

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:

«Вебдодаток для організації міських заходів»

Виконав студент групи ПЗ-22-3
ТРОХИМЧУК Андрій Русланович

Керівник роботи:
ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:
ТОЛСТОЙ Ігор Анатолійович

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: **ТРОХИМЧУК Андрій Русланович**

Керівник роботи: **ЛОБАНЧИКОВА Надія Миколаївна**

Тема роботи: **«Вебдодаток для організації міських заходів»**,
затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №74/с**

Вихідні дані для роботи: **процес автоматизації продажу квитків та взаємодії між організаторами заходів та відвідувачами, який використовує веб-платформу для візуалізації каталогу подій, інтерактивного вибору місць та підвищення ефективності реалізації квитків.**

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Лобанчикова Н.М.	01.03.2026р.	01.03.2026р.
2	Лобанчикова Н.М.	01.04.2026р.	01.04.2026р.
3	Лобанчикова Н.М.	01.05.2026р.	01.05.2026р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з вебдодатку для організації міських заходів та пояснювальної записки. Пояснювальна записка до випускної роботи містить 88 сторінок, 52 ілюстрацій та 16 таблиць.

Метою роботи є розробка та програмна реалізація вебдодатку для організації міських заходів, суттю якої є автоматизація взаємодії між організаторами заходів та відвідувачами з використанням каталогу подій та інструментів інтерактивного бронювання місць для візуалізації залів та оформлення замовлень у реальному часі. Архітектура сервісу базується на використанні фреймворку Next.js [2] для реалізації як клієнтської, так і серверної частини (Full-stack підхід) у поєднанні з реляційною базою даних PostgreSQL [15]. Використання технології SSR (Server Side Rendering) забезпечує високу швидкість завантаження сторінок івентів та ефективну SEO-оптимізацію, що є критичним для залучення органічного трафіку та просування платформи.

Додаток має інтуїтивно зрозумілий інтерфейс із застосуванням сучасних UI-патернів, що дозволяє користувачам зручно переглядати афішу, фільтрувати події за категоріями чи датами, взаємодіяти зі схемою залу та оперативно купувати квитки.

КЛЮЧОВІ СЛОВА: EVENT-ПЛАТФОРМА, ПРОДАЖ КВИТКІВ, NEXT.JS, POSTGRESQL, БРОНЮВАННЯ, ОНЛАЙН-ОПЛАТА, SSR, CLR, ISR, SSG

					ІПЗ.КР.Б-121-26-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Розробка вебдодатку для організації міських заходів	Лім.	Арк.	Аркушів
Розроб.		Трохимчук А.Р.						
Керівник		Лобанчикова Н.М.					3	88
Н. контр.						Житомирська політехніка, група ІПЗ-22-3		
Затвердив								

ABSTRACT

The bachelor's thesis consists of a web application for organizing city events and an explanatory note. The explanatory note contains 88 pages, 52 figures, and 11 tables.

The objective of the thesis is to develop an event search and ticket sales platform to automate the interaction between event organizers and attendees, utilizing an event catalog and interactive seat booking tools for venue visualization and real-time order processing. The service architecture is based on the Next.js [2] framework for implementing both the client and server sides (full-stack approach) in combination with a PostgreSQL [15] relational database. The use of SSR (Server-Side Rendering) technology ensures high loading speeds for event pages and effective SEO optimization, which is critical for attracting organic traffic and promoting the platform.

The application features an intuitive interface employing modern UI patterns, allowing users to conveniently browse upcoming events, filter events by category or date, interact with the seating chart, and quickly purchase tickets.

KEYWORDS: EVENT PLATFORM, TICKET SALES, NEXT.JS, POSTGRESQL, BOOKING, ONLINE PAYMENT, SSR, CLR, ISR, SSG.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		4

ЗМІСТ

ВСТУП.....	7
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ВЕБДОДАТКУ ДЛЯ ОРГАНІЗАЦІЇ МІСЬКИХ ЗАХОДІВ	9
1.1 Аналіз інструментів для створення вебдодатків	9
1.2 Аналіз аналогів програмного продукту.....	11
1.3 Вибір архітектури вебдодатку.....	17
Висновки до першого розділу	21
РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБДОДАТКУ ДЛЯ ОРГАНІЗАЦІЇ МІСЬКИХ ЗАХОДІВ	23
2.1 Визначення варіантів використання та об'єктно-орієнтованої структури системи	23
2.2 Проектування компонентної структури та взаємодії модулів	39
2.3 Розробка основних алгоритмів роботи платформи.....	44
Висновки до другого розділу	48
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБДОДАТКОМ ДЛЯ ОРГАНІЗАЦІЇ МІСЬКИХ ЗАХОДІВ	50
3.1 Порядок встановлення та налаштування параметрів системи.....	50
3.2 Структура інтерфейсу вебсистеми для міських заходів	53
3.3 Тестування роботи програмного продукту	68
Висновок до третього розділу	73
ВИСНОВКИ.....	74
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	76
ДОДАТКИ	79
Додаток А:.....	80

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – Application Programming Interface

TS – TypeScript

БД – база даних

CLI – Command Line Interface

HTTP – протокол передавання гіпертексту

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						6
Змн.	Арк.	№ докум.	Підпис	Дата		

ВСТУП

Актуальність теми полягає у стабільно високому попиті на відвідування культурних, розважальних та освітніх заходів, а також у потребі користувачів мати зручний і швидкий інструмент для придбання квитків. У сучасних умовах ринок івент-індустрії вимагає надійних цифрових рішень, оскільки існуючі платформи часто стикаються з проблемами високих навантажень під час старту продажів та недоліками користувацького інтерфейсу. Створення сучасного вебдодатка для пошуку подій та продажу квитків дозволяє автоматизувати процес взаємодії між організаторами та відвідувачами, забезпечити простоту оформлення замовлень та гарантувати безпеку фінансових транзакцій.

Метою роботи є розробка та програмна реалізація вебдодатку для організації міських заходів, суттю якої є автоматизація взаємодії між організаторами заходів та відвідувачами з використанням каталогу подій та інструментів інтерактивного бронювання місць та оформлення замовлень у реальному часі. Для реалізації поставленої мети необхідно виконати низку завдань, зокрема: провести аналіз ринку івент-платформ та сервісів продажу квитків і сформулювати вимоги до функціональності системи, здійснити вибір засобів розробки з обґрунтуванням використання сучасного технологічного стеку, а також розробити зручну клієнтську частину платформи та систему управління подіями для організаторів.

Об'єктом дослідження є процес програмної реалізації створення вебдодатка для агрегації подій та реалізації електронних квитків.

Предметом дослідження є моделі, методи та програмні засоби побудови вебдодатку для організації міських заходів. Дослідження також охоплює питання масштабованості платформи під час пікових навантажень та забезпечення коректного обліку кількості доступних квитків у реальному часі для кінцевих користувачів.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						7
Змн.	Арк.	№ докум.	Підпис	Дата		

Методи дослідження. У роботі застосовувався комплексний підхід, що базується на методах системного та порівняльного аналізу для дослідження існуючих рішень на ринку та вибору оптимального технологічного стеку. Для побудови архітектури та розробки функціональних модулів платформи використовувалися методи об'єктно-орієнтованого проектування, а для перевірки надійності, ефективності алгоритмів обробки замовлень та безпеки створеної системи застосовувалися методи емпіричного тестування.

Практичне значення одержаних результатів. Одержані результати мають безпосереднє прикладне значення, оскільки розроблений програмний комплекс може бути використаний як готова основа для запуску реального сервісу продажу квитків. Впровадження такої системи дозволить організаторам заходів автоматизувати процеси реалізації квитків та знизити витрати на комісії посередників, а кінцевим користувачам отримати швидкий, зрозумілий та безперебійний доступ до купівлі квитків навіть в умовах пікових навантажень на сервери.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		8

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ЗАСТОСУВАННЯ ВЕБДОДАТКУ ДЛЯ ОРГАНІЗАЦІЇ МІСЬКИХ ЗАХОДІВ

1.1 Аналіз інструментів для створення вебдодатків

У сучасному світі вебдодатки стали невід’ємною частиною цифрової інфраструктури. Вони використовуються в електронній комерції, освіті, охороні здоров’я, розвагах та багатьох інших сферах. Для їх створення існує велика кількість інструментів, фреймворків та технологій, які дозволяють розробникам реалізовувати найскладніші функціональні вимоги. Розглянемо основні сучасні інструменти, що використовуються при створенні масштабованих вебдодатків, з поділом на клієнтську та серверну частини, бази даних та інфраструктурні рішення.

Фронтенд (клієнтська частина) відповідає за інтерфейс користувача та взаємодію з ним. Сьогодні стандартом де-факто для створення складних і динамічних інтерфейсів є використання компонентного підходу, лідером якого є бібліотека React [12]. Враховуючи вимоги до пошукової оптимізації (SEO) та швидкості завантаження сторінок, розробники все частіше переходять від класичних односторінкових додатків (SPA) до метафреймворків. Найбільшої популярності серед них набув Next.js [2] повноцінний фреймворк, який забезпечує гібридний рендеринг (серверний та клієнтський), вбудовану маршрутизацію та оптимізацію ресурсів. Для стилізації інтерфейсу сучасним стандартом є використання утилітарного фреймворку Tailwind CSS, який дозволяє швидко створювати адаптивні дизайни без написання громіздких кастомних таблиць стилів. Крім того, розробку значно пришвидшує використання готових компонентних рішень, таких як Radix UI та Shadcn, що гарантують високу доступність (accessibility) та сучасний зовнішній вигляд платформи. Для ефективного управління станом асинхронних серверних даних та їх кешування на клієнті активно застосовується бібліотека TanStack React Query.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		9

Бекенд (серверна частина) у сучасних вебдодатках на базі метафреймворків часто тісно інтегрується з клієнтською частиною. Використання Next.js [2] на базі середовища виконання Node.js [13] дозволяє реалізовувати серверну логіку, обробку запитів та роботу з API в єдиному репозиторії. Для забезпечення надійності даних та типізації на всіх рівнях додатку використовується мова TS [14] у поєднанні з бібліотекою валідації схем Zod. Важливим аспектом будь-якої системи електронної комерції є безпека користувачів: для реалізації сучасних механізмів автентифікації та управління сесіями ефективно застосовується спеціалізована бібліотека Better Auth [4]. Додатково, архітектура сучасного сервісу з продажу квитків вимагає інтеграції надійних сторонніх інструментів, зокрема платіжних шлюзів Stripe [5] для безпечної обробки фінансових транзакцій та поштових сервісів Resend [7] для транзакційних розсилок електронних квитків.

Важливим елементом архітектури є система управління базами даних. Для збереження транзакційної цілісності фінансових операцій та суворого обліку місць оптимальним є вибір реляційних СУБД, зокрема PostgreSQL. Сучасним еволюційним кроком у розвитку баз даних є перехід до хмарних безсерверних (serverless) рішень, таких як Neon Postgres [3]. Це дозволяє системі автоматично масштабувати обчислювальні ресурси під час пікових навантажень та знижувати витрати в періоди простою. Для безпечної та зручної взаємодії серверної частини з базою даних замість написання «сирих» SQL-запитів доцільно використовувати сучасні інструменти об'єктно-реляційного відображення, зокрема Drizzle ORM [19]. Цей інструмент забезпечує сувору типізацію, високу швидкість виконання запитів та зручне управління міграціями бази даних.

Після розробки програмного продукту необхідно забезпечити його надійне розгортання та підтримку. Замість класичних віртуальних серверів або контейнеризації через Docker, сучасні додатки на Next.js [2] ефективно розгортаються на крайових (edge) мережах доставки контенту. Використання глобальної інфраструктури Cloudflare у поєднанні з адаптером OpenNext

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						10
Змн.	Арк.	№ докум.	Підпис	Дата		

дозволяє обробляти запити користувачів на серверах, географічно найближчих до них, що критично зменшує затримки. Для забезпечення високої якості програмного коду та безперебійної роботи логіки застосовуються інструменти тестування. Контроль дотримання стандартів написання коду здійснюється за допомогою лінтерів та форматерів ESLint [27] і Prettier [28], а управління залежностями за допомогою пакетного менеджера Yarn.

Проведений аналіз сучасних інструментів для створення вебдодатків дозволив визначити переваги використання екосистеми Next.js [2], безсерверних баз даних та крайової інфраструктури, що створює міцне підґрунтя для розробки швидкої, безпечної та масштабованої платформи.

1.2 Аналіз аналогів програмного продукту

Платформа «Karabas» (<https://zhytomyr.karabas.com/>) є одним із найстаріших та найбільших національних квиткових операторів України. Її функціонал базується на обробці великого масиву даних, що дозволяє системі підтримувати наймасштабнішу базу подій, включаючи локальні заходи в регіонах.

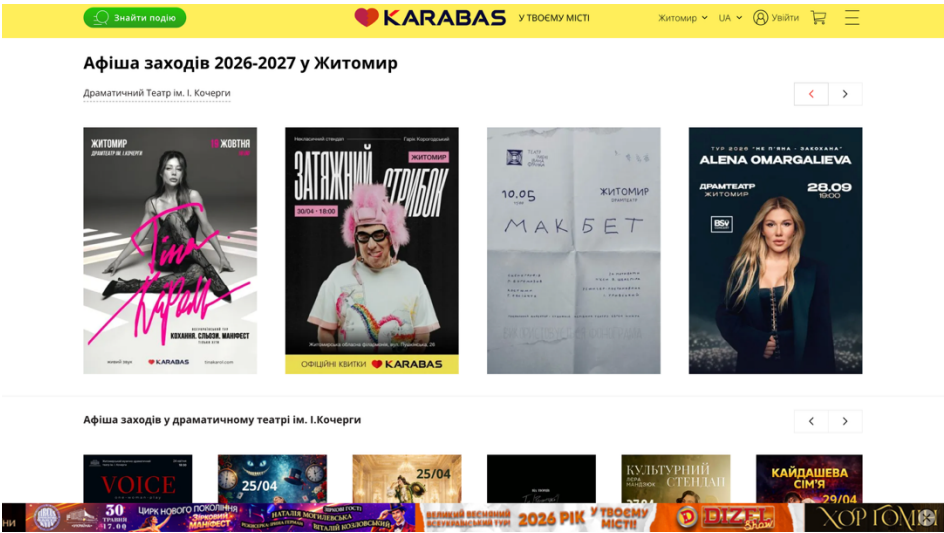


Рис.1.1 Головна сторінка веб-сайту.

На головній сторінці платформи користувача одразу зустрічає структурований каталог подій, згрупований за локаціями та хронологією.

Функціонально верхня навігаційна панель забезпечує швидкий доступ до глобального текстового пошуку, зміни міста для налаштування регіональної видачі результатів, авторизації користувача, а також керування кошиком замовлень. Алгоритм подачі контенту реалізований через компоненти горизонтального прокручування (каруселі), що дає змогу платформі ефективно демонструвати великий обсяг актуальних афіш у межах одного екрана, зменшуючи необхідність довгого вертикального скролінгу.

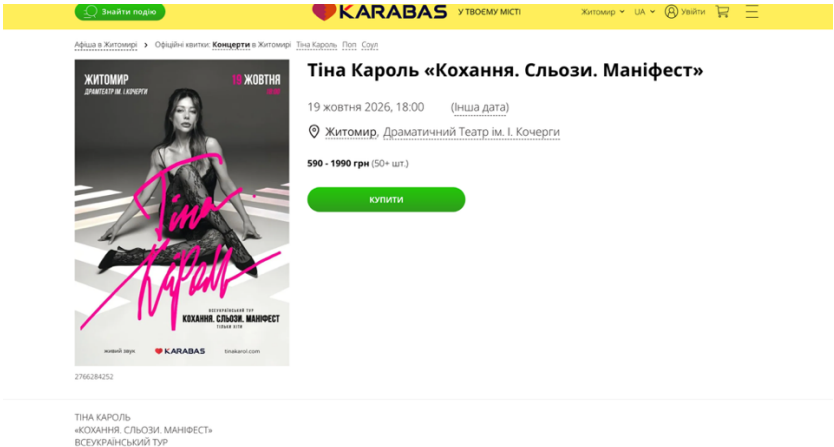


Рис. 1.2. Сторінка події.

На сторінці конкретного заходу зосереджено базовий функціонал ознайомлення та ініціації транзакції. Інтерфейс забезпечує користувача ієрархічною навігацією (хлібними крихтами) для швидкого повернення до каталогу, а також виводить ключові метадані з бази даних: дату, точний час та місце проведення. Важливим функціональним вузлом є модуль відображення ціноутворення, який динамічно підтягує актуальний діапазон вартості та інформує про залишок доступних квитків у реальному часі. Ключовим елементом взаємодії на цьому етапі є кнопка переходу безпосередньо до процесу купівлі, яка запускає логіку оформлення замовлення.

Загальна функціональна характеристика платформи «Karabas» свідчить про високу здатність архітектури агрегувати масштабні бази даних, підтримувати актуальність інформації для різних регіонів та забезпечувати надійну електронну комерцію. Водночас наявність великої кількості паралельних навігаційних шляхів, банерних систем та загальна інформаційна

насиченість інтерфейсу дещо ускладнюють шлях користувача, тоді як сучасні рішення для швидкого продажу квитків вимагають більш лінійного, інтуїтивного та оптимізованого алгоритму переходу від вибору події до фінальної оплати.

Сервіс «Kontramarka» (<https://zhyt.kontramarka.ua/uk/concert>) представляє собою сучасну квиткову платформу з акцентом на технологічність та SEO-оптимізацію

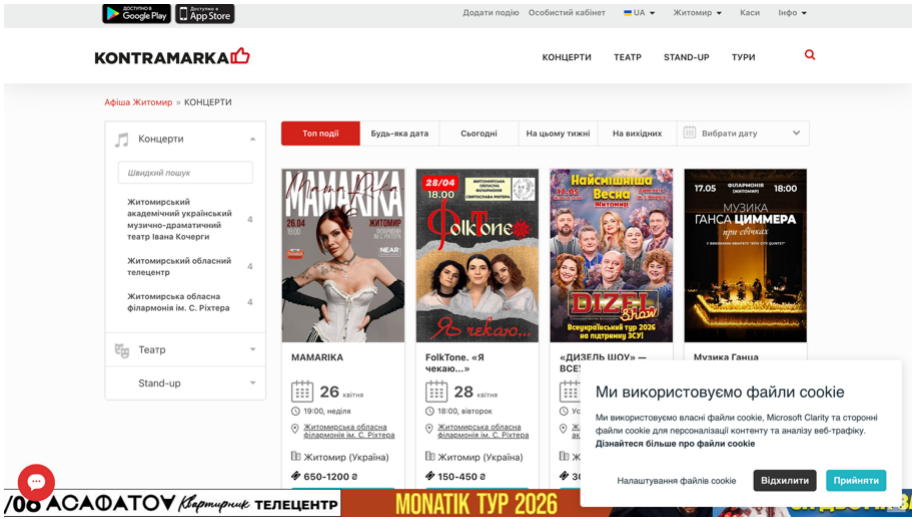


Рис. 1.3. Головна сторінка веб-сайту.

На сторінці каталогу платформи реалізовано розширену систему агрегації та багатокритеріальної фільтрації подій. Зліва розташований навігаційний модуль, який дозволяє здійснювати швидкий текстовий пошук та сегментувати заходи за конкретними майданчиками (театри, концертні зали) або тематичними напрямками. У верхній частині інтерфейсу імплементовано додатковий блок часової фільтрації, де за допомогою вкладок користувач може миттєво відсортувати події за датою («сьогодні», «на вихідних» або через вбудований календар). Основна робоча область відображає результати пошуку у вигляді структурованої сітки карток. Кожна картка функціонує як самостійний інформаційний блок, що одразу виводить усі критично важливі для прийняття рішення метадані: статус доступності, розклад, точну геолокацію та актуальний діапазон цін, мінімізуючи кількість зайвих переходів по сторінках.

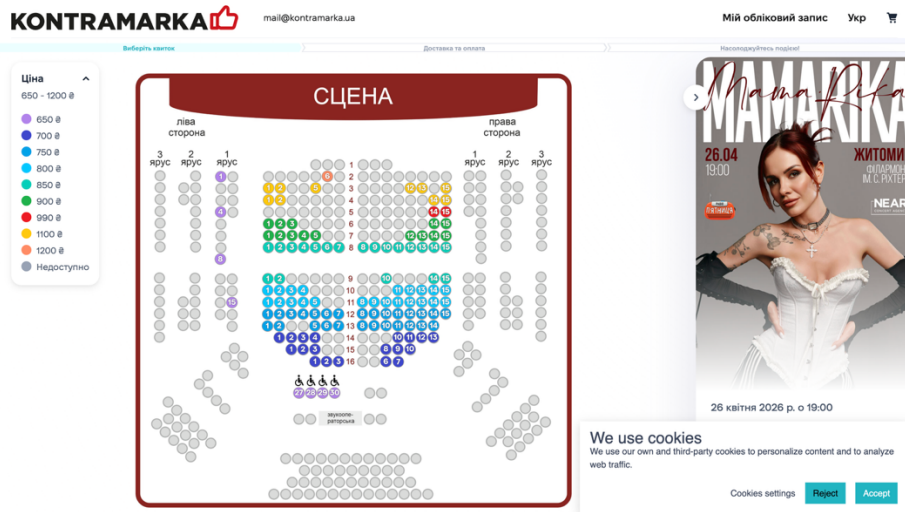


Рис. 1.4. Інтерактивна схема залу.

На етапі ініціації покупки користувач потрапляє до інтерфейсу безпосереднього формування замовлення. У верхній частині екрана впроваджено лінійний індикатор прогресу (кроки оформлення), який візуально структурує алгоритм транзакції від вибору квитка до фінальної оплати. Зліва розміщено модуль категоризації за ціною, що дозволяє миттєво відфільтрувати доступні пропозиції відповідно до бюджету покупця. Центральна область функціонально призначена для визначення цінової категорії та вибору необхідної кількості квитків. Після виконання цієї дії система ініціює алгоритм тимчасового резервування та дозволяє перехід до захищеного модуля оплати. Права панель слугує статичним інформаційним віджетом, який постійно відображає деталі обраного заходу, що є важливим елементом запобігання помилкам з боку користувача під час оформлення.

Загальна функціональна характеристика платформи «Kontramarka» свідчить про глибоку оптимізацію шляху користувача (User Journey). Архітектура системи побудована навколо потужних інструментів фільтрації, що дозволяють ефективно обробляти великі масиви даних та швидко видавати релевантний результат. Сама логіка процесу купівлі реалізована за принципом покрокового алгоритму з чіткою візуальною ієрархією. Це робить флоу транзакції максимально лінійним, зрозумілим та швидким, що в підсумку позитивно впливає на загальний показник успішних конверсій (завершених покупок).

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		14

Платформа «TicketsBox» (<https://zhytomyr.ticketsbox.com/>)

позиціонується як динамічна система для продажу електронних квитків (e-tickets) та управління подіями з фокусом на цифровізацію пропускнуго процесу

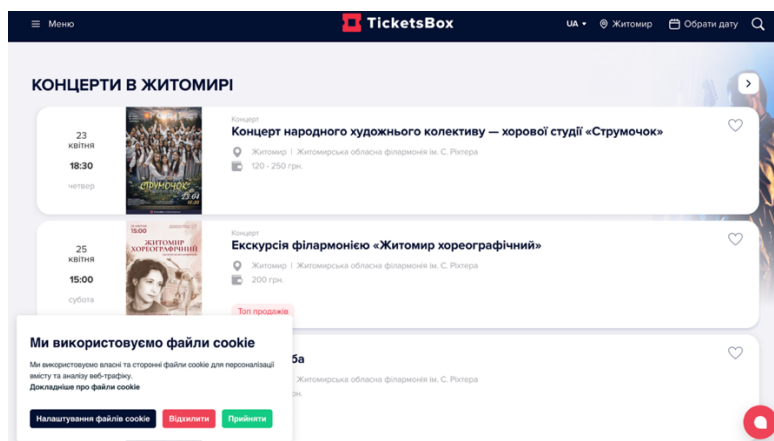


Рис. 1.5. Головна сторінка веб-сайту.

На головній сторінці платформи реалізовано лінійний формат відображення каталогу подій у вигляді списку горизонтальних інформаційних карток. Функціонально верхня панель навігації містить інструменти для глобального пошуку, вибору локації (міста) та фільтрації заходів за конкретною датою за допомогою вбудованого календаря. Кожна картка події виступає інтерактивним модулем, який миттєво надає користувачеві ключові метадані з бази даних: дату та час проведення, назву, точну геолокацію майданчика та діапазон цін. Крім того, в інтерфейс інтегровано мікрофункціонал додавання подій до списку обраного (іконка у вигляді серця), що дозволяє відвідувачам зберігати заходи для подальшої швидкої покупки.

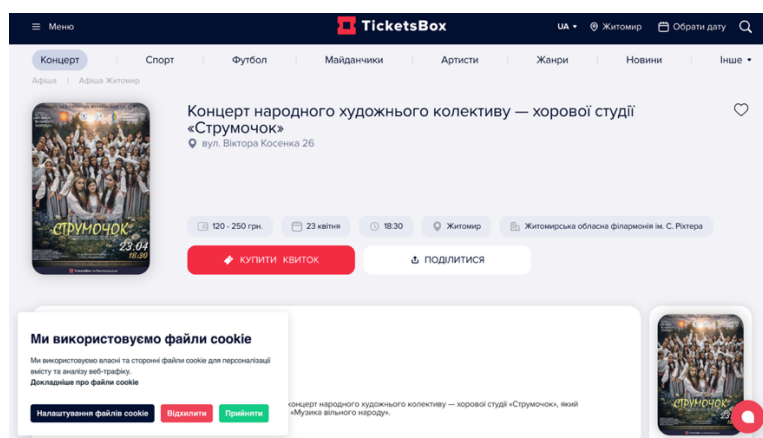


Рис. 1.6 Сторінка події.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		15

На сторінці конкретного заходу реалізовано структуровану подачу розширеної інформації для ініціації транзакції. Під основним навігаційним меню розташовано додатковий блок рубрикатора (концерти, спорт, артисти, жанри), що забезпечує швидкий перехід між різними типами контенту без необхідності повертатися на головну сторінку. Основні атрибути події (ціна, дата, час, місто та назва закладу) згруповані у вигляді компактних інформаційних тегів, що суттєво пришвидшує візуальне зчитування даних. Головним функціональним елементом сторінки є яскраво виділена кнопка «Купити квиток», яка слугує тригером для запуску прямого алгоритму оформлення замовлення. Додатково імплементовано модуль соціальної взаємодії функцію швидкого поширення посилання на подію.

Загальна функціональна характеристика платформи «TicketsBox» демонструє орієнтацію на сучасний компонентний підхід та чистоту інтерфейсу. Архітектура сервісу фокусується на швидкій доставці релевантної інформації: від ефективного текстового та календарного пошуку до чіткого структурування метаданих на сторінці заходу. Відсутність перевантаженості візуальними елементами та лінійна логіка взаємодії дозволяють користувачеві максимально швидко пройти шлях від пошуку необхідної події до ініціації алгоритму швидкого придбання електронного квитка.

Виходячи з виявлених переваг та недоліків, складемо порівняльну таблицю програмних аналогів (табл. 1.1).

таблиця 1.1

Таблиця порівняння платформ

Функціональна характеристика	Karabas	Kontramarka	TicketsBox	Eventik
Наявність каталогу подій (афіші) та пошуку	Так	Так	Так	Так
Багатокритеріальна фільтрація заходів	Так	Так	Так	Так
Інтерактивна схема залу та вибір конкретного місця	Так	Так	Так	Ні
Алгоритм швидкої купівлі квитків (без прив'язки до місць)	Ні	Ні	Ні	Так

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						16
Змн.	Арк.	№ докум.	Підпис	Дата		

Наявність особистого кабінету користувача	Так	Так	Так	Так
Сучасний адаптивний інтерфейс для мобільних пристроїв	Ні	Так	Так	Так
Оптимізація бази даних під високі навантаження (без таймаутів)	Так	Так	Ні	Так
Відсутність перевантаженості візуальними рекламними банерами	Ні	Так	Так	Так

Порівняльний аналіз програмних аналогів свідчить, що існуючі рішення часто мають обмеження у вигляді прихованих комісій, перевантажених застарілих інтерфейсів або недостатньої стабільності під час пікових навантажень, що перешкоджає ефективній реалізації квитків та погіршує досвід користувачів. Розроблювана система забезпечує кращу SEO-оптимізацію, високу швидкість завантаження інтерактивних схем залів завдяки технології SSR та прозорі умови покупки без неочікуваних зборів. Платформа поєднує в собі переваги професійного квиткового оператора з сучасним підходом до користувацького інтерфейсу та високою технічною надійністю, що робить її найбільш ефективним інструментом для автоматизації продажів та комфортної взаємодії між організаторами заходів і відвідувачами.

1.3 Вибір архітектури вебдодатку

Проведемо аналіз популярних архітектурних підходів, таких як MVC (Model-View-Controller), Layered Architecture (Багаторівнева архітектура), Clean Architecture, Microservices, Monolithic та сучасної методології Feature-Sliced Design (FSD), щоб визначити оптимальний варіант для розроблюваної платформи продажу квитків.

Архітектура MVC логічно розділяє програмний код на три основні компоненти: модель (відповідає за бізнес-логіку та взаємодію з базою даних), представлення (відображає дані для користувача) та контролер (обробляє запити та координує систему).

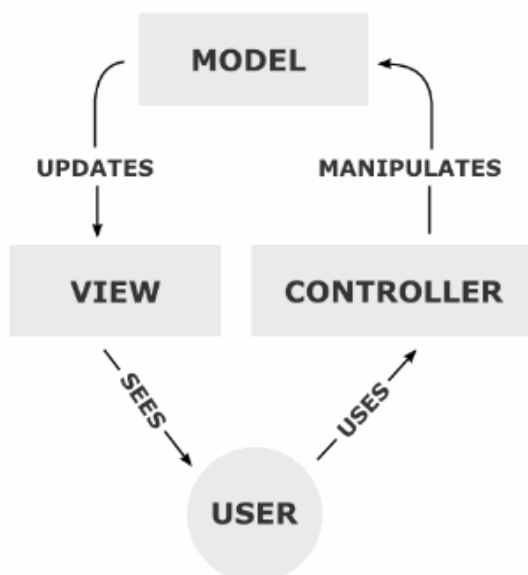


Рис. 1.2. Концепція архітектури MVC

Це розділення дозволяє розробникам зосереджуватись на окремих аспектах програми, проте в умовах сучасних вебдодатків із багатим клієнтським інтерфейсом (Single Page Applications або гібридних застосунків) класичний MVC часто призводить до перевантаження контролерів та ускладнює у масштабуванні фронтенд-частини коду.

Запропонована Робертом Мартіном концепція Clean Architecture ґрунтується на незалежності ядра (доменної логіки) від інфраструктури та фреймворків.

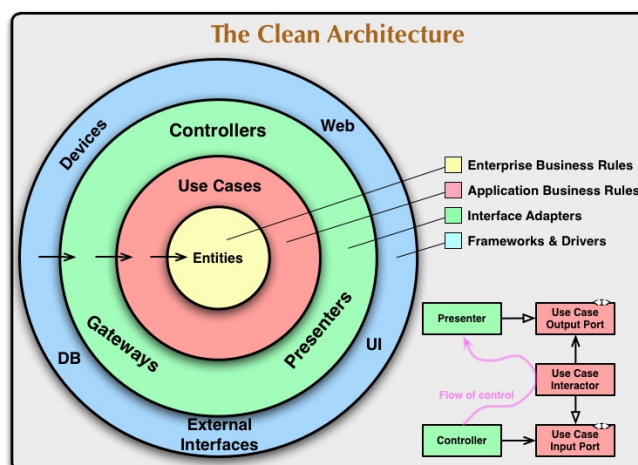


Рис. 1.3. Концепція Clean Architecture

Усі залежності в ній спрямовані до центру: внутрішній шар містить бізнес-правила (Entities), далі йдуть сценарії використання (Use Cases), а зовнішні шари відповідають за інтерфейс користувача та бази даних. Перевагою є абсолютна незалежність від інструментів та легкість тестування. Водночас цей підхід вирізняється високою складністю реалізації, вимагає значного досвіду команди та є надлишковим для проєктів із порівняно прямолінійною бізнес-логікою купівлі квитків.

Наступними підходами, що заслуговують на увагу, є монолітна та мікросервісна архітектури.

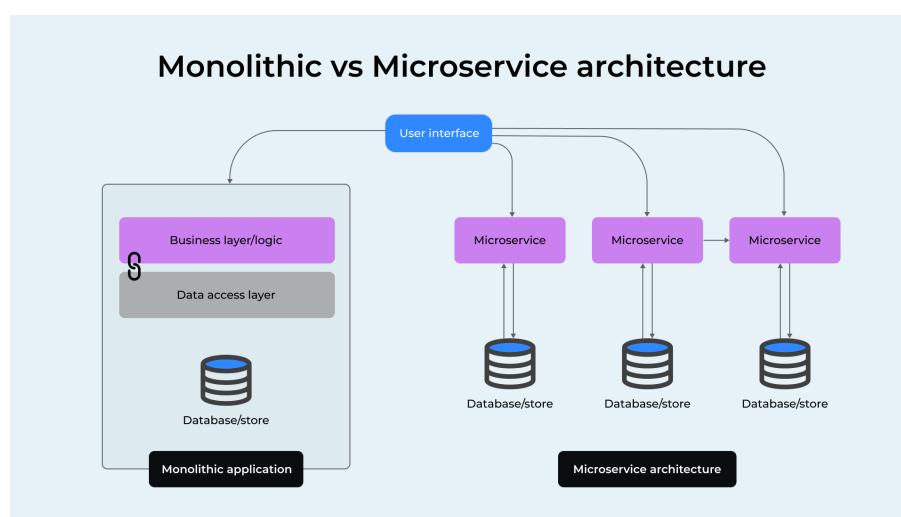


Рис. 1.4. Монолітна архітектура та Мікросервіси

Монолітна архітектура передбачає реалізацію всіх компонентів в одному застосунку, що забезпечує швидкий старт, але створює проблеми з масштабуванням. При використанні мікросервісів система розбивається на набір незалежних модулів. Перевагами такої архітектури є її гнучкість та стійкість до збоїв під високими навантаженнями, однак вона вимагає складної інфраструктури (DevOps), постійного моніторингу та підвищених витрат на підтримку, що не завжди виправдано на початкових етапах запуску продукту.

З огляду на використання сучасного стеку технологій на базі фреймворку Next.js [2], найбільш доцільним для розробки застосунку «Eventik» є використання архітектурної методології Feature-Sliced Design

(FSD). Цей підхід створений спеціально для організації коду сучасних фронтенд- та повностекових додатків і передбачає поділ проєкту на логічні модулі за предметними областями (фічами), а не за типами файлів.

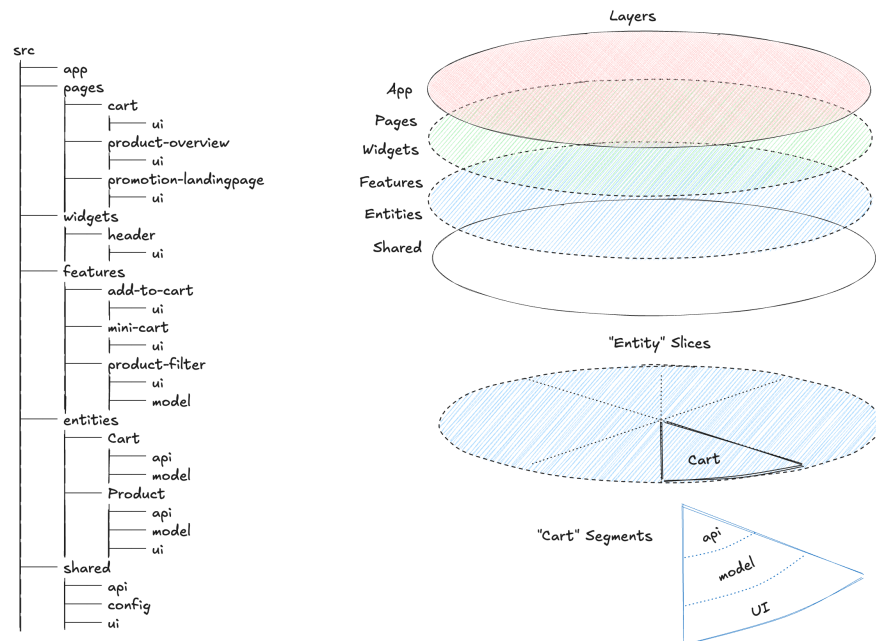


Рис. 1.5. Структура Feature-Sliced Design (FSD)

Згідно з FSD, код поділяється на суворі ієрархічні шари (Layers): App (глобальні налаштування додатку), Widgets (самостійні композиційні блоки, такі як шапка сайту), Features (конкретні бізнес-сценарії, наприклад, модуль купівлі квитка), Entities (бізнес-сутності, такі як квиток або користувач) та Shared (спільний код, UI-компоненти, утиліти). Кожен шар складається зі слайсів (Slices), які групують код за бізнес-доменом, а слайси діляться на сегменти (UI, модель, API). Головне правило FSD полягає в тому, що модулі можуть імпортувати лише ті компоненти, які знаходяться на шарах нижче.

У порівнянні з класичним MVC, архітектура FSD краще підходить для екосистеми React [12] та Next.js [2], оскільки забезпечує високу зв'язність коду всередині одного функціонального модуля. На відміну від мікросервісів, FSD дозволяє розробляти проєкт як єдиний репозиторій, але зберігає модульність, що в майбутньому дозволить легко масштабувати додаток «Eventik», безпечно додавати нові функції (наприклад, інтеграцію нових платіжних систем) та

ізолювати помилки без ризику зламати загальну бізнес-логіку системи. Таким чином, вибір архітектури Feature-Sliced Design є технічно обґрунтованим і оптимальним рішенням для даної випускної кваліфікаційної роботи.

Висновки до першого розділу

У результаті проведеного аналізу встановлено, що створення сучасних вебдодатків для івент-індустрії вимагає комплексного підходу до вибору інструментів та технологій. Розробка таких систем, особливо тих, що передбачають фінансові транзакції та різкі пікові навантаження під час старту продажів, охоплює кілька ключових напрямів – клієнтську частину (фронтенд), серверну логіку (бекенд), безпечну роботу з базами даних, а також сучасну інфраструктуру для безперебійного розгортання.

Було розглянуто спектр сучасних фреймворків та інфраструктурних рішень. Аналіз показав, що для забезпечення високої швидкодії, SEO-оптимізації та зручної розробки повностекових додатків доцільно використовувати гібридні метафреймворки, такі як Next.js [2] на базі екосистеми React [12]. Визначено, що для надійного обліку транзакцій та доступних квитків найкраще підходять реляційні СУБД, зокрема безсерверні хмарні рішення Neon PostgreSQL [3] у поєднанні з типізованими ORM-системами (Drizzle). Окрему увагу приділено розгортанню застосунків на крайових мережах (Edge-інфраструктура Cloudflare), що дозволяє значно мінімізувати затримки та забезпечити стійкість платформи до високих навантажень.

Проведений аналіз провідних українських платформ-аналогів («Karabas», «Kontramarka», «TicketsBox») дозволив виявити ключові недоліки існуючих рішень: перевантаженість інтерфейсів, наявність прихованих комісій, надмірну складність процесу оформлення замовлення (checkout) та технічні збої (таймаути) при великому напливі відвідувачів. Це підтвердило актуальність та необхідність створення оптимізованого продукту з фокусом на лінійний алгоритм швидкої купівлі електронних квитків, без обтяжливого для

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						21
Змн.	Арк.	№ докум.	Підпис	Дата		

бази даних функціоналу інтерактивного вибору місць.

Визначено, що архітектурна методологія Feature-Sliced Design (FSD) успішно вирішує проблеми структурування та масштабування сучасних повностекових застосунків. Завдяки суворому поділу коду на бізнес-домени (слайси) та ієрархічні шари, FSD забезпечує високу модульність, чітке розмежування зон відповідальності, легке тестування та безпечне розширення функціоналу системи в майбутньому.

Отже, в контексті створення сучасного вебдодатка для агрегації подій та швидкого продажу квитків, використання екосистеми Next.js [2] разом із методологією Feature-Sliced Design є технічно виправданим та оптимальним рішенням, що гарантує надійність, швидкодію та максимальний комфорт для кінцевого користувача.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						22
Змн.	Арк.	№ докум.	Підпис	Дата		

РОЗДІЛ 2. ПРОЕКТУВАННЯ ВЕБДОДАТКУ ДЛЯ ОРГАНІЗАЦІЇ МІСЬКИХ ЗАХОДІВ

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

Моделювання варіантів використання є одним із ключових етапів об'єктно-орієнтованого проектування. Воно дозволяє визначити межі програмного комплексу, ідентифікувати основних дійових осіб та формалізувати сценарії їхньої взаємодії з системою. Цей етап перетворює текстові функціональні вимоги у візуальну архітектуру поведінки.

На основі розробленої моделі управління доступом, в архітектурі платформи для управління подіями було виділено 3 незалежних акторів. Детальний опис кожної ролі наведено у таблиці 2.1.

Таблиця 2.1

Ідентифікація акторів системи

Актор	Опис ролі та зона відповідальності
Користувач	Авторизований відвідувач. Шукає події, купує квитки, зберігає їх у власному кабінеті, залишає відгуки та керує особистими даними.
Організатор	Користувач із розширеними правами. Відповідає за створення та редагування подій, відстеження статистики продажів та управління прибутком.
Адміністратор	Технічний та фінансовий керівник платформи. Керує всіма учасниками системи, налаштовує комісії, погоджує виплати та аналізує загальну статистику.

Зважаючи на специфіку платформи, усі варіанти використання системи логічно розділені на функціональні зони взаємодії:

Ключові варіанти використання системи для звичайного користувача охоплюють процес реєстрації та авторизації зі створенням облікового запису та обов'язковим підтвердженням електронної пошти. У рамках взаємодії з платформою користувач має змогу переглядати каталог заходів, зберігати їх у розділі обраного, а також залишати відгуки з відповідною оцінкою події. Безпосередній процес купівлі включає вибір бажаного квитка, його безпечну онлайн-оплату та подальше отримання електронного документа з автоматично згенерованим QR-кодом. Крім того, передбачено функціонал управління профілем, що дозволяє клієнту переглядати й редагувати особисті дані, а також мати швидкий доступ до історії придбаних квитків.

Для ролі організатора ключові прецеденти зосереджені на управлінні контентом, що включає створення заходу шляхом заповнення спеціальної форми, а також подальше редагування або видалення власних публікацій. Важливим аспектом роботи організатора є доступ до аналітичного та фінансового модулів, які забезпечують можливість перегляду детальної статистики продажів зі звітами по реалізованих квитках, а також оформлення запитів на виплату отриманого прибутку.

У просторі адміністрування платформи реалізовано інструменти глобального управління, які дозволяють адміністратору контролювати всіх учасників системи: користувачів, організаторів та модераторів, зокрема здійснювати їх блокування або розблокування. Адміністративний функціонал охоплює процеси глибокої модерації контенту перевірку деталей нових подій, їх підтвердження або відхилення, розгляд скарг, а також блокування заходів із обов'язковим зазначенням причини таких дій. Додатково на рівні адміністрування здійснюється фінансовий контроль із налаштуванням системних комісій та виплат, проводиться моніторинг загальної аналітики платформи та відстежуються дії модераторів із можливістю зняття відповідних прав.

На основі виявлених акторів та формалізованих прецедентів побудовано розширену діаграму варіантів використання системи (рис. 2.1).

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						24
Змн.	Арк.	№ докум.	Підпис	Дата		

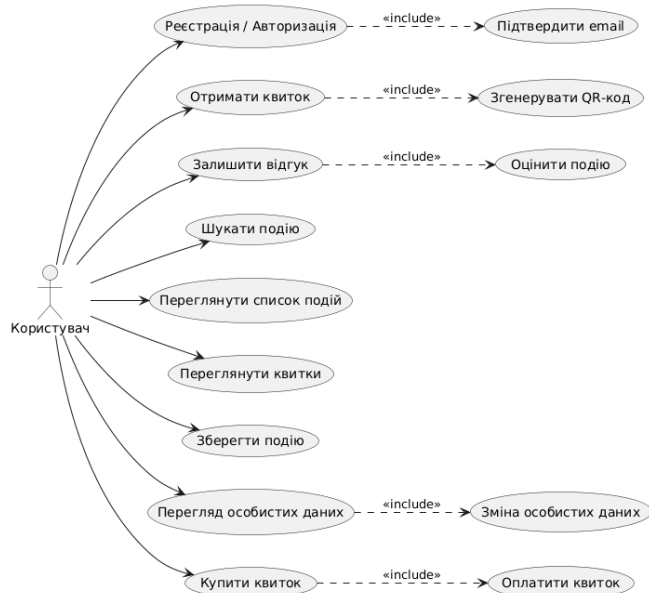


Рис. 2.1. Діаграма варіантів використання актора «Користувач»

На діаграмі для актора «Користувач» відображено базовий набір прецедентів, що забезпечують взаємодію клієнта з платформою. У рамках пошуку та вибору контенту користувач має можливість шукати заходи, переглядати загальний список подій та зберігати їх в обране для подальшого доступу. Процес бронювання місць змодельовано через прецедент купівлі квитка, який нерозривно пов'язаний з його онлайн-оплатою. Успішна транзакція ініціює варіант отримання квитка з обов'язковою автоматичною генерацією унікального QR-коду для пропуску. Додатково актор може переглядати історію своїх замовлень, керувати профілем через перегляд та зміну особистих даних.



Рис. 2.2. Діаграма варіантів використання актора «Адміністратор»

Варіанти використання для простору адміністрування охоплюють процеси глобального контролю та управління всією екосистемою вебдодатка. Адміністратор має найвищі повноваження для керування всіма учасниками системи, зокрема ініціює прецедент керування користувачами, який включає можливість їх блокування або розблокування у разі порушення правил. Управління організаторами виділено в окремий варіант використання. Модерація контенту реалізована через прецедент керування подіями, який нерозривно пов'язаний із перевіркою статистики та відгуків для прийняття рішень щодо публікацій. Фінансовий та системний контроль відображено у варіанті налаштування комісій та виплат, що обов'язково включає процедуру погодження транзакцій для організаторів, а також у прецеденті перегляду глобальної аналітики системи для моніторингу загальної ефективності платформи.

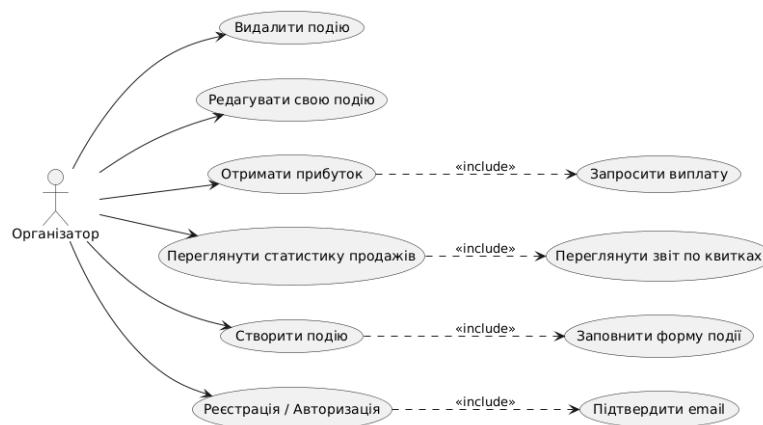


Рис. 2.3. Діаграма варіантів використання актора «Організатор»

Діаграма для актора «Організатор» моделює бізнес-процеси управління контентом та фінансову взаємодію з платформою. Як і клієнт, організатор проходить обов'язковий етап реєстрації з підтвердженням електронної пошти. Центральними прецедентами в його роботі є створення події, що обов'язково включає етап заповнення детальної форми заходу, а також подальше управління контентом через редагування або видалення власних публікацій. Окремим важливим блоком є модуль аналітики та монетизації: організатор може переглядати статистику продажів, що розширюється за рахунок

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						26
Змн.	Арк.	№ докум.	Підпис	Дата		

формування детального звіту по проданих квитках. Прецедент отримання прибутку також є комплексним і безпосередньо включає процес формування запиту на виплату зароблених коштів.

Для більш детального розуміння логіки роботи системи та підготовки до проєктування алгоритмів у середовищі Next.js [2], було розроблено специфікації для ключових прецедентів платформи. Специфікація деталізує кроки взаємодії актора з системою, визначаючи контракти між клієнтськими компонентами та серверними маршрутами.

Таблиця 2.2

Специфікація прецеденту “Реєстрація / Авторизація”

Атрибут	Опис
Назва прецеденту	Реєстрація / Авторизація
Актор	Користувач, Організатор
Мета	Отримати безпечний доступ до функціоналу купівлі квитків або управління подіями.
Передумови	Користувач знаходиться на сторінці входу, має стабільне інтернет-з’єднання.
Основний сценарій (Успішний)	Користувач заповнює форму авторизації (email, пароль). Клієнтський React-компонент валідує введені дані. Виконується запит до системи авторизації. Серверна частина перевіряє наявність користувача у базі даних та звіряє хеш пароля. Сервер генерує JWT-токен [29] і повертає успішний статус.

	Next.js маршрутизатор перенаправляє користувача до особистого кабінету або дашборду організатора.
Альтернативний сценарій (Помилки)	Невірний пароль або email: сервер повертає статус 401 Unauthorized, на клієнті виводиться повідомлення про помилку без уточнення конкретного поля. При реєстрації (Email вже існує): сервер повертає статус 409 Conflict.
Постумови	У системі створюється активна сесія. Користувач отримує доступ до захищених маршрутів (Private Routes).

У наведеній таблиці деталізовано специфікацію прецеденту «Реєстрація / Авторизація», який є фундаментальним для забезпечення безпеки та розмежування прав доступу в вебдодатку «Eventik». Головними акторами цього процесу виступають як звичайні користувачі, так і організатори подій, спільною метою яких є отримання санкціонованого доступу до захищених функціональних модулів платформи. Ключовою передумовою для успішної ініціації процесу є коректне завантаження сторінки автентифікації на пристрої клієнта та наявність стабільного мережевого з'єднання із сервером.

Одним із найважливіших модулів системи є процес придбання квитка з динамічною генерацією ідентифікатора. У таблиці 2.3 описано специфікацію цього прецеденту.

Специфікація прецеденту “Купити квиток та Отримати квиток”

Атрибут	Опис
Назва прецеденту	Купити квиток (включає: Оплатити квиток, Згенерувати QR-код)
Актор	Користувач
Мета	Успішно провести транзакцію та отримати унікальний цифровий квиток.
Передумови	Користувач авторизований, знаходиться на сторінці обраної події, є доступні вільні місця.
Основний сценарій (Успішний)	Користувач натискає кнопку щоб купити квиток. Клієнт викликає серверну дію (Server Action) для ініціалізації сесії оплати. Система інтегрується з платіжним шлюзом і користувач проводить оплату. Після успішного вебхука від платіжної системи, бекенд підтверджує транзакцію. Сервер генерує унікальний криптографічний токен для квитка і на його основі створює QR-код. Запис про квиток зберігається в реляційній базі даних із прив’язкою до ID користувача та ID події. Користувач перенаправляється на сторінку квитків, де рендериться його квиток із QR-кодом.
Альтернативний сценарій (Помилки)	Відхилення платежу банком: система скасовує резерв місця та виводить повідомлення про помилку транзакції. Втрата з’єднання під час генерації: якщо платіж

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						29
Змн.	Арк.	№ докум.	Підпис	Дата		

	пройшов, але зв'язок обірвався, квиток все одно генерується у фоновому режимі та з'явиться в профілі або буде надісланий на email.
Постумови	Кількість доступних місць на подію зменшується на 1. У базі даних формується запис про транзакцію та створений квиток.

У таблиці наведено детальну специфікацію найважливішого бізнес-процесу платформи прецеденту «Купити квиток», який комплексно охоплює етапи оплати та генерації унікального QR-коду. Головним актором у цьому процесі виступає авторизований користувач, метою якого є успішне проведення транзакції для отримання цифрового пропуску на захід. Обов'язковими передумовами для запуску цього алгоритму є наявність активної сесії користувача, його перебування на сторінці цільової події та фізична наявність вільних квитків у базі даних.

Окрім клієнтського функціоналу, стабільність платформи підтримується адміністративними процесами. Специфікація прецеденту модерації (табл. 2.4) демонструє механізми контролю якості контенту.

Таблиця 2.4

Специфікація прецеденту “Заблокувати подію”

Атрибут	Опис
Назва прецеденту	Заблокувати подію (розширюється: Додати причину блокування)
Актор	Модератор, Адміністратор
Мета	Приховати подію з публічного каталогу через порушення правил платформи.
Передумови	Модератор авторизований, розглядає скаргу або перевіряє нову подію в панелі управління.

Основний сценарій (Успішний)	Модератор ініціює дію “Заблокувати подію”. Система відкриває модальне вікно та вимагає обов’язкового заповнення поля “Причина блокування”. Модератор вводить причину та підтверджує дію. Next.js Server Action оновлює статус події в базі даних (з active на blocked).
Основний сценарій (Успішний)	Сервер автоматично формує та відправляє email організатору із зазначенням причини блокування. Інтерфейс миттєво оновлює таблицю подій без перезавантаження сторінки.
Альтернативний сценарій (Помилки)	Спроба відправити форму без вказання причини: клієнтська валідація блокує запит і підсвічує поле червоним.
Постумови	Подія більше не доступна для публічного перегляду та купівлі квитків. Гроші за вже куплені квитки (якщо такі є) можуть бути заморожені до рішення адміністрації.

У наведеній таблиці представлено детальну специфікацію прецеденту «Заблокувати подію», який є ключовим інструментом для забезпечення модерації контенту та дотримання правил на розробленій платформі. Акторами у цьому процесі виступають адміністратор або уповноважений модератор, головною метою яких є оперативне приховування заходу з публічного каталогу у випадку виявлення порушень. Обов'язковою передумовою для виконання цієї дії є наявність відповідних прав доступу в авторизованого працівника, який розглядає скаргу або проводить планову перевірку нової публікації в захищеній панелі управління.

Розроблені специфікації прецедентів визначають архітектурні контракти системи. Вони демонструють, що кожна дія актора супроводжується серверною логікою: управлінням транзакціями, перевіркою прав доступу та синхронізацією бази даних. Ця деталізація є вхідними даними для наступного етапу – проєктування об’єктно-орієнтованої структури класів та API платформи.

2.2 Проєктування логічної структури бази даних

Для забезпечення надійного зберігання інформації та підтримки транзакційної цілісності даних, що є критично важливим аспектом під час оформлення фінансових операцій та швидкого продажу квитків, як основну систему управління базами даних було обрано реляційну СУБД PostgreSQL [15] (хмарне безсерверне рішення Neon [3]). Ця система гарантує суворе виконання вимог ACID (атомарність, узгодженість, ізольованість, довговічність), що унеможливорює втрату або пошкодження даних під час пікових навантажень. Для ефективної взаємодії серверної частини на базі Next.js [2] із базою даних інтегровано сучасний інструмент об’єктно-реляційного відображення Drizzle ORM [19]. Завдяки глибокій інтеграції з TS [14], він забезпечує строгу типізацію коду на етапі компіляції та гарантує високу продуктивність за рахунок генерації максимально оптимізованих SQL-запитів. Додатково архітектура бази даних передбачає використання спеціалізованого типу даних JSONB для збереження специфічної інформації, як-от масиви тегів або нестандартні параметри подій. Це рішення дозволяє гармонійно поєднати надійність класичної реляційної моделі з гнучкістю NoSQL-підходів у тих сегментах, де структура даних може динамічно змінюватися.

Логічне проєктування бази даних платформи передбачає створення комплексної структури взаємопов’язаних таблиць, які повністю покривають бізнес-логіку управління користувачами, автентифікації, публікації подій, формування розкладів, збереження медіафайлів та функціонування системи відгуків. Ключові зв’язки між сутностями системи побудовані з урахуванням

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						32
Змн.	Арк.	№ докум.	Підпис	Дата		

суворих правил реляційної цілісності. Зокрема, між таблицею користувачів (users) та сутностями відгуків (reviews), системних сесій (sessions) і пов'язаних акаунтів (accounts) встановлено зв'язок типу «один-до-багатьох» (1:N) із налаштованим каскадним видаленням, що забезпечує автоматичне очищення всіх пов'язаних даних у разі видалення профілю. Аналогічний зв'язок «один-до-багатьох» реалізовано між організаторами (organizers) та заходами (events), що дозволяє одному організатору публікувати та керувати необмеженою кількістю подій. Сама ж сутність події (events) виступає центральним вузлом бази даних і має прямі зв'язки формату «один-до-багатьох» із таблицями галереї (event_galleries) для збереження ілюстративних матеріалів та розкладу (event_schedules) для детального структурування програми заходу.

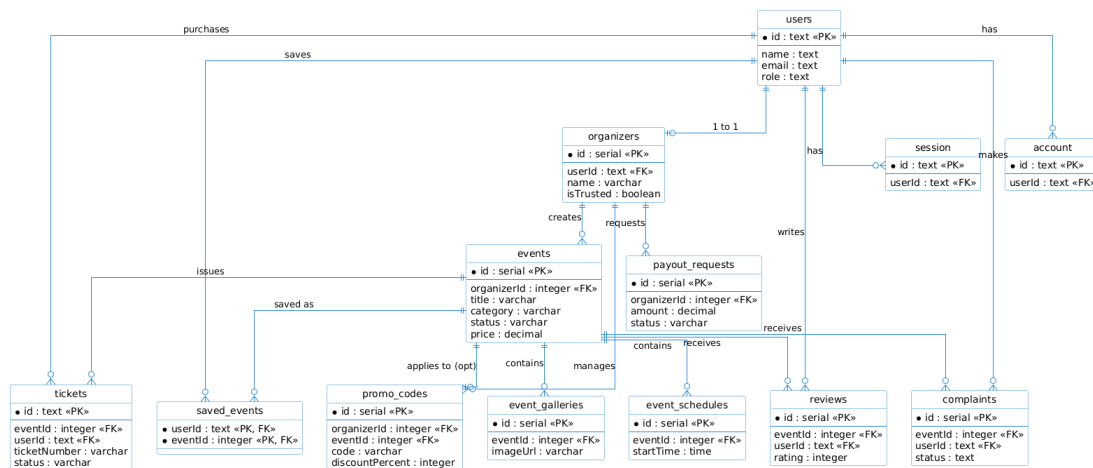


Рис. 2.4 ER-діаграма реляційної бази даних

Фізична структура БД Для реалізації логічної моделі за допомогою Drizzle ORM [19] було створено фізичні таблиці. У таблицях 2.5 – 2.9 наведено специфікації найважливіших сутностей системи.

Для управління доступом та збереження профілів розроблено таблицю users. Її структуру наведено в таблиці 2.5.

Структура таблиці users

Поле	Призначення
id	Унікальний ідентифікатор користувача в системі (формат UUID).
firstName, lastName	Ім'я та прізвище особи.
name	Повне ім'я (обов'язкове поле для базового відображення в інтерфейсі).
email	Унікальна електронна адреса, що використовується для авторизації та зв'язку.
emailVerified	Булеве значення, що підтверджує валідність вказаної електронної пошти.
image, avatarUrl	Посилання на завантажену фотографію профілю.
	Роль у системі, що розмежовує права доступу (за замовчуванням звичайний користувач).
banned, banReason, banExpires	Комплекс полів для керування доступом: статус блокування, його причина та термін дії.
createdAt, updatedAt	Системні мітки часу створення та останнього оновлення запису.

Це базова сутність системи, яка зберігає облікові дані всіх зареєстрованих осіб (як звичайних відвідувачів, так і майбутніх організаторів). Вона відповідає за глобальну ідентифікацію, авторизацію та зберігання базової інформації профілю, до якої прив'язуються всі інші дії користувача.

Структура таблиці organizers

Поле	Призначення
id	Унікальний внутрішній ідентифікатор профілю організатора.
userId	Зовнішній ключ (посилання) на базовий профіль у таблиці users.
name	Офіційна назва компанії, бренду або ім'я організатора.
description	Текстовий опис діяльності або специфіки організатора.

logoUrl, website, phone, address	Набір контактних даних та логотип для формування публічної сторінки організатора.
instagram, facebook	Посилання на офіційні сторінки в соціальних мережах.
isTrusted	Системна відмітка верифікованого (надійного) організатора від адміністрації.

Таблиця логічно розширює базовий профіль користувача, додаючи специфічні бізнес-дані. Вона необхідна для того, щоб відокремити звичайних покупців від організаторів, які отримують розширені права на створення подій, управління продажами, генерацію промокодів та отримання прибутків.

Таблиця 2.7

Структура таблиці events

Поле	Призначення
id	Унікальний ідентифікатор заходу.
Title	Публічна назва події, що виводиться в каталозі.
Description	Детальний текстовий опис формату та особливостей заходу.
mainImageUrl	Посилання на головний постер (зображення) події.
startDate, endDate	Точні системні дата та час початку і завершення заходу.
durationHours	Розрахункова тривалість заходу в годинах.
Category	Жанр або категорія для роботи системи фільтрації.
Status	Поточний стан події (наприклад: відкрита, заблокована).
Price, currency	Базова вартість квитка та прив'язана валюта.
totalSeats, soldSeats	Загальна місткість локації та лічильник вже успішно проданих місць.
Address, mapUrl	Фізична адреса проведення заходу та посилання на географічну мапу.
organizerId	Посилання на профіль організатора, який створив публікацію.
Tags, perks	Додаткові характеристики та переваги події (зберігаються у гнучкому форматі JSONB).

Це центральна таблиця всієї архітектури додатка. Вона агрегує вичерпну інформацію про конкретний захід і виступає головним зв'язуючим вузлом для квитків, галерей, розкладів та відгуків.

Таблиця 2.8

Структура таблиці tickets

Поле	Призначення
id	Унікальний ідентифікатор квитка (формат UUID) для безпеки.
eventId	Посилання на подію, для якої придбано цей квиток.
userId	Посилання на профіль покупця (може бути порожнім, якщо реалізована гостьова покупка).
ticketNumber	Зрозумілий для людини унікальний номер (наприклад, EVT-000123) для візуального контролю.
Status	Стан транзакції та квитка (очікує оплати, оплачено, використано, скасовано).
quantity	Кількість квитків у рамках одного сформованого замовлення.
unitPrice, totalPrice	Зафіксована вартість одного квитка та загальна сума замовлення на момент транзакції.
buyerFirstName та ін.	Набір полів для збереження контактних даних безпосереднього покупця.
paymentMethod, paymentId	Інформація про обраний платіжний шлюз та ідентифікатор банківської транзакції.
qrToken	Зашифрований секретний токен для генерації QR-коду та його безпечної валідації на вході.
promoCode	Інформація про застосований під час покупки промокод.

Таблиця фіксує всі фінансові операції бронювання. Вона відповідає за облік проданих місць, збереження історії покупок та містить критично важливий токен (qrToken), який забезпечує створення унікального електронного пропуску для користувача.

Структура таблиці promo_codes

Поле	Призначення
id	Унікальний ідентифікатор створеного промокоду.
organizerId	Посилання на організатора-власника, який згенерував код.
eventId	Посилання на конкретну подію (якщо код діє лише на один захід).
code	Текстове значення коду, яке безпосередньо вводить покупець (наприклад, EARLYBIRD).
discountPercent	Розмір знижки у відсотковому еквіваленті.
maxUses, usedCount	Встановлений ліміт можливих використань та лічильник фактичних застосувань коду.
isActive	Прапорець, що вмикає або примусово вимикає дію промокоду.
validFrom, validUntil	Часові рамки, в межах яких система дозволяє застосувати знижку.

Це модуль для реалізації маркетингових кампаній. Таблиця дозволяє організаторам гнучко налаштовувати системи знижок на свої події з чітким контролем лімітів за часом дії та кількістю активацій покупцями.

Таблиці соціальної взаємодії та модерації (reviews, complaints, saved_events) Кластер таблиць соціальної взаємодії та модерації включає сутності відгуків, скарг та збережених подій. Базовими атрибутами для цих таблиць є універсальні ідентифікатори eventId та userId, які забезпечують чітку реляційну прив'язку конкретної дії користувача до відповідного заходу. Для системи відгуків використовуються спеціальні поля rating та comment, що зберігають числову оцінку та текстове враження відвідувача, дозволяючи платформі автоматично формувати публічний рейтинг кожної події. Таблиця скарг оперує полями reason та status, які фіксують детальний опис виявленого порушення та поточний етап розгляду цього звернення модераторами. Загалом цей кластер є критично важливим для розширення користувацького досвіду та підтримки якості контенту: він надає клієнтам можливість формувати власні

списки збережених заходів (закладки), ділитися об'єктивними враженнями після подій, а також допомагати адміністрації своєчасно виявляти та блокувати публікації, що порушують правила платформи.

Таблиця фінансових запитів (payout_requests) Таблиця запитів на виплату відіграє роль основи внутрішнього фінансового документообігу розробленого додатка. Її головне призначення надійна фіксація та обробка запитів від організаторів на виведення коштів, акумульованих у результаті успішного продажу електронних квитків. Кожен такий запит ідентифікується унікальним системним ключем id та містить обов'язкову прив'язку до ініціатора через зовнішній ключ organizerId. Фінансова складова операції описується полем amount, що фіксує точну суму запитаних коштів, та текстовим атрибутом bankDetails, де безпечно зберігаються банківські реквізити отримувача для здійснення транзакції. Контроль за виконанням зобов'язань забезпечується через поле status, яке динамічно відображає поточний стан запиту (очікує на перевірку, схвалено до виплати або відхилено адміністрацією). Впровадження цієї сутності дозволяє системі прозоро керувати взаєморозрахунками з партнерами та вести точний аудит руху коштів.

Таблиці системи автентифікації (session, account, verification) Група системних таблиць авторизації забезпечує інтеграцію сучасних стандартів безпеки та керування сесіями користувачів. Усі складові цього кластера нерозривно пов'язані з основним профілем через ідентифікатор userId. Сутності сесій та пов'язаних акаунтів оперують криптографічними даними, зокрема полями token та accessToken, які роблять можливим безпечний доступ до системи, включаючи авторизацію через сторонніх OAuth-провайдерів [30] (ідентифікатор яких зберігається в атрибуті providerId). Для захисту від несанкціонованого доступу та перехоплення сесій використовується поле expiresAt, яке встановлює суворий часовий ліміт життєдіяльності кожного токена. Ці таблиці є фундаментальною частиною інфраструктури безпеки платформи, оскільки вони відповідають за підтримку активних підключень,

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						38
Змн.	Арк.	№ докум.	Підпис	Дата		

обробку інтеграцій із зовнішніми сервісами, а також за захищене керування процесами верифікації електронної пошти та відновлення втрачених паролів.

2.2 Проєктування компонентної структури та взаємодії модулів

Діаграма розгортання та інфраструктура Оскільки система проєктується як сучасне вебрішення з високими вимогами до швидкодії та SEO-оптимізації, її архітектура спирається на можливості хмарних платформ та Serverless-технологій. На рис. 2.3.1 представлена розроблена діаграма розгортання програмного комплексу.

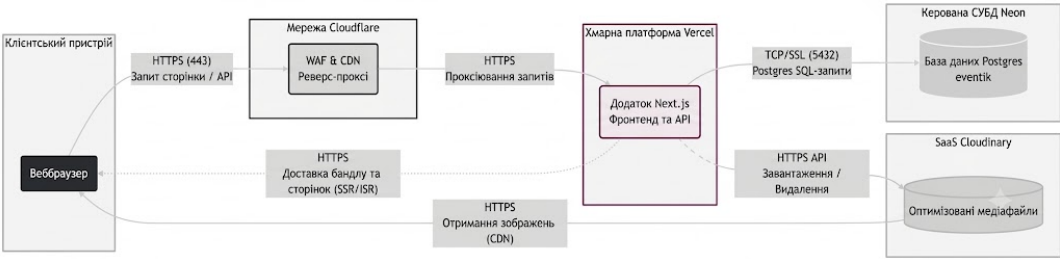


Рис. 2.5 Діаграма розгортання системи

Архітектура платформи базується на ефективній взаємодії кількох ключових інфраструктурних вузлів. Початковою точкою взаємодії є клієнтський пристрій, а саме веббраузер користувача, який приймає згенерований сервером HTML/CSS-код та виконує процес гідратації React-компонентів [12] для забезпечення інтерактивності інтерфейсу. Замість традиційних виділених серверів, розгортання та виконання логіки додатка здійснюється на базі глобальної мережі Cloudflare із використанням адаптера OpenNext. Цей вузол виконує роль як реверс-проксі сервера для забезпечення захисту від кібератак WAF, управління DNS та кешування статичних ресурсів CDN, так і основного крайового Edge [20] середовища виконання фреймворку Next.js [2]. Саме тут відбувається генерація статичних сторінок SSG/ISR, рендеринг динамічних маршрутів (SSR), а також обробка API-запитів та серверних дій (Server Actions) [21].

Збереження та обробка транзакційних даних здійснюється за допомогою керованої СУБД Neon [3] хмарної безсерверної бази даних PostgreSQL, до якої

додаток підключається через безпечне TCP/SSL-з'єднання для виконання оптимізованих SQL-запитів за допомогою інструменту Drizzle ORM [19]. Для управління медіаконтентом інтегровано спеціалізоване SaaS-рішення Cloudinary, яке виконує функції надійного сховища, автоматичної оптимізації та швидкої доставки графічних файлів, зокрема банерів подій та аватарів користувачів.

Замість класичного розділення на відокремлені бекенд та фронтенд, розроблена платформа використовує повностекові можливості екосистеми Next.js [2]. Архітектуру модулів та їхню взаємодію наведено на рис. 2.5.

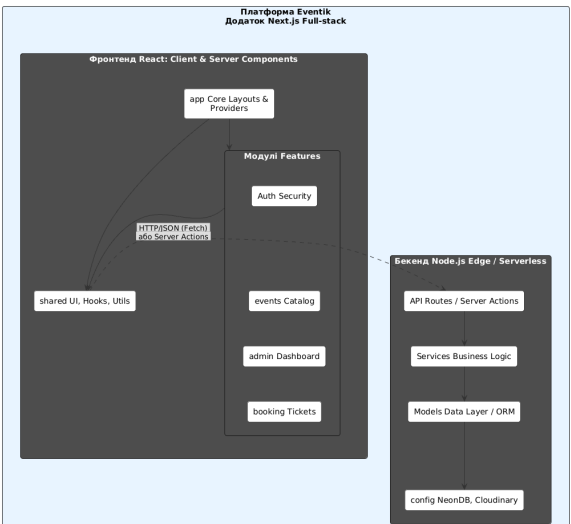


Рис. 2.6 Структура Next.js Full-stack додатку

Розроблена система концептуально поділена на два основні логічні рівні. Рівень клієнтської частини (фронтенду), побудований на базі клієнтських та серверних компонентів React [12], структурований за предметними областями (фічами) відповідно до обраної архітектурної методології. Він включає незалежні модулі безпеки та автентифікації, відображення каталогу подій, системи швидкого бронювання квитків та адміністративної панелі керування. При цьому всі універсальні елементи користувацького інтерфейсу та спільні функції (хуки) винесені в окремий логічний блок спільного використання (shared).

Рівень серверної частини (бекенду), що функціонує в середовищі Node.js

[13] або на крайових Edge [20] обчислювальних вузлах, забезпечує безпосередню взаємодію з клієнтом переважно через механізм серверних дій (Server Actions). Водночас для обробки асинхронних подій від сторонніх сервісів, зокрема для прийому вебхуків від платіжних систем, застосовуються класичні маршрути програмного інтерфейсу (API Routes).

Уся бізнес-логіка платформи інкапсульована у відповідних сервісах, які безпосередньо звертаються до рівня абстракції даних (Data Layer). На цьому рівні інструмент об'єктно-реляційного відображення Drizzle ORM [19] формує безпечні та оптимізовані запити до хмарної бази даних Neon [3].

Для детального розуміння механізмів обміну даними між клієнтом, сервером та базою даних під час виконання ключових бізнес-процесів платформи було розроблено відповідні діаграми послідовності. Зокрема, на рисунку 2.3.3 детально зображено логічний процес ініціації, валідації та збереження нової події в системі з боку авторизованого організатора.

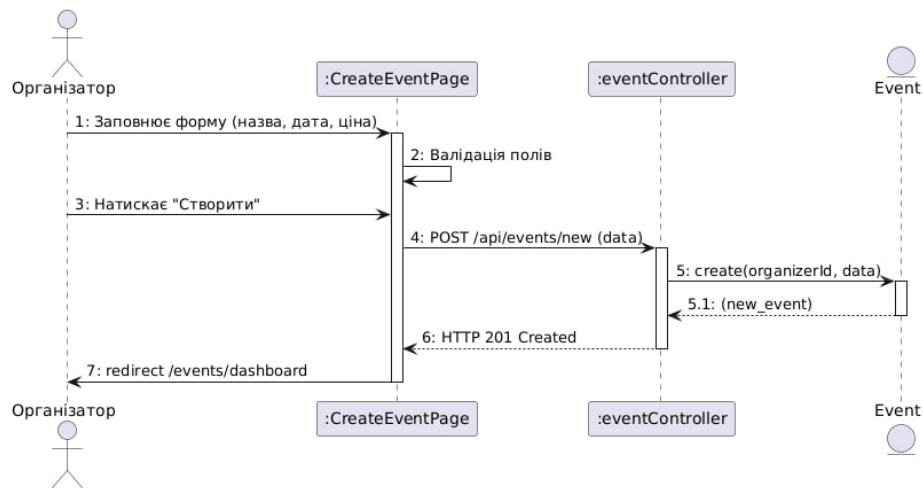


Рис. 2.7 Діаграма послідовності |”Створення події”

Процес розпочинається із заповнення організатором форми створення заходу на спеціалізованій клієнтській сторінці інтерфейсу. Після успішної локальної валідації введених даних клієнтська частина ініціює запит (через POST-метод або механізм Server Action) до відповідного маршруту програмного інтерфейсу. На серверному рівні архітектури контролер здійснює обов'язкову перевірку прав доступу ініціатора запиту, після чого формує та

зберігає новий об’єкт події у базі даних, суворо прив’язуючи його до ідентифікатора поточного організатора. У разі успішного виконання транзакції та підтвердження збереження даних, маршрутизатор системи автоматично перенаправляє користувача до панелі управління (дашборду) подіями.

Для детальнішого розкриття логіки взаємодії клієнта з платформою під час фінансових операцій, на рисунку 2.8 проілюстровано наступний критичний бізнес-процес алгоритм оформлення та купівлі електронного квитка користувачем.

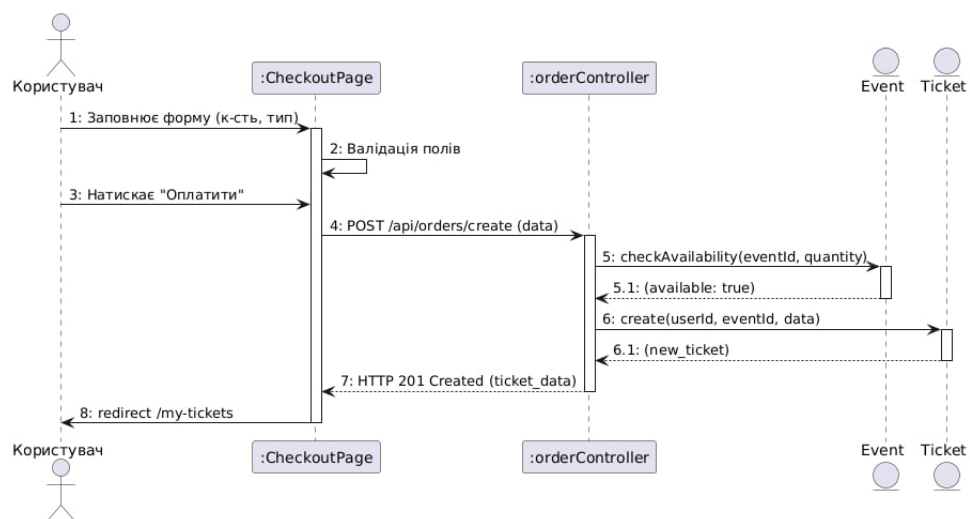


Рис. 2.8 Діаграма послідовності “Купівля квитка”

Алгоритм ініціюється в момент, коли користувач визначає необхідну кількість квитків та активує процес оплати на клієнтській сторінці оформлення замовлення. Відповідний запит передається до серверного контролера обробки замовлень, першочерговим завданням якого є перевірка наявності достатньої кількості вільних місць у базі даних для цільової події. За умови успішної валідації квот, система створює новий транзакційний запис, формуючи сутність квитка із чіткою реляційною прив'язкою до ідентифікаторів поточного покупця та обраного заходу. Завершується сценарій успішною видачею згенерованого цифрового документа та автоматичним перенаправленням клієнта до розділу особистого кабінету з

історією покупок.

Як логічне продовження аналізу системних процесів, на рисунку 2.9 проілюстровано алгоритм скасування події організатором. Цей прецедент є архітектурно складнішим, оскільки його виконання вимагає обов'язкової перевірки фінансових умов, статусу вже здійснених транзакцій та ініціації процедури повернення коштів покупцям.

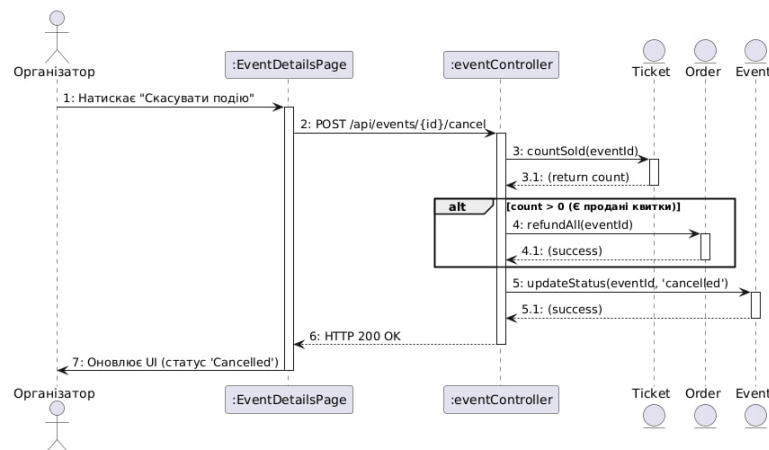


Рис. 2.9 Діаграма послідовності “Скасування події”

Процес скасування ініціюється відповідною дією на сторінці деталей заходу (EventDetailsPage), після чого серверний контролер першочергово виконує перевірку кількості вже проданих квитків (за допомогою методу countSold). На цьому етапі алгоритму відбувається логічне умовне розгалуження, що відповідає блоку alt на діаграмі послідовності: у випадку, якщо транзакції з продажу вже були успішно здійснені, система автоматично ініціює інтеграційний процес масового повернення коштів покупцям (refundAll). Лише після завершення або обходу цих фінансових процедур статус події у базі даних безпечно оновлюється на скасований (cancelled), а реактивний клієнтський інтерфейс миттєво відображає відповідні зміни.

Загалом, застосування описаної архітектури на базі екосистеми Next.js [2] у поєднанні з інструментом Drizzle ORM [19] дозволяє забезпечити високу швидкість розробки, наскрізну сувору типізацію даних на всіх рівнях програмного забезпечення (від схеми бази даних до клієнтського інтерфейсу)

та гарантує стабільну і безперебійну роботу системи навіть в умовах високих навантажень.

Для глибшого розуміння структурних зв'язків між об'єктами під час складних транзакцій використовується діаграма кооперації. Вона відображає ті самі повідомлення, що й діаграма послідовності, але акцентує увагу не на часі, а на просторовій архітектурі та топології зв'язків між компонентами.

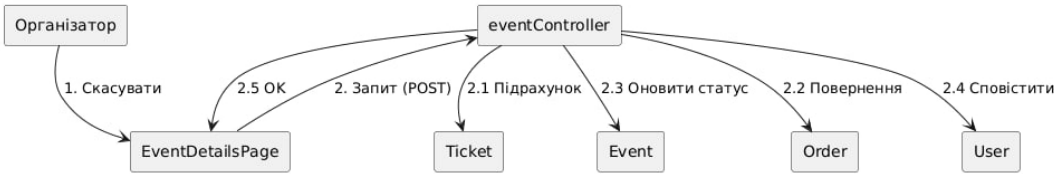


Рис. 2.10 Діаграма кооперації “Скасування події”

Діаграма демонструє радіальну топологію управління, де `eventController` (який у контексті фреймворку Next.js [2] реалізується як серверна дія або API-маршрут) виступає центральним вузлом. Він надійно ізолює клієнтський інтерфейс від безпосередньої взаємодії з базою даних. При ініціації скасування події організатором, контролер бере на себе відповідальність за послідовну координацію відразу чотирьох сутностей. Спочатку він звертається до моделі `Ticket` для підрахунку вже реалізованих квитків, після чого взаємодіє з сутністю `Order` для ініціації процедури масового повернення коштів покупцям. За умови успішного виконання цих операцій, контролер оновлює статус у самій сутності `Event` (змінюючи його на скасований) та викликає сервіс сповіщень для моделі `User`, щоб автоматично поінформувати відвідувачів про скасування заходу. Такий підхід забезпечує виконання принципу єдиної відповідальності (Single Responsibility) та підтримує сувору транзакційну цілісність: якщо помилка виникне на етапі повернення фінансів, статус події залишиться незмінним, що фундаментально гарантує надійність фінансових операцій розробленої платформи.

2.3 Розробка основних алгоритмів роботи платформи

Для детального розуміння внутрішньої логіки платформи Eventik розроблено алгоритми роботи ключових бізнес-процесів. Візуалізація

алгоритмів здійснена за допомогою діаграм активності (Activity Diagrams). Ці діаграми відображають динаміку поведінки системи, показуючи паралельні процеси, розгалуження та чіткий розподіл зон відповідальності між різними акторами та підсистемами за допомогою доріжок (Swimlanes).

Алгоритм “Життєвий цикл події: від реєстрації до публікації” Цей алгоритм описує глобальний процес взаємодії різних ролей у системі, починаючи від входу на платформу і закінчуючи схваленням заходу модератором.

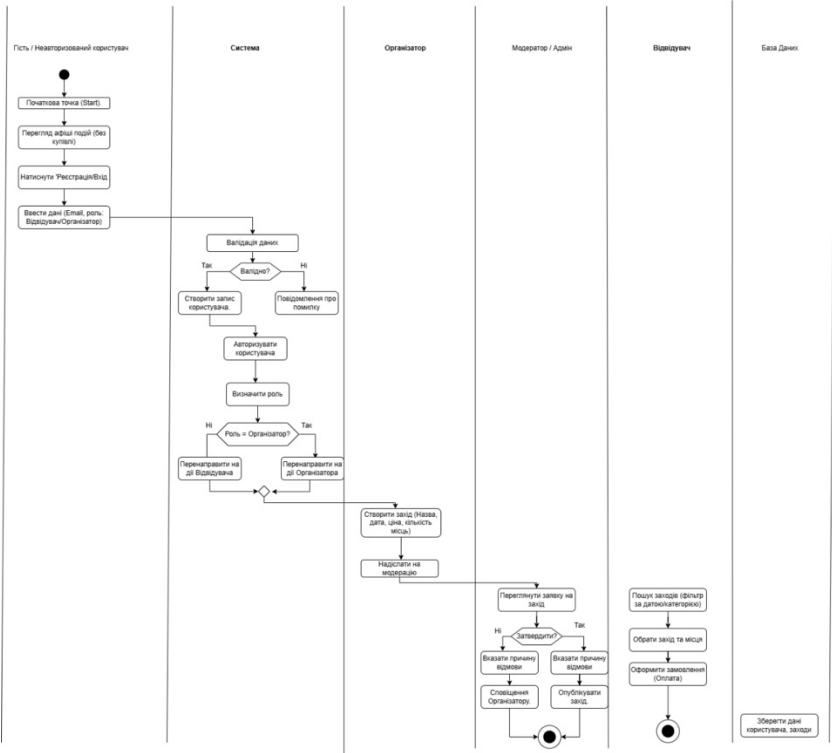


Рис. 2.11 Діаграма активності “Життєвий цикл події”

Покроковий опис базового алгоритму роботи системи розпочинається з етапу ініціалізації. Неавторизований гість має змогу вільно переглядати загальну афішу подій, проте для отримання можливості створення заходу або ініціації покупки квитка він повинен пройти процес реєстрації чи авторизації шляхом введення облікових даних (електронної пошти та пароля). Далі відбувається етап валідації та маршрутизації, під час якого система перевіряє коректність введеної інформації. У разі виявлення невідповідностей інтерфейс виводить повідомлення про помилку. Якщо ж дані коректні, серверна частина створює захищену сесію, визначає системну роль актора (організатор чи

відвідувач) та автоматично перенаправляє його у відповідний робочий простір (дашборд).

Процес створення події ініціюється організатором, який заповнює електронну форму новими даними, зазначаючи назву, дату проведення, ціну та загальну місткість локації (кількість квитків). Після відправлення форми створена сутність автоматично набуває статусу очікування перевірки. На етапі модерації уповноважений адміністратор розглядає заявку, після чого алгоритм передбачає умовне розгалуження. У випадку відхилення публікації модератор обов'язково вказує причину свого рішення, захід залишається прихованим, а організатор отримує відповідне системне сповіщення. Якщо ж заявку схвалено, подія миттєво публікується у відкритому каталозі. З цього моменту розпочинається етап взаємодії з відвідувачем: користувач може швидко знайти опублікований захід через інструменти фільтрації (за датою чи категорією) та перейти безпосередньо до оформлення замовлення.

Окремої уваги вимагає алгоритм процесу бронювання та купівлі квитка. Цей процес є найкритичнішим з точки зору збереження транзакційної цілісності бази даних, оскільки він описує сувору логіку придбання електронного пропуску. Його фундаментальним завданням є забезпечення обов'язкової конкурентної перевірки доступних квитків у реальному часі, що дозволяє системі повністю уникнути проблеми «овербукінгу» критичної помилки продажу більшої кількості місць, ніж фізично закладено організатором.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		46

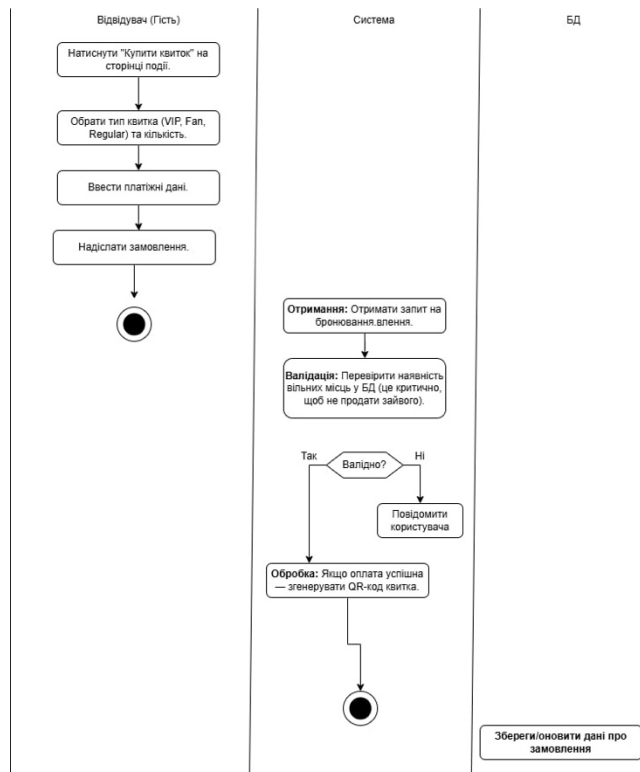


Рис. 2.12 Діаграма активності “Купівля квитка”

Покроковий опис алгоритму купівлі розпочинається з етапу вибору квитків, під час якого відвідувач на сторінці події ініціює транзакцію, обирає бажаний тип пропуску (наприклад, VIP, Fan або Regular) та вказує необхідну їх кількість. Далі слідує етап оформлення, де користувач вводить платіжні реквізити та відправляє запит на формування замовлення. У цей момент відбувається етап критичної перевірки (валідації): система отримує запит і негайно звертається до бази даних для перевірки фактичної наявності вільних місць шляхом порівняння значень полів загальної кількості місць (`total_seats`) та вже проданих квитків (`sold_seats`) у таблиці подій (`events`).

На основі отриманих даних алгоритм виконує суворе умовне розгалуження. Якщо виявляється, що вільних місць недостатньо для задоволення запиту, транзакція негайно скасовується, а користувачеві виводиться відповідне повідомлення про відсутність потрібної кількості квитків. У випадку ж успішної валідації система безпечно переходить до етапу безпосередньої обробки платежу. Після отримання підтвердження про успішну оплату алгоритм ініціює генерацію та збереження даних: система

створює унікальний ідентифікатор і генерує QR-код електронного квитка. Завершальним кроком є синхронізація з базою даних, що включає створення нового запису в таблиці квитків (tickets) та автоматичний інкремент лічильника проданих місць (sold_seats) у сутності заходу.

Розроблені алгоритми наочно демонструють, що ключова бізнес-логіка платформи цілком покладається на надійну серверну валідацію. Делегування відповідальності за перевірку квот та обробку транзакцій виключно на рівень серверних контролерів (у середовищі Next.js Edge [20] або Node.js [13]) унеможлиблює будь-які спроби маніпуляцій з боку клієнтського браузера. Такий архітектурний підхід гарантує стабільну, безпечну та безперебійну роботу системи продажу квитків навіть в умовах різкого зростання навантаження.

Висновки до другого розділу

У другому розділі було виконано комплексне проєктування програмної архітектури платформи для управління подіями та продажу квитків, яке охоплює всі етапи: від розробки високорівневої моделі інфраструктури до деталізації алгоритмів взаємодії компонентів.

Побудовано діаграму розгортання, яка ілюструє сучасний підхід до побудови системи з використанням Full-stack фреймворку Next.js [2] та Serverless-інфраструктури. Такий розподіл між глобальною CDN (Cloudflare), хмарним середовищем виконання Vercel [1] та безсерверною керованою СУБД Neon [3] гарантує високу відмовостійкість, глобальну доступність та безпечне масштабування під час пікових навантажень (наприклад, на старті продажів квитків).

Проведено детальне моделювання варіантів використання системи, де визначено чотирьох незалежних акторів (Користувач, Організатор, Модератор, Адміністратор) та формалізовано їхні права доступу. Для ключових прецедентів, таких як авторизація, управління подіями та купівля квитків, розроблено розширені специфікації, що слугують архітектурними

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						48
Змн.	Арк.	№ докум.	Підпис	Дата		

контрактами між клієнтом та сервером.

Розроблено компонентну структуру системи. Побудовані діаграми ілюструють архітектуру додатку на базі Next.js App Router, яка ефективно розділяє клієнтські та серверні React-компоненти, переносючи важку бізнес-логіку та взаємодію з БД на ізольований серверний шар (Server Actions / API Routes).

Спроектовано логічну та фізичну структуру бази даних PostgreSQL, взаємодія з якою реалізована через Drizzle ORM [19]. Реляційна модель нормалізована та оптимізована для забезпечення транзакційної цілісності фінансових операцій, а використання типу даних JSONB дозволило зберегти гнучкість для зберігання динамічних атрибутів подій (тегів, переваг тощо).

Описано динамічну поведінку системи за допомогою діаграм активності, що деталізують бізнес-логіку ключових процесів (життєвий цикл події та безпечне бронювання квитків). Крім того, побудовано діаграми послідовності та кооперації, які візуалізують хронологію та топологію обміну повідомленнями між інтерфейсом, контролерами та шаром даних під час транзакцій та модерації. Розроблені логічні та структурні моделі повністю задовольняють вимоги технічного завдання і є вичерпною документацією для переходу до етапу безпосередньої програмної реалізації платформи, описаної у наступному розділі.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		49

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБДОДАТКОМ ДЛЯ ОРГАНІЗАЦІЇ МІСЬКИХ ЗАХОДІВ

3.1 Порядок встановлення та налаштування параметрів системи

Система розгортається у хмарному середовищі Vercel [1], а процес від коміту до появи оновленої версії у виробництві є повністю автоматизованим. Розробник не виконує жодних ручних дій для публікації, достатньо внести зміни до основної гілки репозиторію. Решту бере на себе вбудований конвеєр Vercel CI/CD. Система використовує інтегровані хмарні служби (табл. 3.1), де кожна відповідає за окремий аспект роботи і масштабується автоматично залежно від навантаження.

Таблиця 3.1

Перелік використаних хмарних служб та платформ

Служба	Платформа	Призначення
Vercel Edge Network	Vercel	Глобальний хостинг та кешування статичної клієнтської частини
Vercel Serverless Functions [21]	Vercel	Хостинг серверної частини (API) та виконання бізнес-логіки
Vercel Postgres	Vercel	Реляційна база даних
Vercel Blob	Vercel	Сховище бінарних файлів (фотодокази, креслення, медіа)
GitHub / GitLab	GitHub Inc. / GitLab Inc.	Репозиторій вихідного коду та тригер конвеєра розгортання

Клієнтський застосунок функціонує в умовах суворої ізоляції, доставляючись кінцевому користувачеві безпосередньо з найближчого вузла CDN. Будь-які прямі транзакції з браузера до бази даних чи файлового сховища фізично заблоковані на рівні архітектури. Єдиною дозволеною точкою взаємодії виступають Serverless-функції [21] (API-маршрути). Саме цей рівень бере на себе всю оркестрацію: він керує структурованими записами у Vercel Postgres [15] та маніпулює бінарними об'єктами через Vercel Blob. На відміну від традиційної контейнеризації, маршрут доставки коду не потребує проміжних реєстрів на кшталт Docker Hub. Vercel [1] самостійно аналізує код, компілює його та розподіляє: статичні артефакти відправляються на Edge Network [20], а бекенд-логіка автоматично пакується в ізольовані Serverless-функції. Конвеєр безперервної доставки реалізовано засобами внутрішньої інтеграції Vercel з Git-провайдером. Механізм розбито на лінійні етапи із жорсткою залежністю. Логіка пайплайну не допускає компромісів: якщо команда збірки (Build Command) завершується з помилкою, наприклад, через провал перевірки типів або лінера [27], розгортання автоматично скасовується, а попередня робоча версія залишається активною. Додатково платформно налаштовано Preview-середовища. Усі робочі гілки (Pull Requests) проходять фазу ізольованого розгортання за унікальними тимчасовими посиланнями, без жодного впливу на виробниче середовище (Production).

Таблиця 3.2

Етапи безперервної інтеграції та розгортання

Стадія	Завдання	Зміст
Install	Встановлення залежностей	Завантаження пакетів через npm/yarn/pnpm на серверах збірки Vercel
Lint & Test	Перевірка якості	Запуск скриптів lint та test

Build	Компіляція	Транспіляція коду, генерація статичних сторінок та підготовка Serverless-функцій
Migrations	Оновлення БД	Застосування міграцій до Vercel Postgres під час збірки або через окремий скрипт
Deploy	Розподіл артефактів	Публікація функцій та статичних файлів на глобальну мережу маршрутизації

Чутливі параметри конфігурації рядок підключення до бази, ключ підпису токенів не зберігаються у вихідному коді. Вони налаштовуються у панелі керування Vercel [1] як Environment Variables і автоматично шифруються платформою. Завдяки розділенню змінних за типами середовищ (Production, Preview, Development), перемикання між локальною та хмарною базою даних відбувається прозоро для застосунку і не вимагає змін у коді. Взаємодія з бекендом маршрутизується через єдиний локальний шлюз. Таке архітектурне рішення перетворює робочу станцію розробника на точну структурну копію бойового сервера без необхідності підтримувати важкі віртуальні машини або складні конфігурації docker-compose для самого застосунку.

3.2 Структура інтерфейсу вебсистеми для міських заходів

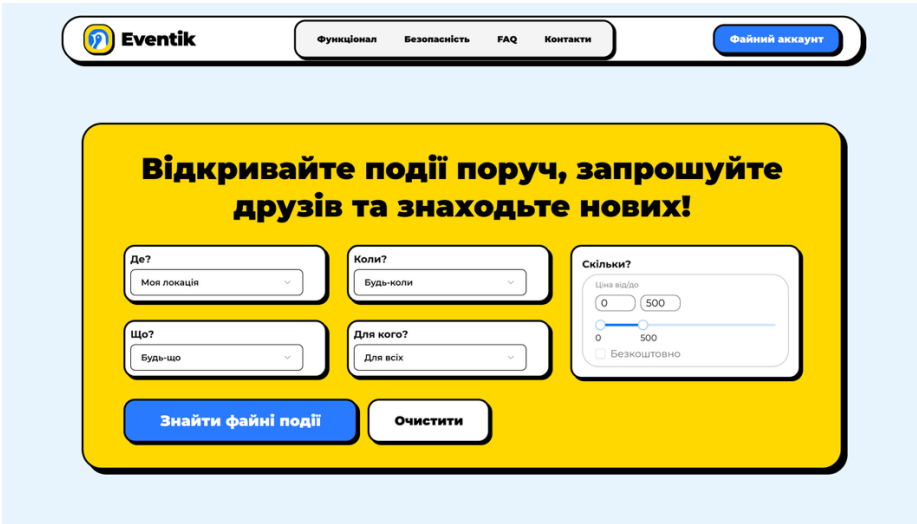


Рис. 3.1 Головна сторінка з фільтром пошуку подій

Головна сторінка вебдодатка (рис. 3.1) слугує інтерактивною точкою входу та містить розширений блок фільтрації для швидкого пошуку заходів. Інтерфейс дозволяє детально налаштувати критерії запиту за локацією, часом, категорією та ціновим діапазоном, після чого система формує релевантну видачу подій.

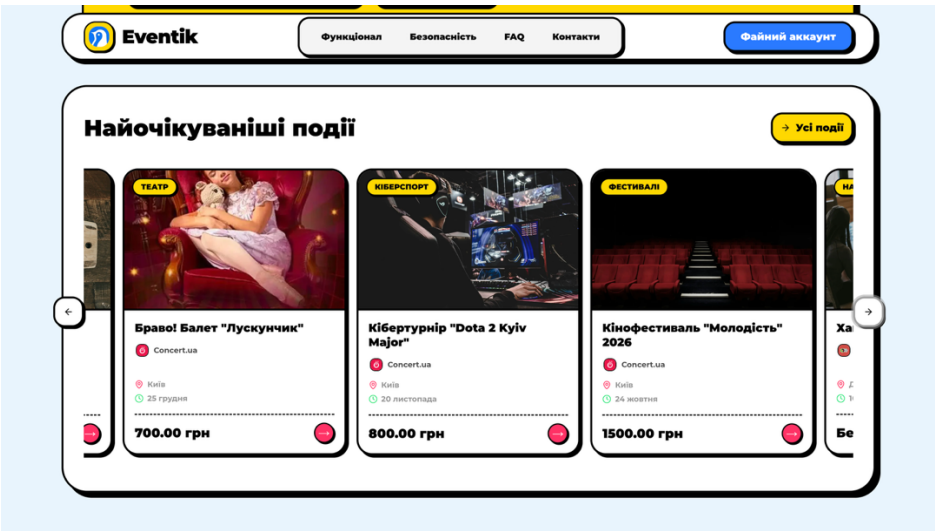


Рис. 3.2 Карусель найочікуваніших подій на головній сторінці

Нижче на головній сторінці (рис. 3.2) розміщено інтерактивний блок «Найочікуваніші події», реалізований у вигляді каруселі з навігаційними кнопками для зручного горизонтального перегляду. Кожна картка містить стислу інформацію про захід, включаючи категорію, назву, локацію, час проведення та вартість, а також інтерактивний елемент для швидкого переходу до детального опису або придбання квитка.

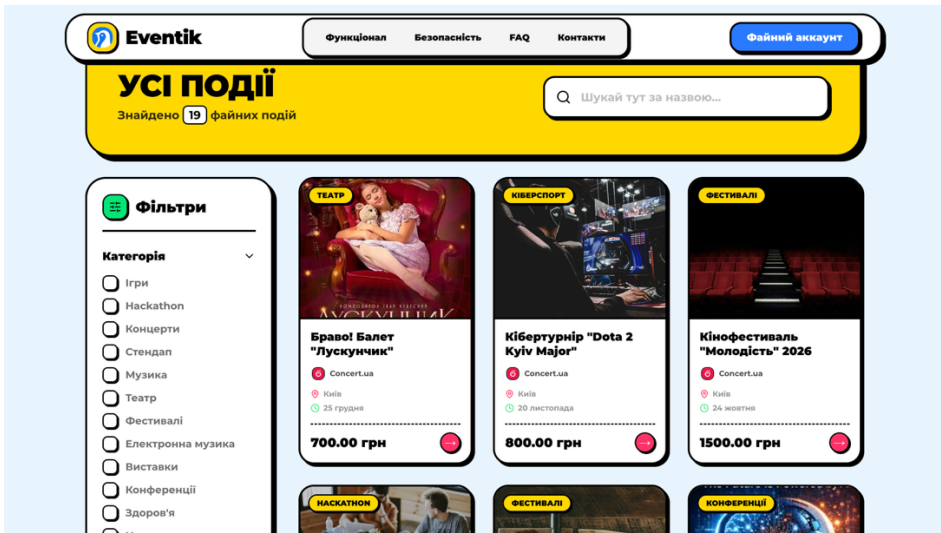


Рис. 3.3 Сторінка каталогу всіх подій із системою фільтрації

Сторінка каталогу (рис. 3.3) відображає загальний перелік доступних заходів у вигляді інформативних карток, організованих у сітку. Для швидкої навігації масивом даних передбачено верхнє поле текстового пошуку за назвою та функціональну бічну панель із чекбоксами для детальної фільтрації подій за обраними категоріями.

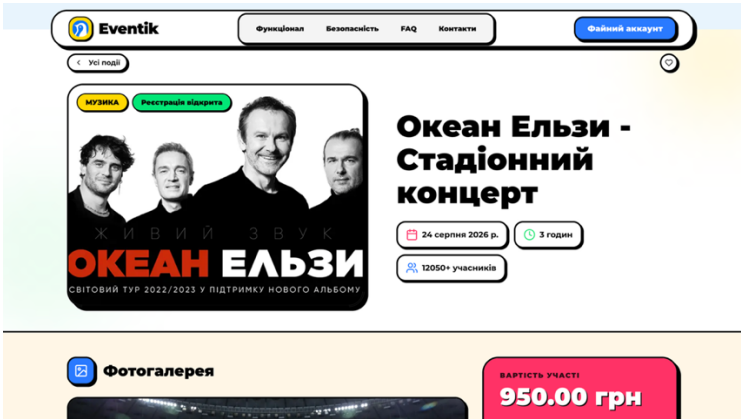


Рис. 3.4 – Сторінка детального опису події

Сторінка детального перегляду (рис. 3.4) надає користувачеві вичерпну інформацію про обраний захід, включаючи ключові організаційні параметри, статус реєстрації, дату проведення та кількість учасників. Структура інтерфейсу також дозволяє детально ознайомитися з програмою майбутньої події, переглянути супутню фотогалерею та дізнатися фіксовану вартість участі для подальшого оформлення квитка.

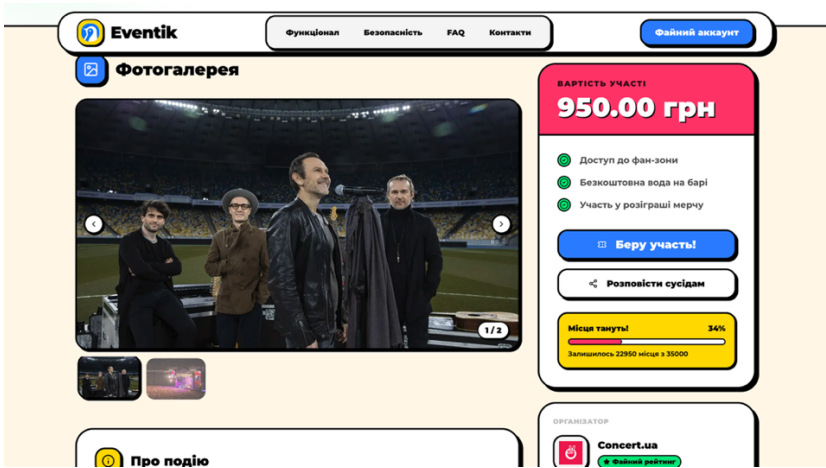


Рис. 3.5 Блок придбання квитка та фотогалерея на сторінці події

Продовження сторінки детального перегляду події (рис. 3.5) містить інтерактивну фотогалерею та комерційний блок оформлення участі. У правій частині інтерфейсу відображається детальна інформація про вартість квитка з переліком переваг, динамічний індикатор залишку вільних місць та головна кнопка дії, яка ініціює процес придбання квитка.

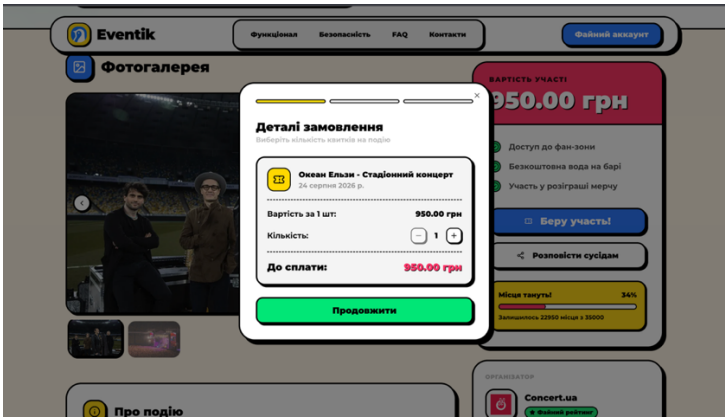


Рис. 3.6 Модальне вікно вибору кількості квитків під час оформлення замовлення

Процес придбання супроводжується викликом модального вікна деталей замовлення (рис. 3.6), де користувач може вказати необхідну кількість квитків за допомогою відповідних елементів керування. Система автоматично перераховує загальну суму до сплати згідно з обраною кількістю, після чого дозволяє перейти до наступного кроку оформлення через кнопку підтвердження.

A screenshot of the Eventik website interface. A modal window titled "Ваші дані" (Your data) is centered on the screen, prompting the user to fill in information for ticket delivery. The form includes input fields for "Ім'я" (Name) with "Тарас" entered, "Прізвище" (Surname) with "Шевченко" entered, "Email" with "taras@example.com", "Телефон (опціонально)" (Phone, optional) with "+380", and "Особливі побажання (опціонально)" (Special requests, optional) with "Наприклад: вегетаріанське меню" (For example: vegetarian menu). Navigation buttons "Назад" (Back) and "Продовжити" (Continue) are at the bottom of the modal. The background shows a concert event page for "Concert.ua" with a ticket price of 950.00 грн and a progress bar for ticket sales.

Рис. 3.7 Форма введення персональних даних під час купівлі квитка

Наступним етапом оформлення замовлення є заповнення форми персональних даних у відповідному модальному вікні (рис. 3.7). Інтерфейс передбачає введення обов'язкової інформації (ім'я, прізвище, електронна пошта) для генерування та надсилання квитка, а також містить додаткові опціональні поля для контактного телефону та особливих побажань.

A screenshot of the Eventik website interface showing the payment modal. The modal is titled "Оплата" (Payment) and asks the user to "Вибрати спосіб оплати" (Choose a payment method). It displays the "Сума до сплати: 950.00 грн" (Amount to be paid: 950.00 грн). There is a radio button selected for "Карткою онлайн" (Online card payment). Navigation buttons "Назад" (Back) and "Сплатити" (Pay) are at the bottom. The background is the same concert event page as in Figure 3.7.

Рис. 3.8 Модальне вікно оплати замовлення

Завершальним етапом оформлення замовлення є модальне вікно оплати (рис. 3.8), яке фіксує підсумкову суму до розрахунку. Користувачеві пропонується підтвердити доступний метод платежу та ініціювати транзакцію для успішного завершення купівлі квитка.

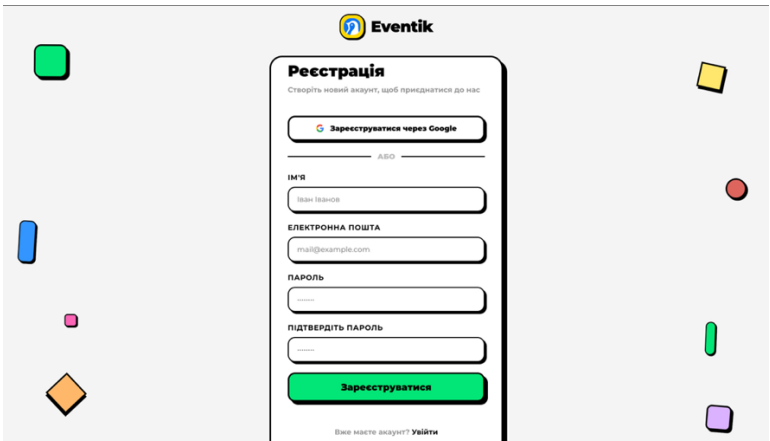


Рис. 3.9 Форма реєстрації та автентифікації користувача

Сторінка реєстрації та автентифікації (рис. 3.9) надає користувачам гнучкі можливості для створення нового облікового запису. Інтерфейс підтримує два базові методи: традиційне заповнення форми з використанням імені, електронної пошти й пароля, або ж швидку авторизацію завдяки інтеграції з існуючим профілем Google.

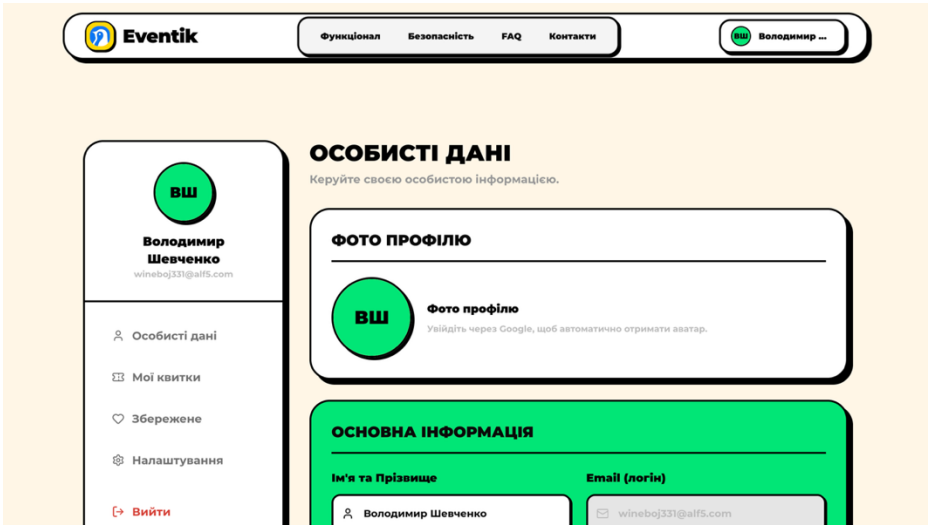


Рис. 3.10 Сторінка особистого профілю користувача

Сторінка особистого профілю забезпечує централізований доступ до управління персональною інформацією та налаштуваннями облікового

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		57

запису. Інтерфейс складається з навігаційного бічного меню та основної робочої області, де користувач може зручно переглядати свої базові дані, такі як ім'я, електронна адреса та аватар.

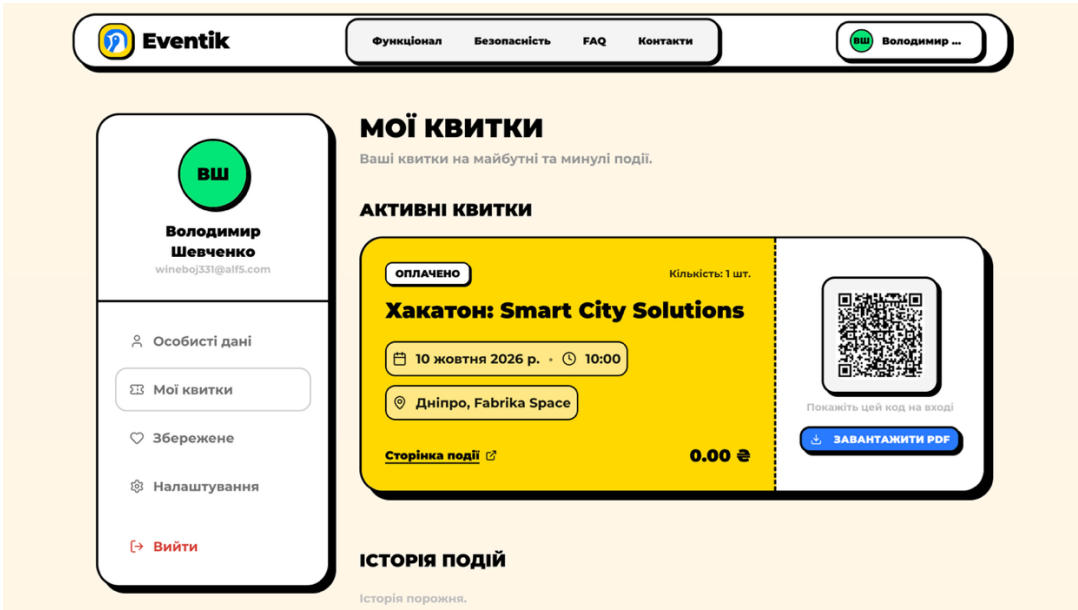


Рис. 3.11 Розділ перегляду придбаних квитків з QR-кодом

Розділ «Мої квитки» в особистому кабінеті (рис. 3.11) надає користувачеві структурований доступ до переліку активних та минулих замовлень. Картка придбаного квитка містить вичерпну інформацію про подію, опцію збереження у форматі PDF, а також генерує унікальний QR-код для зручного безконтактного сканування безпосередньо на локації заходу. Процес візуалізації QR-коду відбувається динамічно на стороні клієнта, спираючись на криптографічні дані транзакції. У лістингу наведено фрагмент коду, що відповідає за безпечну генерацію цього графічного ідентифікатора.

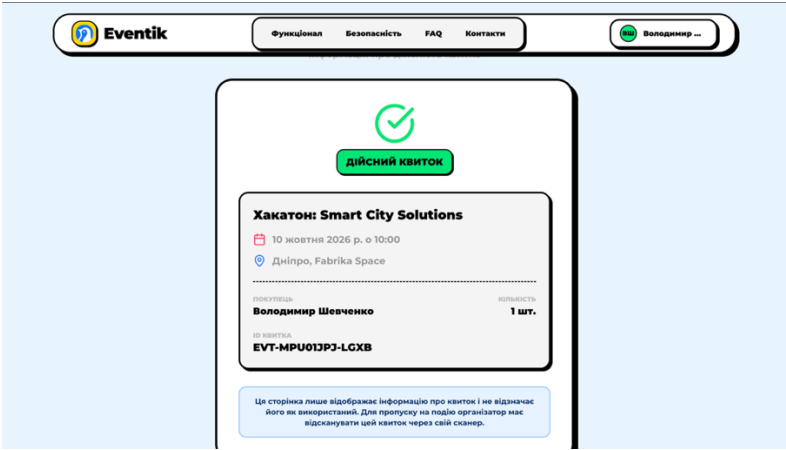


Рис. 3.12 Інформаційна сторінка дійсного квитка після сканування QR-коду

Сторінка результату сканування QR-коду (рис. 3.12) візуально підтверджує дійсність квитка та виводить на екран ключову інформацію про замовлення. Інтерфейс надає вичерпні дані для верифікації учасника, деталізуючи назву, дату і локацію заходу, а також містить персональні дані покупця та унікальний ідентифікатор пропуску.

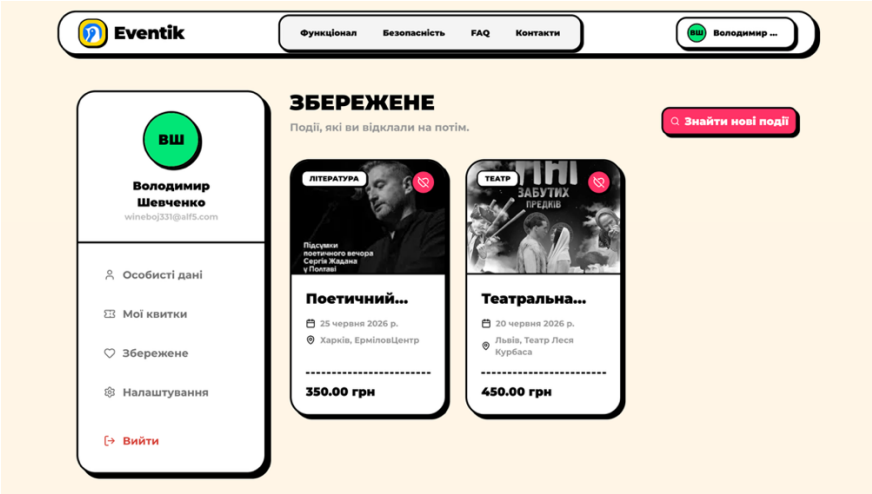


Рис. 3.13 Розділ збережених подій в особистому кабінеті користувача

Розділ «Збережене» в особистому кабінеті (рис. 3.13) призначений для акумулювання та зручного доступу до подій, які користувач відзначив для подальшого ознайомлення. Вміст вкладки відображається у вигляді сітки інформаційних карток, що містять ключові деталі кожного відкладеного заходу, а також спеціальну іконку для швидкого видалення елемента з персонального списку. Для розширення власної добірки у верхній частині

інтерфейсу передбачена акцентна кнопка «Знайти нові події», яка слугує ярликом для зручного переходу до загального каталогу системи.

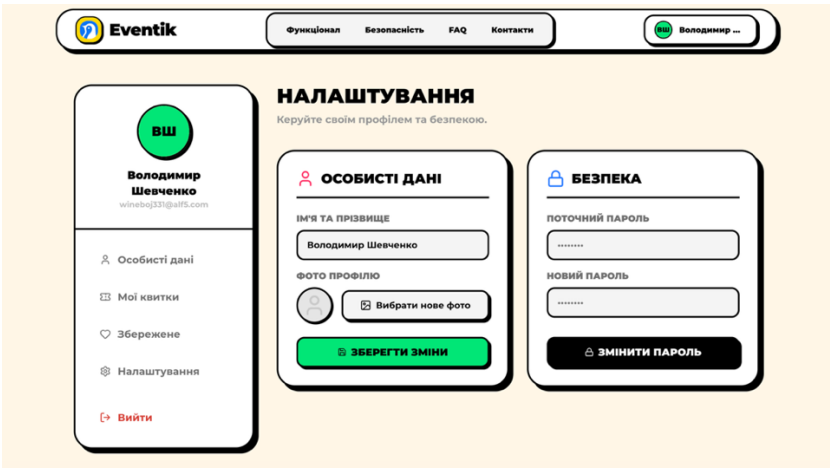


Рис. 3.14 Сторінка налаштувань профілю та безпеки облікового запису

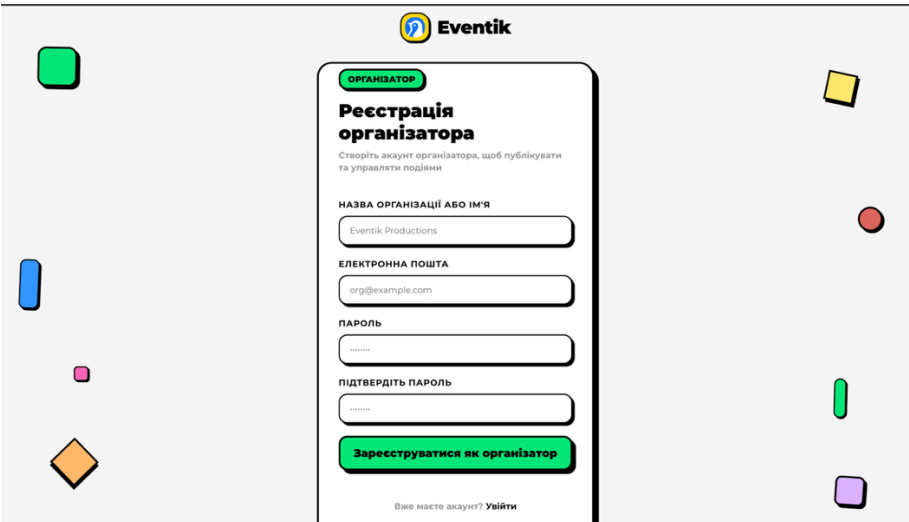


Рис. 3.15 Форма реєстрації облікового запису організатора подій

Система передбачає відокремлену процедуру створення облікового запису для постачальників послуг через форму реєстрації організатора (рис. 3.15). Інтерфейс вимагає заповнення ідентифікаційних даних, зокрема назви компанії або імені приватного представника, разом зі стандартними полями авторизації (електронна пошта та пароль із підтвердженням). Успішне проходження цього етапу формує профіль із розширеними правами доступу, що дозволяє користувачеві публікувати нові заходи та здійснювати повноцінне управління ними в межах платформи.



Рис. 3.16 Панель керування (дашборд) зі зведеною статистикою організатора

Головна панель керування, або дашборд організатора (рис. 3.16), слугує основним аналітичним інструментом для комплексного моніторингу діяльності на платформі.

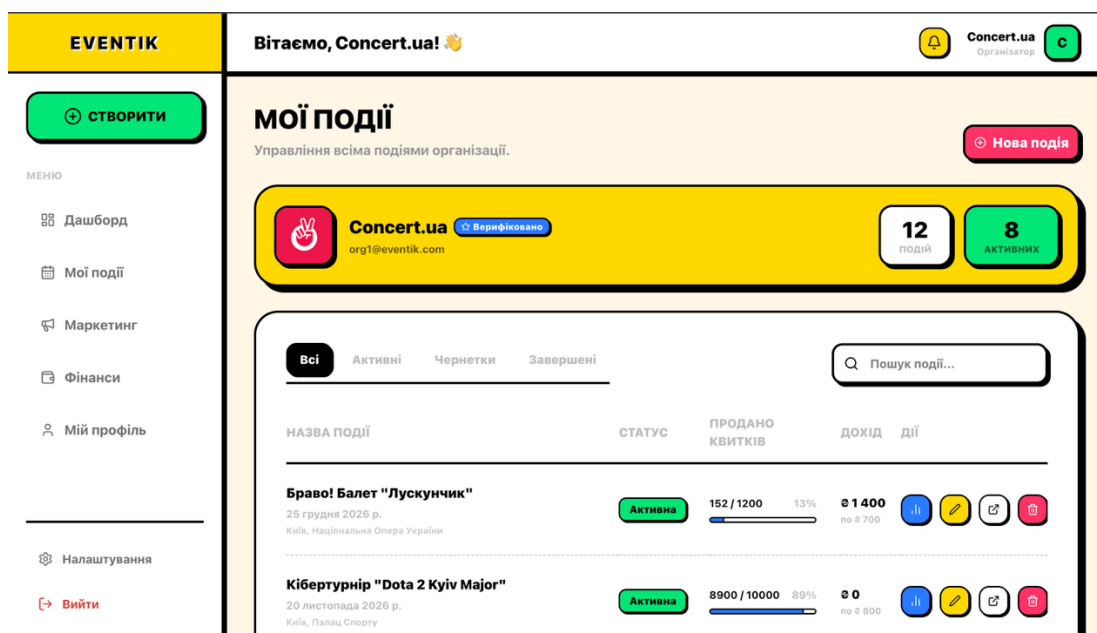


Рис. 3.17 Сторінка управління подіями організатора

Розділ «Мої події» (рис. 3.17) є основним робочим простором організатора для централізованого управління всіма створеними заходами. У верхній частині інтерфейсу розташовано інформаційний блок профілю компанії, що відображає статус верифікації, поточну кількість загальних та активних подій,

а також містить акцентну кнопку для ініціалізації процесу створення нового заходу.

EVENTIK

Вітаємо, Concert.ua!

Concert.ua

Організатор

←

РЕДАГУВАТИ ПОДІЮ

Кінофестиваль "Молодість" 2026

ОСНОВНА ІНФОРМАЦІЯ

Назва події *

Кінофестиваль "Молодість" 2026

Категорія

Опис події

Київський міжнародний кінофестиваль "Молодість". Перегляди конкурсних фільмів, зустрічі з режисерами, майстер-класи та церемонія нагородження.

МЕНЮ

Дашборд

Мої події

Маркетинг

Фінанси

Мій профіль

Налаштування

Вийти

Рис. 3.18 Сторінка редагування основної інформації про подію

Інтерфейс сторінки редагування (рис. 3.18) надає організатору інструменти для модифікації та актуалізації даних раніше створеного заходу. Робоча область містить структуровану форму з текстовими полями для оновлення базової інформації, зокрема назви події, її категорії та детального опису.

EVENTIK

Вітаємо, Concert.ua!

Concert.ua

Організатор

←

РЕДАГУВАТИ ПОДІЮ

Аналітика та список покуців для "Браво! Балет" / Лускунчик

ДОХІД

ПРОДАНО КВИТКІВ

ЗАПОВНЕНІСТЬ

ЗАМОВЛЕНЬ

СПИСОК ЗАМОВЛЕНЬ

НОМЕР	ПОКУПЕЦЬ	КІЛЬКІСТЬ	СУМА	СТАТУС	ДАТА
EVT-MPTSWYSM-E8KO	Taras Andrew 4545andreytr449@gmail.com +380967567567	1 шт.	€ 700	ОЧІКУЄ	31.05.2026, 13:14
EVT-MPTQBQXH-33GJ	Atlas Festival org3@eventik.com +380967567567	1 шт.	€ 700	ОПЛАЧЕНО	31.05.2026, 12:01
EVT-MPTQ7SRT-YMNF	Atlas Festival org3@eventik.com	1 шт.	€ 700	ОЧІКУЄ	31.05.2026, 11:58
EVT-MPTPZ5N2-EH4P	Atlas Festival org3@eventik.com	1 шт.	€ 700	ОЧІКУЄ	31.05.2026, 11:51

МЕНЮ

Дашборд

Мої події

Маркетинг

Фінанси

Мій профіль

Налаштування

Вийти

Рис. 3.19 Сторінка індивідуальної статистики та списку замовлень конкретної події

Сторінка індивідуальної статистики події (рис. 3.19) забезпечує організатора детальними аналітичними даними щодо перебігу конкретного заходу. У верхній частині робочої області розташовано блок ключових показників ефективності, який за допомогою наочних карток відображає поточний фінансовий дохід, співвідношення реалізованих квитків до загальної квоти, відсоток заповненості локації та загальну кількість створених замовлень.



Рис. 3.20 Сторінка розділу «Маркетинг» для управління промокодами

Розділ «Маркетинг» (рис. 3.20) надає організатору спеціалізовані інструменти для стимулювання продажів та управління програмами лояльності. Основну частину робочої області займає блок керування промокодами, який дозволяє генерувати унікальні текстові комбінації для надання знижок покупцям квитків.

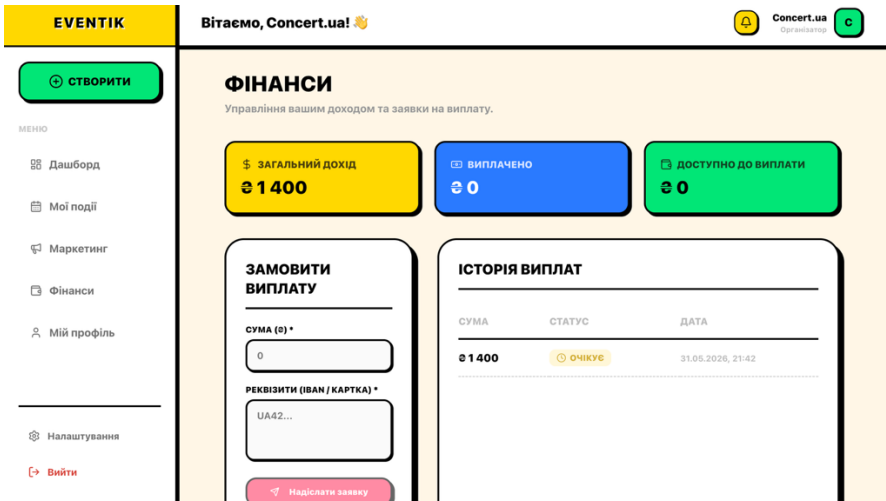


Рис. 3.21 Сторінка управління фінансами та оформлення заявки на виплату

Розділ «Фінанси» (рис. 3.21) слугує ключовим інструментом для управління грошовими потоками організатора та виведення отриманих від продажу квитків коштів. У верхній частині робочої області розташовано панель зведеного балансу, яка за допомогою карток наочно демонструє загальний дохід, суму успішно виведених коштів та актуальний залишок, доступний до виплати.

Рис. 3.22 – Сторінка профілю та налаштувань організатора

Сторінка профілю організатора (рис. 3.22) призначена для управління ідентифікаційними даними компанії та контролю загального стану облікового запису. Інтерфейс робочої області візуально поділений на два основні інформаційні блоки. Ліворуч розташовано панель із логотипом організації та окрему картку статусу акаунта, яка наочно відображає поточний етап верифікації, рівень заповненості профілю та активний тарифний план. Права частина екрану містить розширену форму редагування основної інформації, де організатор має можливість оновити офіційну назву компанії чи ФОП, додати детальний текстовий опис своєї діяльності та переглянути актуальну електронну адресу.

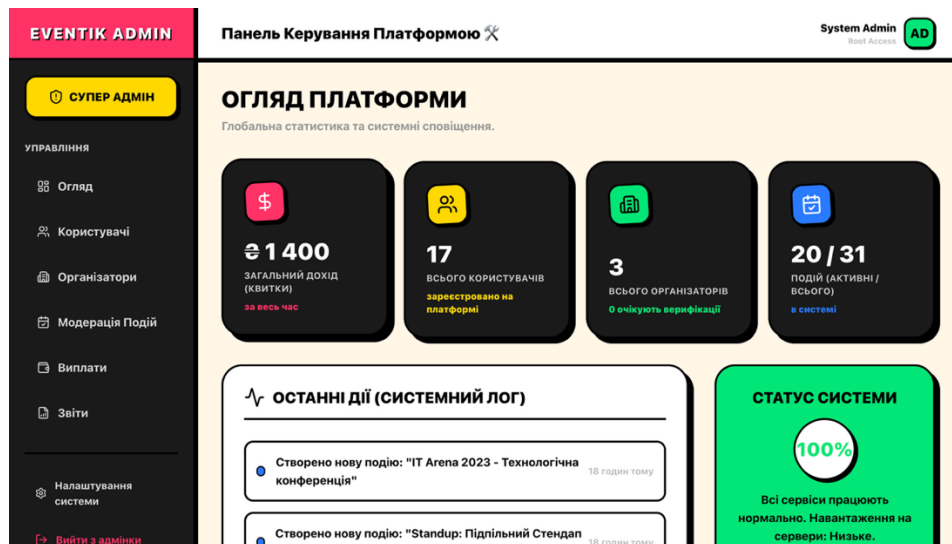


Рис. 3.23 Головна панель керування (дашборд) адміністратора платформи

Головна панель керування адміністратора (рис. 3.23) слугує центральним інструментом для глобального моніторингу роботи платформи та технічного стану системи. У верхній частині робочої області розташовано інформаційний блок із ключовими метриками, який у вигляді контрастних карток відображає зведену макростатистику: загальний фінансовий обіг, кількість зареєстрованих користувачів і верифікованих організаторів, а також співвідношення активних та завершених подій на ресурсі.

The user management page shows a list of users with the following columns: КОРИСТУВАЧ / EMAIL, РОЛЬ, РЕЄСТРАЦІЯ, СТАТУС, ДІЇ.

КОРИСТУВАЧ / EMAIL	РОЛЬ	РЕЄСТРАЦІЯ	СТАТУС	ДІЇ
Atlas Festival org3@eventik.com ID: 5f45c6e2a2e18f8b5c7a9b5d4f0c9b1e	ОРГАНІЗАТОР	30.05.2026	АКТИВНИЙ	Переглянути, Вилучити
Kyiv Music Days org2@eventik.com ID: 15b6b9e0d7d14a0b822043f4a0f9b1e	ОРГАНІЗАТОР	30.05.2026	АКТИВНИЙ	Переглянути, Вилучити
Concert UA org1@eventik.com ID: 1b7f6a0d0c0c0c0c0c0c0c0c0c0c0c0c	ОРГАНІЗАТОР	30.05.2026	АКТИВНИЙ	Переглянути, Вилучити
testauth testauth@gmail.com ID: 358f6e2b0f1f0b2b0b0b0b0b0b0b0b0b	КОРИСТУВАЧ	30.05.2026	АКТИВНИЙ	Переглянути, Вилучити
mylene farmer mylene9490@icloud.com	КОРИСТУВАЧ	30.05.2026	АКТИВНИЙ	Переглянути, Вилучити

Рис. 3.24 Сторінка управління користувачами та модерації облікових записів

Розділ управління користувачами (рис. 3.24) надає адміністратору платформи централізований інструментарій для контролю та модерації всіх облікових записів системи.

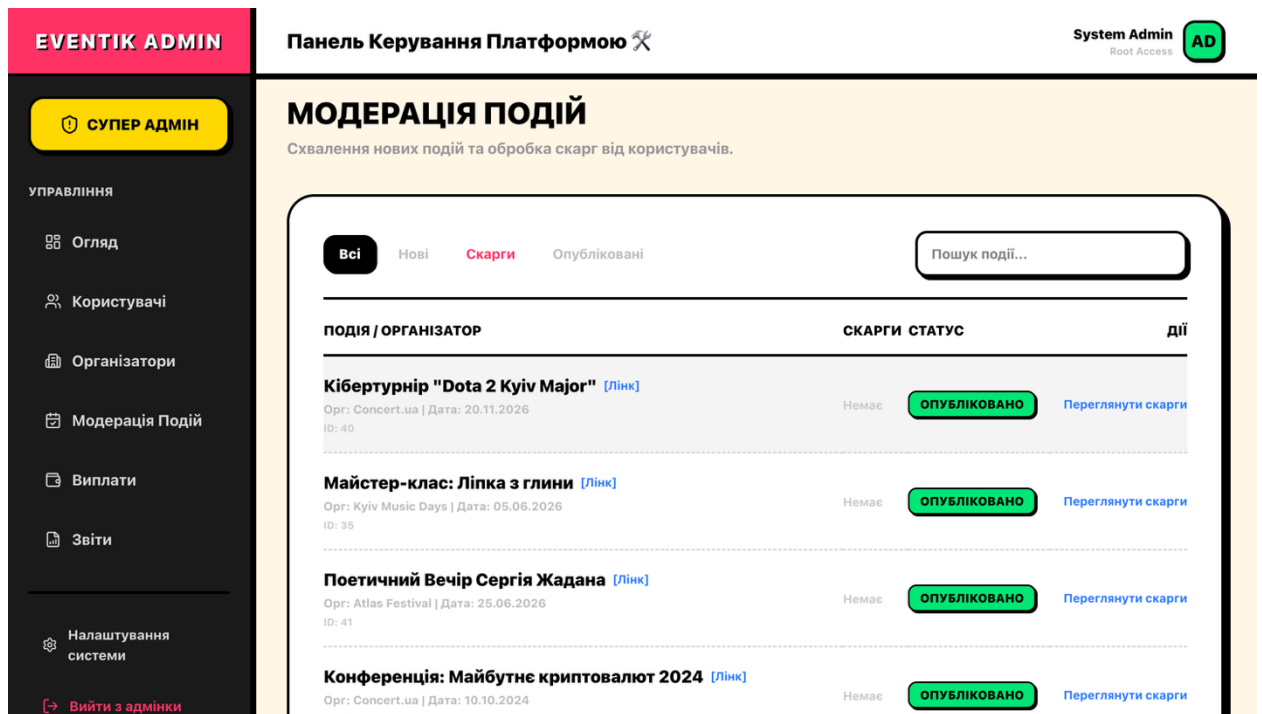


Рис. 3.25 Сторінка модерації подій та обробки скарг користувачів

Розділ «Модерація подій» (рис. 3.25) слугує основним інструментом адміністратора для контролю за контентом на платформі. Робоча область містить структурований перелік заходів із зазначенням їхньої назви, прив'язаного організатора, унікального ідентифікатора та поточного статусу (наприклад, «Опубліковано»).

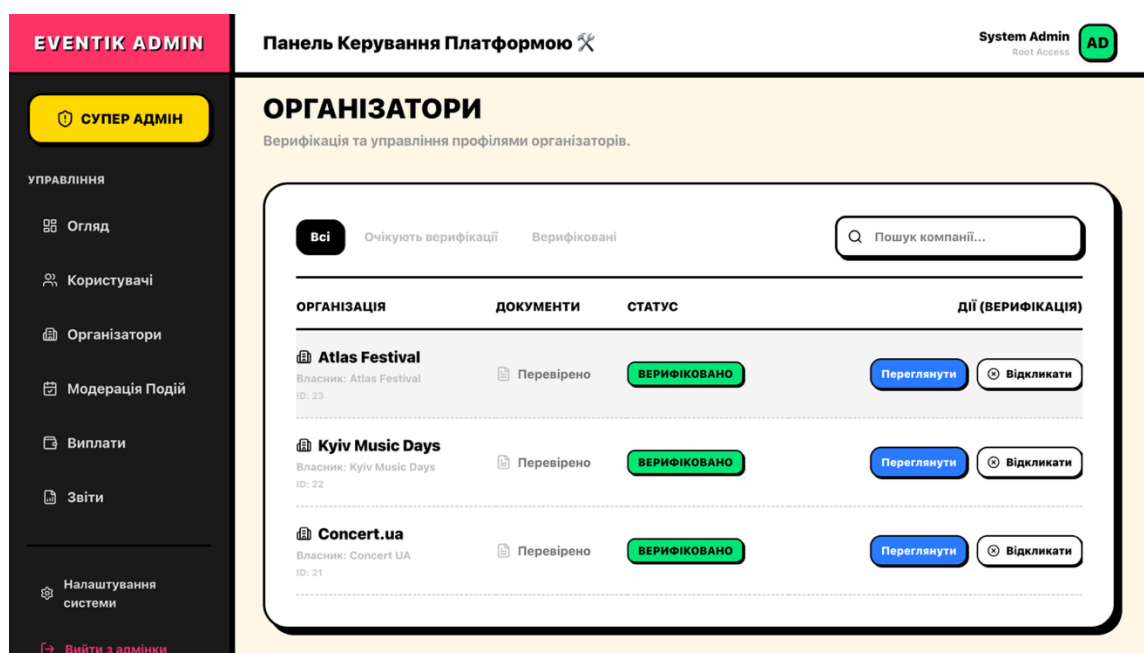


Рис. 3.26 Сторінка управління та верифікації організаторів

Розділ управління організаторами за своєю архітектурою та базовим функціоналом повторює панель модерації звичайних користувачів, надаючи інструменти для перегляду детальної інформації та блокування порушників. Проте ключовою відмінністю цього модуля є наявність опції верифікації.

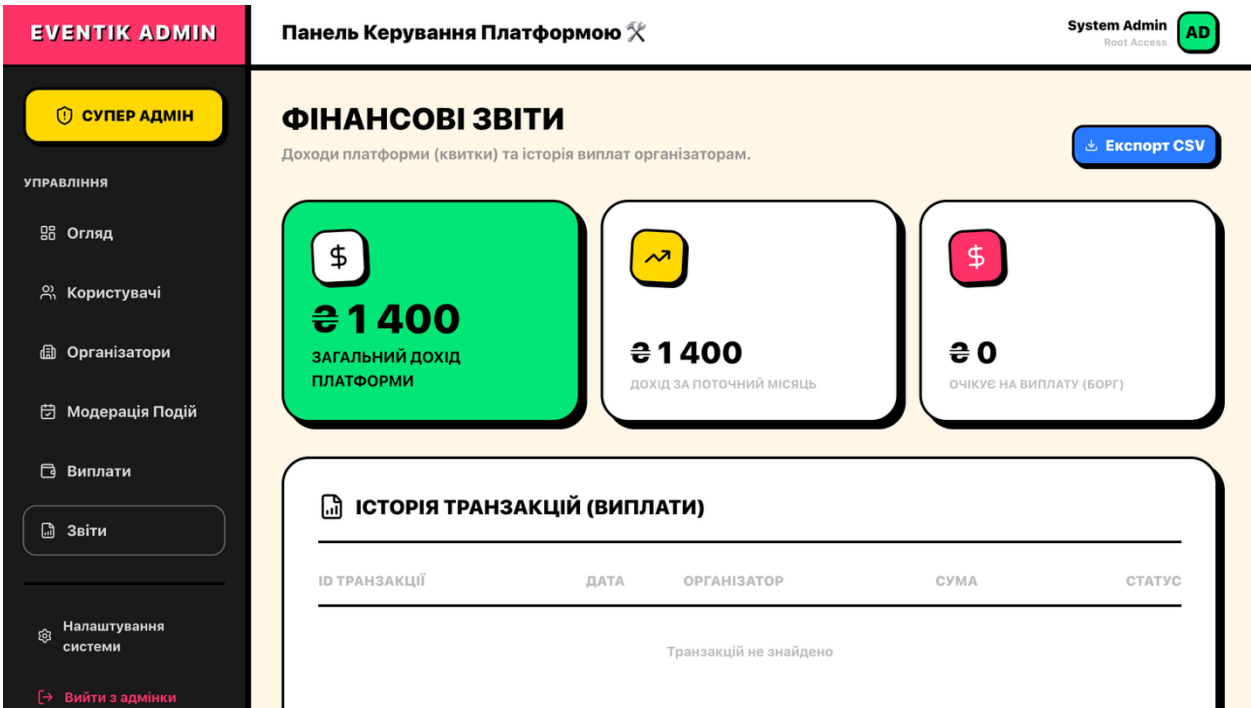


Рис. 3.27 Сторінка фінансових звітів та історії виплат в панелі керування адміністратора

Розділ «Виплати» слугує основним операційним модулем для адміністрування фінансових запитів. Тут адміністратор отримує централізований доступ до всіх сформованих заявок на виведення коштів від організаторів. Інтерфейс дозволяє перевірити вказані банківські реквізити, підтвердити фактичний переказ грошей та змінити статус відповідної транзакції (наприклад, з «очікує» на «виплачено»), закриваючи таким чином поточні зобов'язання платформи перед постачальниками послуг.

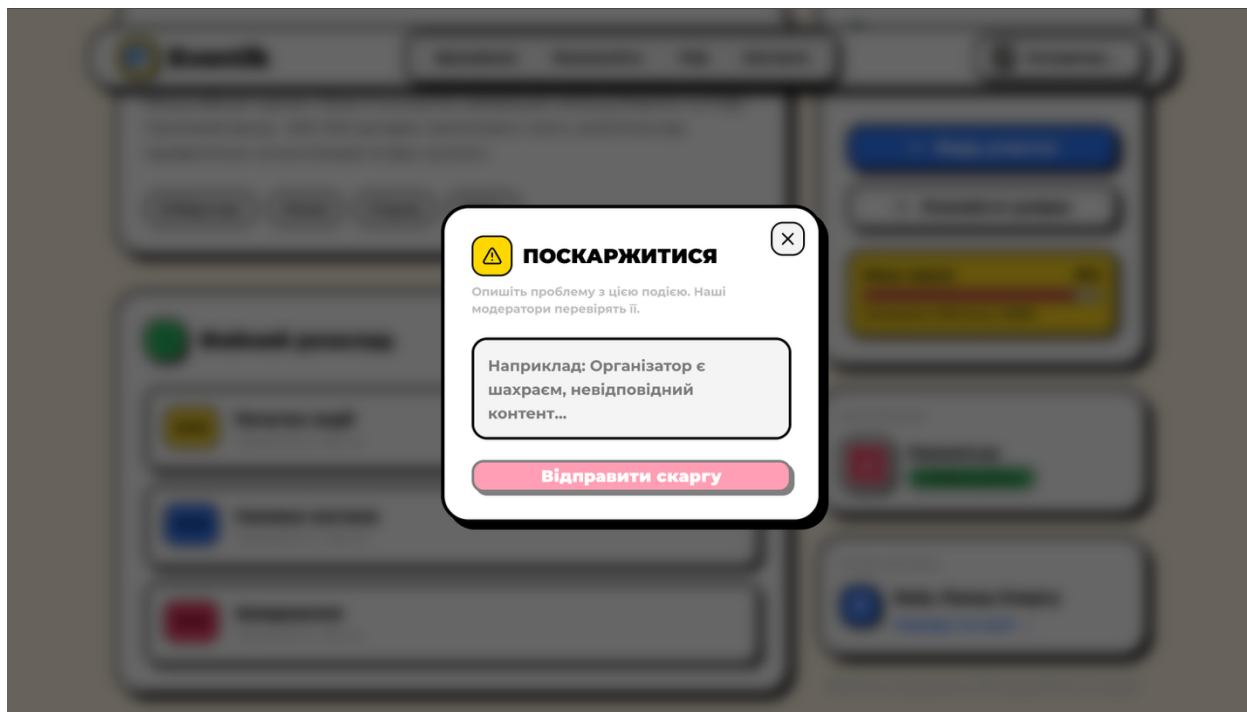


Рис. 3.27 Модальне вікно оформлення скарги на подію

Модальне вікно подачі скарги (рис. 3.27) слугує важливим інструментом користувацького контролю та безпеки на платформі. Інтерфейс дозволяє оперативно повідомити адміністрацію про виявлені порушення, пов'язані з конкретним заходом.

3.3 Тестування роботи програмного продукту

Для підтвердження працездатності, стабільності та відповідності розробленого програмного забезпечення технічним вимогам було проведено комплексне тестування системи. Процес верифікації охоплював два ключові етапи: модульне (Unit) тестування для перевірки коректності ізолюваної логіки окремих компонентів та навантажувальне (Load) тестування для оцінки поведінки системи під впливом інтенсивного потоку вхідних запитів.

Модульне тестування реалізовано з використанням сучасного інструментарію автоматизації Jest [24]. Основною метою цього етапу було забезпечення правильного функціонування критично важливих елементів архітектури додатку, зокрема компонентів користувацького інтерфейсу, утиліт обробки даних та сервісів інтеграції з REST API.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						68
Змн.	Арк.	№ докум.	Підпис	Дата		

Під час запуску тестового сценарію було виконано перевірку чотирьох основних тестових наборів (Test Suites). Результати проведення модульного тестування узагальнено та представлено в таблиці 3.3.

Таблиця 3.3

Результати модульного тестування компонентів системи

№ з/п	Назва тестового набору (Test Suite)	Кількість тестів	Статус
1	Компонент відображення цін (price-badge)	7	PASS (Успішно)
2	Утиліти обробки подій (event.utils)	8	PASS (Успішно)
3	Компонент категорій (category-badge)	5	PASS (Успішно)
4	Сервіс REST API (rest-api.service)	7	PASS (Успішно)
-	Загальна кількість	27	Успішно

У таблиці 3.3 наведено структуровані результати запуску автоматизованих модульних тестів. Сумарна кількість успішно виконаних атомарних перевірок становить 27 одиниць, при цьому жоден тест не завершився помилкою. Це підтверджує функціональну стабільність кодової бази додатку на рівні окремих ізольованих модулів.

Графічне підтвердження успішного завершення процесу модульного тестування у середовищі розробки наведено на рисунку 3.28.

```
Log file: report.json
admin@Mac eventik % yarn test
yarn run v1.22.22
$ jest
PASS src/app/shared/components/price-badge/price-badge.test.tsx
PASS src/__tests__/event.utils.test.ts
PASS src/app/shared/components/category-badge/category-badge.test.tsx
PASS src/pkg/rest-api/service/rest-api.service.test.ts

Test Suites: 4 passed, 4 total
Tests: 27 passed, 27 total
Snapshots: 0 total
Time: 1.233 s
Ran all test suites.
✔ Done in 2.26s.
admin@Mac eventik %
```

Рис. 3.28. Візуалізація результатів виконання модульних тестів у консолі

На рисунку 3.28 продемонстровано знімок екрана інтерфейсу командного рядка (CLI) після завершення виконання команди модульних тестів. Загальний час виконання перевірок склав 1.233 секунди. Відсутність маркерів помилок верифікує готовність системи до наступного етапу тестування.

Для оцінки поведінки ПЗ за умов високої інтенсивності користувацьких запитів було виконано сценарій навантажувального тестування за допомогою інструменту Artillery. Тестовий експеримент імітував фазу стабільного навантаження (Sustained load) з тривалістю активної генерації запитів 10 секунд. Сценарій моделював роботу користувачів: відвідування головної сторінки веб-додатку (маршрут /) та перевірку стану працездатності серверної частини (маршрут /api/health).

Загальні кількісні та якісні показники, зафіксовані за результатами навантажувального експерименту, наведено в таблиці 3.4.

Таблиця 3.4

Загальні агреговані показники продуктивності системи під навантаженням

Найменування інтегральної метрики продуктивності	Значення
Загальна кількість створених віртуальних сесій (vusers.created)	50
Кількість успішно завершених сесій (vusers.completed)	50
Кількість сесій, що завершилися аварійно (vusers.failed)	0
Сумарна кількість згенерованих HTTP-запитів	100
Кількість успішних відповідей (код статусу 200 OK)	100
Середня швидкість надходження запитів (Request rate)	20 запит./сек
Загальний обсяг успішно завантажених даних	3600 байт

У таблиці 3.4 відображено зведені макроскопічні метрики навантажувального тесту. Аналіз продемонстрував нульовий коефіцієнт відмов (vusers.failed = 0), що вказує на стійкість веб-сервера. Усі 100 надісланих HTTP-запитів повернули позитивний код відповіді 200 OK при інтенсивності навантаження 20 запитів за секунду. Оскільки різні кінцеві

точки (endpoints) мають різну обчислювальну складність, було зібрано диференційовану статистику часу відповіді (Response Time). Дані наведено в таблиці 3.5.

Таблиця 3.5

Диференційовані метрики часу відповіді за цільовими маршрутами (мс)

Цільовий серверний маршрут	К-сть запитів	Мінімальний час	Медіана (p50)	95-й перцентиль	Максимальний час
Головна сторінка (/)	50	1462.0	3197.8	7117.0	7302.0
API Health (/api/health)	50	268.0	340.4	391.6	408.0
Загальний показник	100	268.0	407.5	6187.2	7302.0

У таблиці 3.5 зафіксовано статистичний розподіл затримок обробки запитів. Ендпоінт /api/health демонструє високу швидкодію (медіанний час 340.4 мс). Для маршруту головної сторінки (/) спостерігаються значно вищі затримки (медіана 3197.8 мс), що аргументує необхідність подальшої оптимізації важких запитів до бази даних або рендерингу.

Окрім часу реакції на окремі запити, було проаналізовано інтегральну тривалість повної користувацької сесії (Session Length). Дані наведено в таблиці 3.6.

Таблиця 3.6

Параметри тривалості користувацьких сесій (мс)

Метрика	Мінімальне значення	Медіана (p50)	Середнє значення	95-й перцентиль	Максимальне значення
Тривалість сесії	4725.7	9047.6	9316.7	13770.3	13914.1

У таблиці 3.6 відображено часові інтервали життєвого циклу віртуальних користувачів. Середній час виконання сесії склав 9316.7 мс, при цьому 95%

усіх користувачів успішно виконали повний сценарій взаємодії менш ніж за 13.77 секунди.

Текстовий вивід фінального звіту Artillery (Summary report) після завершення симуляції представлено на рисунку 3.29.

```
admin@Mac eventik % npx artillery run --output report.json load-test.yml
Test run id: tq7t6_9c7baj66fexen8rydy3cdz9ddqz9t_bb7h
Phase started: Sustained load (index: 0, duration: 10s) 15:56:48(+0300)

-----
Metrics for period to: 15:56:50(+0300) (width: 0.218s)
-----

http.request_rate: ..... 5
/sec
http.requests: ..... 5
vusers.created: ..... 5
vusers.created_by_name.Visit home page and API health: ..... 5

Phase completed: Sustained load (index: 0, duration: 10s) 15:56:58(+0300)

-----
Metrics for period to: 15:57:00(+0300) (width: 9.623s)
-----

http.codes.200: ..... 4
7
http.request_rate: ..... 5
/sec
http.requests: ..... 4
5
http.response_time:
  min: ..... 1
  462
  max: ..... 7
  302
  mean: ..... 3
  999.7
  median: ..... 3
  534.1
  p95: ..... 7
  117
  p99: ..... 7
  260.8
http.response_time.2xx:
  min: ..... 1
  462
  max: ..... 7
  302
  mean: ..... 3
  999.7
  median: ..... 3
  534.1
```

Рис 3.29. Консольний звіт навантажувального тестування

На рисунку 3.29 представлено підсумкову статистику навантажувального тесту. Звіт підтверджує стабільну швидкість обробки даних та успішну обробку всіх транзакцій без виникнення системних таймаутів.

На основі отриманих результатів комплексного тестування доведено, що розроблене програмне забезпечення відповідає критеріям надійності. Модульні тести підтвердили 100% коректність бізнес-логіки. Навантажувальний аналіз довів стійкість архітектури додатка (0% помилок),

проте виявив необхідність подальшої оптимізації швидкодії завантаження головної сторінки.

Висновок до третього розділу

У третьому розділі визначено ключові особливості програмної реалізації та детально описані проєктні рішення, необхідні для забезпечення стабільного функціонування розробленого програмного продукту. Було проведено ґрунтовний опис інтерфейсної частини вебдодатку, а також продемонстровано принципи взаємодії користувача із системою для перегляду інформації, пошуку актуальних подій, фільтрації за категоріями та ціновими пропозиціями. Крім того, сформовано чіткий порядок розгортання клієнт-серверної архітектури, де вказано логічну послідовність дій та технічні особливості налаштування як клієнтського середовища, так і серверної частини.

За результатами проведеного комплексного тестування зокрема модульного та навантажувального вдалося підтвердити, що розроблений вебдодаток має високий рівень працездатності, стійко витримує конкурентні запити користувачів, а внутрішня бізнес-логіка компонентів працює абсолютно коректно.

Таким чином, реалізована функціональність повною мірою відповідає поставленим цілям і завданням дослідження, забезпечуючи зручний та ефективний інструментарій для роботи з платформою. Протестовані модулі демонструють безперебійність, стабільність і точність виконання операцій, що робить цей програмний продукт надійним та придатним для використання у практичній діяльності.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						73
Змн.	Арк.	№ докум.	Підпис	Дата		

ВИСНОВКИ

Результатом виконання кваліфікаційної роботи є розроблений та програмно реалізований вебдодаток для організації міських. Система призначена для автоматизації процесів публікації заходів та зручного бронювання квитків. Вона переводить у цифровий формат увесь життєвий цикл івенту: від створення сторінки події та налаштування лімітів місць до фінальної реєстрації відвідувача.

1. Проведено аналіз ринку івент-платформ та сервісів продажу квитків виявив низку проблем. Наявні на ринку аналоги часто перевантажені складним функціоналом, стягують високі комісії або не враховують потреб локальних та незалежних організаторів. Більшість із розглянутих платформ не мали інтуїтивно зрозумілого механізму, який би гармонійно поєднував ролі звичайного користувача-відвідувача та творця контенту.

2. Сформовано вимоги до функціональності системи, що дозволило визначити необхідний стек технологій, зокрема фреймворку Next.js для забезпечення швидкого серверного рендерингу та оптимальної маршрутизації. В основу архітектури покладено надійну доменну модель управління подіями. Вона жорстко контролює інваріанти сутностей: динамічний розрахунок статусів подій, автоматичне переведення в стан «Sold Out» та валідацію доступних місць, що захищає бізнес-процес бронювання від некоректних дій чи перевищення лімітів.

3. Здійснено програмну реалізацію клієнтської частини платформи та системи управління подіями для організаторів за рахунок використання сучасного стека технологій на базі фреймворку Next.js та мови програмування TypeScript. Завдяки Next.js реалізовано швидке завантаження сторінок і високу SEO-оптимізацію через механізми серверного рендерингу. Водночас строга типізація TypeScript забезпечила надійність архітектури, мінімізацію помилок на етапі розробки та зручне масштабування проєкту.

Релізи нових версій застосунку виконуються повністю автоматично через налаштований CI/CD-пайплайн на хмарній платформі Vercel. Така

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		74

комбінація дозволяє швидко знаходити події за критеріями та безпечно бронювати квитки. Організатор публікує заходи безпосередньо в систему і контролює заповненість та статус продажів у реальному часі.

4. Для перевірки коду налаштовано тестове середовище на базі фреймворку Jest . Вони успішно покривають найвразливіший рівень доменної логіки та гарантують безпомилкову роботу ключових алгоритмів застосунку. За результатами модульного тестування у середовищі Jest система продемонструвала 100% коректність бізнес-логіки, успішно виконавши всі перевірки за 1.233 секунди без жодної помилки. Навантажувальне тестування підтвердило стійкість вебсервера з нульовим коефіцієнтом відмов, при цьому 95% повних користувацьких сесій успішно завершилися менш ніж за 13.77 секунди. Водночас диференційований аналіз часу відповіді засвідчив високу швидкодію технічних ендпоінтів (340.4 мс), проте виявив необхідність оптимізації головної сторінки, медіанна затримка завантаження якої склала 3197.8 мс.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						75
Змн.	Арк.	№ докум.	Підпис	Дата		

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vercel Inc. Vercel Documentation. 2026. URL: <https://vercel.com/docs> (дата звернення: 14.03.2026).
2. Vercel Inc. Next.js Documentation. 2026. URL: <https://nextjs.org/docs> (дата звернення: 14.03.2026).
3. Neon Inc. Neon Serverless Postgres Documentation. 2025. URL: <https://neon.tech/docs> (дата звернення: 14.03.2025).
4. Better Auth. Better Auth Documentation. 2026. URL: <https://www.better-auth.com/> (дата звернення: 14.03.2026).
5. Stripe, Inc. Stripe API Reference. 2026. URL: <https://docs.stripe.com/api> (дата звернення: 14.03.2026).
6. Cloudinary Ltd. Cloudinary Documentation. 2025. URL: <https://cloudinary.com/documentation> (дата звернення: 14.03.2025).
7. Resend Inc. Resend Email API Documentation. 2026. URL: <https://resend.com/docs> (дата звернення: 14.03.2026).
8. Мартін Р. С. Чиста архітектура. Мистецтво розробки програмного забезпечення. Харків : Фабула, 2023. 368 с.
9. Fowler M. Refactoring: Improving the Design of Existing Code. Boston : Addison-Wesley Professional, 2023. 448 p.
10. Richardson C. Microservices Patterns: With examples in Java. Shelter Island : Manning Publications, 2022. 544 p.
11. Vercel Inc. Vercel CLI Documentation. 2026. URL: <https://vercel.com/docs/cli> (дата звернення: 14.03.2026).
12. Meta Platforms, Inc. React Documentation. 2026. URL: <https://react.dev/> (дата звернення: 14.03.2026).
13. OpenJS Foundation. Node.js Documentation. 2026. URL: <https://nodejs.org/docs> (дата звернення: 14.03.2026).
14. Microsoft Corporation. TypeScript Documentation. 2026. URL: <https://www.typescriptlang.org/docs> (дата звернення: 14.03.2026).

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		76

- 15.The PostgreSQL Global Development Group. PostgreSQL Documentation. 2026. URL: <https://www.postgresql.org/docs> (дата звернення: 14.03.2026).
- 16.Stripe, Inc. Stripe Webhooks Documentation. 2026. URL: <https://docs.stripe.com/webhooks> (дата звернення: 14.03.2026).
- 17.Google LLC. Google Identity Services Documentation. 2026. URL: <https://developers.google.com/identity> (дата звернення: 14.03.2026).
- 18.Prisma Data, Inc. Prisma ORM Documentation. 2026. URL: <https://www.prisma.io/docs> (дата звернення: 14.03.2026).
- 19.Drizzle Team. Drizzle ORM Documentation. 2026. URL: <https://orm.drizzle.team/> (дата звернення: 14.03.2026).
- 20.Vercel Inc. Vercel Edge Network Documentation. 2026. URL: <https://vercel.com/docs/edge-network> (дата звернення: 14.03.2026).
- 21.Vercel Inc. Vercel Serverless Functions Documentation. 2026. URL: <https://vercel.com/docs/functions> (дата звернення: 14.03.2026).
- 22.GitHub, Inc. GitHub Actions Documentation. 2026. URL: <https://docs.github.com/actions> (дата звернення: 14.03.2026).
- 23.Vitest Team. Vitest Documentation. 2026. URL: <https://vitest.dev/guide> (дата звернення: 14.03.2026).
- 24.Meta Platforms, Inc. Jest Documentation. 2026. URL: <https://jestjs.io/> (дата звернення: 14.03.2026).
- 25.Microsoft Corporation. Playwright Documentation. 2026. URL: <https://playwright.dev/> (дата звернення: 14.03.2026).
- 26.Cypress.io. Cypress Documentation. 2026. URL: <https://docs.cypress.io/> (дата звернення: 14.03.2026).
- 27.OpenJS Foundation. ESLint Documentation. 2026. URL: <https://eslint.org/> (дата звернення: 14.03.2026).
- 28.Prettier. Prettier Documentation. 2026. URL: <https://prettier.io/> (дата звернення: 14.03.2026).
- 29.Auth0 Inc. Introduction to JSON Web Tokens. 2026. URL: <https://jwt.io/introduction> (дата звернення: 14.03.2026).

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		77

- 30.OAuth Working Group. OAuth 2.0 Documentation. 2026. URL: <https://oauth.net/2/> (дата звернення: 14.03.2026).
- 31.Fielding R., Nottingham M., Reschke J. HTTP Semantics (RFC 9110). Internet Engineering Task Force (IETF), 2022. URL: <https://datatracker.ietf.org/doc/html/rfc9110> (дата звернення: 14.03.2026).
- 32.OWASP Foundation. OWASP Top 10 Documentation. 2026. URL: <https://owasp.org/www-project-top-ten/> (дата звернення: 14.03.2026).
- 33.Mozilla Corporation. Cross-Origin Resource Sharing (CORS). MDN Web Docs. 2026. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS> (дата звернення: 14.03.2026).
- 34.Mozilla Corporation. Content Security Policy (CSP). MDN Web Docs. 2026. URL: <https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP> (дата звернення: 14.03.2026).
- 35.Freeman E., Robson E. Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software. Sebastopol : O'Reilly Media, 2023. 688 p.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						78
Змн.	Арк.	№ докум.	Підпис	Дата		

ДОДАТКИ

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						79
Змн.	Арк.	№ докум.	Підпис	Дата		

Лістинг програми: фрагмент

```

'use client'

import confetti from 'canvas-confetti'
import { FC, useState } from 'react'

import { IEventDetails } from '@app/entities/event'
import { ITicket, PaymentMethod, useConfirmTicket, useCreateTicket } from
 '@app/entities/ticket'
import { Dialog, DialogContent, DialogTitle } from '@pkg/theme/ui/dialog'

import { StepBuyerInfo, StepBuyerInfoFormData } from './steps/step-buyer-info'
import { StepOrderDetails } from './steps/step-order-details'
import { StepPayment } from './steps/step-payment'
import { StepSuccess } from './steps/step-success'

interface IProps {
  isOpen: boolean
  onClose: () => void
  event: IEventDetails
}

type Step = 'order' | 'buyer' | 'payment' | 'success'

export const BuyTicketModal: FC<IProps> = ({ isOpen, onClose, event }) => {
  const [currentStep, setCurrentStep] = useState<Step>('order')

  // Order state
  const [quantity, setQuantity] = useState(1)
  const [buyerInfo, setBuyerInfo] = useState<Partial<StepBuyerInfoFormData>>({})
  const [ticket, setTicket] = useState<ITicket | null>(null)

  // API Hooks
  const createTicket = useCreateTicket()
  const confirmTicket = useConfirmTicket()

  const resetState = () => {
    setCurrentStep('order')
    setQuantity(1)
    setBuyerInfo({})
    setTicket(null)
  }

  const handleClose = () => {
    resetState()
    onClose()
  }

  const handleOrderNext = (qty: number) => {
    setQuantity(qty)
  }

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						80
Змн.	Арк.	№ докум.	Підпис	Дата		

```

setCurrentStep('buyer')
}

const handleBuyerNext = async (data: StepBuyerInfoFormData) => {
  setBuyerInfo(data)

  // Create ticket (pending state)
  try {
    const newTicket = await createTicket.mutateAsync({
      eventId: event.id,
      quantity,
      ...data
    })
    setTicket(newTicket)
    setCurrentStep('payment')
  } catch (error) {
    console.error('Failed to create ticket', error)
    // TODO: Add toast notification
  }
}

const handlePayment = async (method: PaymentMethod) => {
  if (!ticket) return

  try {
    const response = await confirmTicket.mutateAsync({
      id: ticket.id,
      payload: { paymentMethod: method }
    })

    if (response.checkoutUrl) {
      window.location.href = response.checkoutUrl
    } else if (response.ticket) {
      setTicket(response.ticket)
      setCurrentStep('success')
      confetti({ particleCount: 150, spread: 70, origin: { y: 0.6 } })
    } else {
      setTicket(response as any)
      setCurrentStep('success')
      confetti({ particleCount: 150, spread: 70, origin: { y: 0.6 } })
    }
  } catch (error) {
    console.error('Payment failed', error)
  }
}

return (
  <Dialog open={isOpen} onOpenChange={open => !open && handleClose()}>
    <DialogContent className='max-w-md gap-0 overflow-hidden rounded-[32px] border-3 border-black bg-white p-8 shadow-[8px_8px_0px_0px_rgba(0,0,0,1)]'>
      </* Visually Hidden Title for Accessibility */>
      <DialogTitle className='sr-only'>Купівля квитка</DialogTitle>

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						81
Змн.	Арк.	№ докум.	Підпис	Дата		

```

    { /* Progress Indicator (hide on success) */ }
    { currentStep !== 'success' && (
    <div className='mb-8 flex gap-2'>
    <div
    className={`h-2 flex-1 rounded-full border-2 border-black ${currentStep === 'order' ||
    currentStep === 'buyer' || currentStep === 'payment' ? 'bg-[#FFD700]' : 'bg-[#F4F4F4]'}`} `}
    ></div>
    <div
    className={`h-2 flex-1 rounded-full border-2 border-black ${currentStep === 'buyer' ||
    currentStep === 'payment' ? 'bg-[#FF3366]' : 'bg-[#F4F4F4]'}`} `}
    ></div>
    <div
    className={`h-2 flex-1 rounded-full border-2 border-black ${currentStep === 'payment' ? 'bg-
    [#00E676]' : 'bg-[#F4F4F4]'}`} `}
    ></div>
    </div>
    )}

    <div className='min-h-[400px]'>
    { currentStep === 'order' && (
    <StepOrderDetails event={event} initialQuantity={quantity} onNext={handleOrderNext} />
    )}

    { currentStep === 'buyer' && (
    <StepBuyerInfo
    defaultValues={buyerInfo}
    onNext={handleBuyerNext}
    onBack={() => setCurrentStep('order')}
    />
    )}

    { currentStep === 'payment' && ticket && (
    <StepPayment
    ticket={ticket}
    event={event}
    onPay={handlePayment}
    onBack={() => setCurrentStep('buyer')}
    isPaying={confirmTicket.isPending}
    />
    )}

    { currentStep === 'success' && ticket && <StepSuccess ticket={ticket} onClose={handleClose}
    /> }
    </div>
    </DialogContent>
    </Dialog>
    )
  }

  import { eq, sql } from 'drizzle-orm'
  import { headers } from 'next/headers'
  import { NextResponse } from 'next/server'
  import Stripe from 'stripe'

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						82
Змн.	Арк.	№ докум.	Підпис	Дата		

```

import { db } from '@/pkg/db'
import { events } from '@/pkg/db/schema/events'
import { tickets } from '@/pkg/db/schema/tickets'
import { sendTicketConfirmationEmail } from '@/pkg/auth/email'

const stripe = new Stripe(process.env.STRIPE_SECRET_KEY as string)
const endpointSecret = process.env.STRIPE_WEBHOOK_SECRET as string

export async function POST(req: Request) {
  const body = await req.text()
  const sig = (await headers()).get('stripe-signature') as string

  let event: Stripe.Event

  try {
    event = stripe.webhooks.constructEvent(body, sig, endpointSecret)
  } catch (err: any) {
    console.error(`Webhook signature verification failed: ${err.message}`)
    return NextResponse.json({ error: `Webhook Error: ${err.message}` }, { status: 400 })
  }

  if (event.type === 'checkout.session.completed') {
    const session = event.data.object as Stripe.Checkout.Session
    const ticketId = session.metadata?.ticketId

    if (ticketId) {
      // Find ticket
      const [ticket] = await db.select().from(tickets).where(eq(tickets.id, ticketId)).limit(1)

      if (ticket && ticket.status === 'pending') {
        // Find event
        const [dbEvent] = await db.select().from(events).where(eq(events.id, ticket.eventId)).limit(1)

        if (dbEvent) {
          // Mark ticket as paid
          await db
            .update(tickets)
            .set({
              status: 'paid',
              paymentMethod: 'card',
              paymentId: session.payment_intent as string || `STRIPE-${session.id}`,
              updatedAt: new Date(),
            })
            .where(eq(tickets.id, ticketId))

          // Increment sold seats
          await db
            .update(events)
            .set({ soldSeats: sql`${events.soldSeats} + ${ticket.quantity}` })
            .where(eq(events.id, ticket.eventId))

          // Send confirmation email

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		83

```

sendTicketConfirmationEmail({
  email: ticket.buyerEmail,
  firstName: ticket.buyerFirstName,
  ticketNumber: ticket.ticketNumber,
  eventTitle: dbEvent.title,
  eventDate: dbEvent.startDate.toISOString(),
  eventAddress: dbEvent.address,
  quantity: ticket.quantity,
  totalPrice: ticket.totalPrice,
  currency: dbEvent.currency,
}).catch(console.error)
}
}
}
}

return NextResponse.json({ received: true })
}

import { betterAuth } from 'better-auth'
import { drizzleAdapter } from 'better-auth/adapters/drizzle'
import { admin, emailOTP } from 'better-auth/plugins'
import { db } from '@/pkg/db'
import * as schema from '@/pkg/db/schema'
import { sendOtpEmail } from './email'
import { envClient, envServer } from '@/config/env'

const googleCredentials = envServer.GOOGLE_CLIENT_ID &&
envServer.GOOGLE_CLIENT_SECRET
? {
  clientId: envServer.GOOGLE_CLIENT_ID,
  clientSecret: envServer.GOOGLE_CLIENT_SECRET,
}
: null

export const auth = betterAuth({
  baseURL: process.env.BETTER_AUTH_URL ||
envClient.NEXT_PUBLIC_CLIENT_WEB_URL,
  secret: envServer.BETTER_AUTH_SECRET,
  database: drizzleAdapter(db, {
    provider: 'pg',
    schema: {
      ...schema,
      user: schema.users,
      session: schema.sessions,
      account: schema.accounts,
      verification: schema.verifications,
    },
  }),
  emailAndPassword: {
    enabled: true,
  },
  session: {

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						84
Змн.	Арк.	№ докум.	Підпис	Дата		

```

cookieCache: {
  enabled: true,
  strategy: 'jwe',
  maxAge: 60 * 5, // 5 хвилин
},
},
...(googleCredentials && {
  socialProviders: {
    google: googleCredentials,
  },
}),
user: {
  additionalFields: {
    firstName: {
      type: 'string',
      required: false,
      fieldName: 'first_name',
    },
    lastName: {
      type: 'string',
      required: false,
      fieldName: 'last_name',
    },
  },
  },
  },
  },
  plugins: [
    admin(),
    emailOTP({
      async sendVerificationOTP({ email, otp, type }) {
        await sendOtpEmail({ email, otp, type }) // Кастомна відправка OTP через Resend
      },
    }),
  ],
})

```

```
import { boolean, integer, pgTable, serial, text, timestamp, varchar } from 'drizzle-orm/pg-core'
```

```
import { events } from './events'
```

```
import { organizers } from './organizers'
```

```
export const promoCodes = pgTable('promo_codes', {
  id: serial('id').primaryKey(),

```

```
  organizerId: integer('organizer_id')
    .notNull()
    .references(() => organizers.id, { onDelete: 'cascade' }),

```

```
  eventId: integer('event_id').references(() => events.id, { onDelete: 'cascade' }),

```

```
  code: varchar('code', { length: 50 }).notNull(),
  discountPercent: integer('discount_percent').notNull(),

```

```
// Ліміти використання
```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						85
Змн.	Арк.	№ докум.	Підпис	Дата		

```
maxUses: integer('max_uses'), // null = безлімітно
usedCount: integer('used_count').notNull().default(0),
```

```
isActive: boolean('is_active').notNull().default(true),
```

```
description: text('description'),
```

```
validFrom: timestamp('valid_from', { withTimezone: true }),
validUntil: timestamp('valid_until', { withTimezone: true }),
```

```
createdAt: timestamp('created_at', { withTimezone: true }).notNull().defaultNow(),
updatedAt: timestamp('updated_at', { withTimezone: true }).notNull().defaultNow(),
})
```

```
'use server'
```

```
import { headers } from 'next/headers'
import { revalidatePath } from 'next/cache'
import { eq } from 'drizzle-orm'
import { auth } from '@pkg/auth'
import { db } from '@pkg/db'
import { complaints } from '@pkg/db/schema'
```

```
// Екшен для подання скарги з боку користувача
export async function submitComplaint(eventId: number, reason: string) {
  try {
    const session = await auth.api.getSession({ headers: await headers() })
    if (!session) {
      return { success: false, error: 'Unauthorized' }
    }

```

```
    await db.insert(complaints).values({
      eventId,
      userId: session.user.id,
      reason,
    })

```

```
    return { success: true }
  } catch (error) {
    console.error('Failed to submit complaint:', error)
    return { success: false, error: 'Internal Server Error' }
  }
}
```

```
// Екшен для модерації скарг (для адмінки)
export async function updateComplaintStatus(complaintId: number, status: 'pending' | 'reviewed'
| 'dismissed') {
  try {
    const session = await auth.api.getSession({ headers: await headers() })
    if (!session) {
```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		86

```

return { success: false, error: 'Unauthorized' }
}

// Optionally check if the user is an admin here

await db.update(complaints)
.set({ status })
.where(eq(complaints.id, complaintId))
revalidatePath('/admin/events')
revalidatePath(`/admin/events/[id]/complaints`, 'page')
return { success: true }
} catch (error) {
console.error('Failed to update complaint:', error)
return { success: false, error: 'Internal Server Error' }
}
} import { NextResponse } from 'next/server'
import type { NextRequest } from 'next/server'

import { auth } from '@/pkg/auth'

export async function proxy(request: NextRequest) {
  const { pathname } = request.nextUrl
  const session = await auth.api.getSession({ headers: request.headers })

  const isDashboard = pathname.startsWith('/dashboard')
  const isAdmin = pathname.startsWith('/admin')
  const isProfile = pathname === '/profile'

  // Protect dashboard: must be authenticated + organizer or admin
  if (isDashboard) {
    if (!session) {
      return NextResponse.redirect(new URL('/login', request.url))
    }
    if (session.user.role !== 'organizer' && session.user.role !== 'admin') {
      return NextResponse.redirect(new URL('/profile', request.url))
    }
  }

  // Protect admin: must be authenticated + admin
  if (isAdmin) {
    if (!session) {
      return NextResponse.redirect(new URL('/login', request.url))
    }
    if (session.user.role !== 'admin') {
      return NextResponse.redirect(new URL('/profile', request.url))
    }
  }

  // Redirect organizer from /profile to /dashboard
  if (isProfile && session?.user.role === 'organizer') {
    return NextResponse.redirect(new URL('/dashboard', request.url))
  }
}

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
Змн.	Арк.	№ докум.	Підпис	Дата		87

```

    }

    return NextResponse.next()
  }

export const config = {
  matcher: ['/dashboard/:path*', '/admin/:path*', '/profile'],
}

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						88
Змн.	Арк.	№ докум.	Підпис	Дата		