти

Немає облікового запису? [Зареєструватися](#)

Рис. Д.2 Модуль ідентифікації та управління профілем: Форма авторизації користувача

НОВА СПІЛЬНОТА

Натисніть, щоб додати обкладинку

СТВОРИТИ ГРУПУ

Ваша група з'явиться у пошуку одразу після створення.

Назва групи

English Speaking Club

Опис

B / S H1 H2 H3 ≡ 1≡ " </> ↶ ↷

Розкажіть про вашу спільноту...

Категорії (оберіть одну або кілька)

#Volleyball

#Basketball

#Tennis

#Hiking

#Coffee Meetup

#Board Games

#Book Club

#Pub Quiz

#Coding

#Photography

#Language Exchange

#Yoga

↑

Завантажити обкладинку

Натисніть, щоб обрати файл

Видимість

Публічна — всі можуть знайти

Створити групу

Скасувати

Рис. Д.3 Модуль управління групами: Організатор створює групу

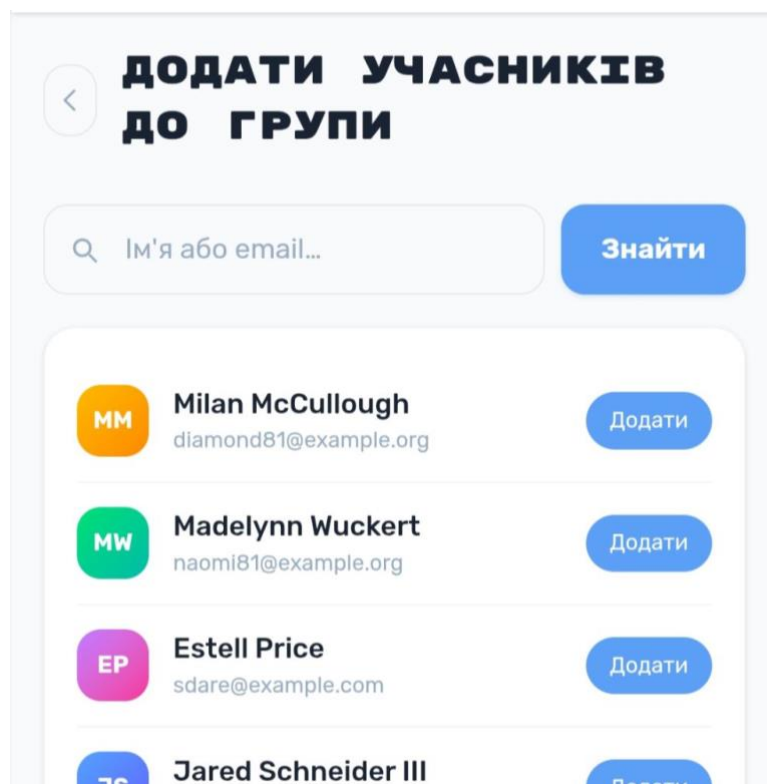


Рис. Д.4 Модуль управління групами: Організатор додає учасника зі списку користувачів

РЕДАГУВАННЯ ГРУПИ

CHES

<

РЕДАГУВАТИ

Зміни збережуться одразу після натискання «Зберегти».

Назва групи

Chess

Короткий опис

B

/

S

H1

H2

H3

=

1=

"

</>

↶

↷

chess

I

Категорії (оберіть одну або кілька)

#Volleyball

#Basketball

#Tennis

#Hiking

#Coffee Meetup

#Board Games

#Book Club

#Pub Quiz

#Coding

#Photography

#Language Exchange

#Yoga

Видимість

🌐

Публічна — всі можуть знайти

▼

Зберегти

Скасувати

Рис. Д.5 Модуль управління групами: Організатор редагує назву, опис та теги групи

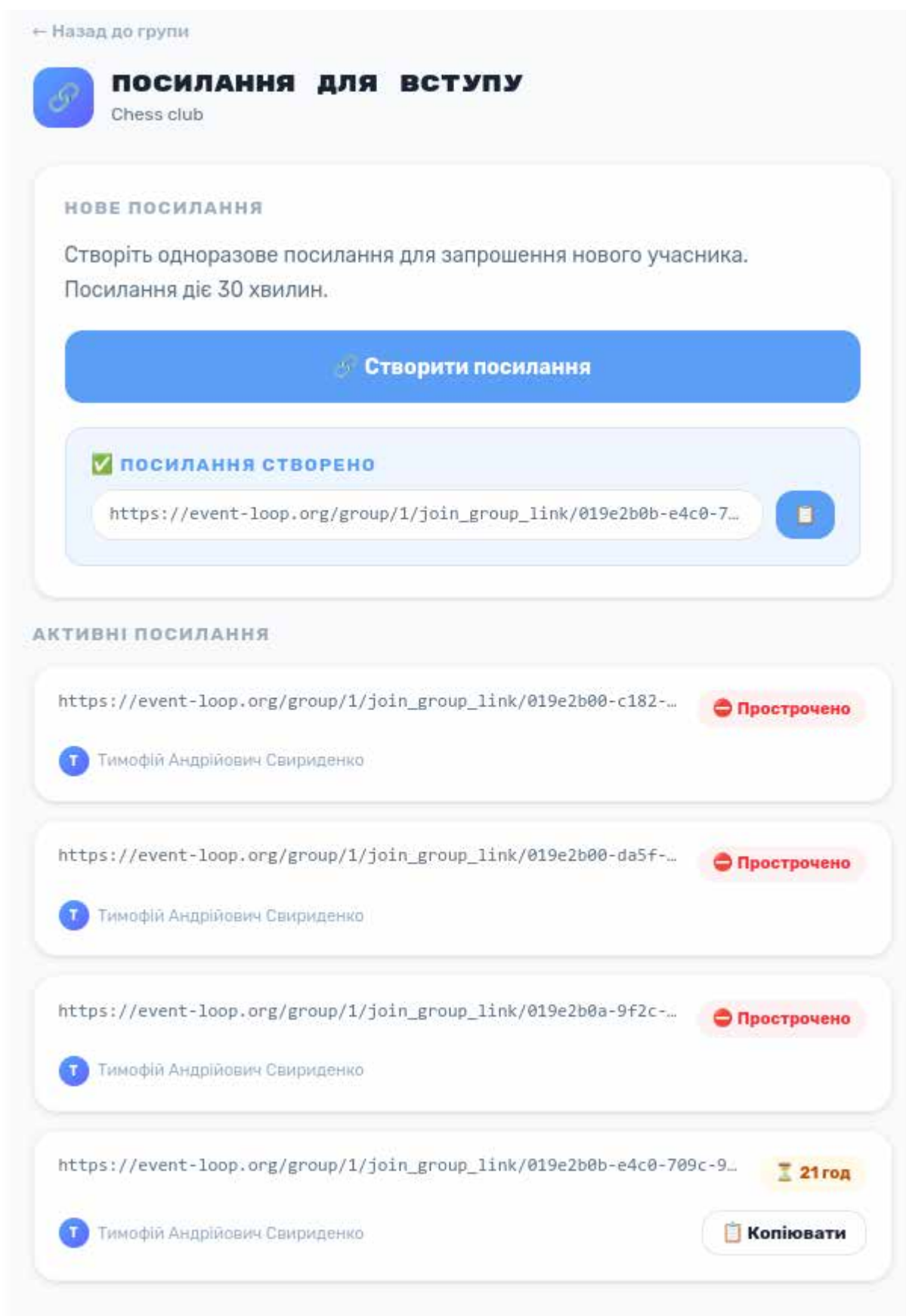


Рис. Д.6 Модуль управління групами: Організатор створює та ділиться посиланням для автоматичного вступу до групи



Рис. Д.7 Модуль управління групами: Сторінка групи очима учасника

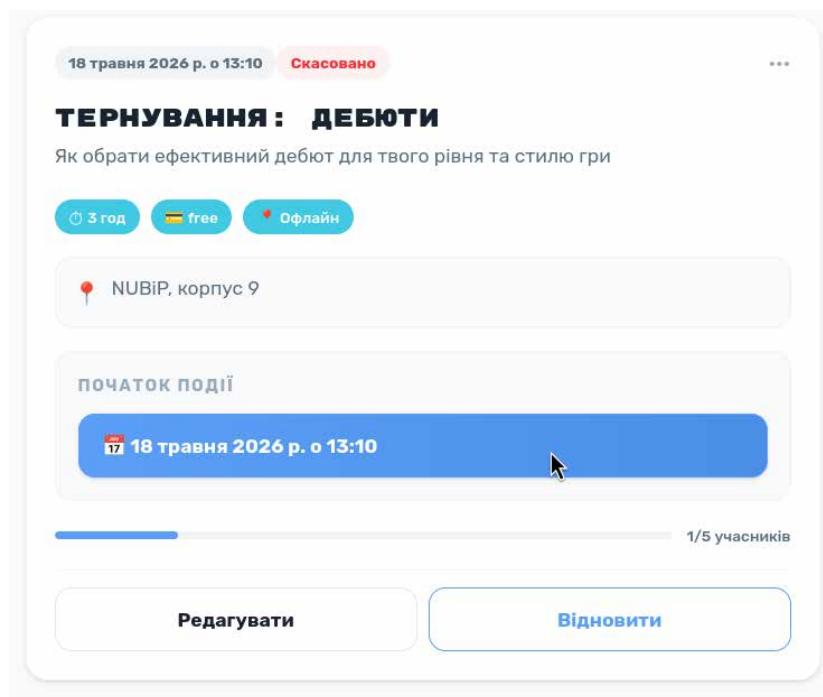


Рис. Д.8 Модуль управління подіями: Відображення скасованої організатором події

НОВА ПОДІЯ

Назва

Тренування еншпілів

Опис події

В

/

S

H1

H2

H3

≡

1≡

"

</>

↶

↷

Пройдемося по типовим помилкам гравців в еншпілі

Час проведення

Фіксований час

Опитування

Час проведення

05 / 18 / 2026 , 01:30 PM

Дедлайн голосування

05 / 17 / 2026 , 12:00 PM

Тривалість (хвилини)

120

Вартість

Безкоштовно

Макс. учасників

10

Формат

Офлайн

Онлайн

Адреса проведення

м. Київ, НУБіП, 15 корпус

Створити подію

Рис. Д.9 Модуль управління подіями: Організатор створює та планує подію

**ДОДАТОК М**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ**

**І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ**

**Факультет інформаційних технологій**

**Декларація про академічну доброчесність та використання ШІ**

Я, Свириденко Тимофій Андрійович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Програмне забезпечення вебзастосунку для планування та керування зустрічами» є результатом моєї особистої праці.
2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.
3. Щодо використання інструментів ШІ зазначаю, що (обрати необхідне):
  - ☐ інструменти генеративного ШІ не використовувалися.
  - ☒ інструменти ШІ Gemini, Claude були використані для:
    - ☐ перекладу іншомовних джерел;
    - ☒ стилістичного редагування та перевірки граматики;
    - ☒ допомоги у написанні/відлагодженні програмного коду✓ ;
    - ☐ інше: \_\_\_\_\_.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України.

«\_\_» \_\_\_\_\_ 2026 р. \_\_\_\_\_ Тимофій СВИРИДЕНКО