

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

**«Вебплатформа для організації волонтерської
діяльності»**

Виконала студентка групи ІПЗ-22-4
ЦАРУК Катерина Андріївна

Керівник роботи:
ЧИЖМОТРИЯ Олексій Володимирович

Рецензент:
САВЦЬКИЙ Роман Святославович

Житомир – 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення

Тетяна Вакалюк

«28» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЦАРУК Катерина Андріївна**

Керівник роботи: **ЧИЖМОТРЕ Олексій Володимирович**

Тема роботи: **«Вебплатформа для організації волонтерської діяльності»**,
затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., № 92/с**

Вихідні дані для роботи: об'єктом дослідження є процес організації волонтерської діяльності та координації взаємодії між волонтерами й організаторами у вебсередовищі; предметом дослідження є методи та інструментальні засоби розробки вебплатформи для реєстрації волонтерів, публікації заходів та управління участю у волонтерських проєктах.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	01.03.2026р	01.03.2026р
2	Чижмотря О.В.	01.03.2026р	01.03.2026р
3	Чижмотря О.В.	01.03.2026р	01.03.2026р

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи бакалавра виконана на тему «Вебплатформа для організації волонтерської діяльності». Випускна кваліфікаційна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел із 37 найменувань (3 стор.), 2 додатки (32 стор.), 23 таблиць та 45 ілюстрацій по тексту. Повний обсяг роботи становить 126 сторінок, з них 82 сторінки основного тексту.

Мета роботи — розробка вебплатформи для ефективної організації волонтерської діяльності, що забезпечує взаємодію між волонтерами, організаторами та адміністраторами.

У роботі проаналізовано предметну галузь організації волонтерської діяльності та розглянуто існуючі програмні рішення — Volunteer Country, Better Impact та GivePulse, виявлено їх функціональні обмеження. Обґрунтовано вибір клієнт-серверної архітектури з REST API та технологічного стеку на основі Node.js, Express.js, PostgreSQL, React і Docker. Спроектовано реляційну базу даних та побудовано UML-діаграми системи. Реалізовано повний функціонал вебплатформи: автентифікацію з підтримкою Google OAuth, управління заходами та завданнями, систему відгуків і рейтингів, збір донатів через Stripe, генерацію PDF-сертифікатів та систему гейміфікації на основі досвіду і рівнів волонтера. Проведено комплексне тестування з використанням Jest, Supertest та Postman.

Результати роботи можуть бути використані волонтерськими організаціями для автоматизації процесів координації діяльності волонтерів.

Ключові слова: ВЕБПЛАТФОРМА, ВОЛОНТЕРСЬКА ДІЯЛЬНІСТЬ, ГЕЙМІФІКАЦІЯ, REST API, ТЕСТУВАННЯ.

					ІІЗ.КР.Б – 121 – 26 – ІІЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Вебплатформа для організації волонтерської діяльності	Літ.	Арк.	Аркушів
Розроб.		Царук К.А.						
Керівник		Чижмотря О.В.					3	126
Рецензент		Савіцький Р.С.				Житомирська політехніка, група ІІЗ-22-4		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The explanatory note to the bachelor's qualification work is written on the topic "Web Platform for Organizing Volunteer Activities". The graduation qualification work consists of an introduction, 3 sections, conclusions, a list of references with 37 titles (3 p.), 2 appendices (32 p.), 23 tables and 45 figures in the text. The total volume of the work is 126 pages, of which 82 pages are the main text.

The purpose of the work is to develop a web platform for effective organization of volunteer activities, ensuring interaction between volunteers, organizers and administrators.

The work analyzes the subject area of volunteer activity organization and examines existing software solutions — Volunteer Country, Better Impact and GivePulse, identifying their functional limitations. The choice of client-server architecture with REST API and a technology stack based on Node.js, Express.js, PostgreSQL, React and Docker is justified. A relational database is designed and UML diagrams of the system are constructed. The full functionality of the web platform is implemented: authentication with Google OAuth support, event and task management, a review and rating system, donation collection via Stripe, PDF certificate generation and a gamification system based on volunteer experience and levels. Comprehensive testing was carried out using Jest, Supertest and Postman.

The results of the work can be used by volunteer organizations to automate the processes of coordinating volunteer activities.

Keywords: WEB PLATFORM, VOLUNTEER ACTIVITIES, GAMIFICATION, REST API, TESTING.

		Царук К.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижемотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	7
ВСТУП.....	8
РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ.....	10
1.1. Постановка задачі.....	10
1.2. Аналіз аналогів програмного продукту.....	12
1.3. Вибір архітектури системи.....	16
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	20
Висновки до першого розділу.....	23
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ ВОЛОНТЕРСЬКОЇ ДІЯЛЬНОСТІ.....	25
2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи.....	25
2.2. Розробка бази даних системи.....	31
2.3. Проєктування та реалізація алгоритмів роботи системи.....	39
2.4. Реалізація функціоналу вебплатформи для організації волонтерської діяльності.....	45
2.5. Модульна структура та програмний інтерфейс системи.....	51
Висновки до другого розділу.....	55
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ ДЛЯ ОРГАНІЗАЦІЇ ВОЛОНТЕРСЬКОЇ ДІЯЛЬНОСТІ.....	56
3.1. Порядок встановлення та налаштування параметрів платформи.....	56
3.2. Структура інтерфейсу вебплатформи.....	61
3.3. Статистика та гейміфікація користувачів.....	77
3.4. Тестування вебплатформи.....	81
Висновки до третього розділу.....	86
ВИСНОВКИ.....	88

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		5

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 90
ДОДАТКИ..... 93

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API — прикладний програмний інтерфейс

БД — база даних

ПЗ — програмне забезпечення

СУБД — система управління базами даних

SPA — односторінковий вебзастосунок (Single Page Application)

MVC — шаблон проєктування, що розділяє застосунок на три компоненти: модель, представлення та контролер (Model-View-Controller)

REST — архітектурний стиль взаємодії компонентів розподіленого застосунку (Representational State Transfer)

CRUD — базові операції управління даними: створення, читання, оновлення, видалення (Create, Read, Update, Delete)

HTTP / HTTPS — протокол передавання гіпертексту

JSON — текстовий формат обміну даними (JavaScript Object Notation)

JWT — відкритий стандарт для створення токенів доступу (JSON Web Token)

OAuth — відкритий протокол авторизації

XP — система балів досвіду користувача (Experience Points)

CI/CD — практика безперервної інтеграції та безперервного розгортання (Continuous Integration / Continuous Deployment)

UUID — універсальний унікальний ідентифікатор (Universally Unique Identifier)

CORS — механізм обміну ресурсами між різними джерелами (Cross-Origin Resource Sharing)

SQL — мова структурованих запитів (Structured Query Language)

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		7

ВСТУП

Актуальність теми. Волонтерська діяльність в Україні суттєво активізувалася після 2022 року та охопила широкий спектр напрямів — від гуманітарної допомоги до відновлення інфраструктури [3]. Зростання кількості ініціатив і учасників призвело до ускладнення координаційних процесів: організатори змушені паралельно вести облік волонтерів, розподіляти завдання й відстежувати їх виконання через розрізнені канали зв'язку — месенджери, електронну пошту, таблиці. Це збільшує операційне навантаження та знижує загальну ефективність роботи.

Наявні вебрішення у сфері волонтерства здебільшого орієнтовані на конкретну організацію або вузький функціонал і не забезпечують повноцінного циклу — від публікації заходу до підтвердження участі й звітності [11]. Відсутність універсального інструменту, придатного для різних типів волонтерських об'єднань, зумовлює актуальність розробки спеціалізованої вебплатформи.

Метою кваліфікаційної роботи є проєктування та розробка вебплатформи для організації волонтерської діяльності, що забезпечує реєстрацію учасників, публікацію заходів і керування участю у волонтерських проєктах. У межах роботи досліджено підходи до організації волонтерської діяльності, проаналізовано існуючі аналоги та визначено вимоги до розроблюваної вебплатформи.

Об'єктом дослідження є процес організації та координації волонтерської діяльності у вебсередовищі.

Предметом дослідження є методи та інструментальні засоби розробки вебплатформи для реєстрації волонтерів, публікації заходів і управління участю у волонтерських проєктах.

Для досягнення поставленої мети застосовувалися методи системного аналізу для формування вимог до вебплатформи, принципи об'єктно-орієнтованого проєктування для побудови архітектури системи, а також

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		8

методи візуального моделювання для проєктування бази даних і логіки роботи вебплатформи.

Завданнями роботи є аналіз предметної області волонтерської діяльності та огляд існуючих аналогів, виявлення їх переваг і недоліків, проєктування архітектури системи та схеми бази даних, розробка програмної реалізації основних модулів вебплатформи, налаштування механізмів автентифікації та розмежування ролей користувачів, розгортання вебплатформи та тестування працездатності системи.

Методика роботи передбачала поділ вебплатформи на фронтенд і бекенд із чітким визначенням зон відповідальності кожної частини. Для зберігання даних спроектовано реляційну модель бази даних. Архітектура вебплатформи побудована на перевірених шаблонах проєктування, що забезпечує масштабованість і зручність подальшого супроводу [15]. Тестування охоплює перевірку окремих функцій і повних сценаріїв взаємодії користувача з системою. Розгортання виконується за допомогою контейнеризації, що спрощує перенесення вебплатформи між середовищами.

Практичне значення отриманих результатів. Розроблена вебплатформа може бути використана волонтерськими організаціями, громадськими об'єднаннями та благодійними фондами для автоматизації процесів координації волонтерської діяльності. Платформа забезпечує повний цикл взаємодії між учасниками — від публікації заявки до підтвердженого завершення — та може бути розширена додатковими модулями відповідно до потреб конкретної організації.

Апробація. За темою кваліфікаційної роботи опубліковано 2 тези конференцій [1, 2].

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижемотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		9

РОЗДІЛ 1. АНАЛІЗ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

1.1. Постановка задачі

Волонтерська діяльність в Україні охоплює широкий спектр напрямів — гуманітарну допомогу, медичне забезпечення, відновлення інфраструктури, соціальну підтримку вразливих верств населення та реагування на надзвичайні ситуації. Починаючи з 2022 року масштаби волонтерського руху зросли до безпрецедентного рівня, що суттєво ускладнило процеси координації між учасниками [3], [11]. Організатори заходів змушені одночасно вести облік волонтерів, публікувати оголошення в різних каналах, обробляти заявки вручну та контролювати виконання завдань без єдиного інструменту управління. Таке роздроблення процесів призводить до дублювання інформації, втрати заявок і підвищеного навантаження на координаторів [5, 8].

Аналіз існуючого стану організації волонтерської діяльності виявив кілька системних проблем. Відсутність централізованого простору для публікації заходів змушує волонтерів самостійно збирати інформацію з різних джерел — соціальних мереж, месенджерів та сайтів окремих організацій. При цьому організатори не мають зручних засобів для обліку поданих заявок і контролю складу учасників, що нерідко призводить до перевищення встановлених лімітів або, навпаки, до недобору людей на заплановані заходи. Відсутність системи репутації та відгуків знижує мотивацію до активної участі, оскільки волонтери не мають можливості фіксувати свій досвід і внесок у загальну справу.

На основі аналізу предметної галузі визначено три категорії кінцевих користувачів вебплатформи [10]. Перша категорія — волонтер, тобто особа, готова витратити власний час на допомогу іншим. Для цієї категорії критично важливими є зручний пошук заходів із фільтрацією за напрямом, містом і датою, можливість подачі заявки та відстеження її статусу, а також доступ до власної історії участі. Друга категорія — організатор, представник громадської організації або ініціативної групи, якому необхідні інструменти

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		10

публікації заходів, управління заявками та контроль складу учасників. Третя категорія — адміністратор платформи, який здійснює загальний нагляд за функціонуванням системи, управляє обліковими записами та вирішує спірні питання.

Для вирішення виявлених проблем необхідно розробити вебплатформу, що реалізує повний цикл організації волонтерської діяльності в межах єдиного середовища. Платформа має забезпечувати реєстрацію та автентифікацію користувачів із розмежуванням ролей, повний набір операцій для управління заходами — створення, редагування, перегляд і видалення — з підтримкою категоризації та пошуку. Механізм подачі заявок на участь повинен включати контроль ліміту учасників і можливість для організатора підтверджувати або відхиляти кандидатури. Окрім цього, система має підтримувати управління завданнями всередині заходу, систему відгуків і рейтингів, а також сповіщення для інформування учасників про зміни статусу.

Поряд із функціональними вимогами до вебплатформи висуваються й нефункціональні. З точки зору безпеки паролі користувачів мають зберігатися у хешованому вигляді, а доступ до захищених ресурсів — перевірятися через токени автентифікації. Продуктивність системи повинна забезпечувати час відповіді сервера не більше однієї секунди при типовому навантаженні. Інтерфейс має бути адаптивним і коректно відображатися в сучасних браузерях без встановлення додаткового програмного забезпечення. Архітектура системи повинна допускати подальше розширення функціоналу без необхідності переписування існуючих модулів.

Окремої уваги заслуговує вимога до локалізації та доступності платформи: інтерфейс має бути виконано українською мовою, що забезпечить зручність використання. Система сповіщень повинна своєчасно інформувати волонтерів та організаторів про зміну статусів заявок, наближення дати заходу та інші події, що потребують реакції учасника. Наявність механізму підтвердження участі та фіксації факту виконаного завдання є необхідною умовою для подальшого формування рейтингів і видачі сертифікатів.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		11

Таким чином, головним завданням кваліфікаційної роботи є розробка вебплатформи, що усуне наявні прогалини в організації волонтерської діяльності та забезпечить учасникам і координаторам зручне, надійне середовище для взаємодії в межах єдиного цифрового простору.

1.2. Аналіз аналогів програмного продукту

На сучасному етапі розвитку інформаційних технологій цифрові рішення для координації волонтерської діяльності представлені достатньо широко, однак більшість із них орієнтована або на вузькоспеціалізовані завдання, або на великі організації з розвиненою адміністративною структурою. Для обґрунтування доцільності розробки власної вебплатформи в межах кваліфікаційної роботи було проведено аналіз трьох існуючих програмних рішень: Volunteer Country, Better Impact та GivePulse.

Volunteer Country (<https://volunteer.country/>) є українською вебплатформою, орієнтованою на пошук і залучення волонтерів до соціальних, гуманітарних та громадських ініціатив. Організації можуть публікувати волонтерські пропозиції, а зареєстровані користувачі — переглядати актуальні проєкти й подавати заявки на участь. Інтерфейс платформи виконано у мінімалістичному стилі, що забезпечує відносно низький поріг входу для нових користувачів і зручне ознайомлення з доступними ініціативами (рис. 1.1).

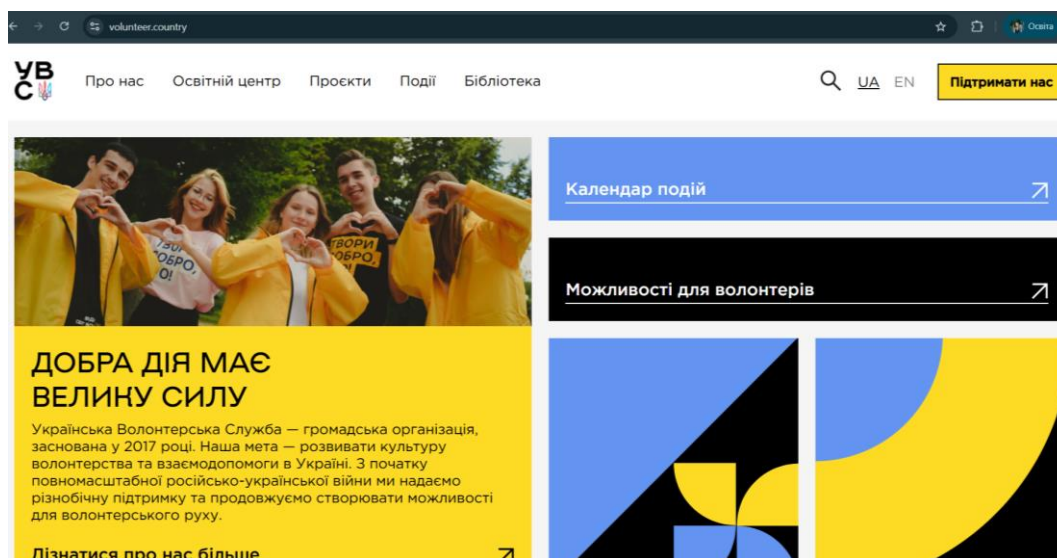


Рис. 1.1. Головна сторінка платформи Volunteer Country

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		12

Разом із тим функціональні можливості Volunteer Country зосереджені переважно на інформаційному рівні: платформа слугує дошкою оголошень, але не надає інструментів для управління завданнями всередині заходу, відстеження статусу заявок, обліку волонтерської активності чи формування звітності. Координатор не має змоги контролювати хід виконання роботи в межах платформи, а вся подальша комунікація переноситься у зовнішні канали. Це суттєво обмежує застосування платформи як повноцінного засобу організації діяльності.

Better Impact (<https://www.betterimpact.com/>) є комплексним програмним рішенням, орієнтованим переважно на організації, що системно працюють із великою кількістю волонтерів. Платформа надає широкі можливості для управління розкладами, обліку відпрацьованих годин, зберігання профілів волонтерів та автоматизованого формування звітності. Значна увага приділяється адміністративному аспекту: керівник організації отримує детальну картину активності кожного учасника, може налаштовувати ролі та встановлювати доступи до різних модулів системи (рис. 1.2).

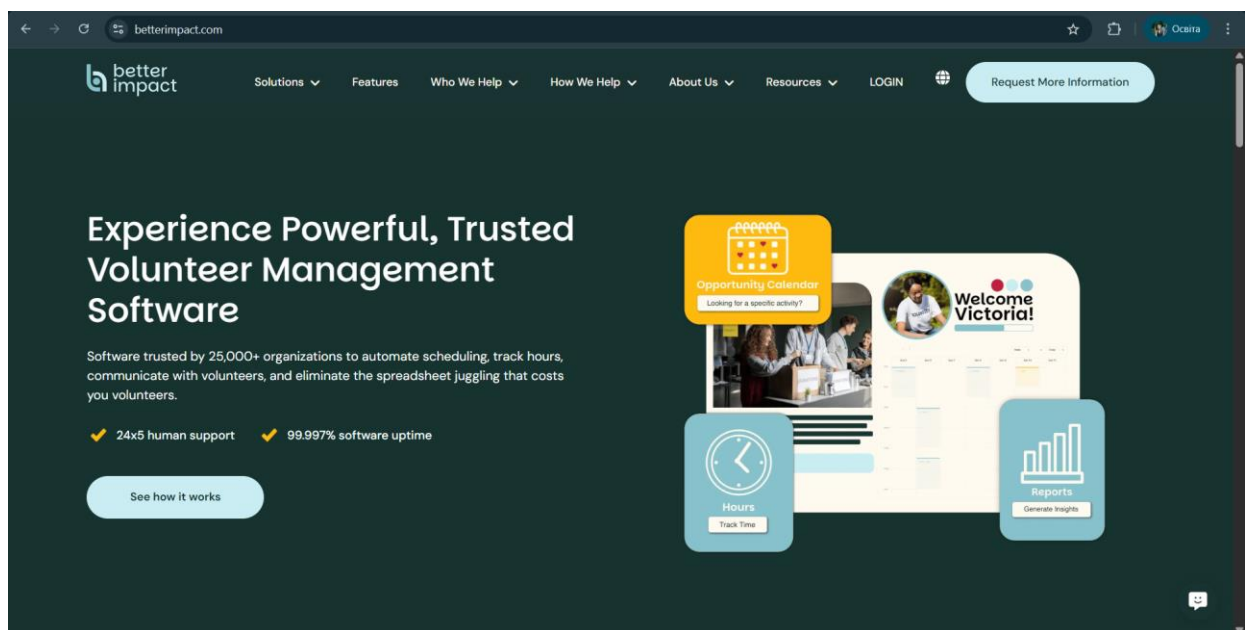


Рис. 1.2. Головна сторінка платформи Better Impact

Незважаючи на функціональну повноту, Better Impact характеризується складним інтерфейсом і тривалим процесом налаштування, що підвищує поріг входу та робить платформу малопридатною для невеликих ініціатив або

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		13

неформальних груп. Орієнтація на корпоративний сегмент означає, що рядовий волонтер виступає переважно об'єктом обліку, а не рівноправним учасником із власним простором для взаємодії. Соціальна складова та інструменти прямої комунікації між учасниками фактично відсутні.

GivePulse (<https://learn.givepulse.com/>) поєднує функції управління волонтерською діяльністю та елементи соціальної взаємодії. Платформа підтримує організацію заходів, реєстрацію учасників, облік активності та збір статистичних даних. Серед її переваг — інтеграція з навчальними закладами і громадськими організаціями, що зробило її популярною у сфері освітнього волонтерства. Інтерфейс є більш доступним порівняно з Better Impact і надає волонтеру власну сторінку активності (рис. 1.3).

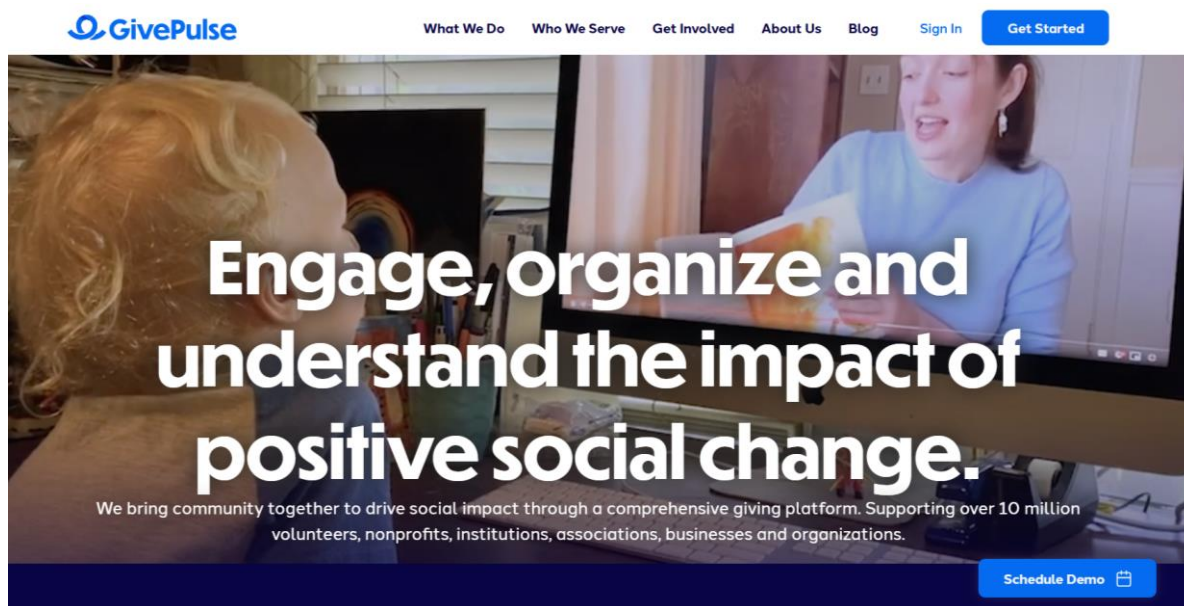


Рис. 1.3. Головна сторінка платформи GivePulse

Водночас інструменти GivePulse для детальної координації завдань і оперативної комунікації між учасниками реалізовані обмежено. Платформа добре справляється зі збором статистики, однак не забезпечує повноцінного управління процесом усередині конкретного заходу: розподілу завдань, контролю їх виконання та структурованого зворотного зв'язку між організатором і волонтерами.

На основі проведеного аналізу сформовано порівняльну таблицю основних функціональних характеристик розглянутих платформ (табл. 1.1).

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		14

Таблиця 1.1

Порівняльна характеристика платформ для організації волонтерської діяльності

Характеристика	Поруч	Better Impact	GivePulse	Volunteer Country
Управління волонтерами	+	+	+	±
Облік волонтерських годин	+	+	+	—
Управління завданнями	+	+	—	—
Централізована комунікація	+	—	±	+
Управління ресурсами	+	+	—	—
Формування звітності	+	+	+	—
Орієнтація на малі ініціативи	+	—	+	+
Простота використання	+	—	+	+
Соціальна взаємодія	+	—	+	+

Аналіз розглянутих платформ свідчить про те, що жодне з існуючих рішень не забезпечує повного набору функцій, необхідних для зручної організації волонтерської діяльності широким колом учасників. Volunteer Country зручна у використанні, але залишається лише інформаційним агрегатором. Better Impact охоплює більшість адміністративних потреб, однак складна у налаштуванні та недоступна для малих груп. GivePulse наближається до балансу між функціональністю та простотою, але має обмеження в інструментах координації та не підтримує українську мову.

Виявлені прогалини обґрунтовують доцільність розробки власної вебплатформи, яка поєднає зручний інтерфейс, повний цикл управління заходами та орієнтацію на потреби як волонтерів, так і організаторів. На відміну від розглянутих аналогів, розроблювана платформа враховує специфіку українського волонтерського середовища, підтримує українську мову інтерфейсу та включає елементи гейміфікації, що підвищують залученість учасників. Це дозволяє позиціонувати її як рішення, адаптоване до реальних потреб вітчизняних волонтерських організацій різного масштабу.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		15

1.3. Вибір архітектури системи

Вибір архітектурного рішення є одним із ключових етапів проектування, оскільки визначає структуру системи, спосіб взаємодії між її компонентами та можливості подальшого масштабування. Для розроблюваної вебплатформи було розглянуто три основні підходи: монолітну архітектуру, мікросервісну архітектуру та клієнт-серверну архітектуру з REST API.

Монолітна архітектура передбачає розробку всіх компонентів системи у вигляді єдиного розгорнутого артефакту, де інтерфейс користувача, серверна логіка та робота з базою даних зосереджені в одному застосунку. Перевагою цього підходу є простота початкової розробки та налагодження, відсутність мережових затримок між компонентами та зручне розгортання на одному сервері. Однак із зростанням обсягу функціоналу монолітна система стає важкою для супроводу: будь-яка зміна вимагає перезбирання всього проєкту, а масштабування можливе лише шляхом вертикального нарощування потужності сервера, що є неефективним при збільшенні навантаження.

Мікросервісна архітектура розбиває систему на набір незалежних сервісів, кожен із яких відповідає за певну бізнес-функцію та взаємодіє з іншими через добре визначений API. Такий підхід забезпечує гнучке горизонтальне масштабування окремих компонентів і їх незалежне розгортання. Разом із тим він суттєво ускладнює інфраструктуру: виникають проблеми розподіленої транзакційності, налаштування міжсервісної мережі та моніторингу, що вимагає зрілих DevOps-практик. Для проєкту з командою розробки з однієї особи така складність є невиправданою на поточному етапі.

Клієнт-серверна архітектура з REST API забезпечує чіткий поділ відповідальності між клієнтською та серверною частинами [14]. Сервер відповідає за бізнес-логіку, автентифікацію та взаємодію з базою даних, тоді як клієнтський застосунок у браузері — за відображення даних та взаємодію з користувачем. Між компонентами встановлено контракт у вигляді REST-інтерфейсу, що дозволяє паралельно розробляти фронтенд і бекенд, підключати різних клієнтів до одного API та ізольовано тестувати кожний

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		16

компонент. Відсутність стану на сервері між запитами спрощує горизонтальне масштабування та підвищує передбачуваність поведінки системи.

Порівняльну характеристику розглянутих архітектурних підходів зведено у табл. 1.2.

Таблиця 1.2

Порівняльна характеристика архітектурних підходів

Критерій	Моноліт	Мікросервіси	Клієнт-сервер з REST API
Складність початкової розробки	Низька	Висока	Середня
Складність масштабування	Висока	Низька	Середня
Зручність тестування	Середня	Висока	Висока
Незалежність компонентів	Низька	Висока	Висока
Вимоги до інфраструктури	Мінімальні	Значні	Помірні
Придатність для малих команд	Так	Ні	Так

Саме клієнт-серверний підхід було обрано для реалізації вебплатформи як оптимальний з огляду на вимоги до продукту та складність реалізації.

Принципи REST та їх застосування у проєкті. Representational State Transfer (REST) — архітектурний стиль, сформульований Роем Філдінгом, що визначає шість обмежень для побудови масштабованих вебслужб [14]. В розроблюваній вебплатформі ці принципи реалізовано таким чином:

єдиний інтерфейс — усі ресурси (заходи, користувачі, завдання) ідентифікуються уніфікованими URI, а операції над ними виражаються стандартними HTTP-методами GET, POST, PUT, PATCH та DELETE;

відсутність стану — кожен запит містить усю необхідну інформацію для його обробки; стан сесії не зберігається на сервері, а передається у вигляді JWT-токена у заголовку Authorization;

кешування — відповіді API позначаються відповідними заголовками Cache-Control, що дозволяє браузеру та проміжним проксі-серверам кешувати незмінні ресурси;

багаторівнева система — між клієнтом і сервером можна прозоро розміщувати проміжні шари (балансувальники навантаження, Nginx), не змінюючи логіку взаємодії.

Патерн MVC на серверній частині. Серверна частина побудована на основі патерну MVC (Model–View–Controller), адаптованого для REST-бекенду. У цьому контексті маршрутизатори (Router) виконують роль диспетчерів, що направляють вхідний HTTP-запит до відповідного контролера. Контролери (Controller) реалізують бізнес-логіку: перевіряють права доступу, викликають функції роботи з базою даних і формують відповідь. Модель (Model) представлена у вигляді окремого шару SQL-запитів до PostgreSQL. Оскільки в REST API відповіддю є JSON-об'єкт, а не HTML-сторінка, роль подання (View) виконує клієнтська частина на React. Такий поділ дозволяє незалежно змінювати логіку кожного шару, не зачіпаючи інших.

Односторінковий застосунок (SPA) на клієнтській частині. Клієнтська частина реалізована як Single Page Application на базі React 18. На відміну від традиційних багатосторінкових застосунків, де кожен перехід між розділами ініціює повне завантаження HTML-сторінки з сервера, SPA завантажує увесь інтерфейс одноразово, а подальша навігація відбувається шляхом часткового оновлення вмісту через API-запити. Це забезпечує плавний користувацький досвід, зменшує кількість мережових запитів і дозволяє реалізувати офлайн-поведінку для окремих функцій. React Router DOM 7 забезпечує декларативну маршрутизацію: кожен URL відображається на відповідний компонент без перезавантаження сторінки. Механізм Virtual DOM, реалізований у React, мінімізує кількість операцій з реальним деревом документа: бібліотека обчислює мінімальний набір змін між попереднім і новим станом та застосовує лише їх, що підвищує продуктивність при частих оновленнях інтерфейсу.

Саме цей підхід було обрано для реалізації вебплатформи як оптимальний з огляду на вимоги до продукту та складність реалізації. Серверна частина побудована на основі патерну MVC: маршрутизатори

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		18

виконують роль диспетчерів запитів, контролери реалізують бізнес-логіку, а PostgreSQL відповідає за збереження даних. Клієнтська частина реалізована як односторінковий застосунок (SPA) на базі React, що взаємодіє з бекендом виключно через REST API за допомогою HTTP-запитів. Схему обраної архітектури наведено на рис. 1.4.

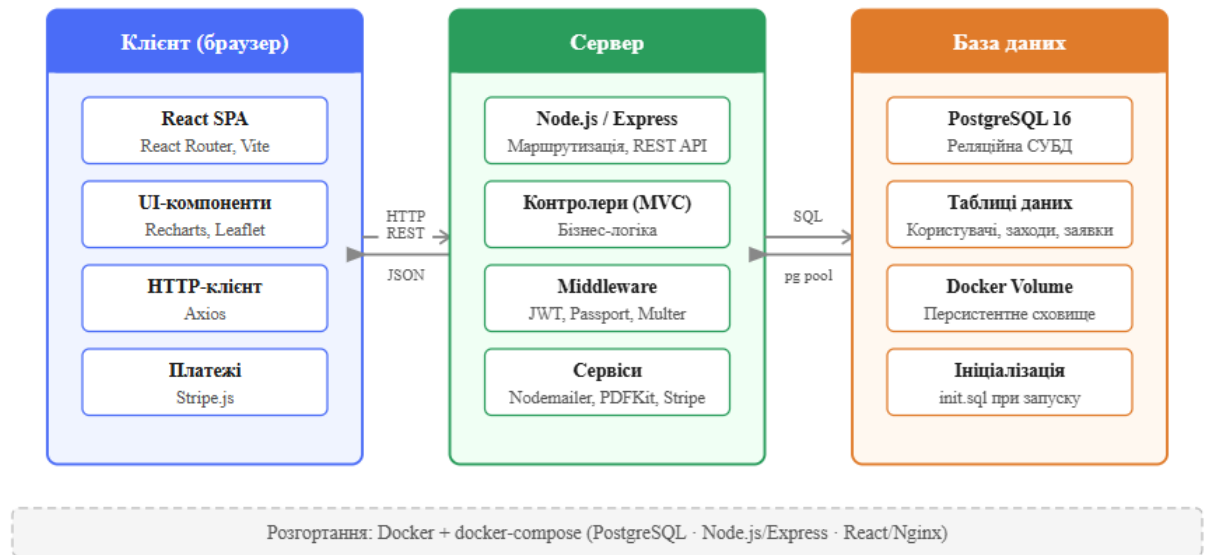


Рис. 1.4. Схема клієнт-серверної архітектури вебплатформи

Серверна частина побудована на платформі Node.js з використанням фреймворку Express, що забезпечує обробку HTTP-запитів, маршрутизацію та роботу з проміжним програмним забезпеченням. Автентифікація реалізована через JWT-токени та підтримує вхід через обліковий запис Google за протоколом OAuth 2.0. Для роботи з базою даних використовується бібліотека pg із пулом з'єднань до PostgreSQL. Завантаження файлів обробляється через Multer, генерація PDF-документів — через PDFKit, а відправка електронних листів — через Nodemailer. Платіжна функціональність інтегрована через Stripe [21].

Клієнтська частина зібрана за допомогою Vite та використовує React Router для навігації між сторінками без перезавантаження. Для відображення географічних даних підключено бібліотеку Leaflet, для побудови діаграм — Recharts. HTTP-запити до API виконуються через Axios.

Структура модулів системи. Серверна частина розбита на функціональні модулі відповідно до сутностей предметної галузі: модуль автентифікації, модуль управління заходами, модуль заявок і учасників, модуль завдань, модуль відгуків і рейтингів, модуль донатів та платежів, модуль сертифікатів і гейміфікації, модуль адміністрування. Кожен модуль складається з файлу маршрутів, контролера та, за потреби, окремого файлу SQL-запитів. Така структура забезпечує слабку зв'язність між модулями: зміни в одному з них не потребують модифікації інших.

Розгортання всіх компонентів системи — бази даних PostgreSQL, бекенду та фронтенду — здійснюється за допомогою Docker і docker-compose, що забезпечує однакове середовище виконання незалежно від платформи та спрощує процес запуску проєкту. Фронтенд роздається через Nginx, що підвищує швидкість завантаження статичних ресурсів.

Обрана архітектура є загальновизнаним стандартом для сучасних вебплатформ середнього масштабу та повністю відповідає функціональним і нефункціональним вимогам до розроблюваного продукту. У разі зростання навантаження система може бути масштабована горизонтально без змін у клієнтській частині та без переробки базової архітектури.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Вибір технологічного стеку базувався на відповідності функціональним вимогам до вебплатформи, зрілості та стабільності технологій, наявності якісної документації та можливості ефективно розробки силами одного розробника [6].

Серверна частина реалізована на платформі Node.js із використанням фреймворку Express.js [7, 17]. Node.js реалізує асинхронну обробку запитів на основі подієво-орієнтованої однопотокової моделі з неблокуючим введенням-виведенням, що дозволяє ефективно обробляти велику кількість паралельних з'єднань за умов інтенсивного звернення до бази даних. Express.js є мінімалістичним інструментом для побудови REST API: він надає

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		20

маршрутизацію запитів, механізм middleware-ланцюжків та обробку HTTP-запитів, не нав'язуючи при цьому жорсткої структури проєкту [18]. Це забезпечує повний контроль над архітектурою та дозволяє організувати код відповідно до потреб конкретного проєкту.

Для збереження даних обрано реляційну СУБД PostgreSQL 16 [19]. Ця система відрізняється суворим дотриманням стандартів SQL, надійними механізмами підтримки ACID-транзакцій, підтримкою перелічуваних типів, складних обмежень цілісності, UUID-ідентифікаторів та типів JSONB. Взаємодія з базою даних реалізована через бібліотеку pg із пулом з'єднань, що підвищує продуктивність при паралельних запитах за рахунок повторного використання відкритих з'єднань. Усі запити формуються у вигляді параметризованих SQL-рядків, що виключає SQL-ін'єкції на рівні архітектури [32].

Автентифікація реалізована за допомогою JSON Web Tokens — стандарту RFC 7519. JWT є компактим підписаним об'єктом, що підтверджує автентичність без збереження стану сесії на сервері, спрощуючи горизонтальне масштабування [22]. Термін дії токена обмежено. Паролі зберігаються у вигляді хеш-значень, обчислених алгоритмом bcrypt, що забезпечує стійкість до брутфорс-атак навіть при компрометації бази даних [35]. Додатково реалізовано вхід через обліковий запис Google за протоколом OAuth 2.0 за допомогою бібліотеки Passport.js, що підвищує зручність реєстрації для користувачів [23].

Завантаження файлів на сервер обробляється через бібліотеку Multer. Для генерації PDF-документів використовується PDFKit, що дозволяє формувати звіти та підтвердження безпосередньо на сервері. Відправка електронних листів реалізована через Nodemailer. Платіжна функціональність інтегрована через Stripe — провідний сервіс обробки онлайн-платежів із розвиненим SDK та надійним захистом транзакцій.

Клієнтська частина побудована на бібліотеці React 18 [16]. Компонентний підхід дозволяє описати кожен елемент інтерфейсу

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		21

ізолюваним компонентом із власним станом, що спрощує повторне використання та тестування. Механізм Virtual DOM мінімізує кількість реальних операцій з деревом документа, забезпечуючи плавну роботу інтерфейсу без повного перезавантаження сторінок. Для збирання застосунку використовується Vite 4 — інструмент із майже миттєвим запуском сервера розробки завдяки нативній підтримці ES-модулів і гарячою заміною модулів [36]. Маршрутизацію реалізовано через React Router DOM 7, HTTP-запити до API — через Axios із централізованим перехоплювачем, що автоматично додає JWT-токен до заголовка кожного запиту. Для відображення географічних даних підключено Leaflet, для побудови діаграм і статистики — Recharts [28], для роботи з платежами на стороні клієнта — Stripe.js [21].

Розгортання всіх компонентів системи здійснюється за допомогою Docker і docker-compose [20]. Кожен сервіс — PostgreSQL, Node.js/Express і React/Nginx — запускається у власному ізолюваному контейнері. Це усуває проблему розбіжності середовищ і гарантує однакову поведінку системи як при локальній розробці, так і на виробничому сервері. Фронтенд роздається через Nginx, що підвищує швидкість завантаження статичних ресурсів.

Технологічний стек системи зведено у табл. 1.3.

Таблиця 1.3

Технологічний стек вебплатформи

Компонент	Технологія / Бібліотека	Призначення
Серверне середовище	Node.js	Виконання серверного JavaScript
Серверний фреймворк	Express.js	Маршрутизація та middleware
СУБД	PostgreSQL	Збереження та управління даними
БД-адаптер	pg (node-postgres)	Підключення до PostgreSQL
Автентифікація	jsonwebtoken	Генерація та верифікація JWT
OAuth 2.0	Passport.js	Вхід через Google
Хешування паролів	bcryptjs	Захищене зберігання паролів

Завантаження файлів	Multer	Обробка multipart/form-data
Генерація PDF	PDFKit	Формування документів
Email-розсилка	Nodemailer	Відправка листів
Платежі (сервер)	Stripe SDK	Обробка транзакцій
UI-бібліотека	React	Компонентний інтерфейс
Збирач	Vite	Розробка та збирання SPA
Маршрутизація клієнта	React Router DOM	Навігація в SPA
HTTP-клієнт	Axios	Запити до REST API
Карти	Leaflet + React-Leaflet	Географічна візуалізація
Графіки	Recharts	Діаграми та статистика
Платежі (клієнт)	Stripe.js	Інтерфейс оплати
Контейнеризація	Docker + docker-compose	Розгортання системи
Вебсервер	Nginx	Роздача статички фронтенду

Вимоги до серверного апаратного забезпечення для розгортання платформи є помірними: процесор з тактовою частотою не менше 1 ГГц (рекомендовано двоядерний), не менше 1 ГБ оперативної пам'яті для одночасної роботи всіх контейнерів, не менше 5 ГБ вільного дискового простору, мережевий адаптер зі швидкістю з'єднання не менше 100 Мбіт/с. Як операційна система може використовуватися Ubuntu 20.04+ або будь-який інший Linux-дистрибутив із підтримкою Docker, а також Windows Server 2019+ із встановленим Docker Desktop. Для клієнтської сторони достатньо сучасного браузера — Chrome 90+, Firefox 90+, Safari 14+ або Edge 90+ — з підтримкою ES2020.

Висновки до першого розділу

У першому розділі проведено аналіз предметної області волонтерської діяльності, досліджено існуючі програмні рішення та обґрунтовано технологічні рішення для розроблюваної вебплатформи.

Аналіз поточного стану організації волонтерської діяльності виявив системні проблеми: відсутність централізованого простору для публікації

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		23

заходів, брак інструментів обліку заявок і контролю складу учасників, відсутність системи репутації та відгуків. На основі цього визначено три категорії користувачів — волонтер, організатор і адміністратор — та сформовано функціональні й нефункціональні вимоги до вебплатформи.

Огляд існуючих аналогів — Volunteer Country, Better Impact і GivePulse — показав, що жодна з платформ не забезпечує повного циклу організації волонтерської діяльності. Volunteer Country обмежується роллю інформаційної дошки оголошень, Better Impact орієнтований на великі організації та характеризується високим порогом входу, GivePulse має обмежені інструменти координації та не підтримує українську мову. Виявлені прогалини підтверджують доцільність розробки власної вебплатформи.

Для реалізації системи обрано клієнт-серверну архітектуру з REST API як оптимальну для проєкту: вона забезпечує чіткий поділ відповідальності між фронтендом і бекендом, зручне паралельне тестування компонентів та можливість горизонтального масштабування без зміни клієнтської частини. Серверна частина побудована на Node.js і Express.js із патерном MVC, клієнтська — на React 18 з Vite, база даних — PostgreSQL 16. Розгортання реалізовано через Docker і docker-compose, що гарантує однакову поведінку системи в усіх середовищах.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		24

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБПЛАТФОРМИ ДЛЯ ОРГАНІЗАЦІЇ ВОЛОНТЕРСЬКОЇ ДІЯЛЬНОСТІ

2.1. Визначення варіантів використання та об'єктно-орієнтованої структури системи

Визначення варіантів використання є першим кроком проєктування системи відповідно до методології об'єктно-орієнтованого аналізу та нотації UML 2.0 [12]. Діаграми розроблено у середовищі StarUML [34]. Варіанти використання описують сценарії взаємодії акторів із системою і є основою для специфікації функціональних вимог у формалізованому вигляді [13].

Акторами системи є чотири зовнішніх суб'єкти. Незареєстрований користувач (гість) має доступ лише до публічної частини: перегляду списку заходів, деталей окремого заходу та публічних профілів. Це дозволяє потенційним волонтерам ознайомитися з платформою без реєстрації. Волонтер — автентифікований користувач із роллю *volunteer*, що може брати участь у заходах, виконувати завдання та залишати відгуки. Організатор — автентифікований користувач із роллю *organizer*, що публікує заходи та управляє їх проведенням. Адміністратор — привілейований користувач із роллю *admin*, що має повний доступ до всіх операцій, управляє обліковими записами та розглядає скарги.

Варіанти використання для волонтера охоплюють такі сценарії. Реєстрація та вхід до системи: користувач вводить ім'я, email і пароль або виконує вхід через обліковий запис Google; система верифікує дані та повертає JWT-токен. Перегляд заходів: волонтер переглядає список доступних заходів із фільтрацією за категорією, містом, статусом і діапазоном дат, а також пошуком за текстом назви та опису. Реєстрація на захід: волонтер подає заявку на участь; якщо ліміт учасників не вичерпано, система створює запис зі статусом *pending* і надсилає сповіщення організатору. Скасування заявки: волонтер може скасувати власну заявку до її підтвердження організатором. Управління завданнями: волонтер переглядає призначені завдання та змінює їхній статус із *open* на *in_progress* або *completed*, при цьому кожна зміна

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		25

фіксується в журналі TaskHistory. Залишення відгуку на захід: після завершення заходу волонтер зі статусом attended може залишити відгук із рейтингом і коментарем. Збереження заходів: волонтер може додавати заходи до закладок для швидкого доступу. Управління профілем: редагування контактних даних, біографії, навичок і аватара. Донати: волонтер може підтримати захід фінансово через інтегрований платіжний модуль Stripe [21].

Варіанти використання для організатора включають усі дії волонтера щодо чужих заходів, а стосовно власних — такі сценарії. Створення заходу: організатор заповнює форму з назвою, описом, категорією, містом, координатами, датами та опціональним лімітом учасників; система перевіряє коректність дат і зберігає запис зі статусом draft або published. Редагування та видалення заходу: доступно лише для заходів у певних статусах. Перегляд заявок і управління учасниками: організатор переглядає список поданих заявок та підтверджує або відхиляє їх; після завершення заходу відзначає факт відвідування кожним учасником. Управління завданнями: організатор створює завдання в межах заходу, призначає їх конкретним підтвердженням волонтерам та відстежує прогрес виконання. Оцінювання волонтерів: після завершення заходу організатор може залишити відгук на кожного учасника з рейтингом і коментарем. Запит верифікації: організатор може подати заявку на верифікацію акаунту, завантаживши підтверджуючі документи.

Варіанти використання для адміністратора охоплюють управління обліковими записами користувачів, зміну ролей, блокування та розблокування акаунтів, перегляд і розгляд скарг, обробку запитів верифікації організаторів, а також публікацію новин та управління категоріями заходів.

Таким чином, кожна роль отримує власний набір функцій, що відповідає її відповідальності в межах платформи та забезпечує чітке розмежування доступу між учасниками системи. Розподіл функцій між акторами є основою для побудови діаграми варіантів використання та подальшого проєктування структури модулів системи. Узагальнена таблиця основних варіантів використання наведена у табл. 2.1.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		26

Таблиця 2.1

Основні варіанти використання системи

Варіант використання	Актор	Передумова	Результат
Реєстрація	Гість	Немає акаунту	Новий акаунт, JWT-токен
Вхід до системи	Будь-який	Існуючий акаунт	JWT-токен
Перегляд заходів	Будь-який	—	Відфільтрований список
Реєстрація на захід	Волонтер	Автентиф., захід active	Заявка pending
Скасування заявки	Волонтер	Заявка pending	Заявку видалено
Зміна статусу завдання	Волонтер	Призначене завдання	Статус оновлено, запис у TaskHistory
Збереження заходу	Волонтер	Автентиф.	Захід у закладках
Відгук на захід	Волонтер	Статус attended	Відгук збережено
Донат на захід	Волонтер	Автентиф., донати увімкнені	Транзакція Stripe
Створення заходу	Організатор	Роль organizer	Новий захід draft/published
Управління заявками	Організатор	Власний захід	Статус confirmed/rejected
Створення завдання	Організатор	Власний захід	Нове завдання open
Відгук на волонтера	Організатор	Завершений захід	Рейтинг волонтера оновлено
Запит верифікації	Організатор	Роль organizer	Заявка pending
Розгляд скарги	Адміністратор	Роль admin	Скаргу resolved/dismissed
Блокування акаунту	Адміністратор	Роль admin	Статус is_active = false
Обробка верифікації	Адміністратор	Роль admin	Статус verified/rejected

Діаграму варіантів використання системи наведено на рис. 2.1.

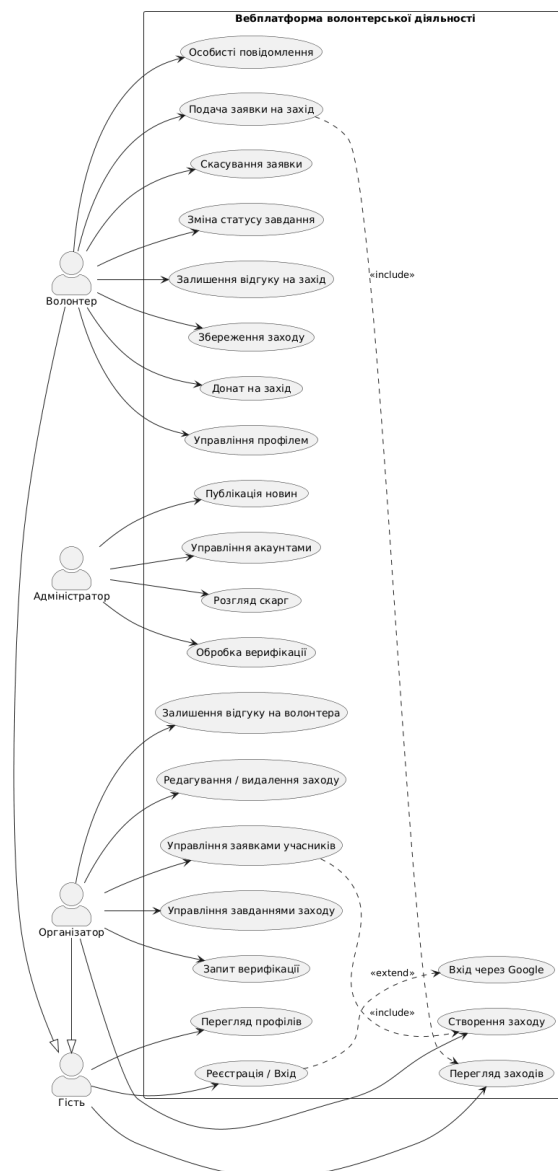


Рис. 2.1. Діаграма варіантів використання системи

Об'єктно-орієнтована модель системи описує структуру класів, їхні атрибути та взаємозв'язки. На основі аналізу варіантів використання та схеми бази даних виділено основні класи серверної частини системи.

Клас User представляє зареєстрованого користувача платформи та є базовою сутністю системи. Він містить атрибути: id (UUID), name, email, passwordHash, googleId, role (перечислення: volunteer, organizer, admin), avatarUrl, bio, phone, city, rating, experience, skills, isActive, isVerified, verificationStatus, stripeAccountId. Паролі зберігаються у хешованому вигляді

алгоритмом bcrypt. Метод generateToken формує JWT-токен із ідентифікатором і роллю користувача.

Клас Event представляє волонтерський захід. Атрибути: id, organizerId, categoryId, title, description, status (перечислення: draft, published, active, overdue, completed, completed_partial, cancelled), location, latitude, longitude, startDate, endDate, maxVolunteers, imageUrl, requiredSkills, isPrivate, inviteToken, donationsEnabled, donationGoal. Метод syncStatus оновлює статус заходу на основі поточного часу відносно startDate та endDate і викликається перед кожним поверненням списку заходів.

Клас EventParticipant фіксує зв'язок між заходом і волонтером. Атрибути: id, eventId, userId, status (перечислення: pending, confirmed, rejected, attended, closed). При скасуванні заходу всі pending-заявки автоматично переводяться у статус closed з надсиланням сповіщень відповідним користувачам.

Клас Task представляє завдання в межах заходу. Атрибути: id, eventId, assignedTo, title, description, status (open, in_progress, completed, cancelled), deadline, completedAt. Кожна зміна статусу фіксується у пов'язаному класі TaskHistory із збереженням попереднього та нового статусів, ідентифікатора автора зміни та мітки часу.

Клас Review описує відгук і реалізує дворівневу систему оцінювання. Атрибути: id, reviewerId, revieweeId, eventId, reviewType (event або volunteer), rating (1–5), comment. При збереженні або видаленні відгуку типу volunteer автоматично перераховується поле rating у таблиці users. Клас має пов'язані сутності ReviewReply та ReviewReaction, що реалізують коментарі та реакції до відгуків.

Клас Notification відповідає за зберігання та доставку сповіщень. Атрибути: id, userId, type (перечислення з 26 значеннями), content, link, isRead. Сповіщення генеруються автоматично при реєстрації на захід, підтвердженні або відхиленні заявки, призначенні завдання, завершенні заходу та інших ключових подіях.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		29

Клас Message реалізує особисті повідомлення між користувачами. Атрибути: id, senderId, receiverId, content, isRead, sentAt. Обмеження на рівні бази даних виключає надсилення повідомлень самому собі.

Клас Complaint фіксує скарги на користувачів або заходи. Атрибути: id, reporterId, targetType (user або event), targetId, reason, details, status (pending, resolved, dismissed), adminNote.

Клас VerificationRequest описує заявку організатора на верифікацію. Атрибути: id, organizerId, status (pending, verified, rejected), rejectionReason, submittedAt, reviewedAt, reviewedBy.

На серверній стороні класи реалізовані у функціональному стилі Express-контролерів: authController реалізує методи класу User, eventController — класу Event [37], participantController — класу EventParticipant, taskController — класів Task і TaskHistory, reviewController — класу Review та пов'язаних ReviewReply і ReviewReaction, notificationController — класу Notification. Розмежування доступу між ролями забезпечується middleware-функціями auth, requireOrgOrAdmin та requireAdmin. Взаємозв'язки між класами відображають реальні бізнес-залежності предметної галузі та слугують основою для побудови схеми бази даних. Діаграму класів вебплатформи наведено на рис. 2.2.

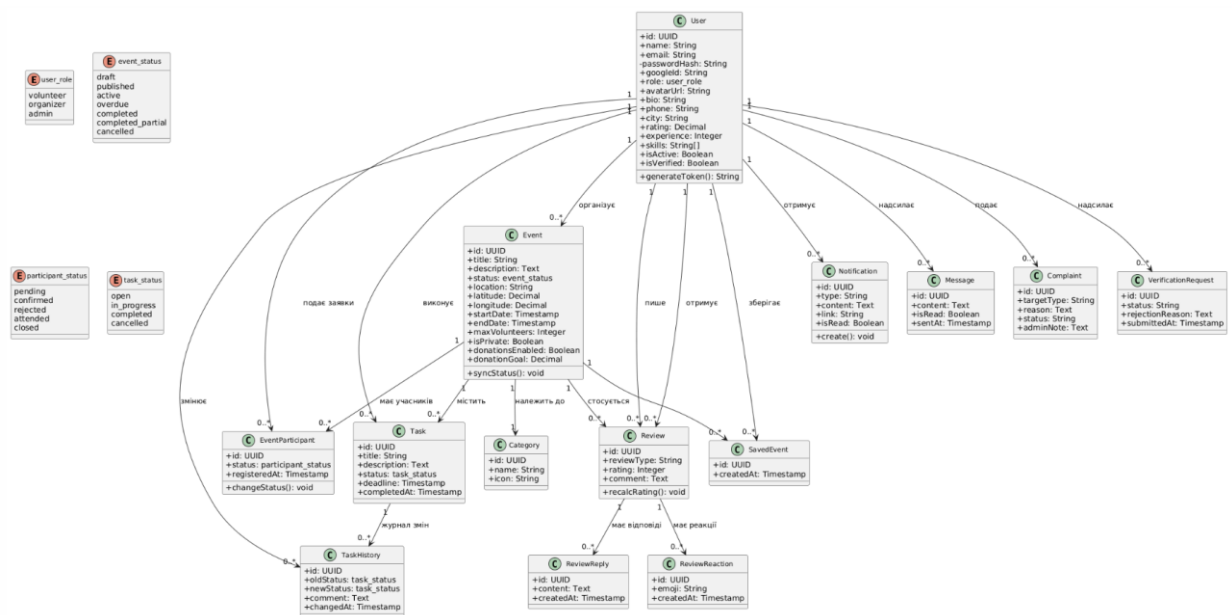


Рис. 2.2. Діаграма класів вебплатформи

2.2. Розробка бази даних системи

З метою забезпечення надійного збереження, цілісності та ефективної обробки інформації було спроектовано структуру реляційної бази даних вебплатформи на основі СУБД PostgreSQL 16 [19]. Архітектура бази даних логічно відображає всі ключові процеси системи та базується на сутностях, виділених у процесі об'єктно-орієнтованого аналізу.

При проєктуванні структури особливу увагу приділено зв'язкам між таблицями через первинні та зовнішні ключі. Для композиційних залежностей — зокрема між заходом і його учасниками, завданнями та документами — налаштовано каскадне видалення, щоб унеможливити появу «осиротілих» записів. Для слабких посилань, де видалення батьківського запису не повинне знищувати пов'язані дані, застосовується стратегія SET NULL. Для підвищення продуктивності на часто використовуваних полях створено індекси — за статусом заходу, датою початку, ідентифікатором організатора, учасника, відправника повідомлень та іншими ключовими атрибутами.

Центральною таблицею системи є users (табл. 2.2), що зберігає облікові дані та профільну інформацію всіх користувачів. Поле role визначає рівень доступу (volunteer, organizer, admin), is_active — статус акаунту, is_verified — підтверджений статус організатора. Поля experience та rating забезпечують роботу системи репутації, а stripe_account_id — інтеграцію з платіжним сервісом для організаторів.

Таблиця 2.2

Опис таблиці users

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор користувача
name	VARCHAR(100)			Ім'я користувача
email	VARCHAR(255)			Електронна пошта (унікальна)
password_hash	VARCHAR(255)			Хеш пароля (bcrypt)
google_id	VARCHAR(255)			Ідентифікатор Google OAuth

продовж. табл. 2.2

role	user_role			Роль: volunteer, organizer, admin
rating	DECIMAL(3,2)			Середній рейтинг волонтера
experience	INTEGER			Накопичений досвід (XP)
is_active	BOOLEAN			Статус акаунту
is_verified	BOOLEAN			Підтвердженість організатора
stripe_account_id	VARCHAR(255)			ID Stripe-акаунту організатора
created_at	TIMESTAMP			Дата реєстрації

Таблиця events зберігає дані про волонтерські заходи (табл. 2.3). Статус заходу визначається перелікованим типом event_status і автоматично синхронізується на основі дат startDate та endDate. Підтримуються географічні координати (latitude, longitude) для відображення на карті, прапорець is_private із відповідним invite_token для закритих заходів, а також параметри збору коштів (donations_enabled, donation_goal).

Таблиця 2.3

Опис таблиці events

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор заходу
organizer_id	UUID		+	Організатор заходу
category_id	UUID		+	Категорія заходу
title	VARCHAR(255)			Назва заходу
status	event_status			Статус: draft, published, active, completed, cancelled
latitude / longitude	DECIMAL(9,6)			Координати для карти
start_date / end_date	TIMESTAMP			Дати проведення
max_volunteers	INTEGER			Максимальна кількість учасників
is_private	BOOLEAN			Закритий захід
invite_token	VARCHAR(128)			Токен для запрошення
donations_enabled	BOOLEAN			Чи відкрито збір коштів
donation_goal	DECIMAL(12,2)			Цільова сума збору

Таблиця event_participants фіксує заявки волонтерів на участь у заходах (табл. 2.4). Унікальне обмеження на пару (event_id, user_id) унеможливлює дублювання заявок. Статус учасника проходить цикл: pending → confirmed/rejected → attended, або закривається автоматично (closed) при скасуванні заходу.

Таблиця 2.4

Опис таблиці event_participants

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор запису
event_id	UUID		+	Захід
user_id	UUID		+	Волонтер
status	participant_status			Статус: pending, confirmed, rejected, attended, closed
registered_at	TIMESTAMP			Дата подачі заявки

Таблиці tasks та task_history реалізують управління завданнями всередині заходу з повним журналуванням змін (табл. 2.5). Кожна зміна статусу завдання фіксується в task_history із збереженням попереднього стану, ідентифікатора автора зміни та мітки часу.

Таблиця 2.5

Опис таблиці tasks

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор завдання
event_id	UUID		+	Захід, якому належить завдання
assigned_to	UUID		+	Призначений волонтер
title	VARCHAR(255)			Назва завдання
status	task_status			Статус: open, in_progress, completed, cancelled
deadline	TIMESTAMP			Дедлайн виконання
completed_at	TIMESTAMP			Фактичний час завершення

Таблиці reviews, review_replies, review_reactions та reply_reactions реалізують дворівневу систему відгуків (табл. 2.6). Поле review_type розрізняє відгуки на захід і відгуки на волонтера. При зміні кількості відгуків типу volunteer автоматично перераховується поле rating у таблиці users.

Таблиця 2.6

Опис таблиці reviews

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор відгуку
reviewer_id	UUID		+	Автор відгуку
reviewee_id	UUID		+	Об'єкт оцінювання (волонтер)
event_id	UUID		+	Пов'язаний захід
review_type	VARCHAR(20)			Тип: event або volunteer
rating	INTEGER			Оцінка від 1 до 5
comment	TEXT			Текст відгуку

Таблиця notifications охоплює 26 типів системних подій і реалізує механізм внутрішніх сповіщень (табл. 2.7). Поле link дозволяє сформувати пряме посилання на відповідний розділ платформи, а is_read — відображати лічильник непрочитаних сповіщень у навігаційному заголовку.

Таблиця 2.7

Опис таблиці notifications

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор сповіщення
user_id	UUID		+	Отримувач сповіщення
type	notification_type			Тип події (26 значень)
content	TEXT			Текст сповіщення
link	VARCHAR(500)			Посилання для переходу
is_read	BOOLEAN			Статус прочитання

Таблиця complaints забезпечує модуль обробки скарг між користувачами платформи (табл. 2.8). Поле target_type дозволяє скаргу адресувати як до конкретного користувача, так і до заходу. Адміністратор змінює статус скарги та може залишити коментар у полі admin_note.

Таблиця 2.8

Опис таблиці complaints

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор скарги
reporter_id	UUID		+	Автор скарги
target_type	VARCHAR(10)			Об'єкт скарги: user або event
target_id	UUID			Ідентифікатор об'єкта скарги

продовж. табл. 2.8

reason	TEXT			Причина скарги
status	VARCHAR(20)			Статус: pending, resolved, dismissed
admin_note	TEXT			Коментар адміністратора

Таблиця donations фіксує кожен платіж у межах збору коштів (табл. 2.9). Підтвердження оплати відбувається через webhook Stripe, після чого статус запису змінюється з pending на completed. Прапорець is_anonymous дозволяє донору приховати своє ім'я від публічного відображення.

Таблиця 2.9

Опис таблиці donations

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор донату
event_id	UUID		+	Захід-отримувач
fundraising_id	UUID		+	Збір коштів (якщо є)
donor_id	UUID		+	Донор
amount	DECIMAL(12,2)			Сума донату
is_anonymous	BOOLEAN			Анонімність донату
status	VARCHAR(20)			Статус: pending, completed, blocked

Таблиця messages реалізує модуль особистих повідомлень між користувачами платформи (табл. 2.10). Обмеження на рівні бази даних унеможливорює надсилання повідомлень самому собі. Поле is_read дозволяє відображати кількість непрочитаних повідомлень у інтерфейсі.

Таблиця 2.10

Опис таблиці messages

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор повідомлення
sender_id	UUID		+	Відправник повідомлення
receiver_id	UUID		+	Отримувач повідомлення
content	TEXT			Текст повідомлення
is_read	BOOLEAN			Статус прочитання
sent_at	TIMESTAMP			Час відправлення

Таблиця categories зберігає перелік категорій волонтерських заходів (табл. 2.11). Кожна категорія має унікальну назву та може містити іконку для

відображення в інтерфейсі. Категорії використовуються для фільтрації заходів і формування персоналізованих рекомендацій через таблицю user_category_stats.

Таблиця 2.11

Опис таблиці categories

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор категорії
name	VARCHAR(100)			Назва категорії (унікальна)
description	TEXT			Опис категорії
icon_url	VARCHAR(500)			URL іконки для відображення в інтерфейсі
created_at	TIMESTAMP			Дата створення категорії

Таблиця event_fundraisings забезпечує зберігання інформації про збори коштів, прив'язані до конкретних заходів (табл. 2.12). Поле current_amount оновлюється автоматично при підтвердженні кожного нового донату через webhook Stripe. Організатор може вимкнути збір коштів, змінивши прапорець is_active.

Таблиця 2.12

Опис таблиці event_fundraisings

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор збору коштів
event_id	UUID		+	Захід, до якого прив'язано збір
title	VARCHAR(255)			Назва збору коштів
goal_amount	DECIMAL(12,2)			Цільова сума збору
current_amount	DECIMAL(12,2)			Поточна зібрана сума
is_active	BOOLEAN			Активність збору коштів
created_at	TIMESTAMP			Дата створення збору

Таблиця sponsors зберігає профілі партнерських організацій, які можуть підтримувати волонтерські заходи фінансово, матеріально або транспортно (табл. 2.13). Після верифікації адміністратором (is_verified = true) спонсор може бути прив'язаний до заходів через таблицю event_sponsors із зазначенням типу підтримки.

Таблиця 2.13

Опис таблиці sponsors

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор спонсора
name	VARCHAR(255)			Назва організації-спонсора
description	TEXT			Опис діяльності організації
logo_url	VARCHAR(500)			URL логотипу для відображення
contact_email	VARCHAR(255)			Контактна електронна пошта
website_url	VARCHAR(500)			Офіційний сайт організації
is_verified	BOOLEAN			Статус верифікації спонсора

Таблиця charity_needs реалізує облік матеріальних потреб волонтерського заходу (табл. 2.14). Організатор зазначає перелік необхідних ресурсів із потрібною кількістю та одиницею вимірювання. Волонтери й спонсори можуть частково або повністю закривати потреби через таблицю need_contributions. Статус потреби автоматично змінюється на fulfilled після досягнення required_quantity.

Таблиця 2.14

Опис таблиці charity_needs

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор потреби
event_id	UUID		+	Захід, до якого належить потреба
title	VARCHAR(255)			Назва необхідного ресурсу
description	TEXT			Детальний опис потреби
required_quantity	INTEGER			Необхідна кількість ресурсу
current_quantity	INTEGER			Поточна зібрана кількість
unit	VARCHAR(50)			Одиниця вимірювання (кг, шт, л тощо)
status	VARCHAR(20)			Статус: open, fulfilled

Таблиця verification_requests зберігає заявки організаторів на верифікацію облікового запису (табл. 2.15). Після подання заявки з підтвердними документами адміністратор змінює статус на verified або rejected із зазначенням причини відмови. Верифікований організатор отримує

спеціальну позначку довіри, що підвищує рівень довіри волонтерів до публікованих заходів.

Таблиця 2.15

Опис таблиці verification_requests

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор запиту
organizer_id	UUID		+	Організатор, що подав запит
status	VARCHAR(20)			Статус: pending, verified, rejected
rejection_reason	TEXT			Причина відмови (за наявності)
submitted_at	TIMESTAMP			Дата подання заявки
reviewed_at	TIMESTAMP			Дата розгляду заявки
reviewed_by	UUID		+	Адміністратор, що розглядав запит

Таблиця user_category_stats накопичує статистику участі кожного волонтера в розрізі категорій заходів (табл. 2.16). Записи оновлюються автоматично при підтвердженні участі у заході. Ці дані використовуються алгоритмом рекомендацій для формування персоналізованого списку заходів — система пропонує заходи категорій, у яких волонтер вже брав участь найчастіше.

Таблиця 2.16

Опис таблиці user_category_stats

Назва	Тип даних	PK	FK	Опис
id	UUID	+		Унікальний ідентифікатор запису статистики
user_id	UUID		+	Волонтер
category_id	UUID		+	Категорія заходів
events_count	INTEGER			Кількість заходів цієї категорії
total_hours	DECIMAL(10,2)			Загальна кількість відпрацьованих годин
last_participation	TIMESTAMP			Дата останньої участі у цій категорії

Повну схему бази даних у вигляді ER-діаграми наведено в додатку А. Спроектowana структура бази даних забезпечує цілісність даних на рівні обмежень, ефективність запитів завдяки індексам та гнучкість для подальшого розширення функціоналу платформи.

2.3. Проектування та реалізація алгоритмів роботи системи

Для формалізованого опису логіки роботи вебплатформи розроблено діаграми активності та послідовності у нотації UML 2.0. Вони відображають ключові сценарії взаємодії акторів із системою та внутрішню поведінку серверних компонентів.

Діаграма активності на рис. 2.3 демонструє алгоритм автентифікації користувача та реєстрації на волонтерський захід.



Рис. 2.3. Діаграма активності для реєстрації на захід

Процес починається з відкриття головної сторінки платформи. Незареєстрований відвідувач переглядає список доступних заходів без авторизації. При спробі подати заявку система перевіряє наявність активного JWT-токена. Якщо токен відсутній або прострочений, користувач

перенаправляється на форму входу, де може авторизуватися через email і пароль або через обліковий запис Google за протоколом OAuth 2.0. Після успішної автентифікації сервер генерує токен і повертає користувача до сторінки заходу.

На етапі подачі заявки система перевіряє кілька умов: чи не подав користувач заявку раніше, чи не вичерпано ліміт учасників, чи захід має статус active або published. За виконання всіх умов створюється запис у таблиці event_participants зі статусом pending, організатору надсилається сповіщення про нову заявку. У разі невиконання будь-якої умови користувач отримує відповідне повідомлення про помилку.

Діаграма активності на рис. 2.4 описує алгоритм управління заходом з боку організатора — від створення до завершення.

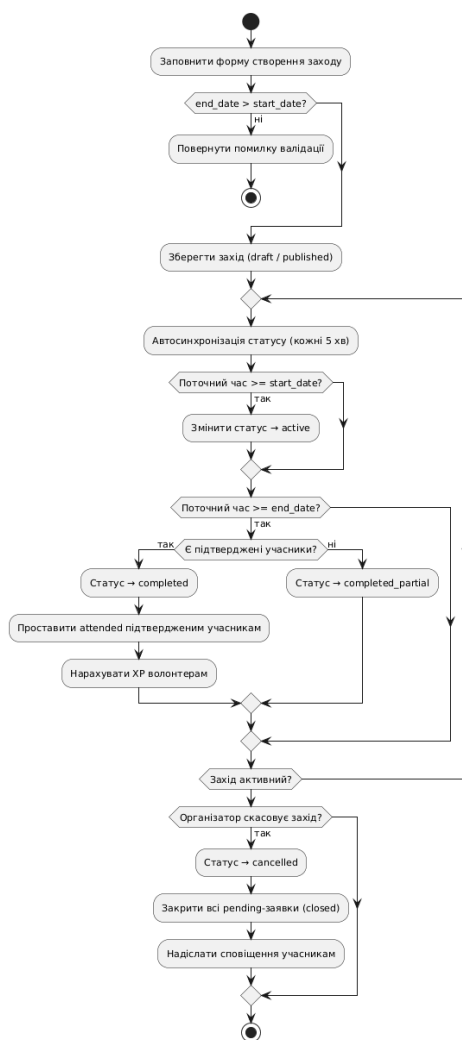


Рис. 2.4. Діаграма активності для управління заходом

Організатор заповнює форму з назвою, описом, категорією, датами, координатами та опціональним лімітом учасників. Сервер перевіряє коректність дат ($\text{end_date} > \text{start_date}$) та зберігає захід зі статусом draft або одразу published. Після публікації система автоматично синхронізує статус заходу залежно від поточного часу: при настанні start_date статус змінюється на active, при настанні end_date — на completed або completed_partial залежно від наявності підтверджених учасників. При скасуванні заходу всі pending-заявки автоматично переводяться у статус closed, а учасникам надсилаються сповіщення.

Діаграма послідовності на рис. 2.5 детально описує взаємодію між клієнтом, сервером та базою даних у процесі розгляду заявки організатором і виконання завдань волонтером.

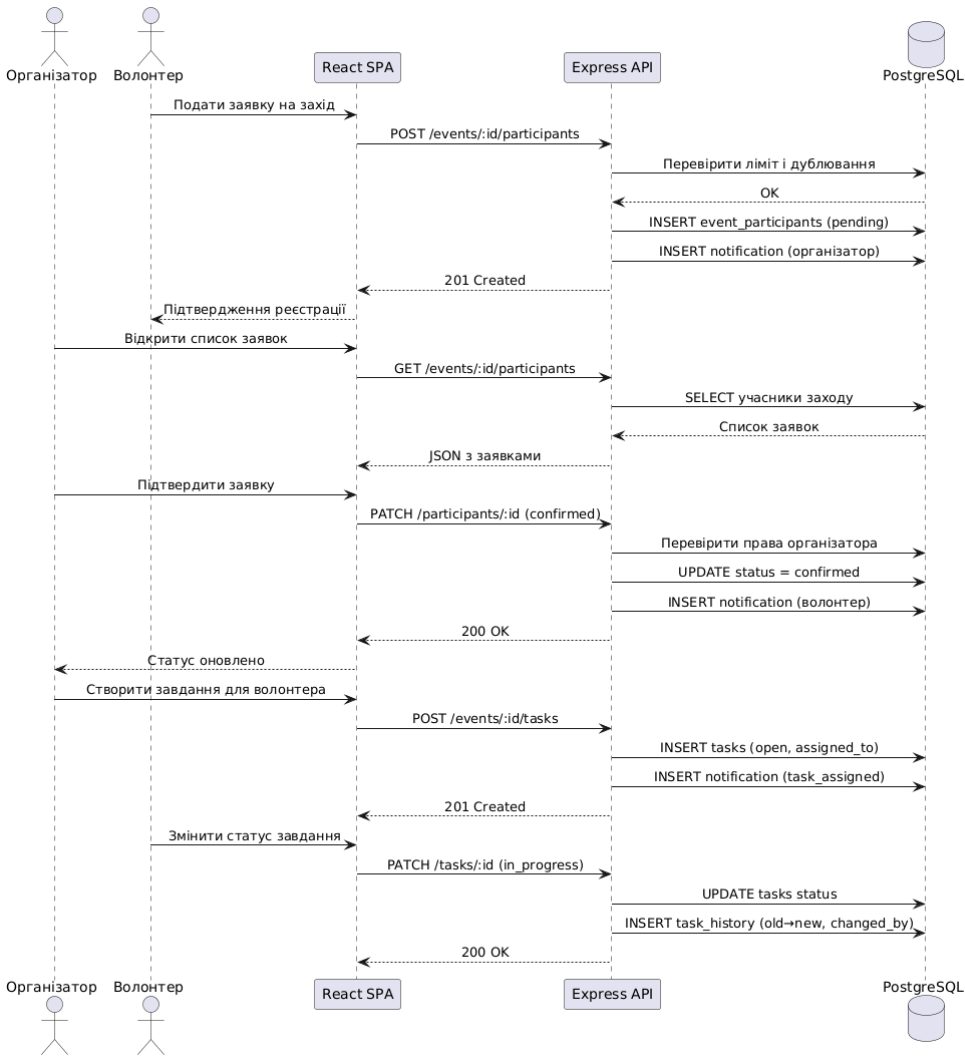


Рис. 2.5. Діаграма послідовності для управління учасниками та завданнями

Після подачі заявки організатор отримує сповіщення і переходить до панелі управління учасниками. При підтвердженні заявки сервер перевіряє, чи є поточний користувач організатором саме цього заходу, після чого змінює статус запису на `confirmed` і надсилає сповіщення волонтеру. Організатор може призначити підтвердженому волонтеру завдання: сервер створює запис у таблиці `tasks` зі статусом `open`, волонтер отримує сповіщення типу `task_assigned`. При зміні статусу завдання волонтером кожна зміна атомарно фіксується у таблиці `task_history` із збереженням попереднього стану, автора зміни та мітки часу.

Діаграма послідовності на рис. 2.6 описує алгоритм обробки донату через платіжний сервіс Stripe. Волонтер відкриває сторінку збору коштів і вводить суму. Клієнтська частина через `Stripe.js` формує платіжний намір на стороні браузера без передачі даних картки на власний сервер. Сервер звертається до `Stripe API` для створення `PaymentIntent` і повертає `client_secret` клієнту. Після підтвердження оплати Stripe надсилає `webhook`-подію на сервер, який верифікує підпис події та фіксує донат у таблиці `donations` зі статусом `completed`. Організатор отримує сповіщення типу `donation_received`.

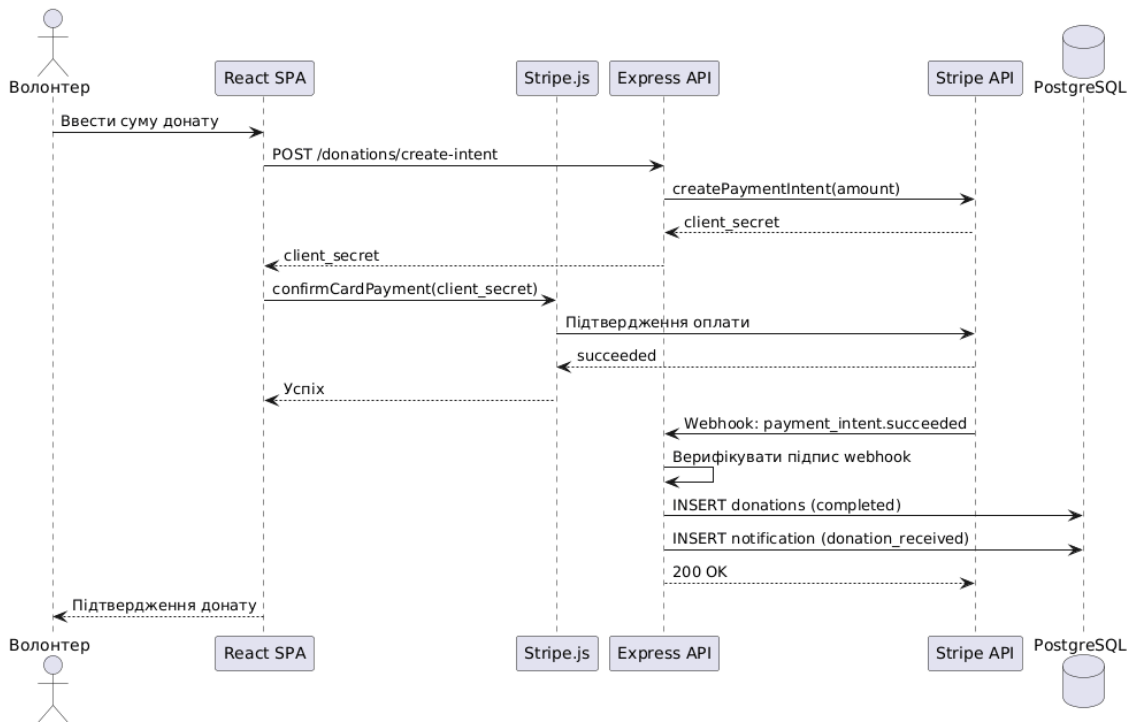


Рис. 2.6. Діаграма послідовності для обробки донату

Для формалізованого опису життєвого циклу волонтерського заходу розроблено діаграму станів (рис. 2.7). Захід проходить шість можливих станів. Після створення організатором він отримує статус *draft* і може бути опублікований (*published*) або скасований до публікації. При досягненні *start_date* функція *syncStatus* автоматично переводить захід у стан *active*. Після настання *end_date* система розрізняє три варіанти завершення: *completed* — за наявності підтверджених учасників, *completed_partial* — якщо підтверджених учасників не було, *overdue* — якщо синхронізація не відбулась вчасно. Скасування (*cancelled*) доступне організатору на будь-якому етапі; при цьому всі заявки зі статусом *pending* автоматично закриваються з надсиланням сповіщень.

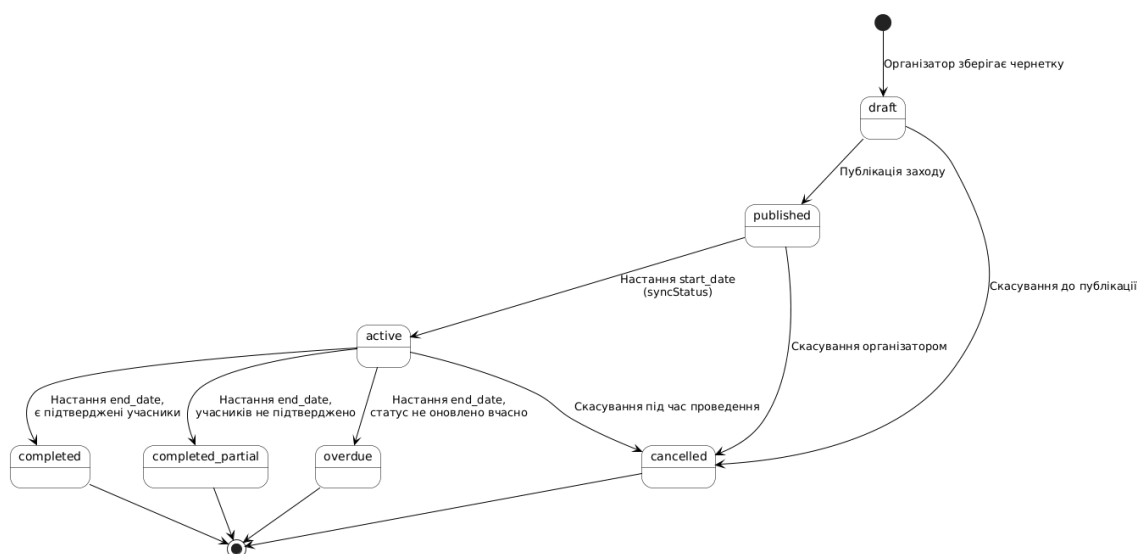


Рис. 2.7. Діаграма станів заходу

Окремо побудовано діаграму станів заявки учасника (рис. 2.8). Після подачі заявки волонтером запис у таблиці *event_participants* отримує статус *pending*. Організатор переводить заявку у *confirmed* або *rejected*; в обох випадках волонтеру надсилається сповіщення. Після завершення заходу організатор відзначає фактичну присутність кожного учасника, змінюючи статус на *attended*, — це є передумовою для нарахування досвіду (XP) та отримання PDF-сертифіката. При скасуванні заходу всі активні заявки (*pending* та *confirmed*) автоматично переводяться у стан *closed* засобами серверного обробника подій.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		43

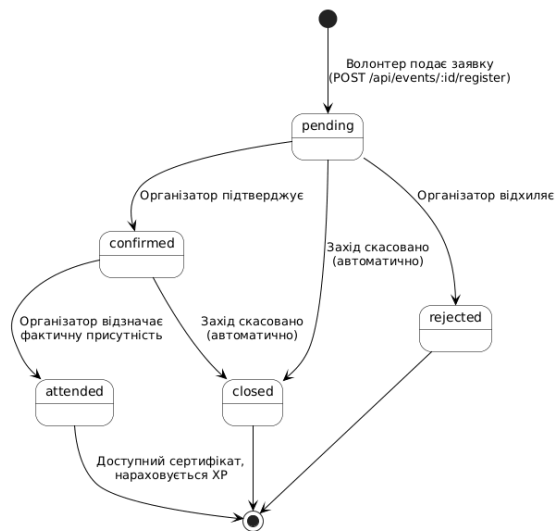


Рис. 2.8. Діаграма станів заявки учасника

Архітектуру розгортання системи відображено на рис. 2.9. Платформа функціонує у трьох Docker-контейнерах в межах єдиної внутрішньої мережі. Контейнер nginx роздає статичний React-білд і проксіює API-запити на контейнер backend. Контейнер backend містить Node.js-сервер з Express, обробляє бізнес-логіку та взаємодіє з контейнером postgres через драйвер pg. Зовнішні запити потрапляють виключно на порт 80 (nginx); порти бекенду та БД не відкриті назовні. Сервер взаємодіє з трьома зовнішніми сервісами: Stripe API (обробка платежів і верифікація webhook-підписів), Google OAuth 2.0 (автентифікація через Passport.js) та SMTP-сервером (email-сповіщення через Nodemailer).

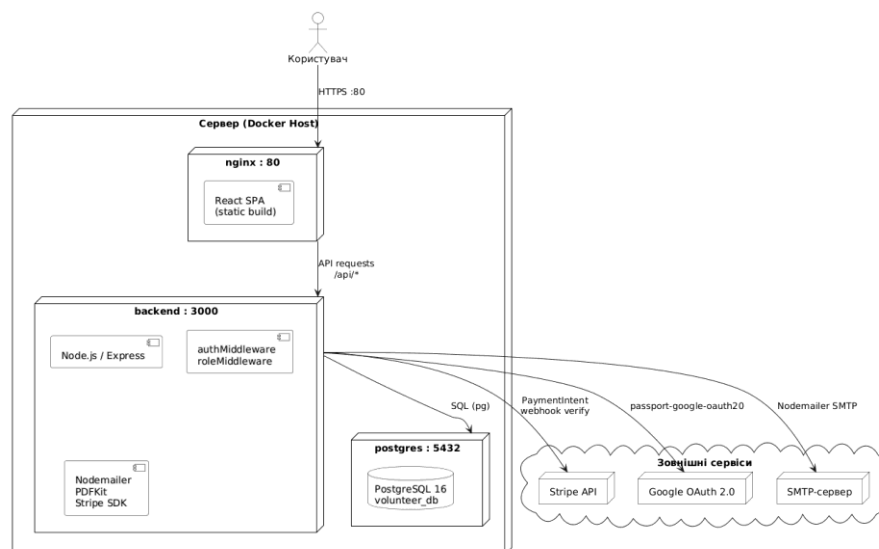


Рис. 2.9. Діаграма розгортання

2.4. Реалізація функціоналу вебплатформи для організації волонтерської діяльності

Головним центральним елементом взаємодії між волонтером і організатором на платформі є механізм реєстрації на захід. Саме на цьому прикладі доцільно розглянути логіку обробки запитів, перевірки бізнес-правил та оновлення пов'язаних даних у реальному часі.

На рівні API обробку запитів виконує функція `registerToEvent` у модулі `participantController`. Вона отримує HTTP POST-запит від автентифікованого користувача, витягує ідентифікатор заходу з параметрів маршруту та ідентифікатор користувача з декодованого JWT-токена. Перед записом до бази даних виконується послідовна перевірка бізнес-обмежень: існування заходу, відповідність його статусу (`published` або `active`), а також заборона організатору реєструватися на власний захід. Унікальне обмеження на комбінацію полів `event_id` та `user_id` на рівні бази даних виключає дублювання заявок — при спробі повторної реєстрації PostgreSQL повертає код помилки 23505, який перехоплюється та перетворюється у зрозуміле повідомлення для клієнта.

Лістинг (реалізація функції реєстрації на захід із перевіркою бізнес-правил та обробкою помилки дублювання):

```
const registerToEvent = async (req, res) => {
  const { id: event_id } = req.params;
  const user_id = req.user.id;
  try {
    const event = await pool.query(
      'SELECT * FROM events WHERE id = $1', [event_id]
    );
    if (event.rows.length === 0)
      return res.status(404).json({ error: 'Подію не знайдено' });
    if (event.rows[0].status !== 'published' &&
      event.rows[0].status !== 'active')
      return res.status(400).json({ error: 'Реєстрація недоступна' });
    if (event.rows[0].organizer_id === user_id)
      return res.status(400).json({
        error: 'Організатор не може реєструватись на власну подію'
      });
    const result = await pool.query(
      INSERT INTO event_participants (event_id, user_id)
      VALUES ($1, $2) RETURNING *
```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		45

```

    ` , [event_id, user_id]);
    res.status(201).json(result.rows[0]);
  } catch (err) {
    if (err.code === '23505')
      return res.status(400).json({ error: 'Ви вже зареєстровані' });
    res.status(500).json({ error: 'Помилка сервера' });
  }
};

```

Після успішного збереження запису зі статусом pending система автоматично виконує два побічні ефекти. По-перше, організатору надсилається внутрішнє сповіщення через функцію createNotification із зазначенням імені волонтера та назви заходу. По-друге, на email організатора надсилається лист через Nodemailer із деталями заявки — датою, місцем проведення та ім'ям кандидата. Обидві операції виконуються після підтвердження запису в базі, що гарантує їх виконання лише при успішній транзакції.

Управління завданнями всередині заходу реалізовано у модулі taskController. Завдання мають пріоритет (low, medium, high) і статус із повним циклом: open → in_progress → completed або cancelled. При отриманні списку завдань запит сортується за двома рівнями: спочатку за пріоритетом, потім за статусом, що забезпечує відображення найважливіших і найактивніших завдань на початку списку. Кожна зміна статусу в межах однієї операції фіксується у таблиці task_history із збереженням попереднього стану, ідентифікатора автора зміни та мітки часу. Після того як всі завдання заходу переведено у статус completed, система автоматично надсилає організатору сповіщення про можливість дострокового завершення заходу. Ця перевірка виконується допоміжною функцією checkAllTasksDone, яка використовує агрегатний SQL-запит із фільтром по статусах замість завантаження всіх записів у пам'ять застосунку.

Система відгуків реалізована через два окремих ендпоінти залежно від типу: відгук на захід і відгук на волонтера. Для кожного запиту виконується перевірка допустимого діапазону оцінки (від 1 до 5), статусу заходу (лише completed), а також факту участі рецензента у заході. Унікальний індекс на

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		46

рівні бази виключає повторні відгуки від одного автора на той самий об'єкт. Після збереження відгуку типу `volunteer` або `organizer` автоматично викликається функція `recalcRating`, яка оновлює середній рейтинг відповідного користувача через єдиний `UPDATE`-запит із підзапитом `AVG`. Такий підхід усуває потребу в агрегації під час читання профілю та забезпечує цілісність даних у межах однієї операції.

Для формування відповіді клієнту відгуки повертаються разом із вкладеними колекціями відповідей і реакцій. Це реалізовано через підзапити з `json_agg` та `json_build_object` безпосередньо у `SQL`, що дозволяє отримати повністю готовий `JSON`-об'єкт за один запит до бази даних без додаткових `round-trip` на сервері.

Однією з особливостей вебплатформи є система досвіду (`XP`) та рівнів волонтера, реалізована у модулі `experience.js`. Константи `XP` визначають кількість очок за кожну дію: участь у заході приносить 50 `XP`, виконане завдання — 20 `XP`, виконання завдання достроково — додатково 10 `XP`, позитивний відгук від організатора (4–5 зірок) — 15 `XP`, участь у заходах категорії «Надзвичайні ситуації» — бонусні 30 `XP`. Ці значення синхронізовані між бекендом і фронтендом, що виключає розбіжності у відображенні прогресу.

Функція `addExperience` зберігає поточний `XP` користувача до нарахування, обчислює новий рівень і, якщо рівень підвищився, автоматично надсилає внутрішнє сповіщення типу `level_up` та `email`-лист із привітанням. Система визначає п'ять рівнів: Новачок (0 `XP`), Активіст (100 `XP`), Досвідчений (300 `XP`), Ветеран (600 `XP`) та Експерт (1000 `XP`). Перевірка підвищення рівня виконується шляхом порівняння порогових значень до і після нарахування — це дозволяє коректно обробляти випадки, коли одна операція приносить достатньо `XP` для переходу через кілька рівнів одразу.

Лістинг (реалізація функції нарахування досвіду з перевіркою підвищення рівня):

```
const addExperience = async (userId, amount) => {
```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижомотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		47

```

if (!userId || amount <= 0) return;
const before = await pool.query(
  'SELECT experience, email, name FROM users WHERE id = $1',
  [userId]
);
const { experience: xpBefore, email, name } = before.rows[0];
const xpAfter = (xpBefore || 0) + amount;
await pool.query(
  'UPDATE users SET experience = $1 WHERE id = $2',
  [xpAfter, userId]
);
const levelBefore = getLevel(xpBefore || 0);
const levelAfter = getLevel(xpAfter);
if (levelAfter.min > levelBefore.min) {
  await createNotification(
    userId, 'level_up',
    `Вітаємо! Ви досягли рівня "${levelAfter.name}"! (${xpAfter} XP)`,
    '/profile'
  );
  await sendLevelUp({ to: email, name, levelName: levelAfter.name });
}
};

```

З архітектурної точки зору логіка перерахунку рейтингів, нарахування досвіду та надсилання сповіщень свідомо розміщена у допоміжних функціях, а не дублюється між контролерами. Це забезпечує єдину точку зміни при потребі коригування алгоритмів і спрощує написання модульних тестів для кожного з компонентів системи.

Модуль автентифікації реалізовано у файлі `authController.js` і охоплює три сценарії входу: реєстрацію з email і паролем, вхід за обліковими даними та авторизацію через Google OAuth 2.0. При реєстрації пароль хешується бібліотекою `bcrypt` із коефіцієнтом складності 10, що відповідає рекомендаціям OWASP [32] і унеможливорює відновлення оригінального значення з хешу навіть при компрометації бази даних. Після успішного входу сервер генерує JWT-токен із терміном дії 24 години та підписом алгоритму HS256. Токен передається клієнту через тіло відповіді і зберігається у `localStorage` браузера.

Лістинг (реалізація функції реєстрації користувача з хешуванням пароля та генерацією JWT-токена):

```

const register = async (req, res) => {
  const { name, email, password, role: rawRole } = req.body;

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		48

```

const role = ['volunteer','organizer'].includes(rawRole)
  ? rawRole : 'volunteer';
if (password.length < 6)
  return res.status(400).json({ error: 'Пароль занадто короткий' });
if (!/^[^s@]+@[^s@]+\.[^s@]+$/ .test(email))
  return res.status(400).json({ error: 'Невірний формат email' });
const existing = await pool.query(
  'SELECT id FROM users WHERE email = $1', [email]
);
if (existing.rows.length > 0)
  return res.status(400).json({ error: 'Email вже зареєстровано' });
const password_hash = await bcrypt.hash(password, 10);
const result = await pool.query(
  `INSERT INTO users (name, email, password_hash, role)
  VALUES ($1, $2, $3, $4) RETURNING id, name, email, role`,
  [name, email, password_hash, role]
);
const token = jwt.sign(
  { id: result.rows[0].id, role: result.rows[0].role },
  process.env.JWT_SECRET,
  { expiresIn: '7d' }
);
res.status(201).json({ user: result.rows[0], token });
};

```

Авторизація через Google OAuth 2.0 реалізована за допомогою бібліотеки Passport.js зі стратегією passport-google-oauth20. При першому вході через Google сервер перевіряє, чи існує обліковий запис із відповідним email. Якщо ні — автоматично створюється новий запис із роллю volunteer та збереженням google_id. При повторному вході виконується пошук за google_id, оскільки автентифікацію здійснює Google. Після успішної OAuth-авторизації сервер переспрямовує браузер на фронтенд з JWT-токеном у параметрі URL, який фронтенд зберігає та очищає з адресного рядка [23].

Для захисту API-ендпоінтів реалізовано middleware authMiddleware, що перевіряє наявність та дійсність JWT-токена в заголовку Authorization кожного захищеного запиту. Middleware декодує токен, витягує ідентифікатор і роль користувача та додає їх до об'єкта req для використання в контролерах. Окремий middleware roleMiddleware перевіряє відповідність ролі користувача вимогам маршруту — наприклад, лише організатор може створювати заходи, а адміністратор — керувати обліковими записами. Валідація вхідних даних виконується через бібліотеку express-validator: перевіряється формат email,

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		49

довжина пароля, коректність дат та числових значень перед передачею даних у бізнес-логіку.

Генерацію персональних PDF-сертифікатів реалізовано у модулі `certificateController.js` з використанням бібліотеки `PDFKit` [29]. Сертифікат створюється динамічно при запиті і не зберігається на сервері — документ формується в оперативній пам'яті та передається клієнту потоком через `res.pipe(doc)`. Структура сертифіката включає заголовок з логотипом платформи, ім'я волонтера, назву заходу, дати участі, унікальний ідентифікатор у форматі `UUID` для верифікації, підпис організатора та QR-код з посиланням на сторінку заходу. Перед генерацією виконується перевірка: сертифікат доступний лише учасникам зі статусом `attended`, тобто тим, хто фактично відвідав захід.

Система сповіщень охоплює 26 типів подій і реалізована у двох рівнях: внутрішні сповіщення у базі даних та дублювання критичних подій через email. Внутрішні сповіщення зберігаються у таблиці `notifications` із прив'язкою до отримувача і посиланням для переходу; фронтенд отримує їх кількість при завантаженні сторінки та відображає лічильник непрочитаних у навігаційному заголовку. Email-сповіщення надсилаються через бібліотеку `Nodemailer` [30] з SMTP-підключенням до поштового сервера. Для кожного типу події сформовано окремий HTML-шаблон із динамічними значеннями — іменем отримувача, назвою заходу та прямим посиланням для переходу до відповідного розділу платформи.

Модуль управління збором коштів реалізовано у файлах `fundraisingController.js` та `donationController.js`. Організатор може прив'язати до заходу збір коштів із зазначенням цільової суми та опису. Поточний прогрес відображається як відсоток від цілі через агрегатний SQL-запит `SUM` по полю `amount` таблиці `donations` зі статусом `completed`. Підтвердження платежу обробляється через вебхук `Stripe`: сервер верифікує підпис події за допомогою `stripe.webhooks.constructEvent` і `STRIPE_WEBHOOK_SECRET`, що унеможливорює підроблення статусу оплати без звернення до `Stripe API`. Після

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		50

верифікації запис у таблиці donations переводиться зі статусу pending у completed, і організатор отримує сповіщення donation_received.

Адміністративний модуль охоплює управління обліковими записами, обробку скарг та верифікацію організацій. Скарги (таблиця complaints) подають волонтери на окремих користувачів або заходи; адміністратор переглядає їх у відповідній панелі й виставляє статус resolved або dismissed із поясненням. Верифікація організацій передбачає подачу організатором запиту з підтвердними документами; адміністратор вручну переводить статус організації у verified, після чого на сторінці організатора з'являється відповідна позначка довіри. Доступ до всіх адміністративних маршрутів захищений перевіркою ролі admin у roleMiddleware.

2.5. Модульна структура та програмний інтерфейс системи

Серверна частина вебплатформи організована за принципом модульної архітектури, де кожен функціональний блок системи відповідає окремому контролеру та маршрутизатору. Точкою входу є файл app.js, який ініціалізує Express-застосунок, підключає middleware та реєструє маршрутизатори. Конфігурація середовища зчитується з файлу .env через бібліотеку dotenv, що забезпечує відокремлення параметрів розгортання від програмного коду [33].

Маршрутизація побудована на основі окремих Router-модулів Express, кожен із яких обслуговує одну предметну область: authRouter — автентифікацію, eventRouter — заходи, participantRouter — управління заявками, taskRouter — завдання, reviewRouter — відгуки, notificationRouter — сповіщення, userRouter — профілі користувачів, adminRouter — адміністрування, donationRouter — платежі, certificateRouter — генерацію сертифікатів. Кожен маршрутизатор підключено до головного файлу з відповідним префіксом /api/[resource].

Захист маршрутів реалізовано через два рівні middleware. Функція authMiddleware перевіряє наявність дійсного JWT-токена в заголовку Authorization: Bearer і передає декодовані дані користувача в об'єкт req.user. Функція roleMiddleware отримує перелік дозволених ролей і повертає статус

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		51

403 Forbidden для запитів із недостатніми правами. Валідація вхідних параметрів виконується через express-validator безпосередньо у визначенні маршруту, до передачі запиту в контролер.

Структуру фронтенд-модулів платформи наведено в табл. 2.17.

Таблиця 2.17

Структура фронтенд-модулів вебплатформи

Модуль / файл	Призначення	Ключові компоненти
src/context/ AuthContext.js	Глобальний стан автентифікації	Провайдер токена, ролі та даних поточного користувача
src/services/ api.js	HTTP-клієнт (axios)	Централізований interceptor: додає JWT до заголовків, перехоплює 401
src/services/ eventService.js	Запити до /api/events	getEvents, getEventById, createEvent, updateEvent, deleteEvent
src/services/ participantService.js	Запити до /api/participants	registerToEvent, updateStatus, getParticipants
src/components/ EventCard.jsx	Картка заходу у списку	Назва, дата, категорія, кількість учасників, кнопка реєстрації
src/components/ TaskList.jsx	Список завдань заходу	Пріоритет, статус, дедлайн, кнопки зміни статусу
src/components/ ReviewForm.jsx	Форма відгуку	Зіркова оцінка, текстове поле, вибір типу відгуку
src/components/ DonationWidget.jsx	Віджет збору коштів	Прогрес-бар, поле суми, інтеграція Stripe.js
src/pages/ HomePage.jsx	Головна сторінка	Фільтри заходів, карта, список заходів, рекомендації
src/pages/ EventDetailPage.jsx	Сторінка заходу	Деталі, учасники, завдання, відгуки, збір коштів
src/pages/ ProfilePage.jsx	Профіль користувача	ХР, рівень, статистика, відгуки, сертифікати
src/pages/ OrganizerDashboard.jsx	Панель організатора	Управління заходами, заявками, завданнями, спонсорами
src/pages/ AdminDashboard.jsx	Адмін-панель	Управління обліковими записами, скарги, верифікації
src/router/ PrivateRoute.jsx	Захищені маршрути	Перевірка токена, перенаправлення на /login

Перелік основних ендпоінтів REST API вебплатформи наведено у табл. 2.18.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		52

Основні ендпоінти REST API вебплатформи

Метод	Маршрут	Роль	Призначення
POST	/api/auth/register	Будь-який	Реєстрація нового користувача
POST	/api/auth/login	Будь-який	Вхід до системи, отримання JWT
GET	/api/auth/google	Будь-який	Ініціалізація OAuth 2.0
GET	/api/events	Будь-який	Список заходів із фільтрами
POST	/api/events	Організатор	Створення заходу
GET	/api/events/:id	Будь-який	Деталі заходу
PUT	/api/events/:id	Організатор	Редагування заходу
DELETE	/api/events/:id	Організатор	Видалення заходу
POST	/api/events/:id/register	Волонтер	Подача заявки на участь
GET	/api/events/:id/participants	Організатор	Список заявок на захід
PUT	/api/participants/:id/status	Організатор	Підтвердження або відхилення заявки
POST	/api/events/:id/tasks	Організатор	Створення завдання
GET	/api/events/:id/tasks	Авторизований	Перегляд завдань заходу
PUT	/api/tasks/:id/status	Волонтер	Зміна статусу завдання
POST	/api/reviews	Волонтер/Орг.	Додавання відгуку
GET	/api/users/:id	Будь-який	Публічний профіль користувача
PUT	/api/users/:id	Власник	Оновлення профілю
GET	/api/notifications	Авторизований	Сповіщення поточного користувача
PUT	/api/notifications/read	Авторизований	Позначення сповіщень прочитаними
GET	/api/events/:id/certificate	Волонтер	Генерація PDF-сертифіката
POST	/api/donations	Волонтер	Ініціалізація платежу Stripe
POST	/api/stripe/webhook	Stripe	Підтвердження оплати (webhook)
GET	/api/admin/users	Адмін	Управління обліковими записами
GET	/api/admin/complaints	Адмін	Перегляд скарг
PUT	/api/admin/verification/:id	Адмін	Розгляд запиту верифікації

Клієнтська частина побудована на основі бібліотеки React 18 із використанням функціональних компонентів та хуків. Глобальний стан автентифікації зберігається в AuthContext і доступний усім компонентам без передачі через props. Маршрутизацію сторінок забезпечує React Router v6; захищені маршрути обгорнуті у компонент PrivateRoute, який перевіряє наявність токена і перенаправляє неавторизованого користувача на сторінку входу.

Структура фронтенду поділяється на три шари. Шар компонентів (components/) містить перевикористовувані UI-елементи: картки заходів, форми, модальні вікна та таблиці учасників. Шар сторінок (pages/) містить повноекранні компоненти для кожного маршруту: HomePage, EventDetailPage, ProfilePage, AdminDashboard тощо. Шар сервісів (services/) відповідає за HTTP-запити через бібліотеку axios із централізованим перехоплювачем (interceptor), що автоматично додає JWT-токен до заголовків усіх запитів і обробляє відповідь зі статусом 401 — видаляє протермінований токен і перенаправляє користувача на сторінку входу.

Розгортання системи реалізовано через Docker Compose із трьома сервісами: nginx (React-білд і проксіювання API), backend (Node.js/Express) та postgres (PostgreSQL 16). Внутрішня мережа Docker забезпечує ізоляцію: зовні доступний лише порт 80; порти бекенду та бази даних закриті. База даних ініціалізується SQL-скриптом при першому запуску контейнера, що автоматично створює всі таблиці, типи перелічень та індекси. Дані зберігаються в іменованому томі postgres_data, незалежному від життєвого циклу контейнера [31].

Взаємодію між модулями серверної частини схематично можна описати таким чином: вхідний HTTP-запит проходить через ланцюжок middleware — cors, express.json, authMiddleware, roleMiddleware — і лише після цього потрапляє до відповідного контролера. Контролер звертається до бази даних через пул з'єднань pg, за потреби викликає допоміжні функції з utils/ (нарахування досвіду, надсилання email, генерація сповіщень) та формує

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		54

JSON-відповідь. Такий лінійний потік обробки запиту забезпечує передбачуваність поведінки системи та спрощує локалізацію помилок під час налагодження.

Висновки до другого розділу

У другому розділі виконано повне проєктування вебплатформи для організації волонтерської діяльності — від визначення варіантів використання до реалізації ключових алгоритмів.

На етапі аналізу варіантів використання визначено чотири актори системи та побудовано діаграму у нотації UML 2.0. Для кожної ролі сформовано перелік доступних сценаріїв: волонтер отримує інструменти пошуку та реєстрації на заходи, управління завданнями і відгуками; організатор — повний цикл управління заходом від створення до завершення; адміністратор — засоби модерації та управління обліковими записами. Об'єктно-орієнтована модель описує дев'ять основних класів системи з їхніми атрибутами та взаємозв'язками.

Спроектована реляційна база даних на основі PostgreSQL 16 містить 19 логічних груп таблиць. Для забезпечення цілісності даних налаштовано каскадне видалення для композиційних залежностей та стратегію SET NULL для слабких посилань. На ключових полях створено індекси для підвищення продуктивності запитів.

Для опису алгоритмів роботи системи розроблено вісім діаграм у нотації UML 2.0: дві діаграми активності — для процесу автентифікації та реєстрації на захід і для управління заходом організатором; дві діаграми послідовності — для управління учасниками і завданнями та для обробки платежів через Stripe; дві діаграми станів — для життєвого циклу заходу і статусів заявки учасника; діаграму розгортання системи.

Реалізація ключового функціоналу охоплює механізм реєстрації на захід із перевіркою бізнес-правил, управління завданнями з журналуванням змін, дворівневу систему відгуків з автоматичним перерахунком рейтингу та систему досвіду й рівнів волонтера.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		55

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБПЛАТФОРМОЮ ДЛЯ ОРГАНІЗАЦІЇ ВОЛОНТЕРСЬКОЇ ДІЯЛЬНОСТІ

3.1. Порядок встановлення та налаштування параметрів платформи

Вебплатформа для організації волонтерської діяльності має трирівневу архітектуру, розділену на три незалежні компоненти: клієнтська частина на базі React, серверна частина на Node.js/Express та база даних PostgreSQL. Кожен компонент контейнеризований у власному Docker-контейнері, що забезпечує ізольоване середовище виконання та однакову поведінку системи незалежно від операційної системи сервера. Управління усіма контейнерами виконується через docker-compose, що дозволяє запуснути повністю працездатну платформу однією командою.

Для кращого розуміння процесу розгортання нижче наведено діаграму розгортання системи (рис. 3.1).

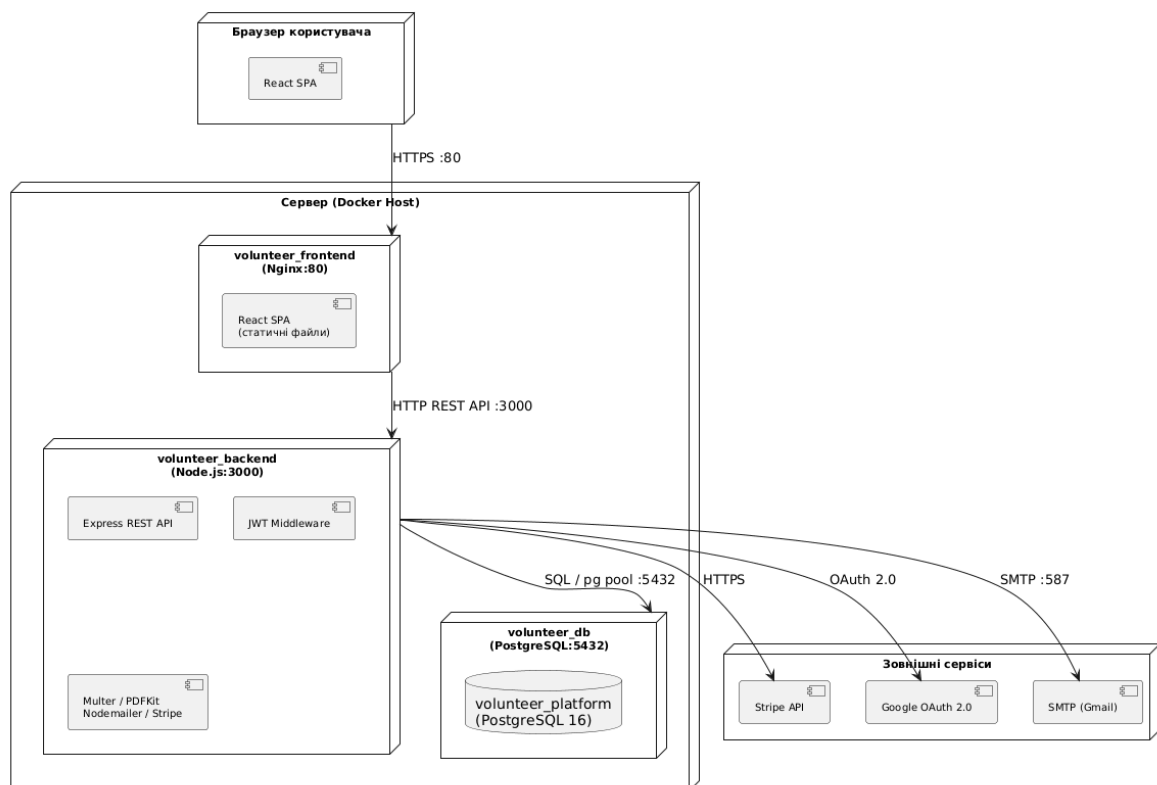


Рис. 3.1. Діаграма розгортання вебплатформи

Вимоги до серверного середовища. Для розгортання платформи необхідно підготувати сервер із такими характеристиками: операційна

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		56

система Linux (Ubuntu 20.04 LTS або новіша) або Windows Server 2019+ із підтримкою Docker; процесор з тактовою частотою не менше 1 ГГц (рекомендовано двоядерний); не менше 2 ГБ оперативної пам'яті для одночасної роботи всіх трьох контейнерів; не менше 10 ГБ вільного дискового простору для образів Docker та бази даних; мережевий адаптер зі швидкістю з'єднання не менше 100 Мбіт/с та відкриті порти 80 (HTTP) і 443 (HTTPS). На сервері обов'язково повинні бути встановлені Docker Engine версії 20.10 або новіший та Docker Compose версії 2.0 або новіший. Для клієнтської сторони достатньо сучасного браузера — Chrome 90+, Firefox 90+, Safari 14+ або Edge 90+ — без встановлення будь-якого додаткового програмного забезпечення.

Підготовка змінних оточення. Перед розгортанням необхідно підготувати файл змінних оточення. У корені проєкту розміщено файл `.env.example` із переліком усіх необхідних параметрів. Його слід скопіювати у файл `.env` та заповнити актуальними значеннями. Перелік змінних наведено у табл. 3.1.

Таблиця 3.1

Змінні оточення вебплатформи

Змінна	Призначення
DB_HOST, DB_PORT, DB_NAME	Параметри підключення до PostgreSQL
DB_USER, DB_PASSWORD	Облікові дані бази даних
PORT	Порт серверної частини (за замовчуванням 3000)
JWT_SECRET	Секретний ключ для підпису JWT-токенів
FRONTEND_URL	URL клієнтської частини (для CORS)
VITE_API_URL	URL API для клієнтської частини
EMAIL_HOST, EMAIL_PORT, EMAIL_USER, EMAIL_PASS	Параметри SMTP для Nodemailer
GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET	Ключі Google OAuth 2.0
GOOGLE_CALLBACK_URL	URL зворотного виклику Google OAuth
STRIPE_SECRET_KEY	Секретний ключ Stripe для обробки платежів

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижомотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		57

Значення JWT_SECRET рекомендується генерувати як випадковий рядок довжиною не менше 64 символів. Для отримання ключів Google OAuth необхідно створити проєкт у Google Cloud Console, увімкнути Google+ API та зареєструвати URI перенаправлення, що відповідає значенню GOOGLE_CALLBACK_URL. Ключі Stripe отримуються в особистому кабінеті розробника на сайті Stripe після реєстрації акаунту; для тестового середовища використовуються тестові ключі з префіксом sk_test_ та pk_test_, що не ініціюють реальних транзакцій.

Структура docker-compose.yml. Файл конфігурації docker-compose.yml визначає три сервіси та їхні взаємозв'язки. Сервіс postgres використовує офіційний образ postgres:16-alpine, монтує іменований том postgres_data для збереження даних та підключає SQL-скрипт ініціалізації з директорії database/ як init-файл. Сервіс backend збирається з Dockerfile у директорії backend/, отримує змінні оточення з файлу .env та залежить від сервісу postgres з умовою service_healthy — запускається лише після успішного проходження healthcheck бази даних. Healthcheck виконує команду pg_isready кожні п'ять секунд із трьома спробами та затримкою старту 10 секунд. Сервіс nginx збирається з Dockerfile у директорії frontend/, відображає порт 80 хоста на порт 80 контейнера та залежить від сервісу backend. Усі три сервіси підключено до спільної внутрішньої мережі volunteer_net із драйвером bridge, що забезпечує комунікацію між контейнерами за іменами сервісів та ізолює їх від зовнішніх мереж.

Конфігурація Nginx. Nginx виконує дві функції: роздачу статичного React-білду та проксіювання API-запитів на бекенд. Конфігураційний файл nginx.conf визначає єдиний блок server на порту 80. Усі запити до маршрутів з префіксом /api/ та /auth/ перенаправляються через директиву proxy_pass на адресу http://backend:3000, де backend — ім'я сервісу у внутрішній мережі Docker. Для коректної роботи SPA-маршрутизації React Router налаштовано директиву try_files: при відсутності запитованого файлу Nginx повертає index.html, дозволяючи React Router самостійно обробити URL на стороні

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		58

клієнта. Заголовки `proxy_set_header` передають оригінальний IP-адрес клієнта та протокол з'єднання на бекенд для коректного формування посилань у email-сповіщеннях.

Ініціалізація бази даних. При першому запуску контейнера `postgres` автоматично виконується SQL-скрипт `volunteer_platform.sql` з директорії `database/`. Скрипт створює всі необхідні перелічувані типи (`user_role`, `event_status`, `participant_status`, `task_status`, `notification_type`), таблиці з обмеженнями цілісності, зовнішніми ключами та індексами. Порядок створення таблиць враховує залежності між ними: спочатку створюються незалежні таблиці (`users`, `categories`, `sponsors`), потім залежні (`events`, `event_participants`, `tasks` тощо). Індеси створюються після таблиць для підвищення продуктивності запитів за найчастіше використовуваними полями. При повторних запусках контейнера `init`-скрипт не виконується повторно, оскільки PostgreSQL перевіряє наявність даних у томі `postgres_data`.

Процес розгортання. Розгортання через `Docker Compose` складається з послідовних кроків, наведених у табл. 3.2.

Таблиця 3.2

Послідовність розгортання платформи

Крок	Команда / дія	Опис
1	<code>git clone <repo-url></code>	Клонування репозиторію проєкту
2	<code>cp .env.example .env</code>	Створення файлу змінних оточення
3	Редагування <code>.env</code>	Заповнення всіх змінних актуальними значеннями
4	<code>docker-compose up --build -d</code>	Збірка образів та запуск контейнерів у фоновому режимі
5	<code>docker-compose logs -f</code>	Перегляд журналів запуску для перевірки відсутності помилок
6	<code>node backend/scripts/createAdmin.js</code>	Створення першого адміністратора платформи

Прапорець `--build` примусово перезбирає образи навіть за наявності кешованих шарів, що гарантує застосування останніх змін у коді. Прапорець `-d` запускає контейнери у фоновому (`detached`) режимі, звільняючи термінал.

Після запуску платформа доступна за адресою `http://localhost` або за IP-адресою сервера на порту 80.

Створення адміністратора. Після першого розгортання необхідно створити обліковий запис адміністратора для доступу до адміністративної панелі. Для цього передбачено окремий скрипт `backend/scripts/createAdmin.js`, який приймає ім'я, електронну пошту та пароль через аргументи командного рядка, хешує пароль алгоритмом `bcrypt` і записує запис у таблицю `users` із роллю `admin`. Скрипт підключається до бази даних через той самий пул з'єднань, що й основний сервер, тому змінні оточення повинні бути доступні в середовищі виконання.

CI/CD пайплайн. Для автоматизації процесів збірки, тестування та розгортання налаштовано CI/CD пайплайн на базі GitLab CI [24]. Пайплайн містить чотири стадії, що виконуються послідовно при кожному `push` у репозиторій. На стадії `build` збираються залежності бекенду та фронтенду, артефакти зберігаються для наступних стадій. На стадії `test` запускаються модульні тести бекенду із збором покриття коду, інтеграційні тести API та перевірка якості коду фронтенду через ESLint. На стадії `docker` виконується збірка Docker-образів із тегуванням за коротким хешем коміту. На стадії `deploy` виконується розгортання на staging-середовище при злитті у гілку `main`. Стадії CI/CD пайплайну наведено у табл. 3.3.

Таблиця 3.3

Стадії CI/CD пайплайну

Стадія	Завдання	Умова запуску
build	Збірка залежностей бекенду та фронтенду	Будь-яка гілка
test	Модульні тести, інтеграційні тести, ESLint	Будь-яка гілка
docker	Збірка Docker-образів	Гілки <code>main</code> , <code>develop</code>
deploy	Розгортання на staging	Лише гілка <code>main</code>

Клієнтська частина є кросбраузерною та коректно відображається у Google Chrome, Mozilla Firefox, Microsoft Edge і Safari. Для роботи платформи

на клієнтській стороні не потрібне встановлення додаткового програмного забезпечення — достатньо сучасного браузера з підтримкою ES2020.

3.2. Структура інтерфейсу вебплатформи

Інтерфейс вебплатформи волонтерської діяльності розроблений на основі принципів адаптивного дизайну та орієнтований на зручність роботи як волонтерів, так і організаторів. Усі сторінки побудовані на базі компонентів React 18 із використанням бібліотеки React Router DOM 7 для клієнтської маршрутизації [16]. Загальна структура застосунку передбачає постійне відображення навігаційного заголовка, а саме компонент Navbar та Footer на всіх сторінках, окрім чату, де застосовується власний повноекранний макет. Залежно від ролі користувача (volunteer, organizer, admin) частина маршрутів захищена компонентами PrivateRoute та AdminRoute, які перенаправляють неавторизованих користувачів на сторінку входу.

Головна сторінка (маршрут «/») є точкою входу для незареєстрованих відвідувачів і слугує оглядовим інструментом для постійних користувачів. Вона містить банер із коротким описом місії платформи та заклик до дії — кнопку переходу до каталогу подій. Нижче розміщено блок «Найближчі події», де відображаються картки трьох-чотирьох актуальних заходів із зазначенням назви, категорії, дати початку, місця проведення та кількості підтверджених учасників. Картки подій реалізовані як компонент EventCard: при натисканні здійснюється перехід на сторінку деталей обраної події. Для заходів, що розпочинаються впродовж тижня, автоматично відображається позначка «🔥 N днів» для привернення уваги. Блок партнерів і спонсорів показує логотипи організацій зі збереженими посиланнями на їхні вебсайти; якщо логотип відсутній, замість нього відображаються ініціали організації у кольоровому колі. Окремий блок «Новини» містить перелік публікацій із датами та можливістю переходу до повного тексту. Нижня частина головної сторінки включає блок збору коштів для благодійних потреб — компонент CharityNeedsBlock, який відображає поточний прогрес зібраної суми у вигляді

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		61

прогрес-бару. Зовнішній вигляд головної сторінки платформи наведено на рис. 3.2.

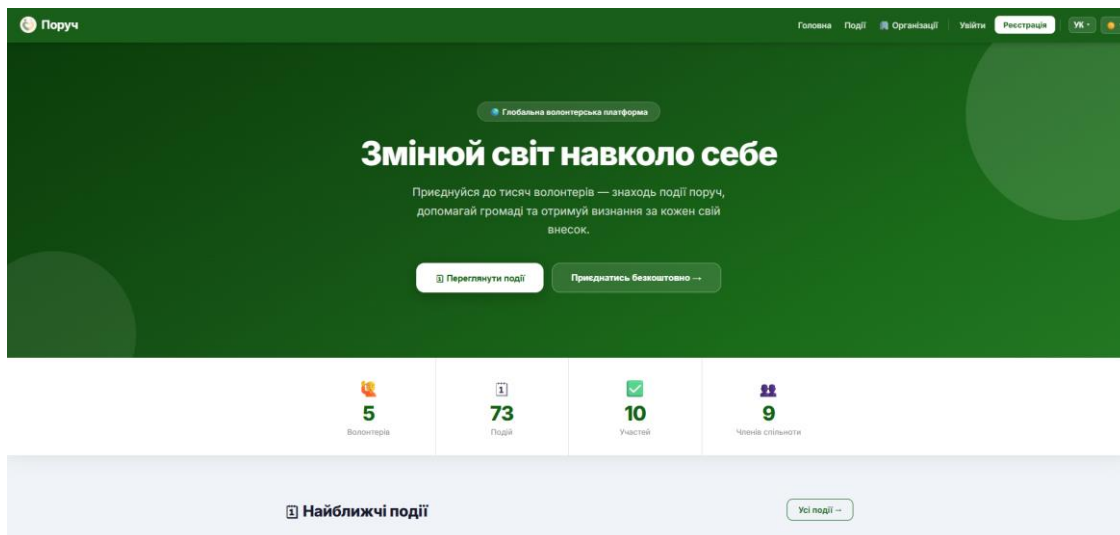


Рис. 3.2. Головна сторінка вебплатформи

Сторінка реєстрації (маршрут «/register») містить форму для введення імені, електронної пошти та пароля, що зображено на рис. 3.3.

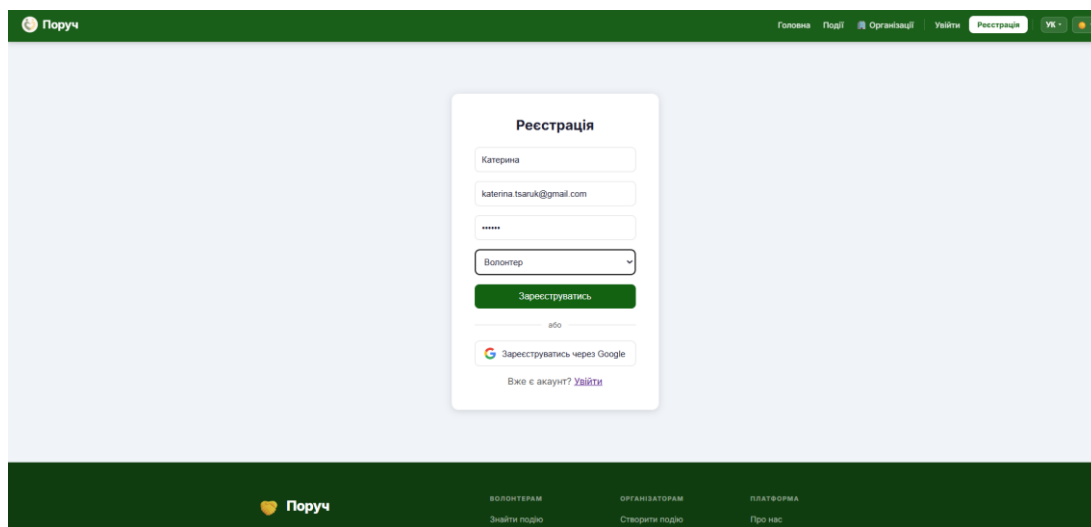


Рис. 3.3. Сторінка реєстрації

Пароль вводиться двічі для підтвердження, а система виконує клієнтську валідацію на відповідність обох полів і мінімальну довжину. Нижче основної форми розміщено посилання на вхід для вже зареєстрованих користувачів. Після успішної реєстрації користувач автоматично перенаправляється на сторінку каталогу подій. Сторінка входу (маршрут «/login») пропонує стандартну форму з полями електронної пошти та пароля, а також кнопку «Увійти через Google», яка ініціює OAuth 2.0-потік через Passport.js. Після

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижевська О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		62

успішної аутентифікації через Google відбувається перенаправлення на маршрут «/auth/callback», де компонент AuthCallbackPage зчитує JWT-токен із параметрів URL та зберігає його у контексті застосунку. Форму входу до системи зображено на рис. 3.4.

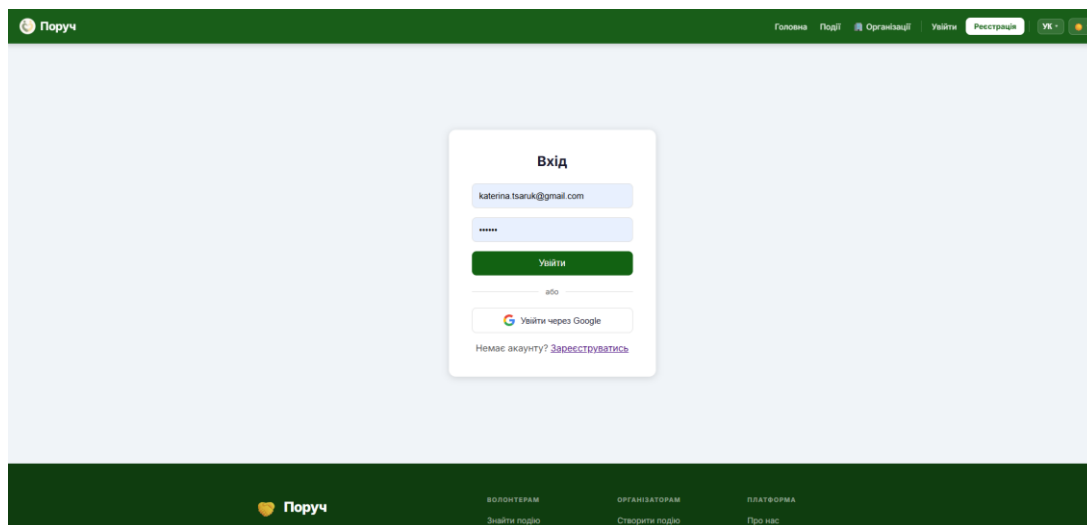


Рис. 3.4. Сторінка входу з підтримкою Google OAuth

Каталог подій (маршрут «/events») є центральним функціональним розділом платформи, де реалізовано розширену систему фільтрації та два режими перегляду. Панель фільтрів, що розгортається по кнопці, дозволяє комбінувати пошук за ключовим словом у назві та описі, фільтрацію за категорією, країною проведення, діапазоном дат початку, а також сортування за датою або популярністю. Результати пошуку можна зберегти під довільною назвою у локальному сховищі браузера та повторно застосувати в будь-який момент. Перемикач вкладок «Активні / Завершені / Скасовані» дозволяє переглядати події за поточним статусом. У режимі «список» події відображаються у вигляді карток із ключовою інформацією; у режимі «карта» завантажується компонент EventMap на основі бібліотеки Leaflet із маркерами на інтерактивній карті — при натисканні на маркер відкривається спливаючий блок із назвою та посиланням на деталі події [27]. Якщо поточний користувач має роль організатора, у верхній частині сторінки відображається кнопка «Створити подію», яка відкриває форму створення нового заходу з полями назви, категорії, опису, місця проведення, дат, максимальної кількості

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		63

волонтерів, переліку необхідних навичок та параметрів збору донатів. Каталог подій із панеллю фільтрації зображено на рис. 3.5.

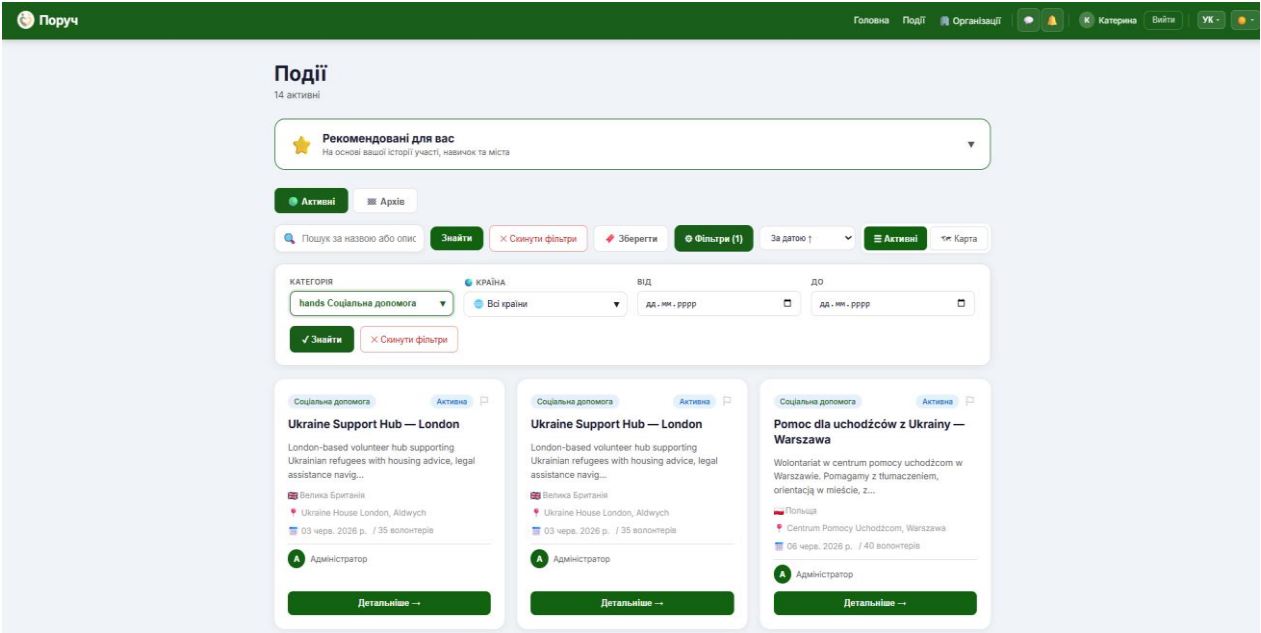


Рис. 3.5. Каталог подій із панеллю фільтрації

Сторінка деталей події (маршрут «/events/:id») відображає повну інформацію про обраний захід. У верхній частині розміщені заголовок, категорія, статусна позначка (draft / published / active / completed / cancelled), дати початку та завершення, місце проведення з посиланням на карту, ім'я організатора із переходом на його профіль. Для авторизованих користувачів відображається кнопка «Зареєструватися» або «Скасувати реєстрацію», а статус поточної реєстрації (pending / confirmed / rejected / attended) виводиться у вигляді кольорового бейджа. Блок «Завдання» показує список завдань заходу, впорядкований за пріоритетом і статусом; волонтери можуть брати завдання до виконання, а організатор — змінювати їхній статус. Якщо організатор увімкнув збір пожертв для події, під основним блоком відображається компонент DonationBlock із цільовою сумою, прогрес-баром поточного збору та формою для Stripe-платежу. У нижній частині сторінки міститься розділ відгуків: після відвідування заходу волонтер може залишити оцінку від 1 до 5 зірок та текстовий коментар; відгуки інших учасників відображаються у вигляді компонентів ReviewCard із можливістю поставити

реакцію «корисно» або «не корисно». Вигляд сторінки деталей події наведено на рис. 3.6 та рис. 3.7.

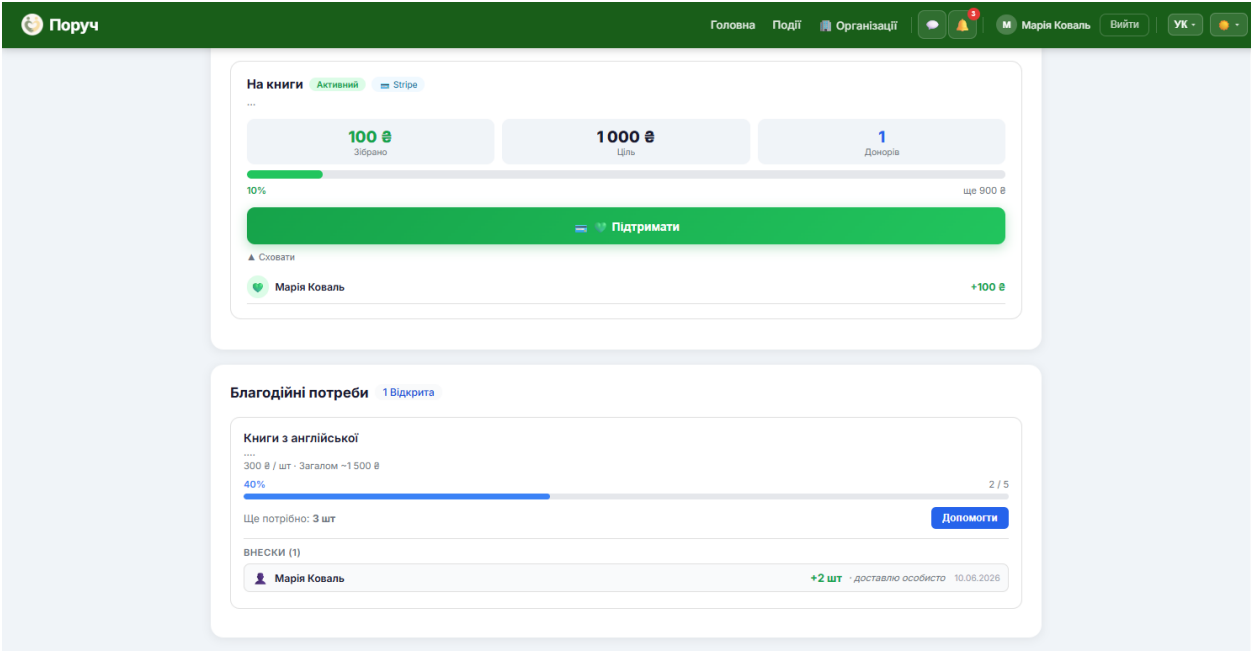


Рис. 3.6. Сторінка деталей події із блоком донатів

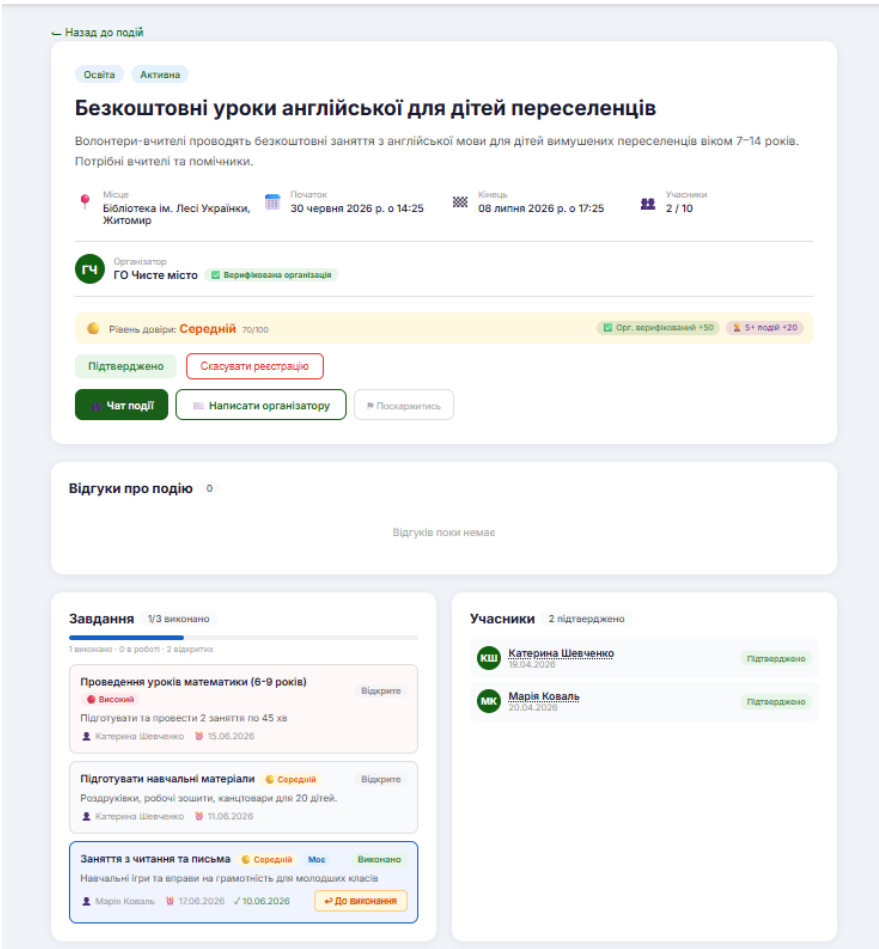


Рис. 3.7. Сторінка деталей події із завданнями

Блок управління завданнями є окремим функціональним модулем сторінки деталей події. Кожне завдання має назву, опис, пріоритет трьох рівнів (high / medium / low) та статус (open / in_progress / pending_approval / completed / cancelled). Сервер повертає завдання відсортованими спочатку за пріоритетом — від найвищого до найнижчого, а в межах однакового пріоритету — за статусом активності та дедлайном. Волонтер може взяти відкрите завдання до виконання, після чого воно переходить у статус in_progress і закріплюється за ним. Після виконання завдання волонтер надсилає його на перевірку зі статусом pending_approval, і організатор підтверджує або відхиляє результат. Організатор також має змогу створювати нові завдання безпосередньо зі сторінки події, призначати їх конкретним волонтерам та встановлювати дедлайни. Після підтвердження виконання завдання сервер автоматично нараховує ХР учасникові, а якщо завдання виконано до дедлайну — додає бонусний досвід. Коли всі завдання заходу переведено у статус completed, організатор отримує сповіщення про можливість дострокового завершення події. Інтерфейс управління завданнями із пріоритетами зображено на рис. 3.8.

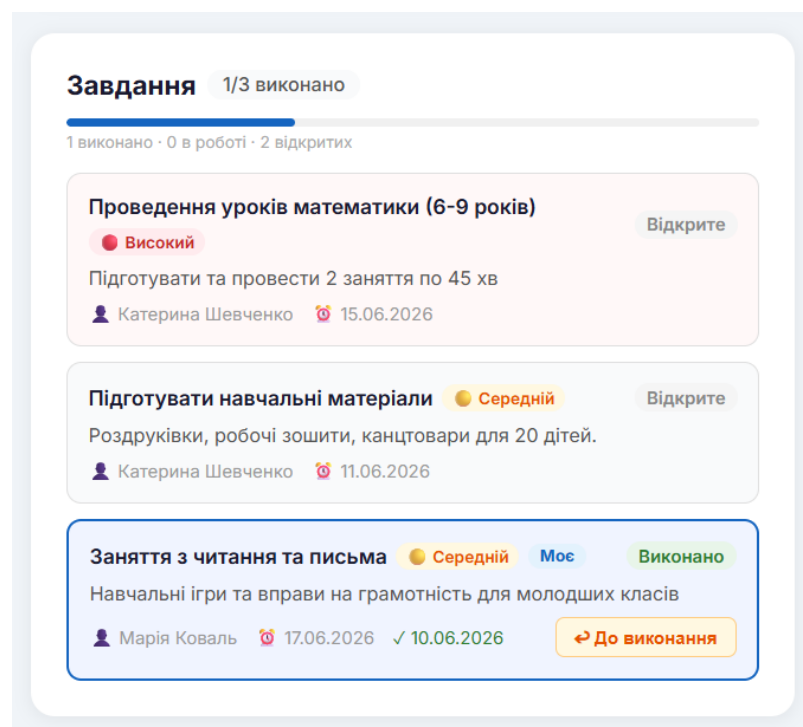


Рис. 3.8. Блок завдань із пріоритетами та статусами

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		66

Процес збору пожертв реалізовано через компонент DonationBlock, який з'являється на сторінці події, якщо організатор увімкнув цю опцію при створенні заходу. Блок відображає назву збору, цільову суму, суму вже зібраних коштів у вигляді прогрес-бару та перелік останніх донатів із іменами жертводавців. Процес внесення пожертви відбувається у три кроки через модальне вікно. На першому кроці користувач обирає або вводить довільну суму (мінімум 25 гривень для Stripe-платежів), вказує необов'язкове повідомлення та може позначити донат як анонімний.

Рис. 3.9. Крок 1: вибір суми донату

На другому кроці відображається форма оплати від Stripe із полями номера картки, терміну дії та CVV-коду — платіжні дані передаються напряму на сервери Stripe без збереження на платформі (рис. 3.10).

На книги

Сума донату
100 €

Stripe · Visa · Mastercard · Google Pay

Secure, fast checkout with Link ✓

Card number
4242 4242 4242 4242

Expiration (MM/YY)
12 / 26

Security code
123

Country
Ukraine

Optional
Save my information for faster checkout

Email
katerina.tsaruk06@gmail.com

Mobile number
067 123 4567

Full name
Катерина Царук

link · By providing phone number and email, you agree to create an account subject to [Terms](#) and [Privacy Policy](#).

← Назад Оплатити 100 €

Захищено Stripe · SSL шифрування

Рис. 3.10. Крок 2: форма оплати Stripe

Після підтвердження оплати Stripe надсилає webhook-подію на сервер, який верифікує підпис та фіксує донат у базі даних зі статусом completed, після чого організатор отримує відповідне сповіщення (рис. 3.11).

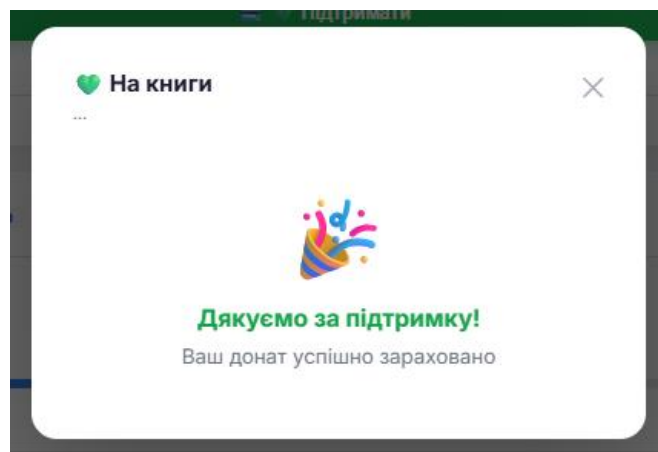


Рис. 3.11. Крок 3: підтвердження успішного платежу

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		68

Особистий кабінет волонтера (маршрут «/profile») є персоналізованим розділом, доступним лише авторизованим користувачам. У верхній частині сторінки розміщено компонент UserAvatar із аватаром або згенерованими ініціалами, ім'я, електронна пошта, роль і рейтинг профілю. Виджет XPWidget відображає поточний рівень користувача (Новачок / Активіст / Досвідчений / Ветеран / Експерт), накопичену кількість досвіду та прогрес-бар до наступного рівня. Нижче розміщено вкладкову навігацію між розділами: «Мої події» — перелік заходів, на які зареєстрований користувач, з кольоровими статусами реєстрації; «Організовані події» — доступний для організаторів список власних заходів із можливістю переходу до управління кожним; «Відгуки» — перелік залишених відгуків із оцінками зірочками; «Збережені» — заходи, додані до закладок; «Налаштування» — форма редагування профілю (ім'я, біо, аватар, пароль). Особистий кабінет волонтера зображено на рис. 3.12.

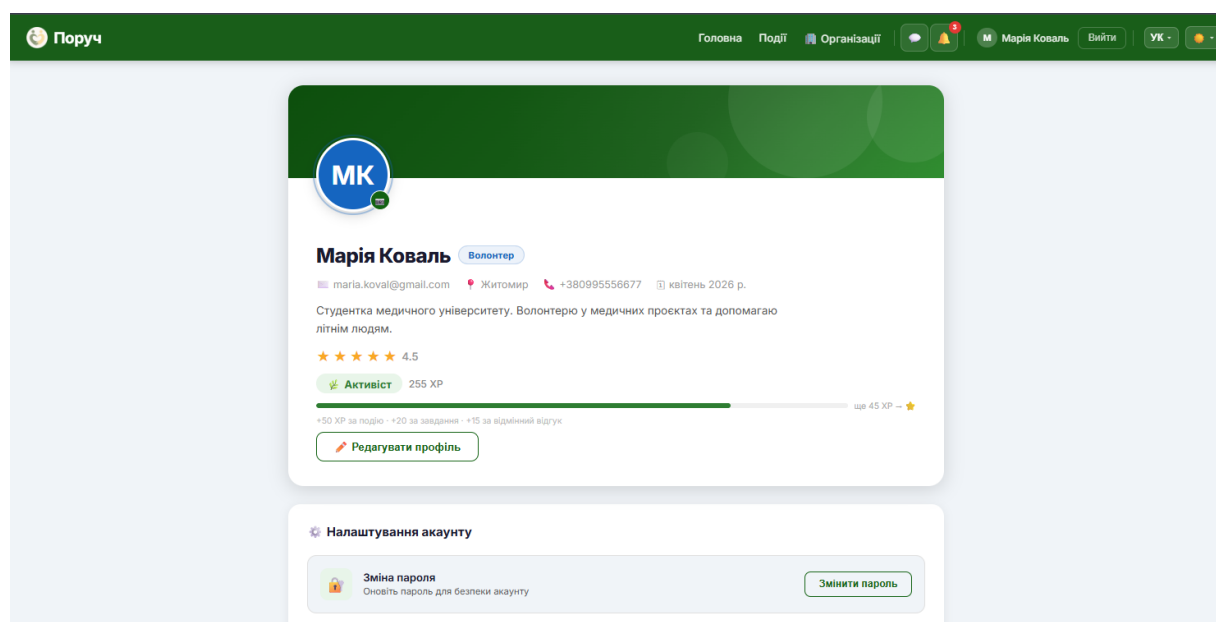


Рис. 3.12 — Особистий кабінет волонтера

Для підтвердження участі у заходах реалізовано функцію генерації PDF-сертифікатів. Після завершення заходу волонтер зі статусом attended може завантажити персональний сертифікат, натиснувши відповідну кнопку на сторінці деталей події або в особистому кабінеті. Серверний контролер certificateController формує документ засобами бібліотеки PDFKit: на сертифікаті розміщуються логотип платформи, ім'я волонтера, назва заходу,

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		69

дати проведення, місце, ім'я організатора та кількість виконаних завдань [29]. Документ оформлено декоративними лініями з ромбоподібними акцентами та містить усі дані у форматі, придатному для друку. Файл формується динамічно на сервері та передається браузеру як потік із заголовком Content-Disposition: attachment, що ініціює автоматичне завантаження. Зразок згенерованого PDF-сертифіката наведено на рис. 3.13.

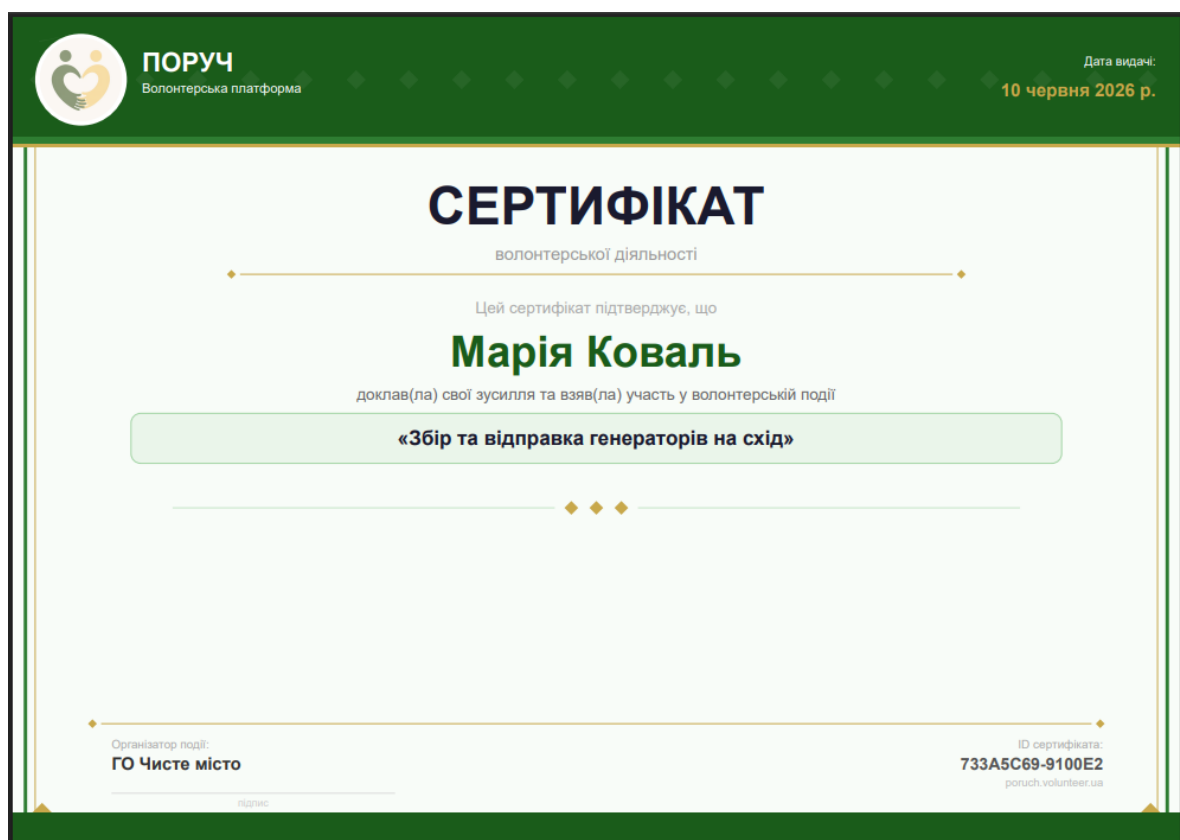


Рис. 3.13. PDF-сертифікат волонтера

Сторінка публічного профілю користувача (маршрут «/users/id») доступна без авторизації та відображає ім'я, аватар, рівень у системі XP, рейтинг та список публічних відгуків. Авторизований відвідувач може надіслати скаргу на профіль через компонент ReportModal, обравши причину зі стандартизованого переліку: неправдива інформація, шахрайство, образливий контент та інші. Сервер перевіряє відсутність дублікату активної скарги від того самого користувача на той самий об'єкт і у разі успіху сповіщає всіх адміністраторів платформи. Аналогічна форма скарги доступна і для подій. Форму подачі скарги зображено на рис. 3.14.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		70

Рис. 3.14. Форма подачі скарги на користувача або подію

Організатори мають можливість подати заявку на отримання статусу верифікованого партнера через особистий кабінет. Форма верифікації дозволяє завантажити підтверджувальні документи (статут організації, реєстраційні свідоцтва тощо) через Multer. Подана заявка отримує статус pending і відображається у адміністративній панелі на вкладці «Запити на верифікацію». Адміністратор переглядає завантажені документи та приймає рішення: схвалити заявку, що встановлює поле `verification_status` користувача у значення `verified`, або відхилити із зазначенням причини. В обох випадках організатор отримує сповіщення із результатом перевірки. Верифікований статус відображається на публічному профілі організації у вигляді відповідної позначки. Інтерфейс подачі заявки на верифікацію та її розгляд в адмін-панелі зображено на рис. 3.15 та рис. 3.16.

Рівень довіри ● Низький — 0/100

- ✗ Не верифікована +50
- 📄 Документи відсутні +30
- 📅 Менше 5 завершених подій (0) +20

📅 Мої події 0
 Заявки
 📊 Статистика
 ⭐ Відгуки 0
 Верифікація

🚩 Мої скарги

Статус верифікації

Не верифікований

Подати заявку на верифікацію

Завантажте документи, що підтверджують діяльність вашої організації. Адміністратор перевірить їх і надасть статус верифікованої організації.

+ Додати документ
 Подати заявку

Рис. 3.15. Подача заявки на верифікацію організації

Користувачі
 Події
 Партнери
 Скарги
 Підтримка подій
 Верифікація
 Донати
 Зв'язок
 Новини

Очікують
 Підтверджені
 Відхилені
 Всі

БФ Разом сильніші
 fond.razom@gmail.com
 Житомир

Підтверджені
Подано: 01 червня 2026 р.

ПОДАНІ ДОКУМЕНТИ (1)

certificate-733a5c69-592e-4a21-a54c-7ae1ce12588e (1).pdf
 Документ реєстрації організації

Відкрити

Скасувати верифікацію

Рис. 3.16. Розгляд заявок на верифікацію в адміністративній панелі

Форма зворотного зв'язку доступна всім відвідувачам платформи без необхідності реєстрації та розміщена у підвалі сайту. Користувач заповнює поля імені, електронної пошти, теми звернення та тексту повідомлення. Після відправки запис зберігається у таблиці feedback бази даних, а всі адміністратори отримують сповіщення про нове звернення з посиланням на

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		72

відповідну вкладку адмін-панелі. Адміністратори переглядають повідомлення у хронологічному порядку та можуть позначати їх як опрацьовані. Форму зворотного зв'язку зображено на рис. 3.17.

Зворотний зв'язок

Маєте питання чи пропозицію? Будемо раді почути вас!

Ваше ім'я *

Email *

Тема

Повідомлення *

Рис. 3.17. Форма зворотного зв'язку

Система сповіщень є невід'ємною частиною взаємодії користувача з платформою. Навігаційний заголовок відображає значок дзвоника з числовим індикатором кількості непрочитаних сповіщень, при натисканні на який розкривається випадаючий список останніх подій. Платформа підтримує 26 типів сповіщень, що охоплюють усі ключові події: підтвердження або відхилення реєстрації на захід, зміну статусу завдання, нові повідомлення в чаті, надходження донату, публікацію відгуку, а також системні сповіщення від адміністратора. Паралельно критичні сповіщення дублюються електронним листом через бібліотеку Nodemailer [30]. Вигляд панелі сповіщень наведено на рис. 3.18.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		73

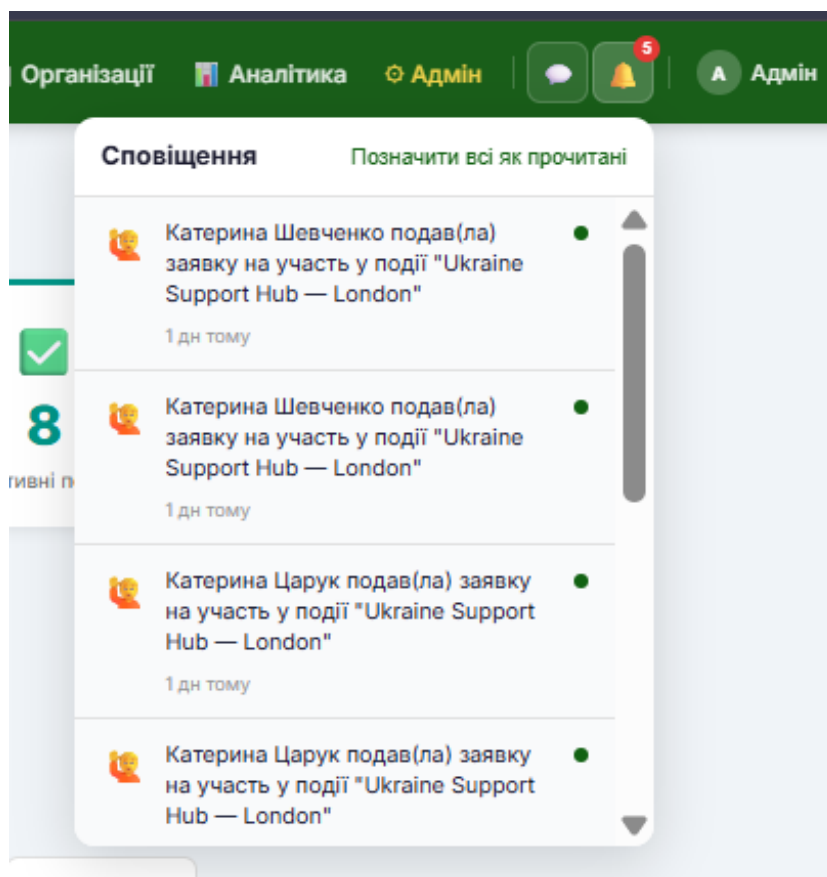


Рис. 3.18. Панель сповіщень користувача

Сторінка каталогу організацій (маршрут «/organizations») містить перелік зареєстрованих на платформі організацій-організаторів з їхніми назвами, описами, посиланнями та кількістю проведених заходів. Фільтрація здійснюється за ключовим словом у пошуковому рядку. Каталог організацій зображено на рис. 3.19.

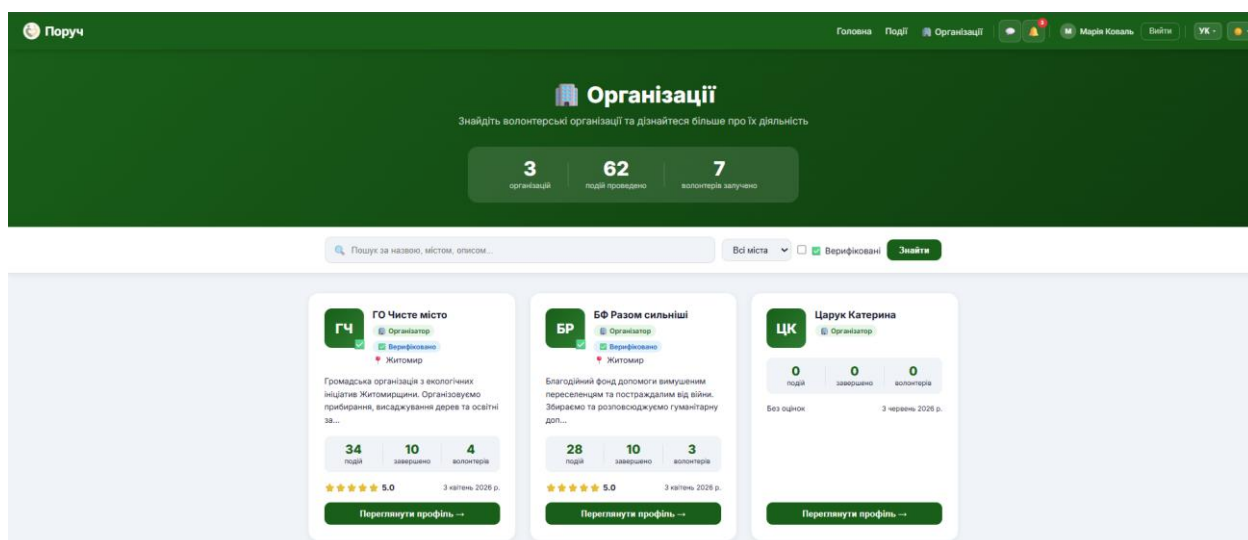


Рис. 3.19. Каталог організацій

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				74
Змн.	Арк.	№ докум.	Підпис	Дата		

Сторінка обміну повідомленнями (маршрут «/messages») реалізована у вигляді повноекранного чату та має власний макет без підвалу. Ліворуч відображається список контактів із превью останнього повідомлення та часом; праворуч — вікно переписки з обраним користувачем. Для нових непрочитаних повідомлень відображається кількісний індикатор. Інтерфейс чату зображено на рис. 3.20.

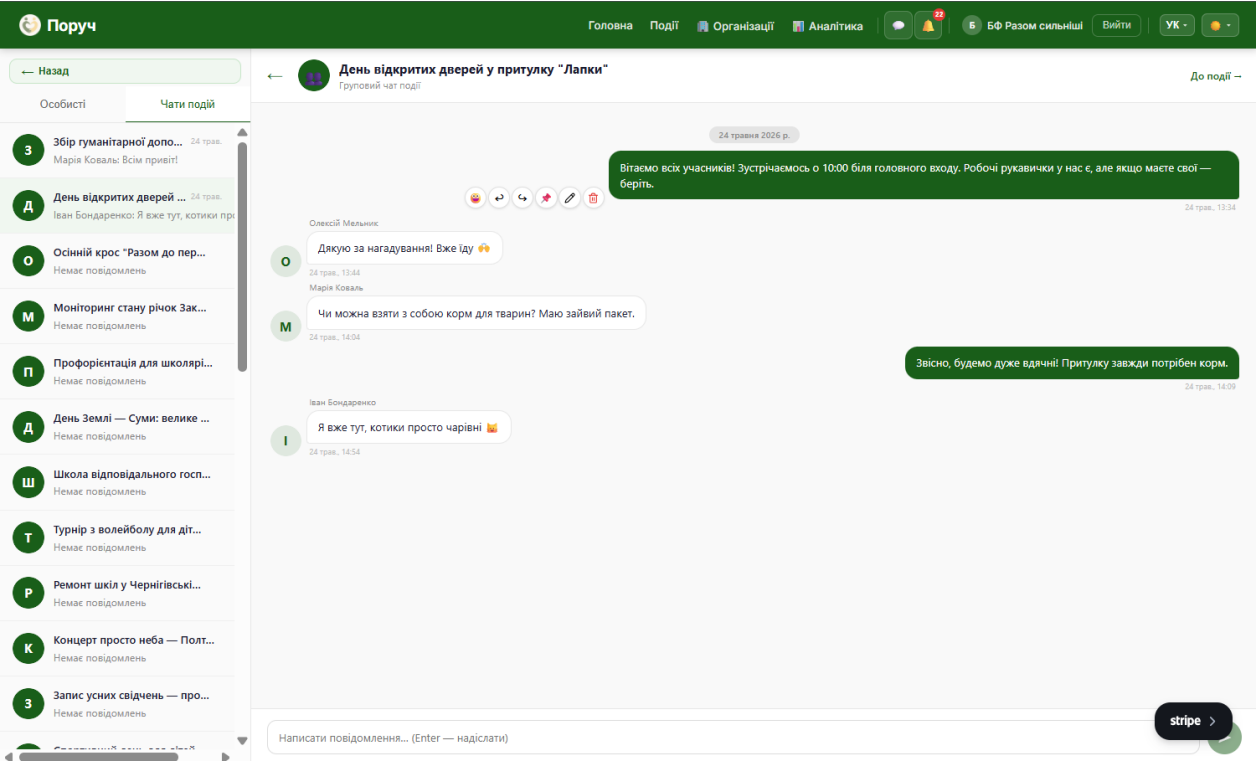


Рис. 3.20. Сторінка обміну повідомленнями

Блок партнерів та спонсорів розміщено на головній сторінці платформи та слугує для відображення організацій, які підтримують волонтерську діяльність. Кожна картка партнера містить логотип або автоматично згенеровані ініціали у кольоровому колі, назву організації та посилання на її вебсайт, яке відкривається у новій вкладці. При наведенні курсору картка піднімається з анімацією для привернення уваги. Адміністратор платформи керує переліком партнерів через адміністративну панель — додає нові організації, завантажує логотипи через Multer та редагує контактні дані. Блок партнерів на головній сторінці зображено на рис. 3.21.

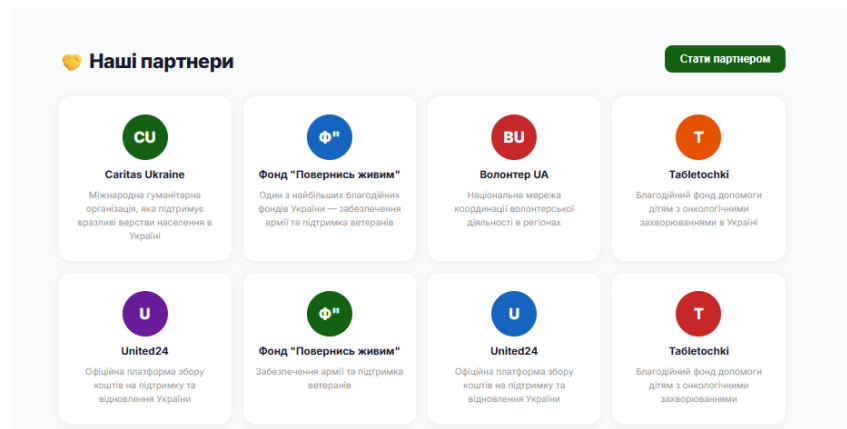


Рис. 3.21. Каталог партнерів

Адміністративна панель (маршрут «/admin») доступна виключно користувачам з роллю admin та захищена компонентом AdminRoute. Панель організована у вигляді вкладкової навігації між розділами: «Користувачі», «Події», «Спонсори», «Скарги», «Запити на верифікацію», «Відгуки», «Новини» та «Статистика». Вкладка «Користувачі» відображає таблицю всіх зареєстрованих акаунтів із можливістю зміни ролі (volunteer / organizer / admin) та видалення облікового запису. Вкладка «Скарги» показує перелік поданих скарг на користувачів або подій із зазначенням причини та дозволяє адміністратору видалити об'єкт скарги або відхилити її. Вкладка «Запити на верифікацію» обробляє заявки організацій на отримання статусу верифікованого партнера. На вкладці «Статистика» за допомогою Recharts відображаються зведені показники платформи: кількість користувачів, подій, донатів у вигляді інтерактивних графіків. Вигляд адміністративної панелі наведено на рис. 3.22.

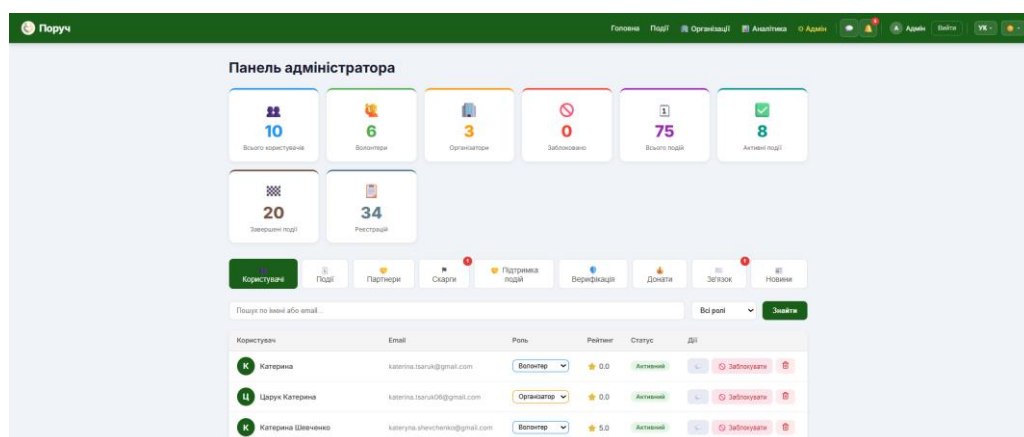


Рис. 3.22. Адміністративна панель платформи

Навігаційний заголовок (компонент NavBar) присутній на всіх сторінках і адаптується залежно від стану авторизації та ролі користувача. Незареєстрованим відвідувачам відображаються посилання «Увійти» та «Реєстрація»; авторизованим — посилання на особистий кабінет, повідомлення, сповіщення та кнопка виходу. Для адміністраторів додатково відображається посилання на адмін-панель. Заголовок також містить перемикачі мови інтерфейсу (контекст LangContext) та теми оформлення (контекст ThemeContext), що дозволяє користувачам вибрати між світлою та темною кольоровою схемою. Сторінка «404 Not Found» (компонент NotFoundPage) відображається для всіх невідомих маршрутів і містить повідомлення про помилку та кнопку повернення на головну сторінку (рис. 3.23).

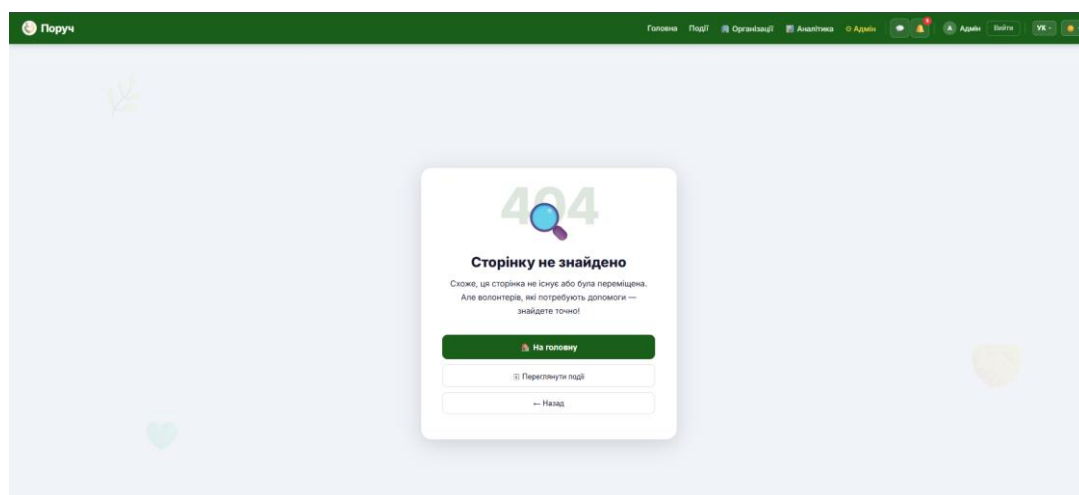


Рис. 3.23. Сторінка «404 Not Found»

Таким чином, інтерфейс платформи охоплює 13 основних сторінок і понад 12 перевикористовуваних компонентів. Архітектура маршрутизації забезпечує розмежування доступу відповідно до ролей, а компонентна структура React дозволяє підтримувати узгоджений стиль та поведінку в межах усього застосунку.

3.3. Статистика та гейміфікація користувачів

Для підвищення залученості учасників та мотивації до регулярної волонтерської діяльності у платформу інтегровано систему гейміфікації на основі досвіду (XP) та рівнів. Механізм нарахування балів реалізовано у

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		77

серверному модулі experience.js, який визначає фіксовані значення винагороди за кожну дію користувача. За відвідування заходу нараховується 50 XP, за виконання завдання — 20 XP, за дострокове виконання завдання — додатково 10 XP, за позитивний відгук від організатора — 15 XP, а за участь у терміновому заході — 30 XP. Нарахування відбувається автоматично після зміни статусу відповідного об'єкта і записується до таблиці users бази даних, де зберігається поточне значення experience кожного користувача.

На основі накопиченого досвіду визначається рівень користувача. Система передбачає п'ять рівнів: «Новачок» (0–99 XP), «Активіст» (100–299 XP), «Досвідчений» (300–599 XP), «Ветеран» (600–999 XP) та «Експерт» (1000 XP і більше). Перехід між рівнями відбувається автоматично при досягненні порогового значення. Поточний рівень і прогрес до наступного відображаються у компоненті XPWidget, що є частиною особистого кабінету та публічного профілю. Віджет показує назву рівня, числове значення XP та прогрес-бар із відсотком заповнення до наступного порогу, що робить прогрес користувача наочним та мотивує до подальшої активності. Вигляд XP-віджету наведено на рис. 3.24.

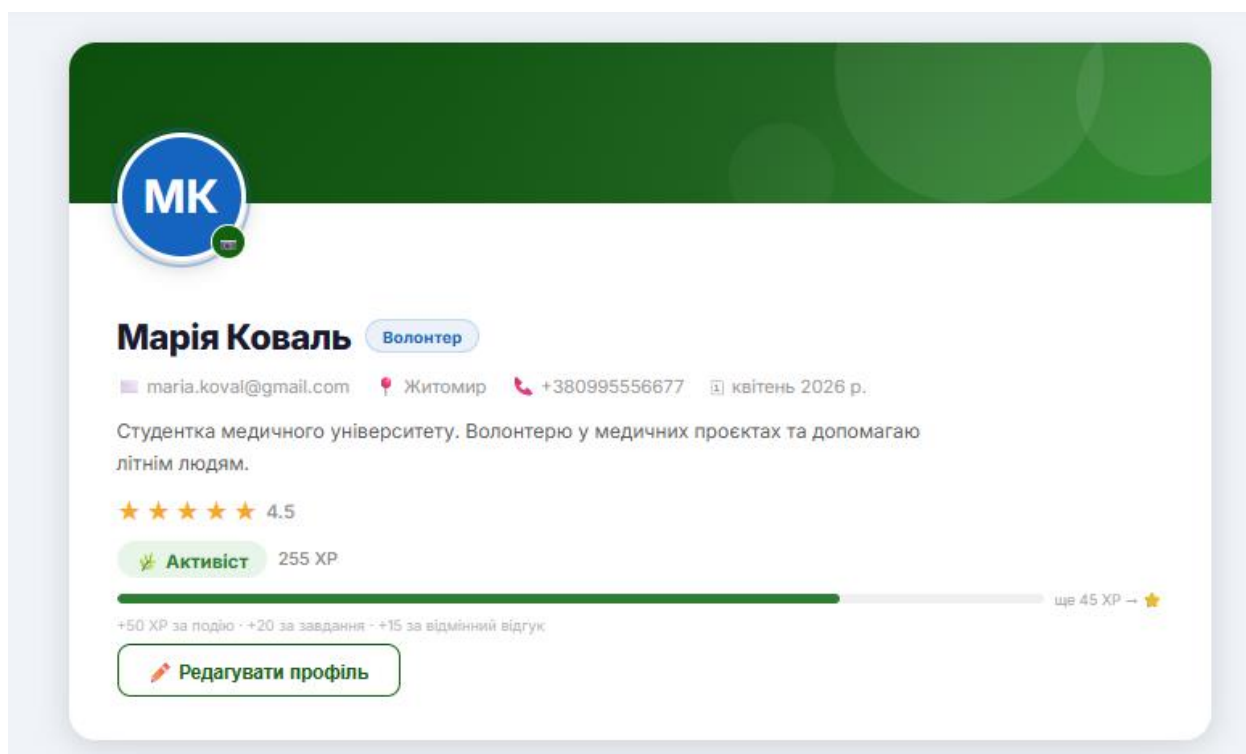


Рис. 3.24. Віджет рівня та досвіду користувача

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		78

Розширена аналітика для організаторів доступна на сторінці «/analytics» та побудована на компонентах Recharts із чотирма аналітичними вкладками. Вкладка «Огляд» містить площинні діаграми (AreaChart) із динамікою реєстрацій та підтверджень волонтерів за останні місяці, а також зведені числові показники: загальну кількість організованих заходів, учасників і донатів. Інформаційна панель дозволяє організатору відстежувати активність платформи в динаміці та порівнювати результати різних заходів між собою (рис. 3.25).

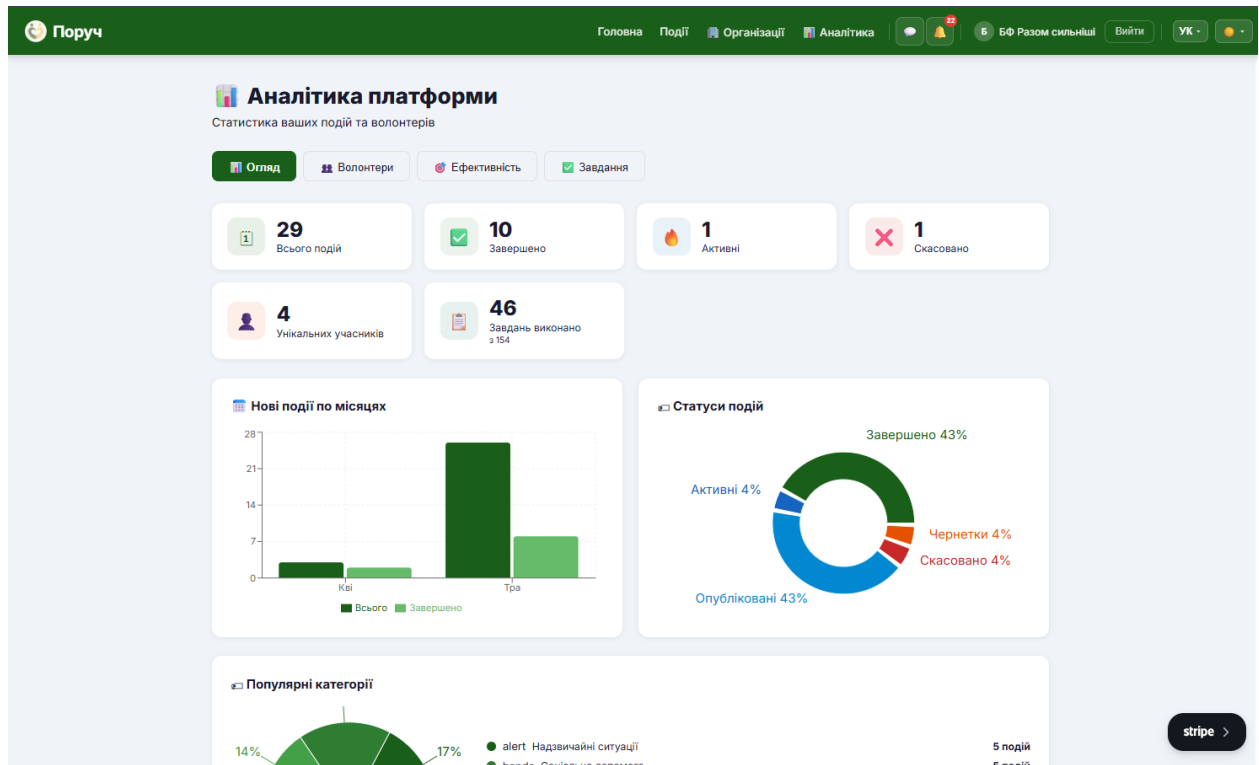


Рис. 3.25. Сторінка аналітики для організатора(Вкладка Огляд)

Вкладка «Волонтери» деталізує розподіл учасників за статусами участі (pending, confirmed, attended, rejected) у форматі стовпчастих діаграм по кожному заходу окремо. Це дозволяє організатору швидко оцінити, який відсоток поданих заявок завершився фактичною участю, та визначити заходи з найнижчою конверсією для подальшого аналізу причин. Окрім діаграм, вкладка відображає загальну кількість волонтерів у кожному статусі у вигляді числових карток, що забезпечує швидкий огляд без необхідності детального аналізу графіків (рис. 3.26).

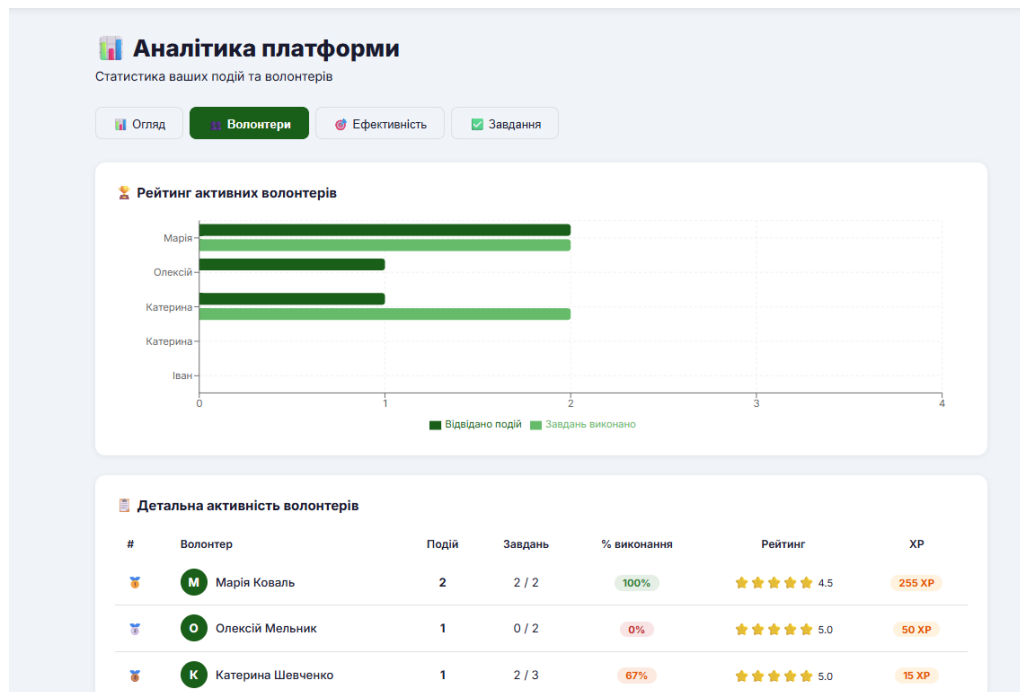


Рис. 3.26. Сторінка аналітики для організатора(Вкладка Волонтери)

Вкладка «Ефективність» відображає середній рейтинг подій на основі відгуків волонтерів, відсоток заповненості від максимальної кількості місць та показник конверсії від реєстрації до фактичного відвідування. Сукупність цих показників дає організатору об'єктивну картину якості проведених заходів і допомагає виявити слабкі місця в організаційному процесі (рис. 3.27).

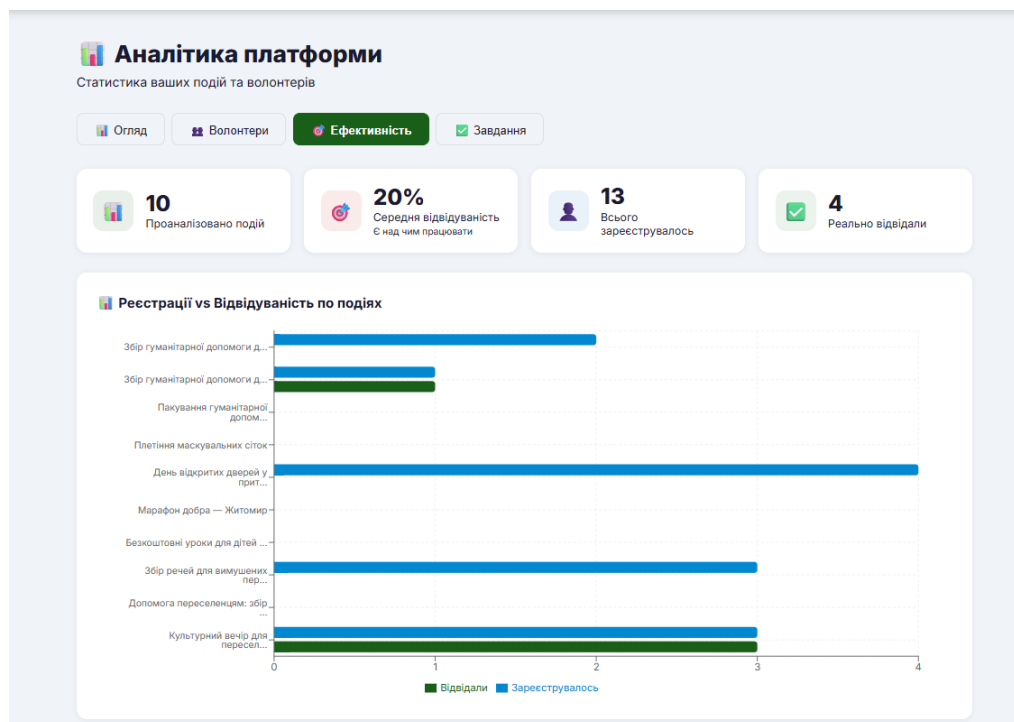


Рис. 3.27. Сторінка аналітики для організатора(Вкладка Ефективність)

Вкладка «Завдання» показує прогрес виконання завдань у розрізі статусів (open, in_progress, completed, cancelled) та пріоритетів. Діаграма дозволяє організатору оцінити загальне завантаження волонтерів, виявити прострочені або незакриті завдання і своєчасно скоригувати розподіл обов'язків між учасниками заходу (рис. 3.28).

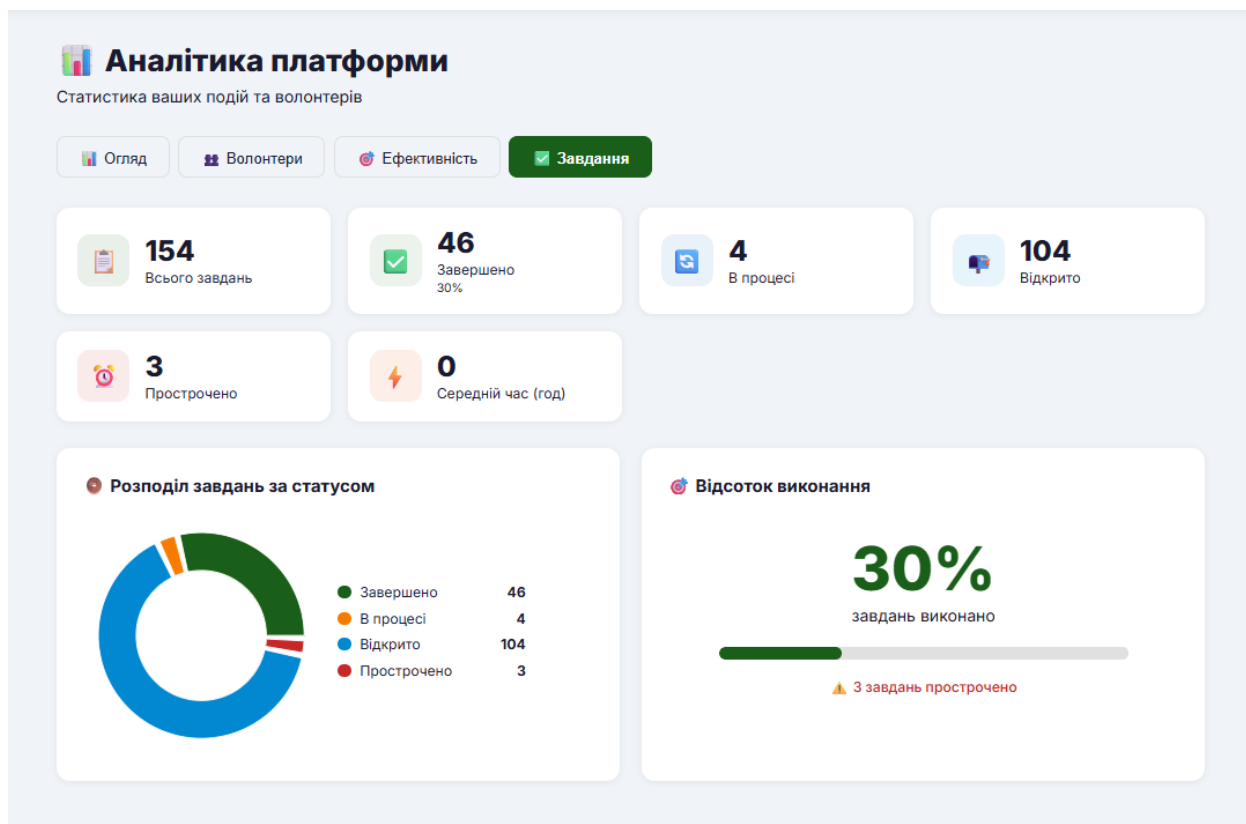


Рис. 3.28. Сторінка аналітики для організатора(Вкладка Завдання)

Поєднання системи ХР та рівнів із детальною особистою статистикою формує у користувача відчуття прогресу та цілеспрямованості. Волонтер бачить не лише перелік відвіданих подій, а й вимірюваний результат своєї участі — рівень та накопичений досвід. Для організаторів аналітична панель слугує інструментом оцінювання якості проведених заходів і планування майбутніх. Завдяки цьому платформа перетворюється на середовище, де волонтерська діяльність стає усвідомленою та відстежуваною, а кожен новий захід додає вимірюваний внесок у розвиток профілю учасника.

3.4. Тестування вебплатформи

Тестування вебплатформи є невід'ємною частиною розробки програмного забезпечення, що дозволяє виявити помилки до впровадження

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		81

продукту в реальне середовище. У процесі розробки вебплатформи волонтерської діяльності було проведено комплексне тестування, яке охоплює автоматизовані юніт-тести, інтеграційні тести, ручне тестування API, перевірку продуктивності та адаптивності інтерфейсу.

Для автоматизованого тестування серверної частини використовується фреймворк Jest у поєднанні з бібліотекою Supertest [25, 26, 38]. Тести розміщено у директорії backend/tests і запускаються автоматично на кожному коміті у рамках CI/CD-конвеєра GitLab на етапі test. Конфігурація Jest у файлі package.json визначає середовище виконання node, шаблон пошуку тестових файлів `**/tests/**/*.test.js` та збір звіту про покриття коду у форматах text, lcov і html.

Юніт-тести модуля `experience.test.js` перевіряють коректність роботи системи нарахування XP та визначення рівнів без підключення до бази даних. Залежність від бази даних замінюється мок-об'єктом через `jest.mock`, що дозволяє тестувати чисту бізнес-логіку ізольовано. Тестовий набір охоплює чотири групи тестів. Перша група «XP constants» перевіряє, що всі значення констант позитивні, що `EVENT_ATTENDED` перевищує `TASK_COMPLETED` та що всі очікувані ключі (`EVENT_ATTENDED`, `URGENT_EVENT`, `TASK_COMPLETED`, `TASK_EARLY`, `POSITIVE_REVIEW`) присутні в об'єкті. Друга група «LEVELS structure» перевіряє, що масив рівнів відсортований за зростанням мінімального порогу XP, що перший рівень починається з нуля та що кожен об'єкт рівня містить поля `name`, `min` та `icon`. Третя група «getLevel» перевіряє граничні значення для кожного рівня: 0 XP повертає «Новачок», 100 XP — «Активіст», 300 XP — «Досвідчений», 600 XP — «Ветеран», 1000 XP та будь-яке значення понад максимум — «Експерт», а виклик без аргументу повертає перший рівень. Четверта група «getNextLevel» перевіряє коректне визначення наступного рівня та повернення null при досягненні максимального рівня «Експерт». Окремо реалізовано сценарний тест «XP progression scenario», який моделює типового волонтера (3 події та 5 завдань дають 250 XP — рівень «Активіст») та активного учасника (10 подій,

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		82

20 завдань, 5 відгуків дають 975 XP — рівень «Ветеран»). Результат запуску юніт-тестів наведено на рис. 3.29.

```
PS C:\ЖДТУ\Дипломна робота\volunteer-platform\backend> npm test

> volunteer-platform-backend@1.0.0 test
> jest --forceExit --detectOpenHandles

PASS tests/api.test.js
  ● Console

    console.error
      TypeError: Cannot read properties of undefined (reading 'rows')
          at rows (C:\ЖДТУ\Дипломна робота\volunteer-platform\backend\src\controllers\eventController.js:133:30)
          at processTicksAndRejections (node:internal/process/task_queues:95:5)
          at getEvents (C:\ЖДТУ\Дипломна робота\volunteer-platform\backend\src\controllers\eventController.js:256:13)

      259 |         res.json(result.rows);
      260 |     } catch (err) {
    > 261 |         console.error(err);
          |         ^
      262 |         res.status(500).json({ error: 'Помилка сервера' });
      263 |     }
      264 | };

      at error (src\controllers\eventController.js:261:17)

PASS tests/experience.test.js
```

Рис. 3.29. Результати виконання юніт-тестів модуля XP

Інтеграційні тести `api.test.js` перевіряють поведінку REST API серверу в умовах, наближених до реальних. Для HTTP-запитів до Express-застосунку використовується бібліотека Supertest, яка запускає сервер у пам'яті без прив'язки до мережевого порту. Зовнішні залежності — підключення до бази даних, Google OAuth стратегія та функції відправки email — замінено мок-об'єктами. Тести охоплюють п'ять груп перевірок. Перевірка ендпоінту GET /health підтверджує, що сервер повертає статус 200 із тілом відповіді, що містить поле status зі значенням «ok» та поле timestamp у форматі ISO. Перевірка захищених маршрутів без JWT-токена підтверджує, що запити до /api/users, /api/notifications та /api/admin повертають коди 401 або 403, тобто система коректно відхиляє неавторизований доступ. Перевірка валідації реєстрації підтверджує, що відправка порожнього тіла або запиту без обов'язкових полів повертає код 400 або 422, а не помилку сервера 500, і що відповідь завжди у форматі JSON. Перевірка публічних маршрутів /api/events, /api/categories та /api/news підтверджує коректну обробку запитів навіть при мок-відповіді бази даних з порожніми рядками. Результат виконання інтеграційних тестів наведено на рис. 3.30.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		83

```

PASS tests/experience.test.js

Test Suites: 2 passed, 2 total
Tests:       34 passed, 34 total
Snapshots:   0 total
Time:        2.016 s, estimated 3 s
Ran all test suites.
PS C:\ЖДТУ\Дипломна робота\volunteer-platform\backend>

```

Рис. 3.30. Результати виконання інтеграційних тестів API

Для ручного тестування REST API серверної частини використовувався інструмент Postman [33]. Postman дозволяє формувати HTTP-запити будь-якого типу (GET, POST, PUT, DELETE), задавати заголовки авторизації, тіло запиту у форматі JSON та перевіряти структуру відповідей. У процесі тестування перевірялися всі ключові ендпоінти платформи: реєстрація та вхід користувача, отримання та фільтрація списку подій, реєстрація на захід, створення завдань та зміна їхнього статусу, відправка донату, генерація PDF-сертифіката, подання скарги та запиту на верифікацію. Для захищених маршрутів у заголовок Authorization передавався дійсний JWT-токен, отриманий після успішного входу. Приклад перевірки запиту в Postman наведено на рис. 3.31.

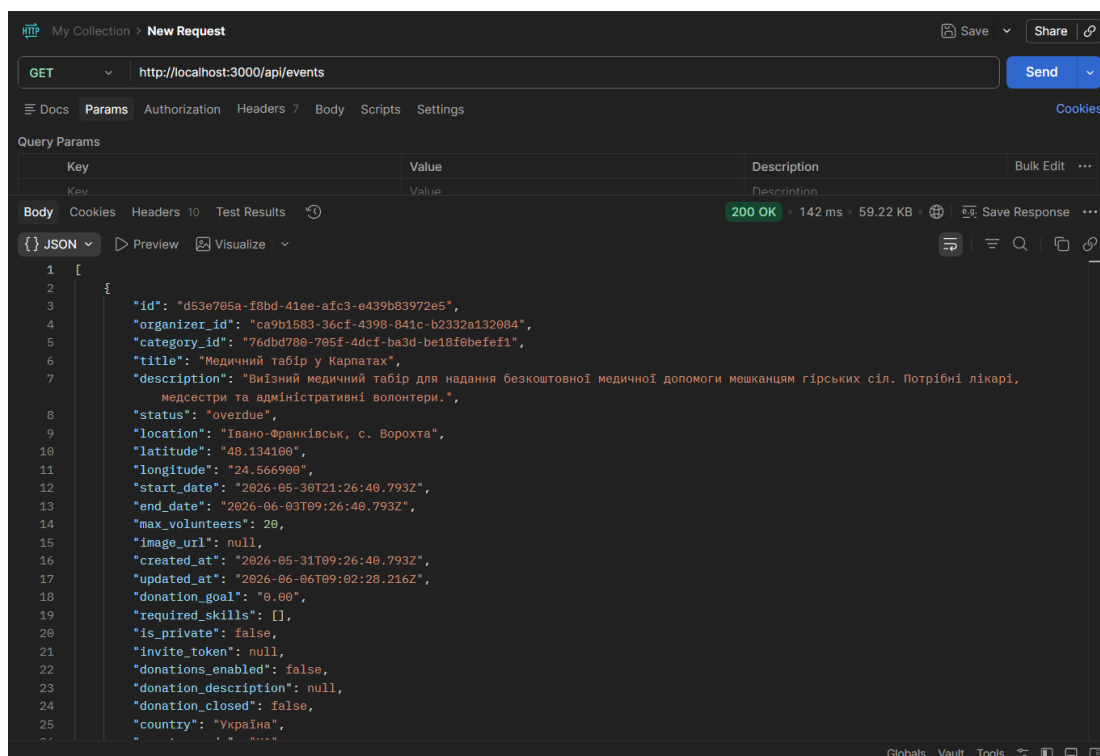


Рис. 3.31. Тестування REST API у середовищі Postman

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижевська О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		84

Тестування адаптивності інтерфейсу проводилося за допомогою вбудованих інструментів розробника браузера Google Chrome. Режим емуляції пристроїв дозволяє перемикатися між різними розмірами екранів — смартфон, планшет, настільний комп'ютер — і оцінювати коректність відображення кожного компонента інтерфейсу. Перевірявся вигляд усіх основних сторінок: головна сторінка, каталог подій, деталі події, особистий кабінет та адміністративна панель. Адаптивне відображення головної сторінки на мобільному пристрої наведено на рис. 3.32.

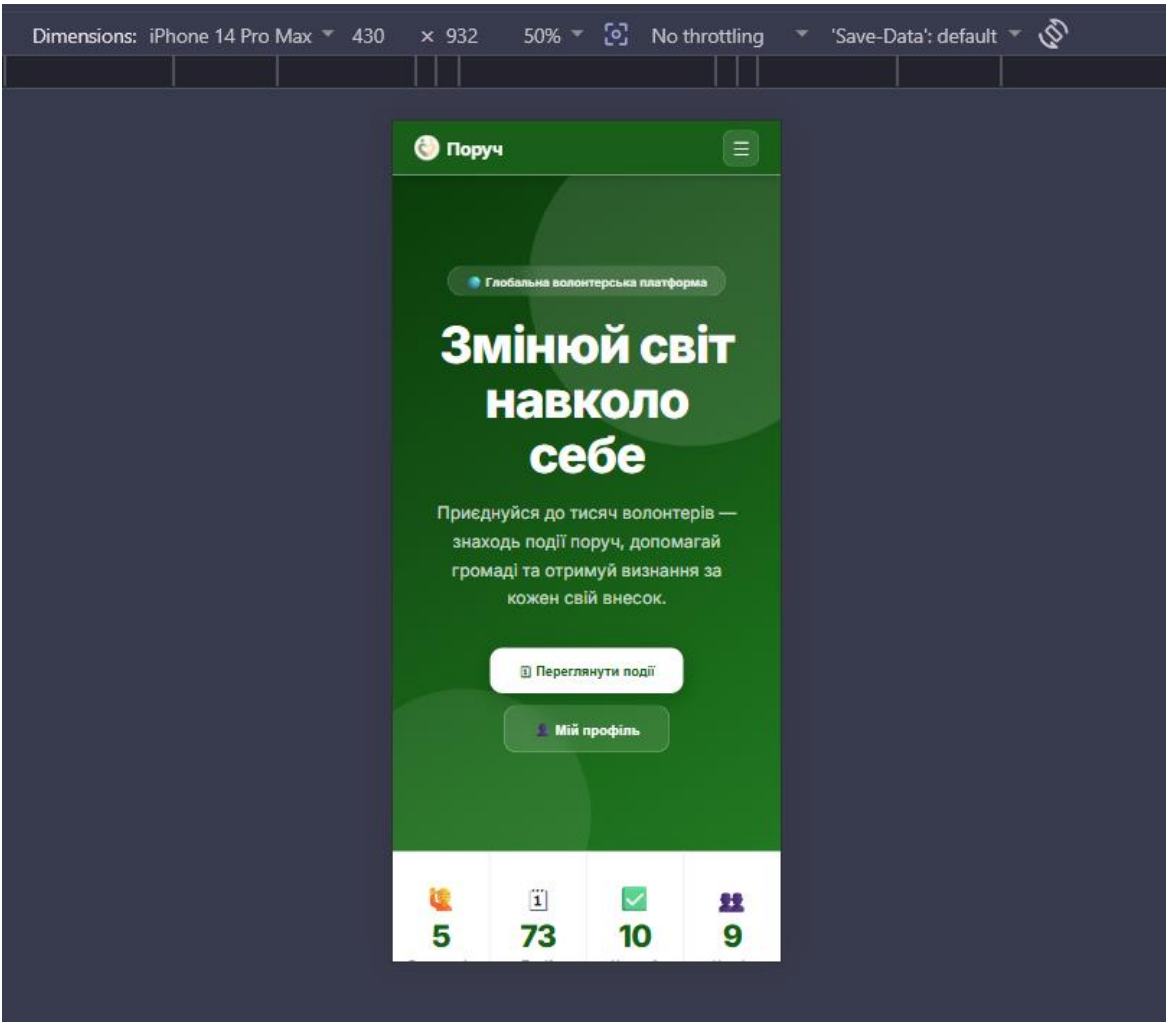


Рис. 3.32. Адаптивне відображення головної сторінки на мобільному пристрої

Окрім функціонального тестування, було проведено перевірку безпеки вебплатформи [4, 9]. Аналіз охоплював коректність роботи JWT-авторизації — перевірку відхилення прострочених та підроблених токенів, захист усіх чутливих ендпоінтів від несанкціонованого доступу, валідацію вхідних даних

на серверній стороні для запобігання ін'єкціям, а також перевірку CORS-налаштувань, що обмежують запити лише з дозволеного домену фронтенду [22, 32]. Дані платіжних карток не зберігаються на сервері платформи — вони передаються безпосередньо на сервери Stripe через захищений канал SSL, що унеможлиблює їх перехоплення [31].

Таким чином, комплексний підхід до тестування — автоматизовані юніт- та інтеграційні тести через Jest і Supertest, ручне тестування API у Postman та перевірка адаптивності інтерфейсу в інструментах розробника браузера — забезпечив підтвердження коректності роботи всіх функціональних модулів платформи та відповідності встановленим вимогам якості. Автоматизовані тести охоплюють 34 тест-кейси у двох файлах і запускаються автоматично в рамках CI/CD-конвеєра GitLab при кожному коміті, що гарантує своєчасне виявлення регресій. Перевірка безпеки підтвердила коректну роботу JWT-авторизації та захист чутливих ендпоінтів від несанкціонованого доступу.

Висновки до третього розділу

У третьому розділі розглянуто практичні аспекти розгортання, використання та тестування вебплатформи волонтерської діяльності.

У підрозділі 3.1 описано порядок встановлення та налаштування платформи. Трирівнева архітектура із контейнеризацією кожного компонента через Docker забезпечує відтворюване середовище розгортання незалежно від операційної системи сервера. Усі параметри конфігурації винесено у файл змінних оточення, що спрощує перенесення системи між середовищами. Налаштований CI/CD-конвеєр GitLab із чотирма стадіями автоматизує збірку, тестування та розгортання при кожній зміні у репозиторії.

У підрозділі 3.2 детально описано структуру інтерфейсу платформи, що охоплює 13 основних сторінок та понад 12 перевикористовуваних компонентів. Розмежування доступу за ролями (volunteer, organizer, admin) реалізовано через компоненти PrivateRoute та AdminRoute. Описано ключові функціональні модулі: систему реєстрації та авторизації з підтримкою Google

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		86

OAuth, каталог подій із розширеною фільтрацією та картографічним відображенням, управління завданнями з пріоритетами, триетапний процес Stripe-платежів, генерацію PDF-сертифікатів, систему скарг та верифікації організацій, форму зворотного зв'язку та панель сповіщень із 26 типами подій.

У підрозділі 3.3 розкрито механізм гейміфікації та аналітики. Система XP нараховує досвід автоматично за участь у заходах, виконання завдань та отримання відгуків, визначаючи один із п'яти рівнів від «Новачок» до «Експерт». Організаторам доступна розширена аналітична панель із чотирма вкладками на основі бібліотеки Recharts, що відображає динаміку реєстрацій, ефективність заходів та прогрес виконання завдань.

У підрозділі 3.4 проведено комплексне тестування платформи. Автоматизовані юніт-тести модуля XP та інтеграційні тести REST API охоплюють 34 тест-кейси у фреймворку Jest із бібліотекою Supertest і виконуються автоматично в CI/CD-конвеєрі. Ручне тестування у Postman підтвердило коректну роботу всіх ключових ендпоінтів. Перевірка безпеки підтвердила надійність JWT-авторизації та захист чутливих даних. Тестування адаптивності засобами Chrome DevTools підтвердило коректне відображення інтерфейсу на мобільних пристроях.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		87

ВИСНОВКИ

У кваліфікаційній роботі вирішено актуальну задачу розробки вебплатформи для організації волонтерської діяльності, що забезпечує повний цикл координації між волонтерами, організаторами та адміністраторами в межах єдиного цифрового середовища.

Проведено аналіз предметної галузі волонтерської діяльності та виявлено системні проблеми: відсутність централізованого простору для публікації заходів, брак інструментів обліку заявок та контролю складу учасників, відсутність системи репутації та мотивації до активної участі. Визначено три категорії кінцевих користувачів — волонтер, організатор і адміністратор — та сформовано функціональні й нефункціональні вимоги до платформи.

Здійснено огляд існуючих аналогів — Volunteer Country, Better Impact і GivePulse. Встановлено, що жодна з розглянутих платформ не забезпечує повного циклу організації волонтерської діяльності: одні обмежуються роллю інформаційної дошки, інші орієнтовані виключно на великі організації з розвиненою інфраструктурою або мають обмежені інструменти координації та не підтримують українську мову. Виявлені прогалини підтвердили доцільність розробки власної платформи.

Обґрунтовано вибір клієнт-серверної архітектури з REST API як оптимальної для проєкту. Визначено технологічний стек: серверна частина на Node.js із фреймворком Express.js і СУБД PostgreSQL 16, клієнтська частина на React 18 із збирачем Vite, розгортання через Docker і docker-compose. Вибір технологій обґрунтовано відповідністю функціональним вимогам, зрілістю рішень та можливістю ефективної розробки.

Виконано повне проєктування системи у нотації UML 2.0. Побудовано діаграму варіантів використання для чотирьох акторів, діаграму класів із дев'ятьма основними сутностями, дві діаграми активності та дві діаграми послідовності для ключових сценаріїв. Спроектовано реляційну базу даних із

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		88

19 логічними групами таблиць із каскадним видаленням, стратегією SET NULL та індексами на ключових полях.

Розроблено та реалізовано повний функціонал вебплатформи, що охоплює: реєстрацію та автентифікацію з підтримкою Google OAuth 2.0, каталог заходів із розширеною фільтрацією та картографічним відображенням через Leaflet, управління завданнями з пріоритетами та журналуванням змін у TaskHistory, дворівневу систему відгуків з автоматичним перерахунком рейтингу, триетапний процес прийому пожертв через Stripe, генерацію персональних PDF-сертифікатів засобами PDFKit, систему верифікації організацій, обробку скарг, форму зворотного зв'язку та панель сповіщень із 26 типами подій із дублюванням через Nodemailer.

Реалізовано систему гейміфікації на основі досвіду (XP) та п'яти рівнів від «Новачок» до «Експерт», що автоматично нараховує бали за участь у заходах, виконання завдань і отримання позитивних відгуків. Для організаторів розроблено аналітичну панель із чотирма вкладками на основі Recharts із площинними та стовпчастими діаграмами динаміки реєстрацій, ефективності заходів і прогресу завдань.

Налаштовано CI/CD-конвеєр GitLab із чотирма стадіями, що автоматизує збірку, тестування та розгортання платформи. Проведено комплексне тестування: автоматизовані юніт- та інтеграційні тести охоплюють 34 тест-кейси у фреймворку Jest із бібліотекою Supertest; ручне тестування API у Postman підтвердило коректну роботу всіх ключових ендпоінтів; перевірка безпеки підтвердила надійність JWT-авторизації; тестування у Chrome DevTools підтвердило адаптивність інтерфейсу на мобільних пристроях.

Розроблена вебплатформа повністю відповідає поставленим функціональним і нефункціональним вимогам та може бути використана громадськими організаціями, ініціативними групами та координаційними центрами для ефективної організації волонтерської діяльності в умовах сучасних викликів.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		89

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Царук К.А., Чижмотря О.В. Розробка вебплатформи для організації волонтерської діяльності. Тези XVI Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології», м. Житомир, 02-03 квітня 2026 р. – Житомир: Житомирська політехніка, 2026.
2. Царук К.А., Чижмотря О.В. Забезпечення інформаційної безпеки вебплатформи для організації волонтерської діяльності. Тези XVI Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології», м. Житомир, 02-03 квітня 2026 р. – Житомир: Житомирська політехніка, 2026.
3. Закон України «Про волонтерську діяльність» від 19.04.2011 № 3236-VI (зі змінами).
4. Закон України «Про захист персональних даних» від 01.06.2010 № 2297-VI (зі змінами).
5. Гуржій А.М., Жалдак М.І. Інформаційні системи та технології : навчальний посібник. Київ : Видавництво НПУ імені М.П. Драгоманова, 2020. 312 с.
6. Кравець О.В. Основи розробки вебзастосунків : навчальний посібник. Львів : Львівська політехніка, 2021. 256 с.
7. Копилов В.Є. Інформаційні технології та системи. Київ : Центр навчальної літератури, 2019. 287 с.
8. Олійник А.О., Субботін С.О. Інформаційні системи і технології в управлінні. Запоріжжя : ЗНТУ, 2018. 318 с.
9. Державна служба спеціального зв'язку та захисту інформації України. Основи кібербезпеки. Київ, 2023. 96 с.
10. Міністерство цифрової трансформації України. Цифрова трансформація в Україні: стратегія та перспективи розвитку. Київ, 2023. 84 с.
11. Державна служба статистики України. Соціальна активність населення та волонтерство в Україні : статистичний збірник. Київ, 2022. 112 с.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				90
Змн.	Арк.	№ докум.	Підпис	Дата		

12. Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. 2nd ed. Boston : Addison-Wesley, 2005. 560 p.
13. Cockburn A. Writing Effective Use Cases. Boston : Addison-Wesley, 2000. 264 p.
14. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner's Approach. 9th ed. New York : McGraw-Hill, 2019. 848 p.
15. Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software. Boston : Addison-Wesley, 1994. 395 p.
16. React Documentation. Quick Start. URL: <https://react.dev/learn> (дата звернення: 10.01.2026).
17. Node.js Documentation. Getting Started Guide. URL: <https://nodejs.org/en/docs/guides/getting-started-guide> (дата звернення: 12.01.2026).
18. Express.js Documentation. Getting started. URL: <https://expressjs.com/en/starter/installing.html> (дата звернення: 12.01.2026).
19. PostgreSQL Documentation. Version 16. URL: <https://www.postgresql.org/docs/16/> (дата звернення: 15.01.2026).
20. Docker Documentation. Get started with Docker. URL: <https://docs.docker.com/get-started/> (дата звернення: 11.02.2026).
21. Stripe Documentation. Accept a payment. URL: <https://stripe.com/docs/payments/accept-a-payment> (дата звернення: 20.02.2026).
22. JWT.io. Introduction to JSON Web Tokens. URL: <https://jwt.io/introduction> (дата звернення: 18.01.2026).
23. Google Identity. OAuth 2.0 for Web Server Applications. URL: <https://developers.google.com/identity/protocols/oauth2/web-server> (дата звернення: 19.01.2026).
24. GitLab Documentation. Get started with GitLab CI/CD. URL: <https://docs.gitlab.com/ee/ci/> (дата звернення: 05.03.2026).
25. Jest Documentation. Getting Started. URL: <https://jestjs.io/docs/getting-started> (дата звернення: 01.03.2026).

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		91

26. Supertest. GitHub Repository. URL: <https://github.com/ladjs/supertest> (дата звернення: 02.03.2026).

27. Leaflet Documentation. Overview. URL: <https://leafletjs.com/reference.html> (дата звернення: 25.01.2026).

28. Recharts Documentation. Getting Started. URL: <https://recharts.org/en-US/guide> (дата звернення: 28.01.2026).

29. PDFKit Documentation. A JavaScript PDF Generation Library for Node and the Browser. URL: https://pdfkit.org/docs/getting_started.html (дата звернення: 03.02.2026).

30. Nodemailer Documentation. About Nodemailer. URL: <https://nodemailer.com/about/> (дата звернення: 06.02.2026).

31. Чинний стандарт безпеки даних індустрії платіжних карток (PCI DSS) : версія 4.0. URL: <https://www.pcisecuritystandards.org> (дата звернення: 05.03.2026).

32. OWASP Top Ten. The Ten Most Critical Web Application Security Risks. URL: <https://owasp.org/www-project-top-ten/> (дата звернення: 10.03.2026).

33. Postman Documentation. Writing tests. URL: <https://learning.postman.com/docs/writing-scripts/test-scripts/> (дата звернення: 08.03.2026).

34. StarUML Documentation. URL: <https://docs.staruml.io> (дата звернення: 02.01.2026).

35. bcrypt.js. GitHub Repository. URL: <https://github.com/dcodeIO/bcrypt.js> (дата звернення: 17.01.2026).

36. Vite Documentation. Getting Started. URL: <https://vite.dev/guide/> (дата звернення: 13.01.2026).

37. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмотря О.В.				92
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ПЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		93

Повна схема бази даних у вигляді ER-діаграми

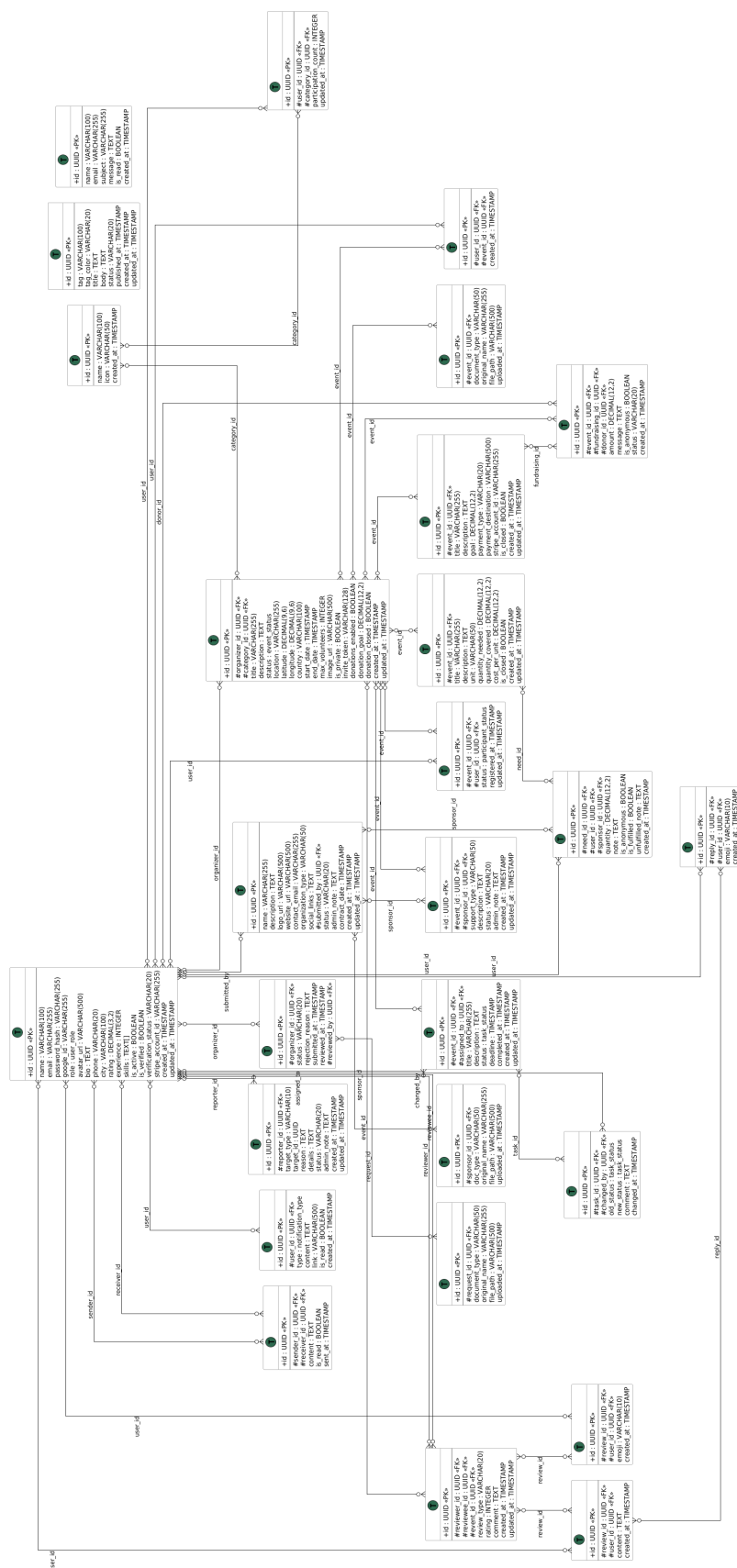


Рис. А.1. ER-діаграма

		Царук К.А.		
		Чижмоторя О.В.		
Змн.	Арк.	№ докум.	Підпис	Дата

backend/src/app.js (Головний файл сервера, підключення всіх маршрутів):

```

const express = require('express');
const cors = require('cors');
const passport = require('./config/googleStrategy');
require('dotenv').config();

const authRoutes = require('./routes/authRoutes');
const eventRoutes = require('./routes/eventRoutes');
const participantRoutes = require('./routes/participantRoutes');
const taskRoutes = require('./routes/taskRoutes');
const userRoutes = require('./routes/userRoutes');
const reviewRoutes = require('./routes/reviewRoutes');
const notificationRoutes = require('./routes/notificationRoutes');
const adminRoutes = require('./routes/adminRoutes');
const savedRoutes = require('./routes/savedRoutes');
const categoryRoutes = require('./routes/categoryRoutes');
const chatRoutes = require('./routes/chatRoutes');
const homepageRoutes = require('./routes/homepageRoutes');
const sponsorRoutes = require('./routes/sponsorRoutes');
const complaintRoutes = require('./routes/complaintRoutes');
const verificationRoutes = require('./routes/verificationRoutes');
const analyticsRoutes = require('./routes/analyticsRoutes');
const feedbackRoutes = require('./routes/feedbackRoutes');
const donationRoutes = require('./routes/donationRoutes');
const charityNeedsRoutes = require('./routes/charityNeedsRoutes');
const fundraisingRoutes = require('./routes/fundraisingRoutes');
const newsRoutes = require('./routes/newsRoutes');
const path = require('path');

const app = express();

app.use(cors({
  origin: process.env.FRONTEND_URL || 'http://localhost:5173',
  credentials: true,
}));
app.use(express.json());
app.use(passport.initialize());

// Статична роздача завантажених файлів
app.use('/uploads', express.static(path.join(__dirname, '../uploads')));

app.use('/api/auth', authRoutes);
app.use('/api/events', eventRoutes);
app.use('/api/events/:id/participants', participantRoutes);
app.use('/api/events/:id/tasks', taskRoutes);
app.use('/api/users', userRoutes);
app.use('/api/reviews', reviewRoutes);
app.use('/api/notifications', notificationRoutes);
app.use('/api/admin', adminRoutes);

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				95
Змн.	Арк.	№ докум.	Підпис	Дата		

```

app.use('/api/events',    savedRoutes);
app.use('/api/categories', categoryRoutes);
app.use('/api/messages',  chatRoutes);
app.use('/api/homepage',  homepageRoutes);
app.use('/api/sponsors',  sponsorRoutes);
app.use('/api/complaints', complaintRoutes);
app.use('/api/verification', verificationRoutes);
app.use('/api/dashboard',  analyticsRoutes);
app.use('/api/feedback',   feedbackRoutes);
app.use('/api/events/id/donations',    donationRoutes);
app.use('/api/events/id/needs',        charityNeedsRoutes);
app.use('/api/events/id/fundraisings', fundraisingRoutes);
app.use('/api/news',                  newsRoutes);

app.get('/', (req, res) => {
  res.json({ message: '☐ Сервер працює!' });
});

app.get('/health', (req, res) => {
  res.json({ status: 'ok', timestamp: new Date().toISOString() });
});

const PORT = process.env.PORT || 3000;
if (require.main === module) {
  app.listen(PORT, () => {
    console.log(`🚀 Сервер запущено на порті ${PORT}`);
  });
}

module.exports = app;

```

backend/src/config/db.js (Підключення до PostgreSQL через пул з'єднань):

```

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

const pool = new Pool({
  host:    process.env.DB_HOST,
  port:    process.env.DB_PORT,
  database: process.env.DB_NAME,
  user:    process.env.DB_USER,
  password: process.env.DB_PASSWORD,
});

pool.connect()
  .then(() => console.log('✓ БД підключена'))
  .catch(err => console.error('✗ Помилка БД:', err.message));
module.exports = pool;

```

backend/src/middleware/auth.js (JWT-автентифікація + перевірка блокування акаунту):

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		96

```

const jwt = require('jsonwebtoken');
const pool = require('../config/db');

const authMiddleware = async (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1]; // Bearer TOKEN

  if (!token) {
    return res.status(401).json({ error: 'Токен відсутній' });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    // Перевіряємо що акаунт не заблоковано
    const userRes = await pool.query(
      'SELECT is_active FROM users WHERE id = $1', [decoded.id]
    );
    if (!userRes.rows[0]?.is_active) {
      return res.status(403).json({ error: 'Ваш акаунт заблоковано. Зверніться до адміністратора.' });
    }

    req.user = decoded; // { id, role }
    next();
  } catch (err) {
    res.status(403).json({ error: 'Невірний або застарілий токен' });
  }
};

module.exports = authMiddleware;

```

backend/src/controllers/authController.js (Реєстрація і логін (хешування паролю, видача токена):

```

const pool = require('../config/db');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const { sendWelcome } = require('../utils/email');

// РЕЄСТРАЦІЯ
const register = async (req, res) => {
  const { name, email, password, role: rawRole } = req.body;
  const allowedRoles = ['volunteer', 'organizer'];
  const role = allowedRoles.includes(rawRole) ? rawRole : 'volunteer';

  if (!name || !email || !password) {
    return res.status(400).json({ error: 'Всі поля обов'язкові' });
  }

  if (password.length < 6) {
    return res.status(400).json({ error: 'Пароль має бути не менше 6 символів' });
  }

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		97

```

if (!/^[^s@]+@[^s@]+\.[^s@]+$/ .test(email)) {
  return res.status(400).json({ error: 'Невірний формат email' });
}

try {
  // Перевірка чи email вже є
  const existing = await pool.query(
    'SELECT id FROM users WHERE email = $1', [email]
  );
  if (existing.rows.length > 0) {
    return res.status(400).json({ error: 'Цей email вже зареєстровано' });
  }

  // Хешуємо пароль
  const password_hash = await bcrypt.hash(password, 10);

  // Зберігаємо користувача
  const result = await pool.query(
    'INSERT INTO users (name, email, password_hash, role)
VALUES ($1, $2, $3, $4)
RETURNING id, name, email, role`,
    [name, email, password_hash, role]
  );

  const user = result.rows[0];

  // Генеруємо токен
  const token = jwt.sign(
    { id: user.id, role: user.role },
    process.env.JWT_SECRET,
    { expiresIn: '7d' }
  );

  // Вітальний email (не блокуємо відповідь у разі помилки)
  sendWelcome({ to: email, name }).catch(() => {});

  res.status(201).json({ user, token });

} catch (err) {
  console.error(err);
  res.status(500).json({ error: 'Помилка сервера' });
}
};

// ЛОГІН
const login = async (req, res) => {
  const { email, password } = req.body;

  if (!email || !password) {
    return res.status(400).json({ error: 'Email та пароль обов'язкові' });
  }

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		98

```

try {
  const result = await pool.query(
    'SELECT * FROM users WHERE email = $1', [email]
  );

  if (result.rows.length === 0) {
    return res.status(400).json({ error: 'Невірний email або пароль' });
  }

  const user = result.rows[0];

  // Акаунт створено через Google — пароля немає
  if (!user.password_hash) {
    return res.status(400).json({ error: 'Цей акаунт створено через Google. Увійдіть через кнопку "Увійти через Google".' });
  }

  // Перевіряємо пароль
  const isMatch = await bcrypt.compare(password, user.password_hash);
  if (!isMatch) {
    return res.status(400).json({ error: 'Невірний email або пароль' });
  }

  if (!user.is_active) {
    return res.status(403).json({ error: 'Ваш акаунт заблоковано. Зверніться до адміністратора.' });
  }

  // Генеруємо токен
  const token = jwt.sign(
    { id: user.id, role: user.role },
    process.env.JWT_SECRET,
    { expiresIn: '7d' }
  );

  res.json({
    user: {
      id: user.id,
      name: user.name,
      email: user.email,
      role: user.role,
      avatar_url: user.avatar_url
    },
    token
  });

} catch (err) {
  console.error(err);
  res.status(500).json({ error: 'Помилка сервера' });
}
};

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		99

```
module.exports = { register, login };
```

backend/src/controllers/eventController.js (Логіка подій (зміна статусів, авто-нарахування участі):

```
const pool = require('../config/db');
const crypto = require('crypto');
const { createNotification } = require('../notificationController');
const { addExperience, XP } = require('../utils/experience');

// Не запускаємо syncEventStatuses частіше ніж раз на 5 хвилин
let lastSyncTime = 0;
const SYNC_INTERVAL_MS = 5 * 60 * 1000;

// — closePendingApplications

// При завершенні або скасуванні події — закриває всі pending-заявки (closed).
const closePendingApplications = async (eventId, eventTitle, reason = 'cancelled') => {
  const pending = await pool.query(
    `SELECT user_id FROM event_participants WHERE event_id = $1 AND status = 'pending'`,
    [eventId]
  );
  if (pending.rows.length === 0) return;

  await pool.query(
    `UPDATE event_participants SET status = 'closed', updated_at = NOW()
    WHERE event_id = $1 AND status = 'pending'`,
    [eventId]
  );

  const msg = reason === 'cancelled'
    ? `Подію "${eventTitle}" скасовано. Вашу заявку закрито автоматично.`
    : `Подію "${eventTitle}" завершено. Нові заявки більше не приймаються.`;

  await Promise.all(pending.rows.map(p =>
    createNotification(p.user_id, 'event_cancelled', msg, `/events/${eventId}`)
  ));
};

// — restoreClosedApplications

// При відновленні події — повертає closed-заявки назад у pending.
const restoreClosedApplications = async (eventId, eventTitle) => {
  const closed = await pool.query(
    `SELECT user_id FROM event_participants WHERE event_id = $1 AND status = 'closed'`,
    [eventId]
  );
  if (closed.rows.length === 0) return;

  await pool.query(

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		100

```

        `UPDATE event_participants SET status = 'pending', updated_at = NOW()
        WHERE event_id = $1 AND status = 'closed`,
        [eventId]
    );

    await Promise.all(closed.rows.map(p =>
        createNotification(
            p.user_id,
            'event_registered',
            `Подію "${eventTitle}" відновлено. Ваша заявка знову активна.`,
            `events/${eventId}`
        )
    ));
};

// — autoAttendParticipants

```

```

// Проставляє 'attended' підтвердженням учасникам що виконали хоча б один таск.
// Якщо у події взагалі немає тасків — attended отримують усі confirmed.
const autoAttendParticipants = async (eventId, eventTitle) => {
    const tasksRes = await pool.query(
        `SELECT COUNT(*) FROM tasks WHERE event_id = $1 AND status != 'cancelled`,
        [eventId]
    );
    const hasTasks = parseInt(tasksRes.rows[0].count) > 0;

    const query = hasTasks
        ? `SELECT DISTINCT ep.user_id
        FROM event_participants ep
        WHERE ep.event_id = $1 AND ep.status = 'confirmed'
        AND EXISTS (
            SELECT 1 FROM tasks t
            WHERE t.event_id = $1 AND t.assigned_to = ep.user_id AND t.status = 'completed'
        )`
        : `SELECT user_id FROM event_participants
        WHERE event_id = $1 AND status = 'confirmed`;

    const participants = await pool.query(query, [eventId]);

    // Отримуємо категорію події для оновлення статистики рекомендацій
    const eventCatRes = await pool.query(
        `SELECT category_id FROM events WHERE id = $1`, [eventId]
    );
    const categoryId = eventCatRes.rows[0]?.category_id;

    for (const p of participants.rows) {
        await pool.query(
            `UPDATE event_participants SET status = 'attended', updated_at = NOW()
            WHERE event_id = $1 AND user_id = $2`,
            [eventId, p.user_id]
        );
        await addExperience(p.user_id, XP.EVENT_ATTENDED);
    }
}

```

		Царук К.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		101

```

    await createNotification(
      p.user_id,
      'event_confirmed',
      `Подію "${eventTitle}" завершено. Дякуємо за участь! Сертифікат доступний на сторінці події`,
      `events/${eventId}`
    );

    // Оновлюємо статистику категорій для системи рекомендацій
    if (categoryId) {
      await pool.query(
        INSERT INTO user_category_stats (user_id, category_id, participation_count,
updated_at)
VALUES ($1, $2, 1, NOW())
ON CONFLICT (user_id, category_id)
DO UPDATE SET participation_count = user_category_stats.participation_count + 1,
updated_at = NOW()
      `, [p.user_id, categoryId]);
    }
  }
};

// — syncEventStatuses
—

// Автоматично оновлює статуси подій на основі поточного часу.
const syncEventStatuses = async () => {
  // published → active: подія розпочалась але ще не закінчилась
  await pool.query(
    UPDATE events SET status = 'active', updated_at = NOW()
    WHERE status = 'published'
    AND start_date <= NOW()
    AND end_date >= NOW()
  );

  // Знаходимо події де кінцева дата минула і статус ще активний
  const expired = await pool.query(
    SELECT id, title FROM events
    WHERE status IN ('published', 'active')
    AND end_date < NOW()
  );

  for (const ev of expired.rows) {
    // Перевіряємо чи всі таски виконані
    const check = await pool.query(
      SELECT
        COUNT(*) AS total,
        COUNT(*) FILTER (WHERE status = 'completed') AS done
      FROM tasks
      WHERE event_id = $1 AND status != 'cancelled'
    `, [ev.id]);
  }
}

```

```

const total = parseInt(check.rows[0].total);
const done = parseInt(check.rows[0].done);
const allDone = (total === 0 || total === done);

if (allDone) {
  // Усі завдання виконані → completed
  await pool.query(
    `UPDATE events SET status = 'completed', updated_at = NOW() WHERE id = $1`,
    [ev.id]
  );
  await autoAttendParticipants(ev.id, ev.title);
  await closePendingApplications(ev.id, ev.title, 'completed');
} else {
  // Є невиконані → overdue, сповіщаємо організатора
  await pool.query(
    `UPDATE events SET status = 'overdue', updated_at = NOW() WHERE id = $1`,
    [ev.id]
  );
  // Закриваємо pending-заявки і для overdue-подій
  await closePendingApplications(ev.id, ev.title, 'completed');
  const orgRes = await pool.query(`SELECT organizer_id FROM events WHERE id = $1`,
  [ev.id]);
  if (orgRes.rows[0]) {
    await createNotification(
      orgRes.rows[0].organizer_id,
      'task_completed',
      `Термін події "${ev.title}" завершився, але є невиконані завдання. Продовжте
або завершіть подію.`,
      `/events/${ev.id}`
    );
  }
}
};

// — getEvents

```

```

const getEvents = async (req, res) => {
  const { category_id, status, city, country_code, search, date_from, date_to, lat, lng, radius } =
  req.query;

  let query = `
  SELECT
    e.*,
    u.name AS organizer_name,
    u.avatar_url AS organizer_avatar,
    c.name AS category_name,
    c.icon AS category_icon,
    COUNT(DISTINCT ep.id) FILTER (WHERE ep.status IN ('confirmed','attended')) AS
  confirmed_count
  FROM events e

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		103

```

        LEFT JOIN users u ON e.organizer_id = u.id
        LEFT JOIN categories c ON e.category_id = c.id
        LEFT JOIN event_participants ep ON e.id = ep.event_id
    WHERE e.status NOT IN ('draft', 'cancelled')
        AND (e.is_private = false OR e.organizer_id = $1)
`;

const userId = req.user?.id || null;
const params = [userId];
let i = 2;

if (category_id) {
    query += ` AND e.category_id = ${i++}`;
    params.push(category_id);
}
if (status === 'archive') {
    // Архів: успішно завершені + завершені частково
    query += ` AND e.status IN ('completed', 'completed_partial')`;
} else if (status) {
    query += ` AND e.status = ${i++}`;
    params.push(status);
} else {
    // За замовчуванням — активні + прострочені (ще не закриті)
    query += ` AND e.status IN ('published', 'active', 'overdue')`;
}
if (city) {
    query += ` AND e.location ILIKE ${i++}`;
    params.push(`%${city}%`);
}
if (country_code) {
    query += ` AND e.country_code = ${i++}`;
    params.push(country_code);
}
if (search) {
    const iSearch = i++;
    query += ` AND (e.title ILIKE ${iSearch} OR e.description ILIKE ${iSearch})`;
    params.push(`%${search}%`);
}
if (date_from) {
    query += ` AND e.start_date >= ${i++}`;
    params.push(date_from);
}
if (date_to) {
    query += ` AND e.end_date <= ${i++}`;
    params.push(date_to);
}
if (lat && lng && radius) {
    const iLat = i++; const iLat2 = i++; const iLng = i++; const iR = i++;
    query += `
        AND e.latitude IS NOT NULL AND e.longitude IS NOT NULL
        AND (
            6371 * 2 * ASIN(SQRT(

```

```

        POWER(SIN((RADIANS($${iLat}) - RADIANS(e.latitude::float)) / 2), 2) +
        COS(RADIANS(e.latitude::float)) * COS(RADIANS($${iLat2})) *
        POWER(SIN((RADIANS($${iLng}) - RADIANS(e.longitude::float)) / 2), 2)
    ))
    ) <= $${iR}`;
    params.push(parseFloat(lat), parseFloat(lat), parseFloat(lng), parseFloat(radius));
}

query += ` GROUP BY e.id, u.name, u.avatar_url, c.name, c.icon`;
query += ` ORDER BY e.start_date ASC`;

try {
    const now = Date.now();
    if (now - lastSyncTime > SYNC_INTERVAL_MS) {
        lastSyncTime = now;
        await syncEventStatuses();
    }
    const result = await pool.query(query, params);
    res.json(result.rows);
} catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
}
};

// — getEventById

const getEventById = async (req, res) => {
    const { id } = req.params;
    try {
        const now = Date.now();
        if (now - lastSyncTime > SYNC_INTERVAL_MS) {
            lastSyncTime = now;
            await syncEventStatuses();
        }
        const result = await pool.query(
            `SELECT
                e.*,
                u.name AS organizer_name,
                u.avatar_url AS organizer_avatar,
                (u.verification_status = 'verified') AS organizer_verified,
                c.name AS category_name,
                c.icon AS category_icon,
                COUNT(DISTINCT ep.id) FILTER (WHERE ep.status IN ('confirmed','attended')) AS
confirmed_count,
                (SELECT COUNT(*) FROM events e2 WHERE e2.organizer_id = u.id AND
e2.status = 'completed') AS organizer_events_count
            FROM events e
            LEFT JOIN users u ON e.organizer_id = u.id
            LEFT JOIN categories c ON e.category_id = c.id
            LEFT JOIN event_participants ep ON e.id = ep.event_id`
        );
    } catch (err) {
        console.error(err);
        res.status(500).json({ error: 'Помилка сервера' });
    }
};

```

		Царук К.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижемоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		105

```

        WHERE e.id = $1
        GROUP BY e.id, u.id, u.name, u.avatar_url, u.verification_status, c.name, c.icon
        `, [id]);

    if (result.rows.length === 0)
        return res.status(404).json({ error: 'Подію не знайдено' });

    const event = result.rows[0];

    // Приватна подія: доступна лише організатору, адміну або по invite_token
    if (event.is_private) {
        const { token } = req.query;
        const isOrganizer = req.user.id === event.organizer_id;
        const isAdmin = req.user?.role === 'admin';
        const tokenOk = token === event.invite_token;

        if (!isOrganizer && !isAdmin && !tokenOk) {
            return res.status(403).json({ error: 'Ця подія є приватною. Потрібне запрошення.' });
        }
    }

    res.json(event);
} catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
}
};

// — createEvent

```

```

const createEvent = async (req, res) => {
    const {
        category_id, title, description,
        location, latitude, longitude,
        country = 'Україна', country_code = 'UA',
        start_date, end_date, max_volunteers, image_url,
        required_skills = [],
        status = 'published',
        is_private = false,
        donations_enabled = false,
        donation_goal,
        donation_description,
    } = req.body;

    if (!title || !start_date || !end_date) {
        return res.status(400).json({ error: 'Назва, дата початку та дата закінчення обов'язкові' });
    }

    const BLOCKED_COUNTRIES = ['RU', 'BY'];
    if (country_code && BLOCKED_COUNTRIES.includes(country_code.toUpperCase())) {

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		106

```

    return res.status(403).json({ error: 'Створення подій для цієї країни заборонено на
платформі.' });
  }
  if (new Date(start_date) >= new Date(end_date)) {
    return res.status(400).json({ error: 'Дата закінчення має бути пізніше дати початку' });
  }

  const allowedStatuses = ['draft', 'published'];
  const eventStatus = allowedStatuses.includes(status) ? status : 'published';
  const inviteToken = is_private ? crypto.randomBytes(32).toString('hex') : null;

  try {
    const lat = latitude !== undefined && latitude !== '' ? parseFloat(latitude) : null;
    const lng = longitude !== undefined && longitude !== '' ? parseFloat(longitude) : null;

    const result = await pool.query(`
      INSERT INTO events
      (organizer_id, category_id, title, description, location,
      latitude, longitude, country, country_code,
      start_date, end_date, max_volunteers, image_url,
      required_skills, status, is_private, invite_token,
      donations_enabled, donation_goal, donation_description)
      VALUES ($1,$2,$3,$4,$5,$6,$7,$8,$9,$10,$11,$12,$13,$14,$15,$16,$17,$18,$19,$20)
      RETURNING *
    `, [
      req.user.id, category_id, title, description, location,
      lat, lng, country || 'Україна', country_code || 'UA',
      start_date, end_date, max_volunteers || 0, image_url,
      required_skills, eventStatus, is_private, inviteToken,
      !!donations_enabled, donation_goal || null, donation_description || null,
    ]);
    res.status(201).json(result.rows[0]);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
  }
};

```

// — updateEvent

```

const updateEvent = async (req, res) => {
  const { id } = req.params;
  const { title, description, location, latitude, longitude,
    country, country_code,
    start_date, end_date, max_volunteers, status, image_url, category_id,
    required_skills, is_private,
    donations_enabled, donation_goal, donation_description } = req.body;

  try {
    const check = await pool.query('SELECT organizer_id, status FROM events WHERE id =
$1', [id]);
  }

```

		Царук К.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		107

```

if (check.rows.length === 0)
    return res.status(404).json({ error: 'Подію не знайдено' });
if (check.rows[0].organizer_id !== req.user.id && req.user.role !== 'admin')
    return res.status(403).json({ error: 'Немає прав' });

const BLOCKED_COUNTRIES = ['RU', 'BY'];
if (country_code && BLOCKED_COUNTRIES.includes(country_code.toUpperCase())) {
    return res.status(403).json({ error: 'Зміна країни на цю заборонена на платформі.' });
}

// При продовженні overdue/completed-події з новою датою — повертаємо в active
let newStatus = status;
if (['overdue', 'completed'].includes(check.rows[0].status) && end_date && !status) {
    newStatus = 'active';
}

// Генеруємо новий токен на випадок якщо подія стає приватною
const newToken = crypto.randomBytes(32).toString('hex');
const isPrivateVal = is_private !== undefined ? Boolean(is_private) : null;

const lat = latitude !== undefined && latitude !== '' ? parseFloat(latitude) : null;
const lng = longitude !== undefined && longitude !== '' ? parseFloat(longitude) : null;

// donations_enabled: якщо передано явно — використовуємо, інакше залишаємо як є
const hasDonations = donations_enabled !== undefined && donations_enabled !== null;
const donationsEnabledVal = hasDonations ? Boolean(donations_enabled) : null;

const result = await pool.query(`
    UPDATE events SET
        title          = COALESCE($1, title),
        description     = COALESCE($2, description),
        location        = COALESCE($3, location),
        latitude        = CASE WHEN $4::text IS NOT NULL THEN $4::numeric ELSE
latitude END,
        longitude       = CASE WHEN $5::text IS NOT NULL THEN $5::numeric ELSE
longitude END,
        start_date      = COALESCE($6, start_date),
        end_date        = COALESCE($7, end_date),
        max_volunteers  = COALESCE($8, max_volunteers),
        status          = COALESCE($9, status),
        image_url       = COALESCE($10, image_url),
        category_id     = COALESCE($11, category_id),
        required_skills  = COALESCE($12, required_skills),
        is_private      = COALESCE($13, is_private),
        invite_token    = CASE
            WHEN $13 = true AND invite_token IS NULL THEN $14
            WHEN $13 = false THEN NULL
            ELSE invite_token
        END,
        donations_enabled = CASE WHEN $16::boolean IS NOT NULL THEN
$16::boolean ELSE donations_enabled END,
        donation_goal    = CASE WHEN $17::text IS NOT NULL THEN $17::numeric

```

		Царук К.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		108

```

ELSE donation_goal END,
    donation_description = COALESCE($18, donation_description),
    country                = COALESCE($19, country),
    country_code           = COALESCE($20, country_code),
    updated_at             = NOW()
WHERE id = $15
RETURNING *
`, [title, description, location, lat, lng,
    start_date, end_date, max_volunteers, newStatus,
    image_url, category_id, required_skills || null,
    isPrivateVal, newToken, id,
    donationsEnabledVal,
    (donation_goal !== undefined && donation_goal !== '') ? donation_goal : null,
    donation_description || null,
    country || null, country_code || null]);

res.json(result.rows[0]);
} catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
}
};

// — deleteEvent

```

```

const deleteEvent = async (req, res) => {
    const { id } = req.params;
    try {
        const check = await pool.query('SELECT organizer_id FROM events WHERE id = $1',
[id]);
        if (check.rows.length === 0)
            return res.status(404).json({ error: 'Подію не знайдено' });
        if (check.rows[0].organizer_id !== req.user.id && req.user.role !== 'admin')
            return res.status(403).json({ error: 'Немає прав' });

        await pool.query('DELETE FROM events WHERE id = $1', [id]);
        res.json({ message: 'Подію видалено' });
    } catch (err) {
        console.error(err);
        res.status(500).json({ error: 'Помилка сервера' });
    }
};

// — cancelEvent

```

```

const cancelEvent = async (req, res) => {
    const { id } = req.params;
    try {
        const check = await pool.query(
'SELECT organizer_id, title, status FROM events WHERE id = $1', [id]

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		109

```

);
if (check.rows.length === 0)
    return res.status(404).json({ error: 'Подію не знайдено' });

const event = check.rows[0];
if (event.organizer_id !== req.user.id && req.user.role !== 'admin')
    return res.status(403).json({ error: 'Немає прав' });
if (event.status === 'cancelled')
    return res.status(400).json({ error: 'Подію вже скасовано' });
if (!['completed', 'completed_partial'].includes(event.status))
    return res.status(400).json({ error: 'Не можна скасувати завершену подію' });

await pool.query(
    "UPDATE events SET status = 'cancelled', updated_at = NOW() WHERE id = $1", [id]
);

// Закриваємо pending-заявки автоматично
await closePendingApplications(id, event.title, 'cancelled');

// Сповіщаємо confirmed учасників про скасування
const confirmed = await pool.query(`
    SELECT user_id FROM event_participants
    WHERE event_id = $1 AND status IN ('confirmed', 'attended')
`, [id]);

await Promise.all(confirmed.rows.map(p =>
    createNotification(
        p.user_id,
        'event_cancelled',
        `Подію "${event.title}" скасовано організатором.`,
        `/events/${id}`
    )
));

const updated = await pool.query('SELECT * FROM events WHERE id = $1', [id]);
res.json(updated.rows[0]);
} catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
}
};

// — changeEventStatus

—

const changeEventStatus = async (req, res) => {
    const { id } = req.params;
    const { status } = req.body;

    const allowed = ['completed', 'completed_partial', 'cancelled', 'active', 'published'];
    if (!allowed.includes(status))
        return res.status(400).json({ error: 'Невірний статус' });

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		110

```

try {
  const evRes = await pool.query('SELECT * FROM events WHERE id = $1', [id]);
  if (evRes.rows.length === 0)
    return res.status(404).json({ error: 'Подію не знайдено' });
  const ev = evRes.rows[0];

  if (ev.organizer_id !== req.user.id && req.user.role !== 'admin')
    return res.status(403).json({ error: 'Немає прав' });

  // Відновлення completed → active: тільки якщо кінцева дата ще не минула
  if (status === 'active' && ev.status === 'completed') {
    if (new Date() >= new Date(ev.end_date)) {
      return res.status(400).json({
        error: 'Неможливо відновити подію — кінцева дата вже минула. Спочатку
змініть дату завершення.'
      });
    }
  }
}

const result = await pool.query(
  'UPDATE events SET status = $1, updated_at = NOW() WHERE id = $2 RETURNING
*',
  [status, id]
);

// При відновленні події — повертаємо closed-заявки у pending
if ((status === 'active' || status === 'published') &&
  ['cancelled', 'completed', 'completed_partial', 'overdue'].includes(ev.status)) {
  await restoreClosedApplications(id, ev.title);
}

// При будь-якому варіанті завершення — autoAttend + закриваємо pending
if (status === 'completed' || status === 'completed_partial') {
  await autoAttendParticipants(id, ev.title);
  await closePendingApplications(id, ev.title, 'completed');

  // Якщо completed_partial — сповіщення організатору про незакриті таски
  if (status === 'completed_partial') {
    const openTasks = await pool.query(
      SELECT COUNT(*) FROM tasks
      WHERE event_id = $1 AND status NOT IN ('completed','cancelled')
    , [id]);
    const openCount = parseInt(openTasks.rows[0].count);
    if (openCount > 0) {
      await createNotification(
        ev.organizer_id,
        'task_completed',
        `Подію "${ev.title}" завершено. ${openCount} завдань залишились
невиконаними.`,
        `events/${id}`
      );
    }
  }
}

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чиждотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		111

```

    }
  }
}

res.json(result.rows[0]);
} catch (err) {
  console.error(err);
  res.status(500).json({ error: 'Помилка сервера' });
}
};

module.exports = {
  getEvents, getEventById, createEvent, updateEvent,
  deleteEvent, cancelEvent, changeEventStatus,
};

```

backend/src/controllers/participantController.js (registerToEvent):

```

const pool = require('../config/db');
const { createNotification } = require('../notificationController');
const { addExperience, XP } = require('../utils/experience');
const {
  sendApplicationReceived,
  sendRegistrationConfirmed,
  sendRegistrationRejected,
} = require('../utils/email');

// Зареєструватися на подію
const registerToEvent = async (req, res) => {
  const { id: event_id } = req.params;
  const user_id = req.user.id;

  try {
    const event = await pool.query(
      'SELECT * FROM events WHERE id = $1', [event_id]
    );
    if (event.rows.length === 0)
      return res.status(404).json({ error: 'Подію не знайдено' });

    if (event.rows[0].status !== 'published' && event.rows[0].status !== 'active')
      return res.status(400).json({ error: 'Реєстрація на цю подію недоступна' });

    if (event.rows[0].organizer_id === user_id)
      return res.status(400).json({ error: 'Організатор не може реєструватись на власну подію' });

    const result = await pool.query(
      INSERT INTO event_participants (event_id, user_id)
      VALUES ($1, $2)
      RETURNING *
    , [event_id, user_id]);

```

```

// Сповіщення організатору: нова заявка
const userRes = await pool.query('SELECT name FROM users WHERE id = $1',
[user_id]);
const userName = userRes.rows[0]?.name || 'Хтось';
await createNotification(
  event.rows[0].organizer_id,
  'event_registered',
  `${userName} подав(ла) заявку на участь у події "${event.rows[0].title}"`,
  `events/${event_id}`
);

// Email організатору: нова заявка
const orgRes = await pool.query('SELECT email FROM users WHERE id = $1',
[event.rows[0].organizer_id]);
if (orgRes.rows[0]?.email) {
  const ev = event.rows[0];
  await sendApplicationReceived({
    to: orgRes.rows[0].email,
    volunteerName: userName,
    eventTitle: ev.title,
    eventDate: ev.start_date
    ? new Date(ev.start_date).toLocaleDateString('uk-UA', { day: '2-digit', month:
'long', year: 'numeric' })
    : null,
    eventLocation: ev.location || null,
  });
}

res.status(201).json(result.rows[0]);
} catch (err) {
  if (err.code === '23505')
    return res.status(400).json({ error: 'Ви вже зареєстровані на цю подію' });
  console.error(err);
  res.status(500).json({ error: 'Помилка сервера' });
}
};

// Отримати учасників події
const getEventParticipants = async (req, res) => {
  const { id: event_id } = req.params;

  try {
    const result = await pool.query(`
      SELECT
        ep.id, ep.status, ep.registered_at,
        u.id AS user_id, u.name, u.email, u.avatar_url, u.rating
      FROM event_participants ep
        JOIN users u ON ep.user_id = u.id
      WHERE ep.event_id = $1
      ORDER BY ep.registered_at ASC
    `, [event_id]);

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижевотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		113

```

        res.json(result.rows);
    } catch (err) {
        console.error(err);
        res.status(500).json({ error: 'Помилка сервера' });
    }
};

// Підтвердити або відхилити учасника (тільки організатор)
const updateParticipantStatus = async (req, res) => {
    const { id: event_id, userId } = req.params;
    const { status } = req.body;

    try {
        const event = await pool.query(`
            SELECT e.organizer_id, e.max_volunteers, e.title, e.status AS event_status,
                   e.start_date, e.location, c.name AS category_name
            FROM events e
                 LEFT JOIN categories c ON e.category_id = c.id
            WHERE e.id = $1
        `, [event_id]);
        if (event.rows.length === 0)
            return res.status(404).json({ error: 'Подію не знайдено' });

        if (event.rows[0].organizer_id !== req.user.id && req.user.role !== 'admin')
            return res.status(403).json({ error: 'Немає прав' });

        const valid = ['confirmed', 'rejected', 'attended'];
        if (!valid.includes(status))
            return res.status(400).json({ error: 'Невірний статус' });

        // Для завершених/прострочених подій — не можна підтверджувати нові заявки
        const eventStatus = event.rows[0].event_status;
        if (status === 'confirmed' && ['completed', 'overdue', 'cancelled'].includes(eventStatus)) {
            return res.status(400).json({ error: 'Неможливо підтвердити заявку для завершеної події' });
        }

        // Перевірка ліміту при підтвердженні
        if (status === 'confirmed' && event.rows[0].max_volunteers > 0) {
            const countRes = await pool.query(`
                SELECT COUNT(*) FROM event_participants
                WHERE event_id = $1 AND status = 'confirmed',
            [event_id]
            );
            const confirmedCount = parseInt(countRes.rows[0].count);
            if (confirmedCount >= event.rows[0].max_volunteers) {
                return res.status(400).json({
                    error: `Досягнуто ліміт учасників (${event.rows[0].max_volunteers}). Спочатку відхиліть іншого учасника.`
                });
            }
        }
    }
};

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		114

```

const result = await pool.query(`
  UPDATE event_participants
  SET status = $1, updated_at = NOW()
  WHERE event_id = $2 AND user_id = $3
  RETURNING *
`, [status, event_id, userId]);

if (result.rows.length === 0)
  return res.status(404).json({ error: 'Учасника не знайдено' });

// XP за участь у події
if (status === 'attended') {
  let xp = XP.EVENT_ATTENDED;
  if (event.rows[0].category_name === 'Надзвичайні ситуації') xp +=
XP.URGENT_EVENT;
  await addExperience(userId, xp);
}

// Сповіщення + email волонтеру про зміну статусу
const eventTitle = event.rows[0].title;
const volRes = await pool.query('SELECT email, name FROM users WHERE id = $1',
[userId]);
const vol = volRes.rows[0] || {};

const evDateStr = event.rows[0].start_date
  ? new Date(event.rows[0].start_date).toLocaleDateString('uk-UA', { day: '2-digit', month:
'long', year: 'numeric' })
  : null;

if (status === 'confirmed') {
  await createNotification(
    userId,
    'event_confirmed',
    `Вашу заявку на подію "${eventTitle}" підтверджено!`,
    `events/${event_id}`
  );
  if (vol.email) {
    await sendRegistrationConfirmed({
      to: vol.email,
      name: vol.name,
      eventTitle,
      eventDate: evDateStr,
      eventLocation: event.rows[0].location || null,
      eventId: event_id,
    });
  }
} else if (status === 'rejected') {
  await createNotification(
    userId,
    'event_rejected',
    `На жаль, вашу заявку на подію "${eventTitle}" відхилено.`
  );
}

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижевська О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		115

```

        `/events/${event_id}`
    );
    if (vol.email) {
        await sendRegistrationRejected({
            to:      vol.email,
            name:    vol.name,
            eventTitle,
            eventId: event_id,
        });
    }
}

res.json(result.rows[0]);
} catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
}
};

// Скасувати свою реєстрацію
const cancelRegistration = async (req, res) => {
    const { id: event_id } = req.params;
    const user_id = req.user.id;

    try {
        const result = await pool.query(`
            DELETE FROM event_participants
            WHERE event_id = $1 AND user_id = $2
            RETURNING *
        `, [event_id, user_id]);

        if (result.rows.length === 0)
            return res.status(404).json({ error: 'Реєстрацію не знайдено' });

        res.json({ message: 'Реєстрацію скасовано' });
    } catch (err) {
        console.error(err);
        res.status(500).json({ error: 'Помилка сервера' });
    }
};

// Відхилити всі pending-заявки події (організатор)
const rejectAllPending = async (req, res) => {
    const { id: event_id } = req.params;
    try {
        const eventRes = await pool.query(
            `SELECT organizer_id, title FROM events WHERE id = $1`, [event_id]
        );
        if (!eventRes.rows.length) return res.status(404).json({ error: 'Подію не знайдено' });
        if (eventRes.rows[0].organizer_id !== req.user.id && req.user.role !== 'admin')
            return res.status(403).json({ error: 'Немає прав' });
    }
};

```

```

    // Шукаємо і pending, і closed (pending може лишитись якщо подія завершена вручну)
    const pending = await pool.query(
      `SELECT user_id FROM event_participants WHERE event_id=$1 AND status IN
('pending','closed')`,
      [event_id]
    );
    if (pending.rows.length === 0)
      return res.json({ count: 0 });

    // Ставимо 'closed' — не 'rejected', щоб при відновленні події заявки поверталися
    // автоматично
    await pool.query(
      `UPDATE event_participants SET status='closed', updated_at=NOW()
      WHERE event_id=$1 AND status IN ('pending','closed')`,
      [event_id]
    );

    const { createNotification } = require('./notificationController');
    await Promise.all(pending.rows.map(p =>
      createNotification(
        p.user_id, 'event_registered',
        `Подія "${eventRes.rows[0].title}" завершена. Ваша заявка закрыта — якщо подію
буде відновлено, ви отримаєте сповіщення.`,
        `/events/${event_id}`
      )
    ));

    res.json({ count: pending.rows.length });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
  }
};

module.exports = {
  registerToEvent, getEventParticipants,
  updateParticipantStatus, cancelRegistration,
  rejectAllPending,
};

```

backend/src/controllers/recommendationController.js:

```

const pool = require('../config/db');

// — getRecommendations
—
// Повертає до 10 активних подій, відсортованих за формулою:
// score = (participation_count × 5) + (skills_match × 3) + (city_match × 2)
const getRecommendations = async (req, res) => {
  const userId = req.user.id;
  try {

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		117

```

// Спочатку отримуємо дані волонтера окремо — уникаємо проблем з типами в CTE
const volRes = await pool.query(
  `SELECT COALESCE(skills, ARRAY[]::text[]) AS skills,
    LOWER(TRIM(COALESCE(city, ''))) AS city
    FROM users WHERE id = $1`,
    [userId]
);
const volSkills = volRes.rows[0]?.skills || [];
const volCity = volRes.rows[0]?.city || '';

const result = await pool.query(
  SELECT
    e.id,
    e.title,
    e.description,
    e.location,
    e.start_date,
    e.end_date,
    e.status,
    e.max_volunteers,
    e.required_skills,
    c.name AS category_name,
    u.name AS organizer_name,
    COALESCE(ep_count.confirmed_count, 0) AS confirmed_count,
    COALESCE(ucs.participation_count, 0) * 5 AS category_score,
    (
      SELECT COUNT(*)::int
      FROM unnest(COALESCE(e.required_skills, ARRAY[]::text[])) AS rs
      WHERE rs = ANY($2::text[])
    ) * 3 AS skills_score,
    CASE
      WHEN $3 <> '' AND LOWER(COALESCE(e.location, '')) LIKE '%' || $3 || '%'
      THEN 2 ELSE 0
    END AS location_score,
    (
      COALESCE(ucs.participation_count, 0) * 5
      + (
        SELECT COUNT(*)::int
        FROM unnest(COALESCE(e.required_skills, ARRAY[]::text[])) AS rs
        WHERE rs = ANY($2::text[])
      ) * 3
      + CASE
        WHEN $3 <> '' AND LOWER(COALESCE(e.location, '')) LIKE '%' || $3 || '%'
        THEN 2 ELSE 0
      END
    ) AS score
  FROM events e
  JOIN users u ON u.id = e.organizer_id
  LEFT JOIN categories c ON c.id = e.category_id
  LEFT JOIN user_category_stats ucs
    ON ucs.user_id = $1 AND ucs.category_id = e.category_id
  LEFT JOIN (

```

		Царук К.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		118

```

        SELECT event_id, COUNT(*) AS confirmed_count
        FROM event_participants
        WHERE status IN ('confirmed', 'attended')
        GROUP BY event_id
    ) ep_count ON ep_count.event_id = e.id
    WHERE e.status IN ('published', 'active')
    AND NOT EXISTS (
        SELECT 1 FROM event_participants ep
        WHERE ep.event_id = e.id AND ep.user_id = $1
    )
    ORDER BY score DESC, e.start_date ASC
    LIMIT 10
`, [userId, volSkills, volCity]);

    res.json(result.rows);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
  }
};

// — getCategoryStats

// Повертає статистику участі волонтера по категоріях (для профілю)
const getCategoryStats = async (req, res) => {
  const userId = req.user.id;
  try {
    const result = await pool.query(`
      SELECT c.name, c.icon, ucs.participation_count
      FROM user_category_stats ucs
      JOIN categories c ON c.id = ucs.category_id
      WHERE ucs.user_id = $1
      ORDER BY ucs.participation_count DESC
    `, [userId]);
    res.json(result.rows);
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: 'Помилка сервера' });
  }
};

module.exports = { getRecommendations, getCategoryStats };

```

backend/src/utils/experience.js (Система XP та рівнів волонтера — унікальна фішка проєкту):

```

const pool = require('../config/db');

// XP за різні дії (синхронізовано з frontend XPWidget.jsx)
const XP = {
  EVENT_ATTENDED: 50, // участь у події (статус attended)

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		119

```

URGENT_EVENT: 30, // бонус за термінову подію (Надзвичайні ситуації)
TASK_COMPLETED: 20, // виконане завдання
TASK_EARLY: 10, // бонус за виконання до дедлайну
POSITIVE_REVIEW: 15, // бонус за відгук 4–5 зірок від організатора
};

// Рівні (дзеркало frontend LEVELS в XPWidget.jsx)
const LEVELS = [
  { name: 'Новачок', min: 0, icon: '🐣' },
  { name: 'Активіст', min: 100, icon: '🌸' },
  { name: 'Досвідчений', min: 300, icon: '★' },
  { name: 'Ветеран', min: 600, icon: '👴' },
  { name: 'Експерт', min: 1000, icon: '🏆' },
];

const getLevel = (xp = 0) => {
  let level = LEVELS[0];
  for (const l of LEVELS) {
    if (xp >= l.min) level = l;
  }
  return level;
};

const getNextLevel = (xp = 0) => LEVELS.find(l => l.min > xp) || null;

/**
 * Нараховує XP волонтеру.
 * Якщо рівень підвищився — надсилає in-апп нотифікацію + email.
 */
const addExperience = async (userId, amount) => {
  if (!userId || amount <= 0) return;

  try {
    // Поточний XP + email до нарахування
    const before = await pool.query(
      'SELECT experience, email, name FROM users WHERE id = $1',
      [userId]
    );
    if (before.rows.length === 0) return;

    const { experience: xpBefore, email, name } = before.rows[0];
    const xpAfter = (xpBefore || 0) + amount;

    // Оновлюємо XP
    await pool.query(
      'UPDATE users SET experience = $1 WHERE id = $2',
      [xpAfter, userId]
    );

    // Перевіряємо підвищення рівня
    const levelBefore = getLevel(xpBefore || 0);

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				120
Змн.	Арк.	№ докум.	Підпис	Дата		

```

const levelAfter = getLevel(xpAfter);

if (levelAfter.min > levelBefore.min) {
  // Level-up!
  const nextLevel = getNextLevel(xpAfter);
  const xpToNext = nextLevel ? nextLevel.min - xpAfter : 0;

  // In-app ноти́фікація
  const { createNotification } = require('../controllers/notificationController');
  await createNotification(
    userId,
    'level_up',
    `${levelAfter.icon} Вітаємо! Ви досягли рівня "${levelAfter.name}"! (${xpAfter}
XP)`,
    '/profile'
  );

  // Email ноти́фікація
  const { sendLevelUp } = require('./email');
  await sendLevelUp({
    to: email,
    name,
    levelName: levelAfter.name,
    levelIcon: levelAfter.icon,
    xp: xpAfter,
    nextLevelName: nextLevel?.name || null,
    xpToNext,
  });
}
} catch (err) {
  console.error('[addExperience error]', err.message);
}
};

module.exports = { addExperience, XP, LEVELS, getLevel, getNextLevel };

```

frontend/src/App.jsx (Маршрутизація, PrivateRoute, AdminRoute):

```

import { BrowserRouter, Routes, Route, Navigate, useLocation } from 'react-router-dom';
import { AuthProvider, useAuth } from './context/AuthContext';
import { LangProvider } from './context/LangContext';
import { ThemeProvider } from './context/ThemeContext';
import Navbar from './components/Navbar';
import Footer from './components/Footer';
import LoginPage from './pages/LoginPage';
import RegisterPage from './pages/RegisterPage';
import EventsPage from './pages/EventsPage';
import EventDetailPage from './pages/EventDetailPage';
import ProfilePage from './pages/ProfilePage';
import UserProfilePage from './pages/UserProfilePage';
import AdminPage from './pages/AdminPage';
import MessagesPage from './pages/MessagesPage';

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		121

```

import HomePage      from './pages/HomePage';
import AnalyticsPage from './pages/AnalyticsPage';
import NotFoundPage  from './pages/NotFoundPage';
import OrganizationsPage from './pages/OrganizationsPage';
import AuthCallbackPage from './pages/AuthCallbackPage';

const PrivateRoute = ({ children }) => {
  const { token } = useAuth();
  return token ? children : <Navigate to="/login" />;
};

const AdminRoute = ({ children }) => {
  const { token, user } = useAuth();
  if (!token) return <Navigate to="/login" />;
  if (user?.role !== 'admin') return <Navigate to="/events" />;
  return children;
};

function AppRoutes() {
  const { pathname } = useLocation();
  const { token } = useAuth();

  return (
    <div style={{ display: 'flex', flexDirection: 'column', height: '100vh', overflow: 'hidden' }}>
      <Navbar />
      {pathname === '/messages' ? (
        /* Messages: власний layout — без Footer, внутрішній скрол */
        <div style={{ flex: 1, overflow: 'hidden', display: 'flex', minHeight: 0 }}>
          {token ? <MessagesPage /> : <Navigate to="/login" />}
        </div>
      ) : (
        /* Всі інші сторінки: звичайний скролл + Footer внизу */
        <div style={{ flex: 1, overflow: 'auto' }}>
          <Routes>
            <Route path="/" element={<HomePage />} />
            <Route path="/login" element={<LoginPage />} />
            <Route path="/register" element={<RegisterPage />} />
            <Route path="/auth/callback" element={<AuthCallbackPage />} />
            <Route path="/events" element={<EventsPage />} />
            <Route path="/organizations" element={<OrganizationsPage />} />
            <Route path="/events/:id" element={<EventDetailPage />} />
            <Route path="/profile" element={
              <PrivateRoute><ProfilePage /></PrivateRoute>
            } />
            <Route path="/users/:id" element={<UserProfilePage />} />
            <Route path="/admin" element={
              <AdminRoute><AdminPage /></AdminRoute>
            } />
            <Route path="/analytics" element={
              <PrivateRoute><AnalyticsPage /></PrivateRoute>
            } />
            <Route path="*" element={<NotFoundPage />} />
          </Routes>
        </div>
      )
    </div>
  );
}

```

		Царук К.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		122

```

        </Routes>
        <Footer />
      </div>
    )}
  </div>
);
}

export default function App() {
  return (
    <ThemeProvider>
      <LangProvider>
        <AuthProvider>
          <BrowserRouter>
            <AppRoutes />
          </BrowserRouter>
        </AuthProvider>
      </LangProvider>
    </ThemeProvider>
  );
}

```

frontend/src/context/AuthContext.jsx (Глобальний контекст автентифікації):

```

import { createContext, useContext, useState, useEffect } from 'react';

const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const [user, setUser] = useState(null);
  const [token, setToken] = useState(localStorage.getItem('token'));

  useEffect(() => {
    const savedUser = localStorage.getItem('user');
    if (savedUser) setUser(JSON.parse(savedUser));
  }, []);

  const login = (userData, tokenData) => {
    setUser(userData);
    setToken(tokenData);
    localStorage.setItem('token', tokenData);
    localStorage.setItem('user', JSON.stringify(userData));
  };

  const updateUser = (userData) => {
    setUser(userData);
    localStorage.setItem('user', JSON.stringify(userData));
  };

  const logout = () => {
    setUser(null);
  };
}

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		123

```

    setToken(null);
    localStorage.removeItem('token');
    localStorage.removeItem('user');
  };

  return (
    <AuthContext.Provider value={{ user, token, login, logout, updateUser }}>
      {children}
    </AuthContext.Provider>
  );
};

export const useAuth = () => useContext(AuthContext);

components/XPWidget.jsx:

import { useLang } from '../context/LangContext';

export const LEVELS = [
  { name: 'Новачок', min: 0, icon: '👶', color: '#607d8b', bg: '#eceff1' },
  { name: 'Активіст', min: 100, icon: '🌿', color: '#2e7d32', bg: '#e8f5e9' },
  { name: 'Досвідчений', min: 300, icon: '★', color: '#1565c0', bg: '#e3f2fd' },
  { name: 'Ветеран', min: 600, icon: '♂', color: '#6a1b9a', bg: '#f3e5f5' },
  { name: 'Експерт', min: 1000, icon: '🏆', color: '#e65100', bg: '#fff3e0' },
];

export function getLevel(xp = 0) {
  let level = LEVELS[0];
  for (const l of LEVELS) {
    if (xp >= l.min) level = l;
  }
  return level;
}

// compact=true — лише бейдж (для навібару/карток)
// compact=false — повний віджет з прогрес-баром
export default function XPWidget({ xp = 0, compact = false }) {
  const { t } = useLang();
  const levelNames = [t('xp.novice'), t('xp.activist'), t('xp.experienced'), t('xp.veteran'),
t('xp.expert')];

  const level = getLevel(xp);
  const nextLevel = LEVELS.find(l => l.min > xp);
  const toNext = nextLevel ? nextLevel.min - xp : 0;
  const progressPct = nextLevel
    ? Math.round(((xp - level.min) / (nextLevel.min - level.min)) * 100)
    : 100;

  const levelIndex = LEVELS.indexOf(level);
  const displayName = levelNames[levelIndex] ?? level.name;
  const nextLevelIndex = nextLevel ? LEVELS.indexOf(nextLevel) : -1;

```

		Царук К.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		124

```

const nextDisplayName = nextLevelIndex >= 0 ? levelNames[nextLevelIndex] : "";

if (compact) {
  return (
    <span style={{
      display: 'inline-flex', alignItems: 'center', gap: '4px',
      background: level.bg, color: level.color,
      borderRadius: '20px', padding: '3px 10px',
      fontSize: '12px', fontWeight: '700',
    }}>
      {level.icon} {displayName}
    </span>
  );
}

return (
  <div style={{ marginTop: '12px' }}>
    { /* Рівень + загальний XP */ }
    <div style={{ display: 'flex', alignItems: 'center', gap: '8px', marginBottom: '8px' }}>
      <span style={{
        display: 'inline-flex', alignItems: 'center', gap: '5px',
        background: level.bg, color: level.color,
        borderRadius: '20px', padding: '4px 14px',
        fontSize: '14px', fontWeight: '700',
      }}>
        {level.icon} {displayName}
      </span>
      <span style={{ fontSize: '13px', color: '#888', fontWeight: '500' }}>
        {xp} XP
      </span>
    </div>

    { /* Прогрес-бар */ }
    <div style={{ display: 'flex', alignItems: 'center', gap: '8px' }}>
      <div style={{
        flex: 1, height: '7px', background: '#f0f0f0',
        borderRadius: '4px', overflow: 'hidden',
      }}>
        <div style={{
          width: `${progressPct}%`, height: '100%',
          background: level.color, borderRadius: '4px',
          transition: 'width 0.6s ease',
          minWidth: xp > 0 ? '6px' : '0',
        }} />
      </div>
      {nextLevel ? (
        <span style={{ fontSize: '11px', color: '#aaa', flexShrink: 0, whiteSpace: 'nowrap' }}
          {t('xp.toNext')} {toNext} XP → {nextLevel.icon}
        </span>
      ) : (
        <span style={{ fontSize: '11px', color: level.color, fontWeight: '600', flexShrink: 0

```

		Царук К.А.			ІІЗ.КР.Б – 121 – 26 – ІІЗ	Арк.
		Чижмоторя О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		125

```

}}>
        {t('xp.maxLevel')}
    </span>
    })
</div>

    { /* Підказка як заробити ХР */ }
    <div style={{ marginTop: '6px', fontSize: '11px', color: '#bbb' }}>
        {t('xp.hint')}
    </div>
</div>
);
}

```

		Царук К.А.			ІПЗ.КР.Б – 121 – 26 – ІЗ	Арк.
		Чижмотря О.В.				
Змн.	Арк.	№ докум.	Підпис	Дата		126