МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Відокремлений структурний підрозділ
«Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя»

Відокремлений структурний підрозділ «Тернопільський фаховий коледж Тернопільського національного технічного університету імені Івана Пулюя»
(повне найменування вищого навчального закладу)

Відділення телекомунікацій та електронних систем
(назва відділення)

Циклова комісія комп'ютерних наук
(повна назва циклової комісії)

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи

фахового молодшого бакалавра

на тему:Розробка веб-сайту для бронювання квитків для концертного залу «Віраж»

Виконав: студент 4 курсу, групи КН-423

спеціальності 122 «Комп'ютерні науки»
(шифр і назва спеціальності)

Кравець Андрій Володимирович
(прізвище та ініціали)

Керівник Болюбаш Роксолана Юріївна
(прізвище та ініціали)

Рецензент _____
(прізвище та ініціали)

м. Тернопіль – 2026

ВІДОКРЕМЛЕНИЙ СТРУКТУРНИЙ ПІДРОЗДІЛ
«ТЕРНОПІЛЬСЬКИЙ ФАХОВИЙ КОЛЕДЖ
ТЕРНОПІЛЬСЬКОГО НАЦІОНАЛЬНОГО ТЕХНІЧНОГО УНІВЕРСИТЕТУ
ІМЕНІ ІВАНА ПУЛЮЯ»

Відділення телекомунікацій та електронних систем
Циклова комісія комп'ютерних наук
Освітньо-професійний ступінь «фаховий молодший бакалавр»
Спеціальність 122 Комп'ютерні науки
Галузь знань 12 Інформаційні технології

ЗАТВЕРДЖУЮ
Голова циклової комісії комп'ютерних наук
_____ Галина МАРЦІЯШ
« 02 » березня 2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Кравцю Андрію Володимировичу
(прізвище, ім'я, по батькові)

1. Тема роботи:Розробка веб-сайту для бронювання квитків для концертного залу «Віраж»

керівник роботи Болюбаш Роксолана Юріївна,
(прізвище, ім'я, по батькові)
затверджені наказом вищого навчального закладу № 4/9-132 від 27.02.2026 р.
2. Строк подання студенткою роботи: 19.06.2026 р.
3. Вихідні дані до роботи: технічне завдання на розробку програмного забезпечення, мови програмування: TypeScript, JavaScript; фреймворки: Next.js, React, Tailwind CSS; бібліотека React, NextAuth, pdf-lib, стандарти IEEE 29148-2018, IEEE 29119, ДСТУ 8302:2015.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити):

1 Загальний розділ
1.1 Аналітичний огляд існуючих рішень
1.2 Технічне завдання
1.2.1 Найменування та область застосування
1.2.2 Призначення розробки
1.2.3 Вимоги до функціоналу вебзастосунку
1.2.4 Вимоги до програмної документації
1.2.5 Техніко-економічні показники
1.2.6 Стадії та етапи розробки
1.2.7 Порядок тестування та прийому

2 Розробка технічного та робочого проєкту

2.1 Розробка структури сайту і web-сторінок

2.2 Створення та верстка сторінок сайту

2.3 Розробка структури бази даних сайту

2.4 Програмування сайту

2.4.1 Написання клієнтської частини

2.4.2 Написання admin-частини

2.5 Тестування web-сайту

3 Спеціальний розділ

3.1 Інструкція з розміщення сайту в Інтернеті

3.2 Інструкція з обслуговування та наповнення сайту

3.3 Інструкція з популяризації та підтримки сайту

4 Економічний розділ

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

4.3 Розрахунок витрат на електроенергію

4.4 Розрахунок суми амортизаційних відрахувань веб-сайту «Віраж»

4.5 Обчислення накладних витрат

4.6 Складання кошторису витрат та визначення собівартості веб-сайту «Віраж»

4.7 Розрахунок ціни робіт

4.8 Визначення економічної ефективності і терміну окупності капітальних вкладень

5 Охорона праці, техніка безпеки та екологічні вимоги

5.1 Нормативно-правова база та характеристика умов праці розробника

5.2 Опис та оснащення робочого місця розробника

5.3 Аналіз небезпечних і шкідливих виробничих факторів

5.4 Техніка безпеки під час роботи з ПЕОМ та програмним забезпеченням

5.5 Електробезпека та пожежна безпека

5.6 Екологічні вимоги при розробці та експлуатації сайту «Віраж»

5.7 Перша допомога та дії при нещасних випадках

6 Висновки

Додаткові вказівки: виконання кваліфікаційної роботи із розробкою програмного продукту – вебсайту.

5. Перелік графічного матеріалу:

1. Схема структурна програми
2. UML-діаграма варіантів використання
3. ER-діаграма бази даних вебсайту
4. Таблиця техніко-економічних показників

6. Консультанти розділів роботи

Розділ	**Ім'я, прізвище та посада консультанта**	**Підпис, дата**	
		завдання видав	**завдання прийняв**
Економічний розділ	Любов КАЛУШКА		
Охорона праці, техніка безпеки та екологічні вимоги	Генадій ГОРЯЧЕК		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Отримання і аналіз технічного завдання	20.03.2026	
2	Збір і узагальнення інформації	01.05.2026	
3	Написання першого розділу	15.05.2026	
4	Розробка технічного та робочого проєкту	29.05.2026	
5	Написання спеціального розділу	05.06.2026	
6	Розрахунок економічної частини	08.06.2026	
7	Написання розділу охорони праці	09.06.2026	
8	Виконання графічної частини	10.06.2026	
9	Оформлення пояснювальної записки	11.06.2026	
10	Погодження нормоконтролю	12.06.2026	
11	Попередній захист кваліфікаційної роботи	09.06.2026	
12	Захист кваліфікаційної роботи	23.06.2026	

7. Дата видачі завдання: 05 березня 2026 р.

Студент _____ Андрій КРАВЕЦЬ
(підпис)

Керівник роботи _____ Роксолана БОЛЮБАШ
(підпис)

ЗМІСТ

АНОТАЦІЯ

Тема кваліфікаційної роботи: Розробка веб-сайту для бронювання квитків для концертного залу «Віраж»

Метою кваліфікаційної роботи є розробка full-stack web-застосунку для афіші подій, бронювання місць на схемі залу (420 місць), генерації PDF-квитків, особистого кабінета, чату підтримки, відгуків та адмін-панелі.

Пояснювальна записка складається з п’яти розділів.

У загальній частині описуються аналітичний огляд існуючих рішень та аналіз технічного завдання на веб-сайт бронювання квитків для концертного залу «Віраж», сформованого за результатами порівняння платформ Concert.ua, Karabas Live і Ticketmaster.

У другому розділі представлено процес створення програмного продукту, опис та обґрунтування вибору структури та методу організації вхідних та вихідних даних, опис алгоритмів, інформаційних зв’язків, зовнішнє проектування програми, тестування та налагодження програм на прикладі сайту «Віраж»

В спеціальній частині описані процес інсталяції програмного продукту, інструкція з використання тестових наборів та інструкція з користування програмою розгортання локально та на хостингу.

Розрахунок вартості розробки та економічної ефективності приведено в економічній частині визначено трудомісткість, собівартість, ціну програмного продукту, очікуваний річний прибуток і термін окупності інвестицій

Основні питання охорони праці та техніки безпеки розглянуто в п’ятому розділі організація робочого місця розробника, безпека при роботі з ПЕОМ та екологічні вимоги під час розробки і експлуатації сайту

Обсяг пояснювальної записки 123 сторінки.

До складу кваліфікаційної роботи входить графічна частина, яка складається з структурної схеми програми, діаграми варіантів використання, техніко-економічних показників, ER-діаграми бази даних, що виконані на окремих аркушах формату А1

ВСТУП

Культурна сфера України після 2022 року переживає поєднання викликів безпеки, економічної стабільності та потреби підтримувати соціальну стійкість через доступні культурні події. Концертні зали, навіть на етапі будівництва або реконструкції, потребують присутності в інформаційному просторі: громадяни очікують афішу в інтернеті, зрозумілі правила відвідування, можливість попереднього вибору місць і отримання електронного підтвердження. Без власного веб-ресурсу майданчик залежить від сторонніх агрегаторів, втрачає контроль над брендом і не може повноцінно демонструвати інвесторам та місту модель майбутнього сервісу [7].

Кваліфікаційна робота присвячена створенню веб-сайту концертного залу «Віраж» у м. Тернопіль. Зал за проєктом розрахований на 420 глядачівих місць, має сектори партеру, амфітеатру, лож і бенуару. Капітальне відкриття об’єкта планується орієнтовно на 2030 рік; на час виконання роботи будівля та касова інфраструктура ще не введені в експлуатацію, тому розроблений веб-застосунок є демонстраційно-експлуатаційним прототипом: він показує, як після відкриття працюватиме цифровий канал для глядача та адміністрації.

Програмний продукт реалізовано автором кваліфікаційної роботи самостійно в середовищі Node.js з використанням фреймворку Next.js (версія 16), бібліотеки React (19), мови TypeScript, Prisma ORM та бази даних SQLite. У складі прототипу передбачено публічні сторінки (головна, афіша, картка події, інтерактивна схема залу, оформлення бронювання, FAQ, контакти з картою, відгуки), особистий кабінет, модуль підтримки в чаті, генерацію PDF-квитка з QR-кодом та адміністративну панель (події, бронювання, чати, відгуки, користувачі). Онлайн-оплата банківською карткою в прототипі свідомо не реалізована: у системі зафіксовано модель оплати в касі залу до початку події, що відповідає дипломному демонстраційному характеру проєкту.

Об’єктом дослідження є процеси представлення афіші концертного залу в мережі Internet та організації бронювання місць через веб-інтерфейс з інтеграцією

в реляційну базу даних.

Предметом дослідження є методи і програмні засоби проєктування та реалізації веб-сайту з інтерактивною схемою залу, серверними API-маршрутами, автентифікацією користувачів і адміністративним інтерфейсом керування контентом.

Метою роботи є розробка цілісного веб-сайту залу «Віраж», який забезпечує повний сценарій взаємодії відвідувача з подією (перегляд, вибір місць, бронювання, отримання квитка) та надає адміністратору засоби ведення афіші, обліку бронювань, відповідей у чаті підтримки й модерації відгуків.

Для досягнення мети поставлено такі завдання:

– провести аналітичний огляд існуючих білетних платформ і сайтів культурних майданчиків;

– сформулювати технічне завдання на веб-сайт відповідно до вимог до функціоналу internet-ресурсу;

– спроєктувати структуру веб-сторінок, логічну модель даних і взаємодію клієнтської та серверної частин;

– реалізувати клієнтську частину з інтерактивною схемою на 420 місць;

– реалізувати адміністративну частину та серверні API;

– провести тестування і підготувати інструкції з розміщення та наповнення сайту;

– виконати економічне обґрунтування та розділ з охорони праці за завданням консультантів.

Методи роботи: аналіз і синтез літературних і веб-джерел; порівняльний аналіз аналогів; об’єктно-компонентне проєктування інтерфейсу; реляційне моделювання (ER-підхід); ітеративна розробка програмного забезпечення; функціональне та інтеграційне тестування за контрольними сценаріями [10].

Практичне значення результатів полягає в тому, що створений сайт можна використовувати на захисті кваліфікаційної роботи як діючий продукт (npm run

dev / production build), для презентації замовникам майбутнього залу, а також як базу для наступних етапів: підключення LiqPay або іншого платіжного шлюзу, перенесення бази на PostgreSQL, розгортання на VPS або хмарному хостингу [16]. Вихідний код структуровано за каталогами app, components, lib, prisma, що полегшує супровід.

Структура пояснювальної записки відповідає методичним вказівкам для кваліфікаційних робіт з розробки веб-сайтів спеціальності 122 «Комп'ютерні науки» [2]. Розділ 1 містить аналітичний огляд і технічне завдання. Розділ 2 описує технічний і робочий проєкт. Розділ 3 інструкції з розміщення, наповнення та популяризації. Розділи 4 і 5 економіка та охорона праці. Додатки містять UML- та ER-матеріали, лістинги, план тестування.

Нормативна база оформлення записки: ДСТУ 3008:2015 щодо структури звіту [1]; вимоги до посилань ДСТУ 8302:2015 [19]

1 ЗАГАЛЬНИЙ РОЗДІЛ

Загальний розділ є першою основною частиною пояснювальної записки. Він містить теоретико-аналітичне обґрунтування розробки веб-сайту бронювання квитків для концертного залу «Віраж» у м. Тернопіль [2].

У цьому розділі послідовно розглянуто: аналітичний огляд існуючих білетних платформ і сайтів культурних майданчиків (п. 1.1); технічне завдання на програмний продукт - вимоги до функціоналу, інтерфейсу, документації, етапів розробки, тестування та прийому прототипу (п. 1.2). Результати загального розділу є вихідними даними для технічного та робочого проєкту (розділ 2), інструкцій з експлуатації (розділ 3), економічного обґрунтування (розділ 4) та розділу з охорони праці (розділ 5).

Об’єктом аналізу виступають існуючі рішення продажу та бронювання квитків; предметом вимоги до власного web-застосунку залу на 420 місць. За результатами огляду обґрунтовано доцільність custom-розробки на стеку Next.js, React, TypeScript, Prisma і SQLite замість обмеження лише сторонніми агрегаторами [3], [7].

1.1 Аналітичний огляд існуючих рішень

Проблема, яку вирішує веб-сайт «Віраж», полягає у відсутності єдиного цифрового каналу між майбутнім концертним залом і глядачем на етапі до відкриття будівлі та після нього. Традиційна модель «лише каса в день події» створює черги, не дає прозорої картини заповненості залу в режимі, близькому до реального часу, і ускладнює аналітику для адміністрації. Агрегатори квитків вирішують частину задач продажу, але не замінюють презентацію конкретного майданчика: історія залу, технічні характеристики, схема місць у прив’язці до бренду «Віраж», чат підтримки, відгуки з відповідями закладу.

Для порівняння аналогів обрано такі критерії:

- наявність інтерактивної схеми з прив’язкою до конкретного місця (ряд,

номер, сектор);

- прозорість цінових категорій на схемі;
- електронний квиток (PDF, QR, mobile);
- можливість власнику майданчика керувати афішею;
- українська локалізація інтерфейсу;
- адаптивність для смартфонів;
- відповідність процесам безпеки (облікові записи, ролі);
- можливість розгортання власного програмного рішення на сервері закладу.

Міжнародні білетні платформи. Ticketmaster та подібні міжнародні системи забезпечують масштабну інфраструктуру продажу для стадіонів і великих арен. Вони включають антифрод-механізми, динамічне ціноутворення, інтеграцію з турпромоутерами. Для залу на 420 місць економічна та організаційна доцільність підключення до глобального оператора обмежена: комісії, вимоги до контракту, брендинг стороннього сервісу на сторінці оплати. Проте як еталон UX варто відзначити чітку схему залу, кошик місць, таймер утримання та підсумок замовлення [8].

Eventbrite орієнтований на події широкого класу (конференції, майстер-класи, клубні заходи). Гнучкість шаблонів місць для класичної концертної посадки з ложами та амфітеатром обмежена. Для «Віраж» приклад корисний щодо простоти реєстрації та email-розсилок, але не як готове рішення «під ключ» [8].

Висновок: міжнародні платформи задають орієнтир для сценарію користувача, проте не замінюють локальний брендований сайт регіонального залу.

Вітчизняні агрегатори Concert.ua та Karabas Live. Concert.ua є одним із найбільш відомих національних каналів продажу квитків. Переваги: широка аудиторія, звичний для користувача процес оплати, підтримка багатьох міст України. Недоліки для окремого залу: домінування бренду агрегатора, обмежені

можливості кастомізації сторінки майданчика, залежність від правил повернення та комісій платформи. Посилання «купити» часто веде на зовнішній домен, що розриває сценарій «сайт залу, вибір місця, квиток» [7].

Karabas Live історично сильний у сегменті розважальних та концертних подій. Функціонал близький до Concert.ua. Для дипломного проєкту важливо зафіксувати: агрегатор вирішує національний маркетинг, але не замінює презентацію майбутнього тернопільського майданчика з технічним описом, планом відкриття 2030 року, демонстрацією схеми саме залу «Віраж» [18].

Сайти окремих театрів і філармоній. Регіональні театри нерідко мають сайти на CMS (WordPress, Joomla, Drupal). Типовий набір: новини, репертуар, контакти, іноді iframe або посилання на зовнішній продаж. Інтерактивна схема залу рідко реалізована власними силами через високу вартість кастомної розробки. Перевага CMS швидке наповнення статичним контентом. Недолік, складність інтеграції транзакційного бронювання з hold місць, транзакціями SQL і генерацією PDF в одному застосунку без плагінів, що часто виходять за рамки типового WordPress-плагіну [7].

Для «Віраж» обрано підхід custom web-застосунку на Next.js саме через потребу в інтерактивній SVG-схемі на 420 місць, API для hold/sold станів і адмін-панелі, тісно пов’язаній з тією ж базою даних [3].

Обраний для реалізації «Віраж» стек веб-розробки. Next.js як фреймворк над React дозволяє поєднати:

- маршрутизацію сторінок (App Router);
- серверні компоненти та серверні маршрути API в одному репозиторії;
- оптимізацію завантаження для SEO (важливо для афіші подій у пошуку);
- зручну збірку production (npm run build) [3].

TypeScript додає статичну перевірку типів, що зменшує кількість помилок при супроводі проєкту після захисту диплому [12].

Prisma ORM надає декларативну схему моделей і типобезпечний клієнт доступу до SQLite (з перспективою PostgreSQL). Це відповідає предметній

області: події, місця, зв’язок EventSeat (статус місця на конкретну подію), бронювання, користувачі, відгуки, чати [5].

Автентифікація реалізована через NextAuth з підтримкою credentials (email + пароль, хеш bcrypt). Розділення ролей user і admin дозволяє закрити /admin через middleware [6], [17].

Генерація PDF-квитків виконана бібліотекою pdf-lib з QR-кодом (qrcode), що дає автономність від зовнішніх PDF-сервісів [14].

Візуальне оформлення побудовано на utility-first CSS Tailwind, що прискорює верстку адаптивних сторінок [13].

Інтерактивні схеми залів і логіка бронювання. Ключова відмінність концертного залу від абстрактного заходу продаж конкретного місця. У прототипі «Віраж» координати місць згенеровані програмно (функція buildHallSeatTemplate у модулі seats.ts), загалом 420 одиниць. Сектори: parterre, amphitheater, amphitheater_l, amphitheater_r, box_dress_l, box_dress_r, box_amph_l, box_amph_r, benoir. Кольори місць відображають цінову категорію; окремі стани: available, held, sold, selected (клієнтське виділення).

Механізм hold необхідний, щоб два користувачі не завершили оплату (у прототипі підтвердження броні) на одні й ті самі місця. Типовий алгоритм: POST /api/events/[id]/seats/hold встановлює heldUntil; при успішному POST /api/bookings місця переходять у sold у транзакції Prisma [5].

Така логіка узгоджується з практикою веб-бронювання та вимогами до цілісності даних у реляційних СУБД [11]. Порівняльні характеристики наведено в таблиці 1.1

Таблиця 1.1 – Порівняння аналогів і проєкту «Віраж»

Критерій	**Ticketmaster / Eventbrite**	**Concert.ua / Karabas**	**Сайт залу на CMS**	**«Віраж»**
1	2	3	4	5
Основні переваги	Сильний продаж квитків	Сильне національне охоплення	Сильний бренд залу	Повний локальний бренд
Бренд залу	Слабкий бренд залу	Зовнішній бренд	Власний бренд залу	Повний бренд

Продовження таблиці 1.1

1	2	3	4	5
Схема залу	Є	Є	Часто відсутня або зовнішня	SVG-схема на 420 місць
Адміністративна панель	Обмежена для залу	Обмежена для залу	Залежить від CMS	Власна адмінка
Чат із користувачами	Немає	Немає	Зазвичай відсутній	Є
Відгуки	На сторонніх сервісах	На сторонніх сервісах	Можуть бути відсутні	Зберігаються в БД
Технологічний стек	Закриті платформи	Закриті платформи	Залежить від CMS	Next.js + Prisma
Стан проєкту	Діючий сервіс	Діючий сервіс	Діючий сайт	Демо до відкриття у 2030 р.

На підставі аналізу доцільно розробляти власний веб-сайт, який:

- презентує зал «Віраж» у Тернополі як окремий культурний бренд;
- надає інтерактивну схему з реальною кількістю місць за проєктом;
- зберігає дані в реляційній базі (події, броні, користувачі, відгуки, чати);
- містить адміністративну панель для персоналу;
- працює українською мовою;
- є розширюваним (оплата, хмарна БД, хостинг).

Обраний стек відповідає сучасним підходам full-stack розробки та дозволяє автору кваліфікаційної роботи пройти повний цикл: від ТЗ до діючого прототипу [3], [5].

1.2 Технічне завдання

Технічне завдання розроблено за структурою рекомендацій для веб-сайтів [2], [10] і узгоджено з фактичною реалізацією в репозиторії virazh.

1.2.1 Найменування та область застосування

Повна назва програмного виробу: веб-сайт бронювання квитків для концертного залу «Віраж» з адміністративною панеллю.

Скорочена назва: веб-сайт «Віраж».

Область застосування: інформаційно-сервісна підтримка планованого концертного залу «Віраж» (м. Тернопіль). Заплановане введення об’єкта в експлуатацію - орієнтовно 2030 рік. До цього терміну сайт використовується як:

– офіційна веб-візитка майбутнього майданчика;

– демонстраційний стенд технології бронювання для громади, партнерів, керівництва навчального закладу;

– тренажер для персоналу (адміністраторів) щодо ведення афіші та обробки звернень.

Категорії користувачів:

– відвідувач (гість) - без реєстрації або з нею;

– зареєстрований користувач (role user);

– адміністратор (role admin) - доступ до /admin.

Географія: Україна, місто Тернопіль; інтерфейс українською. Запланована адреса майданчика за концептом: вул. Театральна, 9. На сторінці контактів вбудовано карту Google Maps із позначенням майбутньої локації.

Об’єкт обліку: глядацький зал на 420 місць, не менше дев’яти секторів.

1.2.2 Призначення розробки

Експлуатаційне призначення:

– інформування про концерти та культурні події;

– зменшення пікового навантаження на фізичну касу;

– облік бронювань і заповненості по кожній події;

– збір відгуків і звернень через чат;

– формування іміджу сучасного залу до завершення будівництва.

Функціональне призначення охоплює такі групи можливостей.

а) Публічний інформаційний модуль: головна сторінка; афіша /events; деталі події /events/[id]; розділи /about (схема залу), /faq, /contacts, /reviews.

б) Модуль бронювання: сторінка /events/[id]/seats; API seats і hold; checkout; success; PDF /api/bookings/[id]/ticket.

в) Модуль користувача: /register, /login, /profile (список бронювань, скасування).

г) Модуль підтримки: віджет чату, API /api/support/chats, збереження в БД.

д) Адміністративний модуль: /admin, /admin/events, /admin/bookings, /admin/chats, /admin/reviews, /admin/users; upload постерів /api/admin/upload.

Межі прототипу: відсутня онлайн-оплата; відсутня інтеграція з фіскальним реєстратором; SMS-сповіщення не реалізовані; одна мова інтерфейсу.

1.2.3 Вимоги до функціоналу web-сайту

Вимоги сформульовано без прив’язки до внутрішнього коду, з точки зору спостерігача-системи [10].

Вхідні дані та дії користувача:

- перегляд списку подій (дані з таблиці Event);
- вибір місць на схемі (ідентифікатори місць Seat, статуси EventSeat);
- введення ПІБ, email, телефону (опційно) на checkout;
- реєстрація (email, пароль, ім’я);
- авторизація (email, пароль);
- відправка повідомлення в чат; публікація відгуку (ім’я, текст, оцінка 1–5);
- дії адміна: створення/редагування події, завантаження зображення, відповідь у чаті, редагування/видалення відгуку, перегляд бронювань.

Вихідні дані:

HTML-сторінки; JSON від API; PDF-квиток; email з квитком (якщо

налаштовано SMTP у .env); оновлення записів у SQLite.

Деталізація функціональних вимог:

а) Афіша: сортування подій за датою; відображення мінімальної ціни, виконавця, постера; перехід на бронювання.

б) Схема: масштабування (коліщатко, кнопки); перетягування поля схеми; легенда цін; підписи секторів; блокування sold/held.

в) Hold і checkout: не більше 8 місць за одне бронювання; утримання перед підтвердженням; розрахунок суми.

г) Бронювання: унікальний номер (VRZ-...); статус confirmed/cancelled; зв’язок з userId або guestEmail.

д) PDF: назва події, дата, час, список місць, сума, QR.

е) Кабінет: скасування з поверненням місць у available.

ж) Адмінка: CRUD подій; перегляд статистики; лічильник відкритих чатів у меню.

з) Відгуки: публічний список; відповідь адміністратора у записі Review.

и) Постери: завантаження JPEG/PNG/WebP/GIF до 5 МБ; збереження URL у Event.imageUrl; відображення на сайті.

Вимоги до надійності:

– валідація Zod на API; bcrypt для паролів; middleware на /admin; транзакція при бронюванні; захист від подвійного продажу місця; зрозумілі повідомлення про помилки українською; basic захист адмін-API (requireAdmin) [9], [17].

Умови експлуатації:

– клієнт: Chrome, Firefox, Edge останніх версій; екран від 360 px ширини;
– сервер: Node.js 20+, 2 ГБ RAM мінімум для прототипу;
– ОС: Windows 10/11 або Linux на VPS;
– мережа: HTTPS у production (рекомендація розділу 3).

Часові характеристики:

– відклик сторінки афіші до 3 с на локальному сервері;

– відповідь API seats до 2 с;

– генерація PDF до 10 с;

– чат: polling 3 с (не real-time WebSocket у прототипі).

Навантаження: прототип розрахований на демонстраційне навантаження (десятки одночасних користувачів); для сотень - потрібен production-хостинг і PostgreSQL [11].

1.2.4 Вимоги до програмної документації

Склад документації:

– завдання на кваліфікаційну роботу (окремий документ);

– анотація (1 сторінка);

– пояснювальна записка (ця робота);

– коментарі в коді TypeScript;

– README (інструкції npm install, db:setup, npm run dev, облікові записи після seed);

– розділ 3 записки (експлуатаційні інструкції);

– графічна частина А1 (ER, UML use case, карта сайту - за завданням на КР) [2].

Вимоги до якості коду:

– модульна структура;

– єдиний стиль іменування;

– секрети в .env;

– окремі store-модулі (events-db, chats-store, reviews-store);

– повторне використання HallMapSvg для схеми на /about і при бронюванні.

1.2.5 Техніко-економічні показники

Орієнтовні трудовитрати автора:

– аналітика, огляд, ТЗ - 16 год;
– проєктування БД, API, схеми місць - 11 год;
– верстка та клієнтські компоненти - 40 год;
– адмін-панель - 8 год;
– тестування, виправлення - 15 год;
– оформлення записки, графіка - 30 год.

Разом близько 120 людино-годин.

Машинний час: розробка на ПК автора; збірки Next.js; локальна SQLite.

Грошова оцінка розробки та експлуатації наводиться в розділі 4. Тут зафіксовано лише ресурсний базис для ТЗ.

1.2.6 Стадії та етапи розробки

Роботи над веб-сайтом бронювання квитків для концертного залу «Віраж» організовано у вигляді послідовних стадій життєвого циклу програмного забезпечення, що відповідає рекомендаціям до кваліфікаційних робіт з розробки веб-сайтів [2]. Такий поділ дозволяє простежити перехід від аналізу предметної області до діючого прототипу та подальшого планового розміщення в мережі Internet.

Підготовча стадія охоплює період практики та початкового аналізу. На цьому етапі уточнено особливості майбутнього концертного залу в м. Тернопіль (місткість 420 місць, орієнтовне відкриття 2030 р.), погоджено формулювання теми кваліфікаційної роботи та зібрано вимоги до цифрового сервісу. Проаналізовано роботу національних агрегаторів Concert.ua і Karabas Live, а також окремих закордонних платформ, що дало змогу сформулювати критерії порівняння та обґрунтувати доцільність власного сайту замість лише зовнішніх каналів продажу [7], [8], [18]. У результаті підготовчої стадії сформовано перелік функцій (афіша, інтерактивна схема, бронювання, PDF-квиток, кабінет користувача, чат підтримки, відгуки, адміністративна панель), який ліг в основу підрозділу 1.1 і подальшого технічного завдання.

Стадія проєктування передбачала перехід від словесного опису вимог до формальних моделей. Розроблено логічну ER-модель даних із сутностями користувача, події, місця в залі, прив'язки місця до події, бронювання, відгуку та чату підтримки. Визначено структуру веб-сторінок і серверних маршрутів: публічна частина (головна, афіша, картка події, схема, оформлення замовлення), особистий кабінет, розділи «Про зал», FAQ, контакти, відгуки, а також закрита зона /admin. Обрано програмний стек Next.js, React, TypeScript, Prisma ORM, SQLite та NextAuth як збалансоване рішення для full-stack прототипу з єдиним репозиторієм [3], [5], [6]. Окремо спроєктовано алгоритм тимчасового утримання місць (hold) і транзакційного підтвердження броні, щоб уникнути подвійного продажу одного місця. Результати проєктування відображені в схемі Prisma, технічному завданні (п. 1.2) та графічних додатках (ER-діаграма, діаграма прецедентів, карта сайту).

Стадія реалізації була найтривалішою за трудовитратами. Створено проєкт на базі Next.js із маршрутизацією App Router. У базу даних завантажено 420 місць відповідно до конфігурації залу (партер, амфітеатр, ложі, бенуар). Реалізовано публічні сторінки з українським інтерфейсом, компонент інтерактивної схеми з масштабуванням і прокруткою, серверні API для отримання статусів місць, утримання вибору та створення бронювання. Додано сторінки оформлення замовлення та підтвердження, генерацію PDF-квитка з QR-кодом [14], реєстрацію та автентифікацію з хешуванням паролів [17]. Адміністративна частина дозволяє вести афішу, переглядати бронювання, відповідати в чатах і модерувати відгуки, завантажувати власні постери подій. На сторінці контактів інтегровано карту Google Maps із позначенням майбутньої адреси майданчика. Підсумком реалізації став репозиторій virazh, який збирається без критичних помилок (npm run build) і демонструється локально (npm run dev).

Стадія тестування виконана згідно з п. 1.2.7. Перевірено відповідність функцій вимогам п. 1.2.3, коректність збереження даних у SQLite після перезапуску сервера, роботу схеми на різних ширинах екрана та реакцію системи на помилкові дії користувача. Виявлені невідповідності (наприклад, колізії між

статусами held і sold, спроби доступу до адмін-панелі без ролі admin) усувалися в ході кількох ітерацій. Підсумкові результати випробувань будуть систематизовані в п. 2.5 розділу 2.

Стадія документування триває паралельно з завершенням розробки. Готується пояснювальна записка за структурою методичних вказівок [2], анотація, файл README для відтворення середовища, коментарі в коді, презентація для захисту та графічна частина. Ця стадія не змінює логіку прототипу, але фіксує хід робіт для комісії.

Планова стадія розміщення передбачена після захисту кваліфікаційної роботи. Передбачається перенесення застосунку на хостинг (Vercel або віртуальний сервер), налаштування доменного імені, протоколу HTTPS і змінних середовища production; у перспективі можливе заміщення SQLite на PostgreSQL для підвищення навантажувальної здатності [16]. Докладний порядок дій описуватиметься в розділі 3. На момент презентації комісії достатньо локальної або тимчасової демонстраційної копії.

За потреби виконувалося повернення до попередніх стадій: наприклад, уточнення моделі відгуків після підключення адміністративного інтерфейсу або коригування API бронювання після перших інтеграційних перевірок. Такий ітеративний підхід є типовим для навчальних веб-проєктів і не суперечить затвердженому технічному завданню.

1.2.7 Порядок тестування та прийому

Тестування програмного прототипу «Віраж» проводиться з метою встановити відповідність реалізованих функцій вимогам п. 1.2.3, перевірити цілісність даних у базі SQLite та підтвердити готовність продукту до демонстрації на захисті кваліфікаційної роботи. Прийом прототипу за результатами випробувань здійснюється за критеріями, наведеними нижче; остаточна перевірка відповідності технічному завданню виконується керівником на захисті.

До функціонального тестування належить перевірка сценаріїв звичайного

відвідувача та адміністратора. Для відвідувача це перегляд головної сторінки та афіші, відкриття картки події, робота з інтерактивною схемою (вибір вільних місць, неможливість вибору вже проданих або утриманих), оформлення бронювання без реєстрації або після входу в обліковий запис, отримання PDF-квитка, публікація відгуку, обмін повідомленнями в чаті підтримки. Для адміністратора - створення та редагування подій, завантаження власного зображення постера, перегляд списку бронювань, відповіді в чаті, редагування відгуків і робота з обліковими записами в межах адмін-панелі.

Інтеграційне тестування спрямоване на ланцюги «веб-інтерфейс - серверний API - Prisma - файл бази даних». Особлива увага приділяється операціям утримання місць (hold), підтвердження бронювання в транзакції, скасуванню броні з поверненням місць у стан «доступно», збереженню повідомлень чату та відгуків після перезавантаження сервера розробки. Саме на цьому рівні виявляються помилки, які не видно при поверхневому перегляді сторінок.

Тестування користувацького інтерфейсу включає перевірку відображення на екрані ноутбука (ширина близько 1920 пікселів) та на мобільному емуляторі (ширина близько 390 пікселів). Оцінюється читабельність легенди цін на схемі, зручність масштабування та прокрутки залу, доступність основних кнопок на сторінках checkout і в адміністративному меню.

Негативне тестування передбачає навмисні некоректні дії: спроба завершити бронювання на вже продане місце, вхід користувача з роллю user на адресу /admin, надсилання повідомлення в уже закритий чат, завантаження зображення постера об’ємом понад установлений ліміт (5 МБ). Очікувана реакція системи - відмова в операції та зрозуміле повідомлення українською мовою без пошкодження даних інших користувачів.

Перед початком випробувань готується однакове середовище: встановлюється Node.js версії 20 або новішої, виконуються команди встановлення залежностей і ініціалізації бази (npm install, npm run db:setup), перевіряються тестові облікові записи з початкового наповнення (користувач і адміністратор). Сервер розробки запускається командою npm run dev; робота ведеться в браузері

за адресою локального хоста.

Порядок проходження сценаріїв побудовано від простішого до складнішого. Спочатку перевіряються публічні сторінки без авторизації: наявність подій на головній та в афіші, коректність картки події, робота схеми залу. Далі виконується повний цикл бронювання гостем із завантаженням PDF-квитка. Потім - реєстрація нового або вхід існуючого користувача, бронювання під обліковим записом і скасування одного з замовлень у профілі. Наступним блоком є адміністративні операції (нова подія, завантаження постера, відповідь у чаті та на відгук). Завершальним кроком є збірка production-версії (npm run build) без помилок компіляції TypeScript.

Контрольний перелік сценаріїв охоплює, зокрема: відкриття головної та афіші; перегляд обраної події; вибір місць і зміну масштабу схеми; бронювання гостем із отриманням номера замовлення формату VRZ; завантаження PDF; реєстрацію, вхід і бронювання авторизованим користувачем; скасування броні; відмову при спробі зайняти продане місце; публікацію відгуку; діалог у чаті з відповіддю адміністратора; створення події з власним постером; відповідь на відгук у публічному списку. Кожен сценарій вважається пройденим, якщо фактична поведінка сайту збігається з очікуванням, сформульованим у технічному завданні.

Критерії прийому прототипу такі: усі перелічені сценарії виконуються успішно; після перезапуску сервера збережені бронювання, відгуки та історія чатів не втрачаються; PDF-квиток відкривається і містить читабельний QR-код; користувач без адміністративних прав не отримує доступ до закритих маршрутів; повідомлення про помилки відображаються українською. Після суттєвих змін у програмному коді доцільно повторювати найбільш ризикові сценарії - повне бронювання гостем і створення нової події з завантаженням зображення, оскільки саме вони зачіпають найбільше сутностей бази даних.

2 РОЗРОБКА ТЕХНІЧНОГО ТА РОБОЧОГО ПРОЄКТУ

Другий розділ пояснювальної записки присвячено проєктуванню та опису веб-сайту бронювання квитків для концертного залу «Віраж». У ньому формулюються рішення щодо побудови застосунку, організації даних, інтерфейсу користувача та перевірки працездатності розробленого програмного засобу.

Опис у цьому розділі ґрунтується на вимогах, сформульованих у технічному завданні, і показує, як загальні положення першого розділу переходять у конкретні проєктні та програмні рішення.

2.1 Розробка структури сайту і web-сторінок

Архітектурний підхід. Програмний прототип побудовано як монолітний full-stack застосунок на базі Next.js з каталогом app (App Router). У одному репозиторії поєднано серверний рендеринг HTML-сторінок, клієнтську інтерактивність (схема залу, форми, чат) і REST-подібні API-маршрути. Такий підхід зменшує кількість окремих сервісів, спрощує розгортання на захисті кваліфікаційної роботи та відповідає сучасній практиці розробки корпоративних веб-застосунків [3], [4].

Логічна структура каталогів репозиторію така: app - маршрути сторінок (page.tsx) і серверні API (route.ts у підкаталозі api); components - React-компоненти інтерфейсу (layout, seating, admin, support тощо); lib - бізнес-логіка, доступ до БД, утиліти (seats.ts, events-db.ts, ticket-pdf.ts); prisma - схема БД (schema.prisma) і скрипт початкового наповнення (seed.ts); public - статичні файли (завантажені постери в public/posters/).

Каталог app містить маршрути App Router (публічні сторінки, /admin, app/api); components - перевикористовувані React-компоненти; lib - бізнес-логіка та утиліти; prisma - декларативна схема БД і seed; public - статичні ресурси та завантажені постери. Такий поділ відповідає рекомендаціям Next.js щодо colocation коду [3]. Структура репозиторію virazh наведена на рисунку 2.1.

Рисунок 2.1 – Структура каталогів репозиторію virazh

Ролі користувачів і сценарії доступу. Система розрізняє три категорії учасників: анонімний відвідувач (гість), зареєстрований користувач (role user у таблиці User) та адміністратор (role admin). Гість може переглядати публічні сторінки, бронювати місця, завантажувати PDF-квиток, писати в чат підтримки та залишати відгук. Авторизований користувач додатково отримує особистий кабінет з історією бронювань і можливістю скасування. Адміністратор працює в закритій зоні /admin; доступ перевіряється middleware і layout адмін-панелі [6], [17].

Карта сайту. Структуру веб-сайту та основні URL подано на структурній схемі в графічній частині кваліфікаційної роботи. У текстовому вигляді карта сайту охоплює такі маршрути.

Кореневий маршрут / - головна сторінка з презентацією залу та фрагментом афіші. Розділ /events - повний список подій з пошуком і фільтром за категорією. Динамічний маршрут /events/[id] - картка конкретної події; /events/[id]/seats -

інтерактивна схема залу для цієї події. Оформлення замовлення виконується на /checkout (після вибору місць) і завершується на /booking/success.

Інформаційні сторінки: /about (про зал, технічні характеристики, схема залу), /faq, /contacts (контакти та карта Google Maps), /reviews (відгуки та форма додавання). Авторизація: /login, /register; особистий кабінет - /profile.

Закрита зона адміністратора має маршрути: /admin (дашборд), /admin/events, /admin/bookings, /admin/chats, /admin/reviews, /admin/users. Усі вони успадковують спільний layout з бічним меню AdminSidebar і верхньою панеллю AdminTopBar..

Взаємодію користувачів із веб-сайтом «Віраж» подано на діаграмі варіантів використання (UML). На ній виділено основні ролі - відвідувач і адміністратор - та типові дії: перегляд афіші, вибір місць, оформлення бронювання, робота з профілем, адміністрування подій і бронювань. Графічне подання діаграми наведено в графічній частині кваліфікаційної роботи

Серверні API-маршрути. Клієнтська частина обмінюється даними з сервером через маршрути в app/api. Публічні та користувацькі: GET /api/events - список подій; GET /api/events/[id] - деталі події; GET /api/events/[id]/seats - місця зі статусами для події; POST /api/events/[id]/seats/hold - тимчасове утримання місць; POST /api/bookings - підтвердження бронювання, генерація PDF; GET /api/bookings/[id] - дані броні; PATCH /api/bookings/[id] - скасування (користувач або адмін); GET /api/bookings/[id]/ticket - завантаження PDF-квитка; POST /api/auth/register - реєстрація; GET/POST /api/auth/[...nextauth] - сесія NextAuth; GET /api/auth/config - чи налаштовано Google OAuth; PATCH /api/user/profile - оновлення імені, телефону, мови, теми; GET/POST /api/reviews - список і створення відгуків; GET/POST /api/support/chats - чат відвідувача; POST /api/support/chat - повідомлення в чат; POST /api/support/chats/[id]/close - закриття чату відвідувачем; GET /api/poster - резервний генеративний постер за id події; GET /api/support/config - параметри підтримки.

Адміністративні (з перевіркою requireAdmin): GET/POST /api/admin/events; PATCH/DELETE /api/admin/events/[id]; GET /api/admin/chats, GET /api/admin/chats/[id]; POST /api/admin/chats/[id]/reply, POST

/api/admin/chats/[id]/close; GET /api/admin/reviews, PATCH/DELETE /api/admin/reviews/[id]; POST /api/admin/upload - завантаження зображення постера. Перелік основних серверних API-маршрутів узагальнено в таблиці 2.1.

Таблиця 2.1 – Основні API-маршрути веб-сайту «Віраж»

Метод	Endpoint	Доступ	Призначення
GET	/api/events	Публічний	Список подій афіші
GET	/api/events/[id]	Публічний	Деталі події
GET	/api/events/[id]/seats	Публічний	Місця зі статусами
POST	/api/events/[id]/seats/hold	Публічний	Тимчасове утримання місць (15 хв)
POST	/api/bookings	Публічний / user	Підтвердження бронювання
GET	/api/bookings/[id]/ticket	Публічний / user	Завантаження PDF-квитка
PATCH	/api/bookings/[id]	user / admin	Скасування броні
POST	/api/auth/register	Публічний	Реєстрація користувача
GET / POST	/api/auth/[...nextauth]	Публічний	Сесія NextAuth
GET / POST	/api/reviews	Публічний	Відгуки
GET / POST	/api/support/chats	Публічний	Чат підтримки
GET / POST	/api/admin/events	Admin	Керування подіями
POST	/api/admin/upload	Admin	Завантаження постера
GET	/api/admin/chats	Admin	Список чатів
PATCH	/api/admin/reviews/[id]	Admin	Відповідь на відгук

Усі адміністративні маршрути перевіряють роль admin через функцію requireAdmin.

Потік даних при бронюванні. Типовий сценарій охоплює ланцюг: сторінка seats (клієнт) → GET seats (сервер читає EventSeat і Seat з Prisma) → POST hold (оновлення status=held, heldUntil) → сторінка checkout → POST bookings (транзакція: status=sold, створення Booking, generateTicketPdf, опційно sendTicketEmail). Номер замовлення формується як VRZ- з додаванням мітки часу в base36. Сума обчислюється як сума basePrice обраних місць. Ідентифікатори

місць у записі Booking зберігаються в полі seats як JSON-рядок [5], [11].

Захист маршрутів. Файл middleware.ts на базі NextAuth перехоплює запити до /admin і /admin/*. Неавторизований користувач перенаправляється на /login з параметром callbackUrl. Користувач без ролі admin перенаправляється на /profile?denied=admin. Конфігурація matcher обмежує middleware лише адмін-зоною, щоб не уповільнювати публічні сторінки [6].

Уточнення щодо поділу серверної та клієнтської логіки. Сторінки, де дані з БД не змінюються від дій користувача на клієнті (головна, about, картка події до переходу на схему), реалізовано як серверні компоненти React: запит до Prisma виконується на сервері під час рендерингу, HTML надходить клієнту вже з наповненим вмістом. Це покращує початкове завантаження та сприяє індексації афіші пошуковими системами [3]. Сторінки зі складною взаємодією (схема місць, списки з фільтрами, форми, чат, адмін-форми) позначено директивою use client і працюють у браузері, викликаючи API fetch. Критичні перевірки (статус місця, роль admin, валідація Zod) залишаються на сервері в route.ts [3], [9].

2.2 Створення та верстка сторінок сайту

Технології верстки. Інтерфейс реалізовано з бібліотекою React 19 і стилізацією Tailwind CSS версії 4 [13]. Глобальні змінні кольорів і тем оголошено в app/globals.css (світла та темна тема через атрибут data-theme на елементі html). Шрифти підключено через next/font/google: DM Sans для основного тексту, Playfair Display для заголовків (клас font-display). Іконки - пакет lucide-react. Такий набір забезпечує сучасний вигляд без важких UI-фреймворків і зберігає контроль над адаптивністю [4].

Загальний каркас сторінок. Кореневий layout.tsx обгортає всі публічні сторінки в структуру: Header (шапка з навігацією), main (вміст), Footer (підвал). Компонент AppProviders інкапсулює SessionProvider (NextAuth), AppSettingsProvider (мова uk/en і тема light/dark з localStorage) та плаваючий віджет SupportChat. Мова сторінки за замовчуванням uk; перемикач

LocaleThemeControls дозволяє змінити тему та мову інтерфейсу без перезавантаження всього застосунку.

Публічні сторінки та їхній зміст. Головна (/) - серверний компонент: блок hero з назвою залу, містом, плановим відкриттям 2030 р., місією з модуля hall.ts; кнопки переходу на афішу та «Про зал»; сітка з чотирьох карток EventCard (дані з getAllEvents); секції HowItWorks (кроки бронювання), ReviewsSection (відгуки з БД), заклик до дії. Сторінка має dynamic = force-dynamic для актуальної афіші з SQLite.

Афіша (/events) - клієнтський компонент EventsList: поле пошуку, фільтр категорій (Усі, Естрада, Рок тощо), сітка карток подій. Кожна картка EventPoster показує зображення, назву, виконавця, дату, мінімальну ціну.

Картка події (/events/[id]) - опис події, бейдж категорії, кнопка «Обрати місця» з переходом на seats.

Схема залу (/events/[id]/seats) - SeatsPageClient з компонентами HallMap, BookingPanel, PriceLegend, SeatLegend. Схема рендериться в SVG (HallMapSvg) з координатами з БД; підтримуються масштабування, перетягування поля перегляду, обмеження до 8 місць за одне бронювання.

Checkout (/checkout) -- форма ПІБ, email, телефон; підсумок обраних місць; кнопка підтвердження з викликом POST /api/bookings.

Success (/booking/success) - підтвердження номера замовлення, посилання на завантаження PDF.

Про зал (/about) - текст з hall.ts, технічні характеристики (місткість 420, сектори, акустика), інтерактивна HallMapPreview.

FAQ (/faq) - FaqAccordion з питаннями з mock-data / БД.

Контакти (/contacts) - адреса, телефон, email, iframe Google Maps (hallMapEmbedUrl).

Відгуки (/reviews) - список ReviewCard і форма ReviewForm (POST /api/reviews).

Login і Register - форми LoginForm, RegisterForm; опційна кнопка GoogleSignInButton за наявності GOOGLE_CLIENT_ID у .env.

Профіль (/profile) - ProfileSettings (ім’я, телефон), список BookingCard з кнопкою скасування. Зовнішній вигляд ключових сторінок узагальнено на рисунку 2.2.

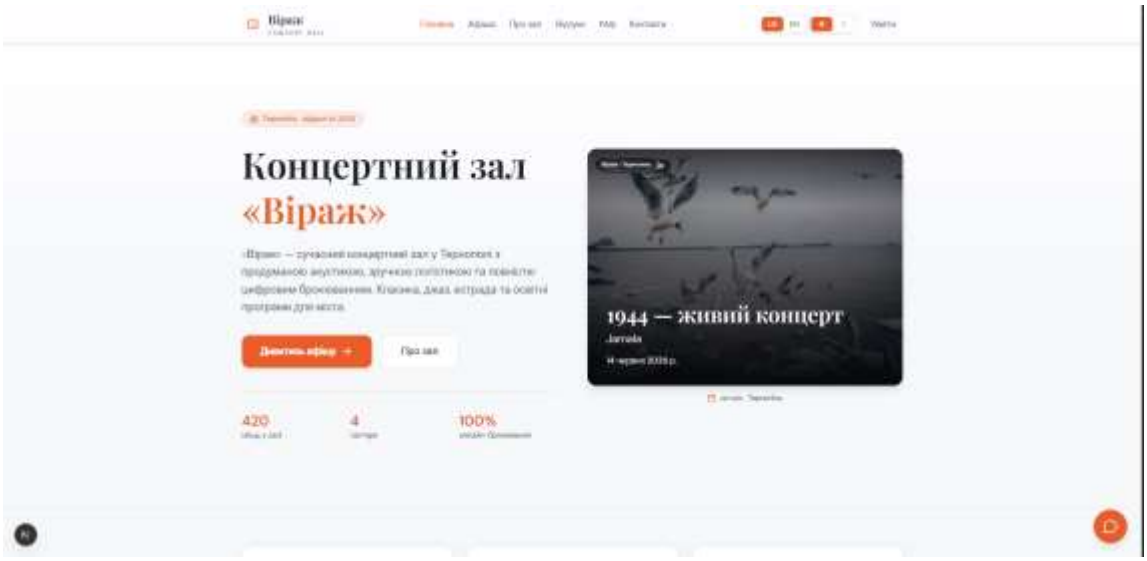


Рисунок 2.2 – Головна сторінка веб-сайту «Віраж»

Сторінка афіші (/events) містить поле пошуку, фільтр за категорією та сітку карток подій з постером, датою і мінімальною ціною. Після переходу на картку події (/events/[id]) відвідувач бачить опис, категорію та кнопку «Обрати місця». Зовнішній вигляд цих сторінок показано на рисунках 2.3 і 2.5.

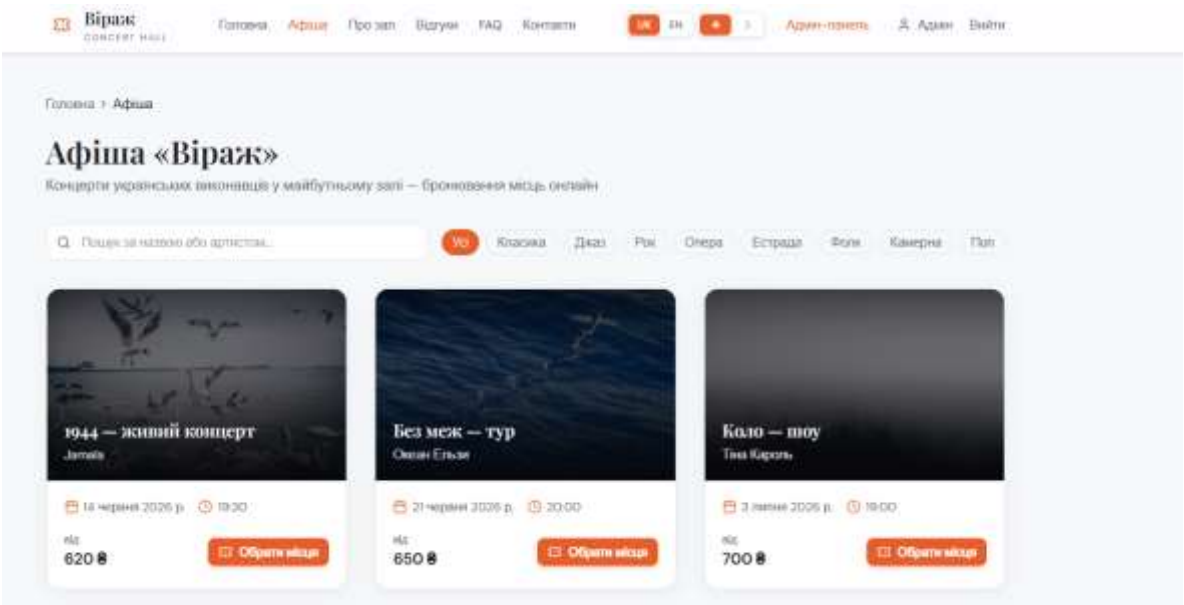


Рисунок 2.4 – Сторінка афіші подій (/events)

Адаптивність і доступність. Верстка побудована на flex- і grid-сітках Tailwind з breakpoints sm, lg. Мінімальна ширина viewport для комфортної роботи - 360 px (відповідає п. 1.2.3). Контраст кольорів місць на схемі та кнопок дій перевірено візуально; орієнтир для подальшого вдосконалення - рекомендації WCAG 2.1 [15]. Темна тема змінює фон, картки та текст через CSS-змінні без дублювання розмітки.

На ширині viewport близько 390 px (мобільний емулятор у Chrome DevTools) навігація згортається в компактне меню, картки подій розташовуються в один стовпець, а схема залу залишається прокручуваною з кнопками масштабу. Приклад відображення головної сторінки на мобільному пристрої наведено на рисунку 2.5.

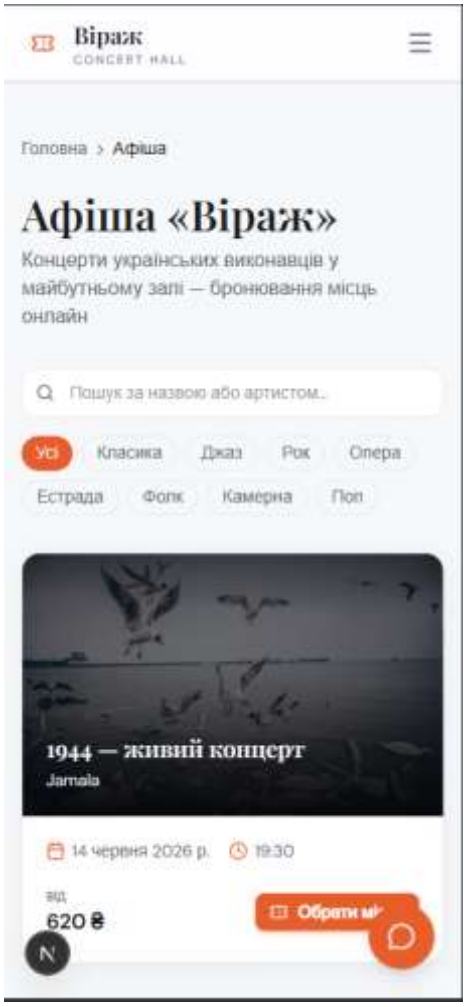


Рисунок 2.5 – Адаптивне відображення головної сторінки (ширина 390 px)

Компоненти повторного використання. Для уніфікації інтерфейсу виділено каталог components/ui: Button (варіанти primary, secondary, ghost, danger), Badge, Breadcrumbs. Картки подій і бронювань використовують спільні стилі карток. EventPoster інкапсулює логіку відображення зображення афіші з fallback. CheckoutStepper візуалізує етапи «Подія → Місця → Оформлення».

Сторінка помилки error.tsx обробляє непередбачені збої рендерингу в межах app і показує користувачу зрозуміле повідомлення замість порожнього екрана. Це відповідає вимозі надійності з п. 1.2.3 щодо інформування про помилки українською мовою.

2.3 Розробка структури бази даних сайту

СУБД і ORM. Для прототипу обрано SQLite - файлову реляційну базу, що не вимагає окремого сервера БД на етапі розробки та демонстрації на захисті [11]. Доступ до даних реалізовано через Prisma ORM: декларативна схема в prisma/schema.prisma, клієнт @prisma/client, міграції через prisma db push [5]. Рядок підключення DATABASE_URL задається в .env (типово file:./dev.db).

Концептуальна модель. База даних відображає предметну область концертного залу: один зал (Hall), фіксований набір місць (Seat) з координатами та ціною, події (Event), прив’язка статусу місця до події (EventSeat), бронювання (Booking), користувачі та сесії автентифікації (User, Account, Session, VerificationToken), відгуки (Review), чати підтримки (SupportChat, SupportChatMessage). ER-діаграма наведена в графічній частині кваліфікаційної роботи.

Опис основних сутностей. Hall - зал «Віраж» (id = virazh), місто Тернопіль; зв’язок один-до-багатьох з Seat і Event. Seat - місце в залі: section (партер, амфітеатр, ложі, бенуар), row, number, координати x/y для SVG, basePrice. Унікальність за комбінацією hallId + section + row + number. Усього 420 записів після seed. Event - афішна подія: title, artist, description, date, time, category, minPrice, availableSeats, imageUrl (опційно), hallId. EventSeat - статус місця на

конкретній події: available, held (з heldUntil), sold. Booking - замовлення: id формату VRZ-…, eventId, userId або guestName/guestEmail, total, status, paymentNote, seats (JSON-масив id місць). User - обліковий запис з email, passwordHash (bcrypt), role (user/admin). Review - відгук з rating 1–5 та adminReply. SupportChat і SupportChatMessage - діалог підтримки. Основні поля таблиць бази даних наведено в таблиці 2.2.

Таблиця 2.2 – Основні поля таблиць бази даних

Сутність	Поля	Опис
Hall	id (PK)	Ідентифікатор залу «virazh»
Hall	name, city	Назва залу та місто
Seat	id (PK), hallId (FK)	Місце в залі, прив’язка до залу
Seat	section, row, number	Сектор, ряд, номер місця
Seat	x, y, basePrice	Координати SVG та базова ціна
Event	id (PK), hallId (FK)	Подія в афіші
Event	title, artist, date, time, category	Опис події
Event	minPrice, availableSeats, imageUrl	Ціна, доступність, постер
EventSeat	eventId (FK), seatId (FK), status, heldUntil	Статус місця на події
Booking	id (PK), eventId (FK), userId (FK)	Номер бронювання VRZ-…, подія, користувач
Booking	guestName, guestEmail, total, status, seats	Дані гостя, сума, статус, місця (JSON)
User	email, passwordHash, role	Обліковий запис користувача та роль
Review	name, text, rating, adminReply	Відгук користувача та відповідь адміністратора
SupportChat	visitorId, status	Чат відвідувача
SupportChatMessage	chatId (FK), role, content	Повідомлення в чаті

Таким чином, структура БД покриває всі сутності, зазначені в технічному завданні.

Генерація місць і початкове наповнення. Координати та ціни 420 місць обчислюються функцією buildHallSeatTemplate() у модулі seats.ts (секції parterre, amphitheater, amphitheater_l/r, box_dress_l/r, box_amph_l/r, benoir). Скрипт

prisma/seed.ts створює зал, усі місця, чотири демонстраційні події, по 12 проданих місць на подію, відгуки, користувача demo@virazh.local і адміністратора admin@virazh-hall.ua. Команда npm run db:setup виконує db push і seed послідовно.

Нормалізація та цілісність даних. Модель відповідає третій нормальній формі в межах прототипу: атрибути місця зберігаються в Seat, а EventSeat містить лише статус для конкретної події. Зовнішні ключі забезпечують посилкову цілісність; onDelete: Cascade для EventSeat і SupportChatMessage запобігає «сиротським» записам. Унікальні індекси на email користувача виключають дублювання облікових записів [5], [11].

2.4 Програмування сайту

Програмна реалізація поділена на клієнтську частину (інтерфейс відвідувача) та адміністративну частину (керування контентом і замовленнями). Серверна логіка зосереджена в API-маршрутах app/api і серверних компонентах сторінок, що безпосередньо звертаються до Prisma.

2.4.1 Написання клієнтської частини

Інтерактивна схема залу. Центральний компонент HallMap (клієнтський, use client) керує станом масштабу (від 0.55 до 2.2), зміщенням поля перегляду та множиною обраних місць selectedIds. Дочірній HallMapSvg малює SVG viewBox 1000×860: підписи секторів (SECTION_LABELS), кола місць з кольором за ціною (seat-pricing.ts) та за статусом (вільне, held, sold, обране). Клік по вільному місцю додає або прибирає його з вибору; sold і held не підлягають вибору. BookingPanel показує список обраних місць у форматі «Амфітеатр, ряд 5, місце 13», суму та кнопку «Продовжити», яка викликає POST hold і перенаправляє на checkout.

Модуль seats.ts експортує HALL_SEAT_COUNT = 420, константи VIEW_W, VIEW_H, SECTION_LABELS. Компонент HallMapPreview на сторінці /about повторно використовує HallMapSvg у спрощеному режимі без бронювання.

Послідовність операцій при бронюванні (від перегляду події до отримання квитка) показано на рисунку 2.6.

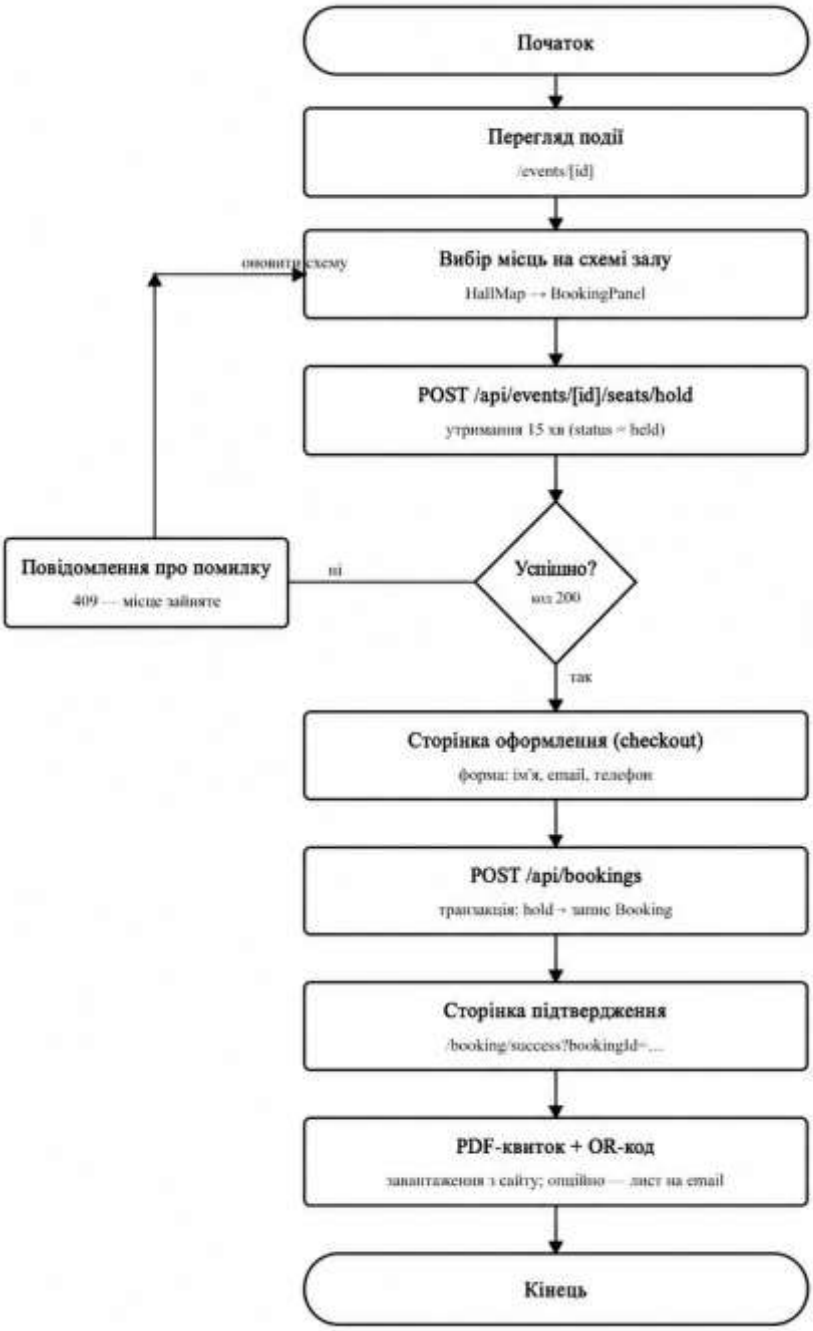


Рисунок 2.6 – Схема процесу бронювання місць

Після натискання «Продовжити» клієнт надсилає запит hold: обрані місця отримують статус held на 15 хвилин, після чого відкривається сторінка checkout. Форма збирає ім’я, email і телефон (для зареєстрованого користувача поля частково підставляються з сесії). Підтвердження викликає POST /api/bookings: у

транзакції місця переводяться в sold, у таблицю Booking записується бронь із JSON-списком seatIds. Користувач переходить на /booking/success, де доступне завантаження PDF-квитка.

Оформлення бронювання та квиток. PDF формує функція generateTicketPdf (pdf-lib): заголовок VIRAZH CONCERT HALL, дані події, список місць, QR-код з bookingId (qrcode) [14]. Маршрут GET /api/bookings/[id]/ticket повертає application/pdf. За наявності SMTP у .env модуль email.ts надсилає лист із вкладенням; інакше квиток доступний лише через завантаження з сайту.

Сторінка /events/[id]/seats поєднує інтерактивну SVG-схему залу та бічну панель бронювання. Користувач може змінювати масштаб і переміщати поле перегляду; легенда пояснює кольори за ціною та статусом місця. У панелі відображаються обрані позиції, підсумкова сума та таймер утримання. Таким чином, ключовий етап вибору місць реалізовано як єдиний екран без переходу на окрему сторінку до підтвердження hold. Екран вибору місць наведено на рисунку 2.7.

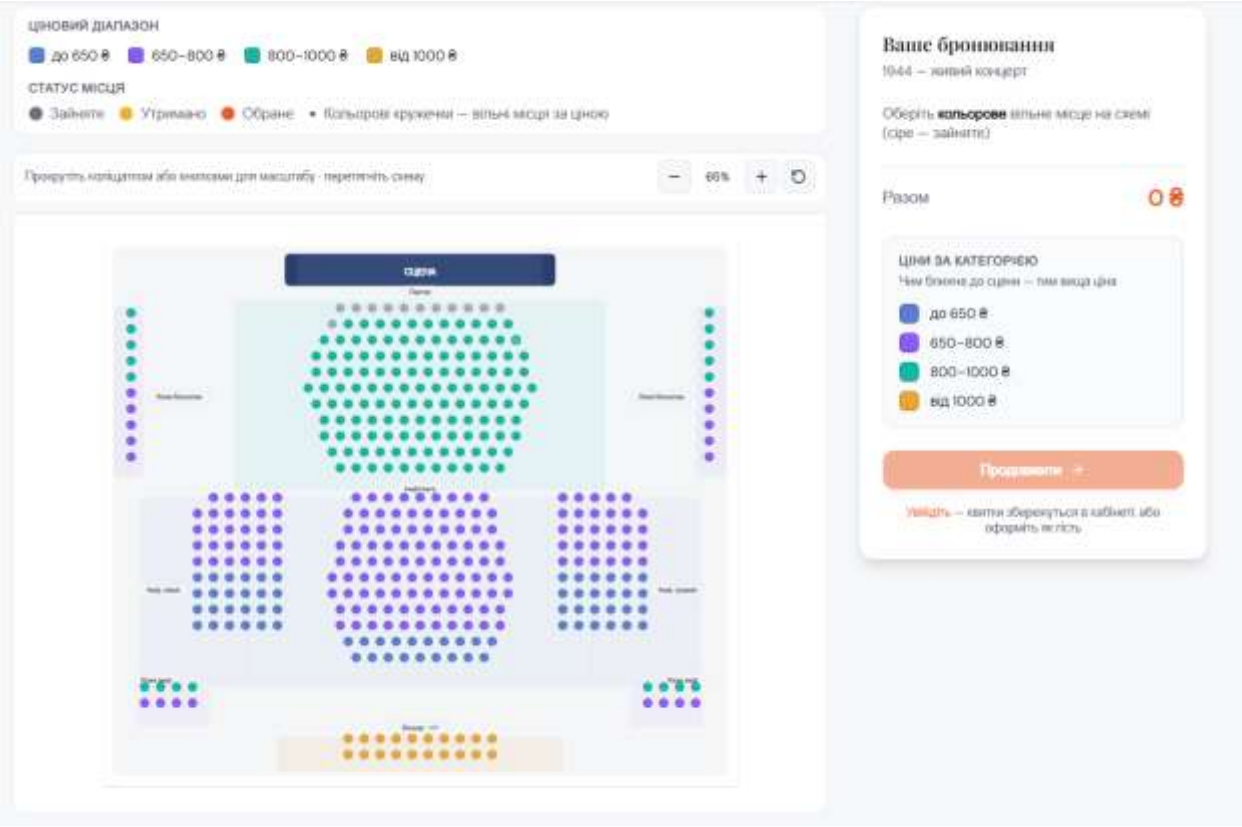


Рисунок 2.7 – Екран вибору місць

Сторінка оформлення замовлення (/checkout) відображає підсумок обраних місць, контактні поля (ПІБ, email, телефон) та кнопку підтвердження бронювання. Інтерфейс цього етапу наведено на рисунку 2.8 і 2.9; згенерований квиток з QR-кодом - на рисунку 2.10.

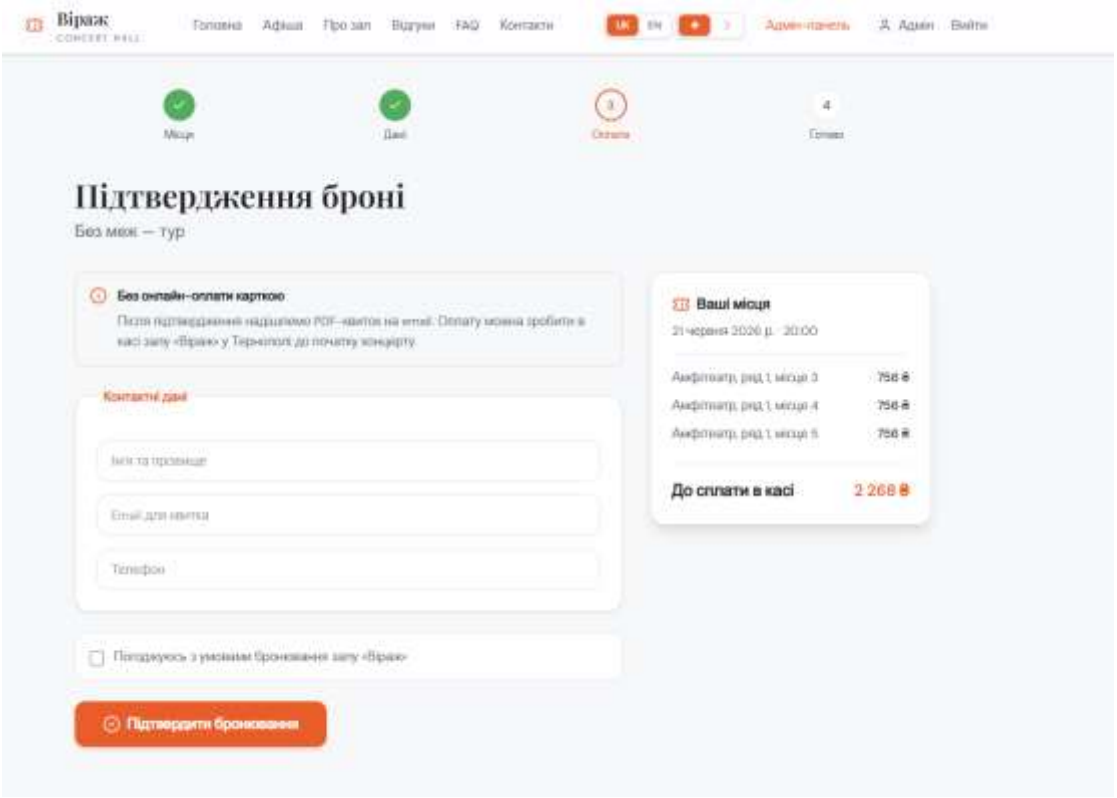


Рисунок 2.8 – Сторінка оформлення бронювання (/checkout)

Після успішного POST /api/bookings користувач переходить на /booking/success, де показано номер замовлення формату VRZ-… і посилання на завантаження PDF-квитка. Інтерфейс цих етапів наведено на рисунку 2.9; згенерований квиток з QR-кодом - на рисунку 2.10.

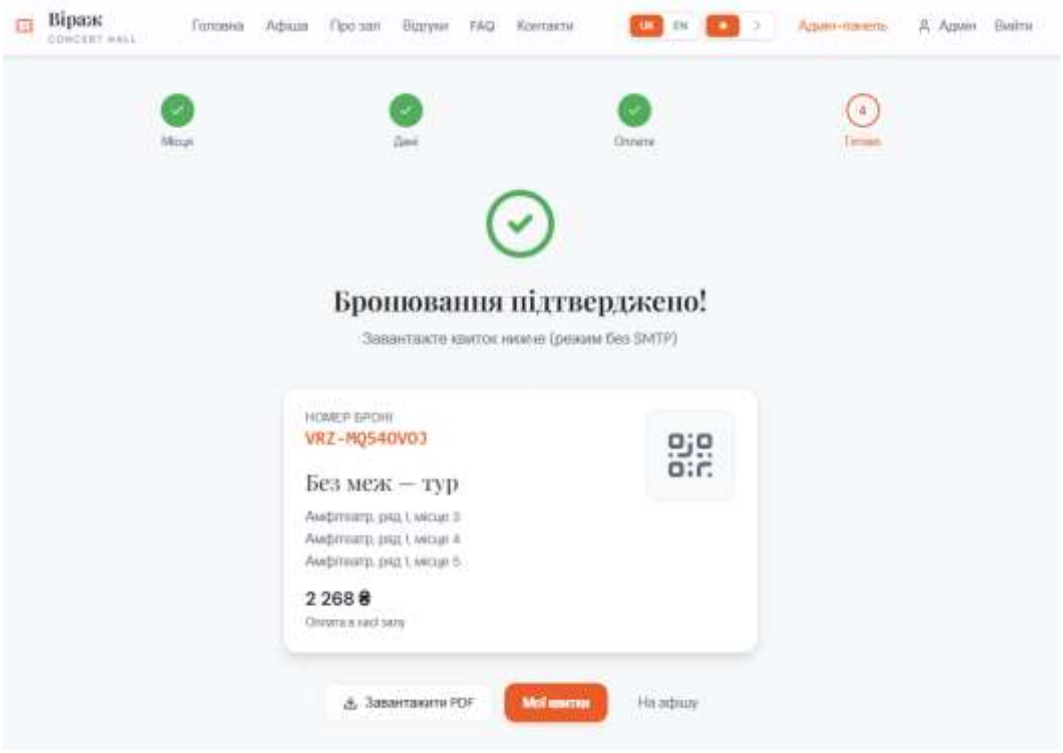


Рисунок 2.9 – Сторінка підтвердження бронювання (/booking/success)

Рисунок 2.10 – PDF-квиток з QR-кодом

Чат підтримки. Компонент SupportChat закріплено в правому нижньому куті (через AppProviders). visitorId зберігається в localStorage; повідомлення завантажуються polling кожні 3 секунди. Відвідувач може закрити діалог; адміністратор відповідає з панелі AdminChatsPanel. На сторінках /admin чат приховано.

Відгуки та афіша. EventsList і EventCard отримують події з сервера; imageUrl вирішується через resolveEventImageUrl. ReviewForm відправляє POST /api/reviews.

Форма входу (/login) і реєстрації (/register) реалізовані компонентами LoginForm та RegisterForm. Вигляд сторінки входу показано на рисунку 2.11.

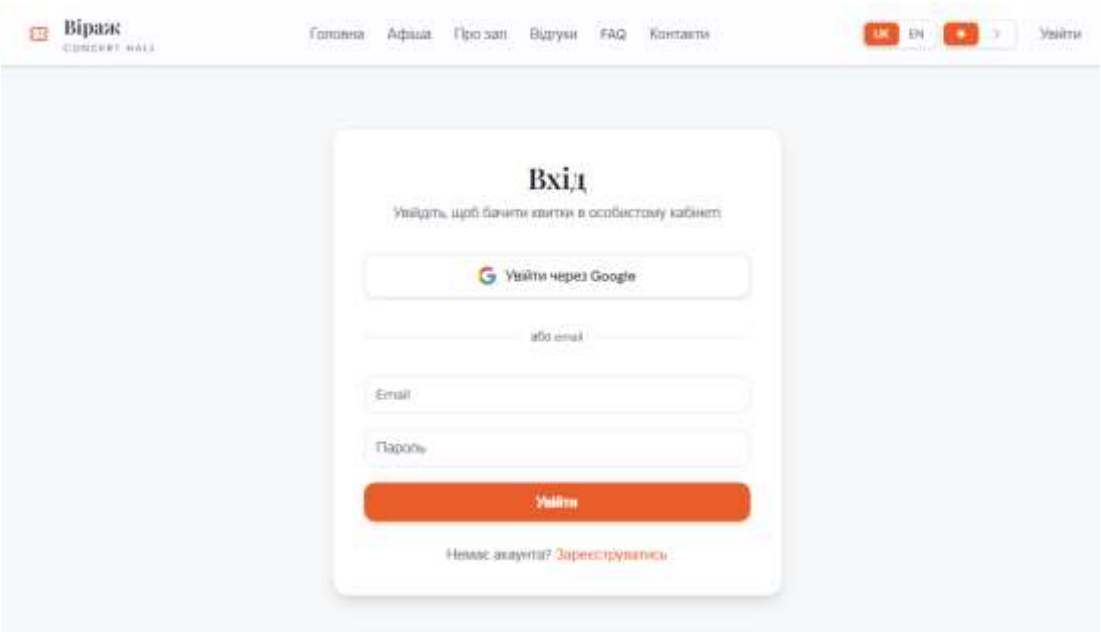


Рисунок 2.11 – Сторінка входу (/login)

Після успішної автентифікації зареєстрований користувач отримує доступ до особистого кабінету (/profile), де відображаються налаштування профілю та список бронювань із можливістю скасування. Вигляд сторінки профілю показано на рисунку 2.12.

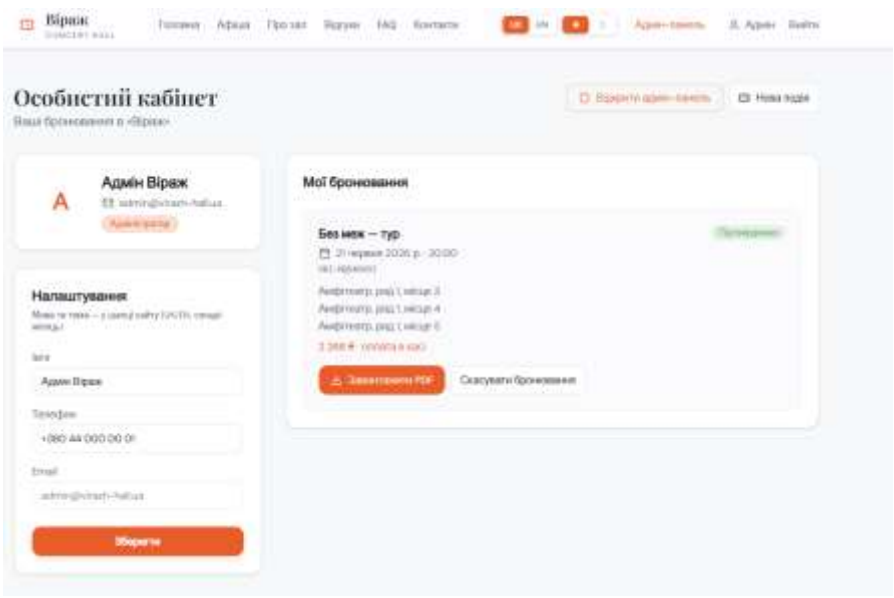


Рисунок 2.12 – Особистий кабінет (/profile)

Авторизація клієнта. LoginForm викликає signIn (credentials). RegisterForm - POST /api/auth/register. Конфігурація auth.config.ts винесена окремо для middleware; auth.ts підключає PrismaAdapter, Credentials і опційно Google OAuth [6]. Паролі зберігаються як bcrypt-хеш [17].

Валідація на сервері. Критичні API використовують схеми Zod: bookings, hold, credentials при логіні. При помилці - JSON з полем error українською та код 400 або 409 [9], [10]. Додаткові клієнтські модулі. Компонент SupportChat (рис. 2.13) закріплено в правому нижньому куті публічних сторінок; відвідувач веде діалог із підтримкою, адміністратор відповідає з панелі /admin/chats.

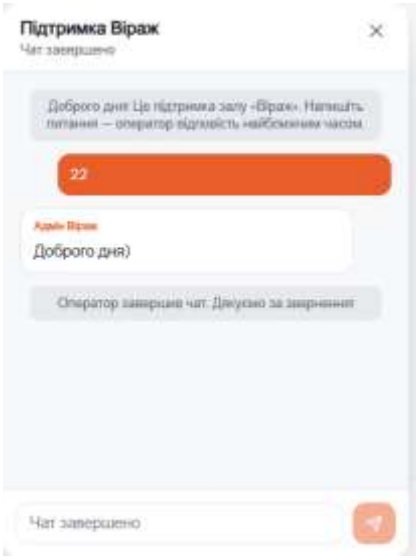


Рисунок 2.13 – Віджет чату підтримки

Сторінка /reviews (рис. 2.14) містить публічний список відгуків і форму ReviewForm для додавання нового відгуку з оцінкою 1–5. Обидва модулі зберігають дані в SQLite через відповідні API-маршрути.

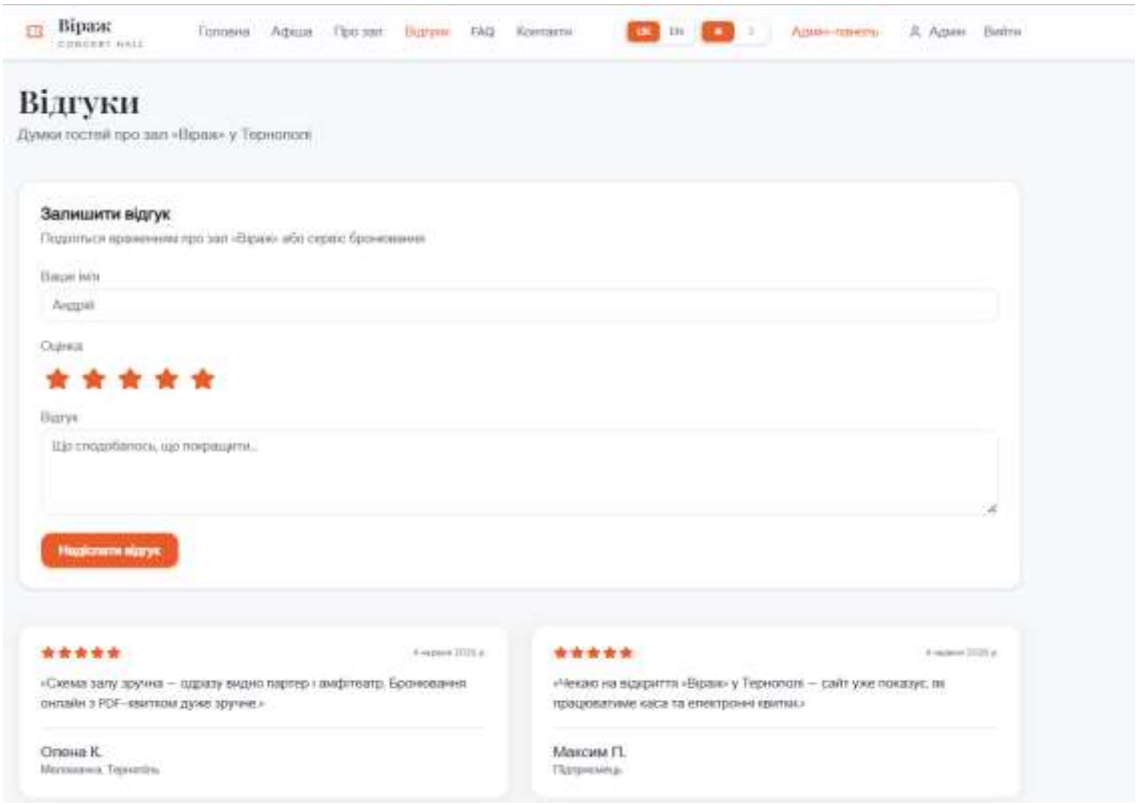


Рисунок 2.14 – Сторінка відгуків (/reviews)

2.4.2 Написання admin-частини

Каркас адмін-панелі. AdminLayout (серверний) дублює перевірку сесії та ролі admin. AdminSidebar містить посилання: Дашборд, Події, Бронювання, Чати підтримки (з лічильником відкритих чатів), Відгуки, Користувачі. AdminTopBar - швидкий перехід на публічний сайт. Структуру адміністративної панелі наведено на рисунку 2.15.

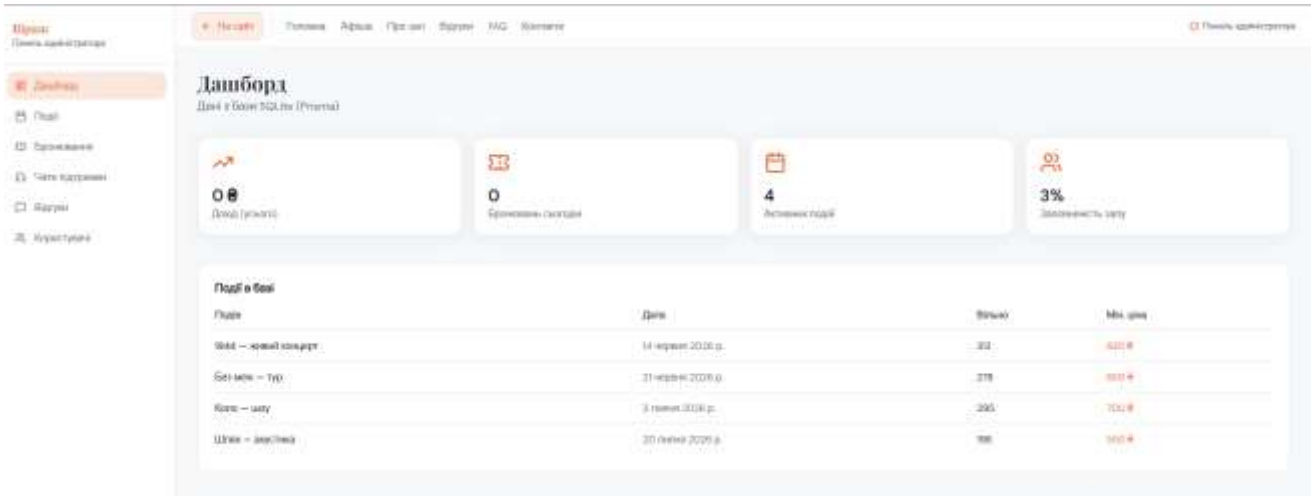


Рисунок 2.15 – Структура адміністративної панелі

Події. Сторінка /admin/events: форма CreateEventForm (назва, виконавець, дата, час, опис, категорія, ціна, URL або файл постера). PosterUpload надсилає multipart на POST /api/admin/upload; файл зберігається в public/posters/. EventEditor дозволяє редагувати подію через PATCH /api/admin/events/[id]. При створенні події сервер генерує EventSeat для всіх 420 місць зі статусом available.

Форма CreateEventForm дозволяє адміністратору ввести назву, виконавця, дату, час, опис, категорію, ціну та завантажити зображення постера через компонент PosterUpload. Після збереження події сервер автоматично створює 420 записів EventSeat зі статусом available. Інтерфейс створення події наведено на рисунку 2.16.

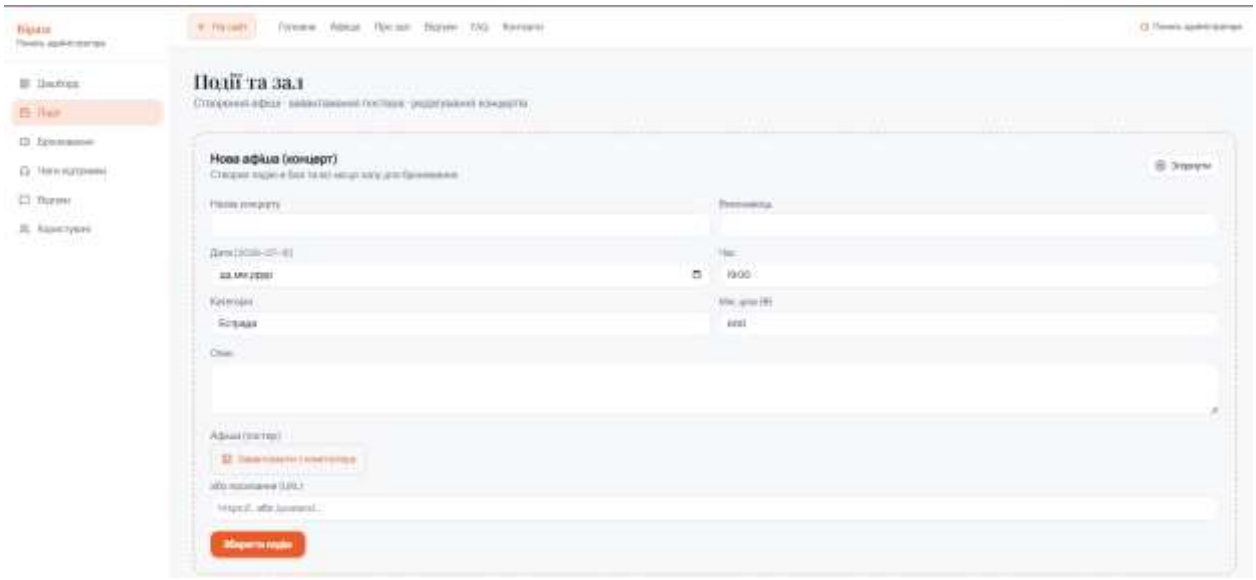


Рисунок 2.16 – Форма створення події в адмін-панелі (/admin/events)

Бронювання. /admin/bookings - таблиця замовлень з номером VRZ, подією, гостем, сумою, статусом. AdminBookingActions дозволяє скасувати бронь (повернення місць у available).

Таблиця бронювань містить номер замовлення VRZ, назву події, дані гостя, суму та статус. Адміністратор може скасувати бронь через AdminBookingActions з поверненням місць у стан available. Вигляд списку замовлень наведено на рисунку 2.17.

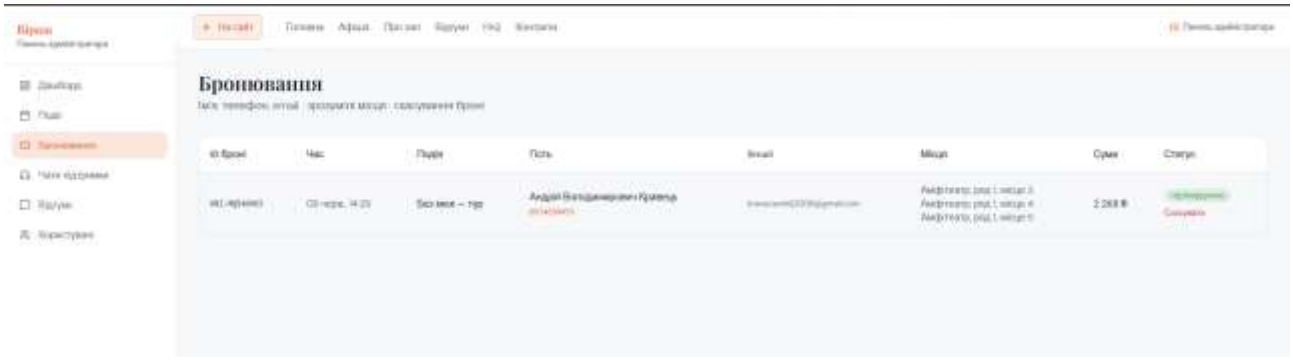


Рисунок 2.17 – Список бронювань в адмін-панелі (/admin/bookings)

Чати та відгуки. AdminChatsPanel: список чатів, історія повідомлень, відповідь адміна, закриття чату. AdminReviewsEditor: редагування відгуку, поле adminReply для публічної відповіді закладу.

Панель AdminChatsPanel відображає список відкритих діалогів, історію повідомлень і поле відповіді адміністратора. AdminReviewsEditor дозволяє редагувати текст відгуку та додати публічну відповідь закладу (adminReply). Приклад роботи з чатом підтримки в адмін-панелі наведено на рисунку 2.18.

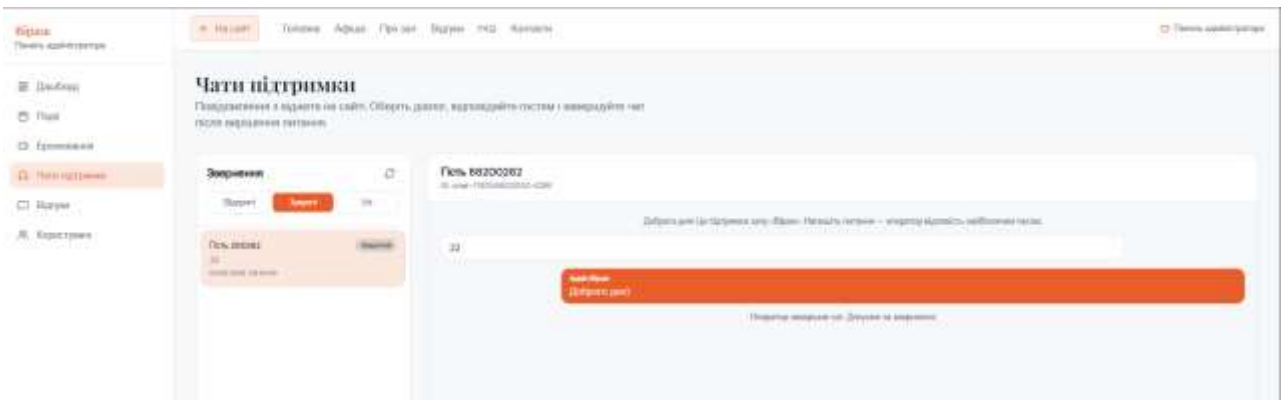


Рисунок 2.18 – Відповідь адміністратора в чаті підтримки (/admin/chats)

Користувачі. /admin/users - перегляд облікових записів без зміни паролів через інтерфейс.

Захист API. Усі маршрути /api/admin/* проходять через requireAdmin: role === admin, інакше HTTP 403 [9].

Дашборд /admin показує зведену статистику: кількість подій, бронювань, відкритих чатів, відгуків - через Prisma count на сервері.

2.5 Тестування web-сайту

Мета тестування - підтвердити відповідність реалізованого прототипу вимогам п. 1.2.3 і п. 1.2.7 технічного завдання, перевірити цілісність даних у SQLite після перезапуску сервера та зафіксувати готовність до демонстрації на захисті кваліфікаційної роботи [2], [10].

Види випробувань. Функціональне тестування - проходження контрольних сценаріїв відвідувача та адміністратора. Інтеграційне - перевірка ланцюгів «браузер → API → Prisma → файл dev.db», зокрема hold, транзакція бронювання, скасування. Тестування інтерфейсу - відображення на ширині екрана близько 1920 px і 390 px. Негативне - спроби некоректних дій. Регресійне - повторення сценаріїв бронювання гостем і створення події з постером після змін у коді.

Середовище: Node.js 20+, Windows 10; команди npm install, npm run db:setup, npm run dev; браузер Chrome; тестові облікові записи з seed. Додатково виконано npm run build без помилок TypeScript. Дати перевірок - березень 2026 р. Результати контрольних сценаріїв узагальнено в таблиці 2.3.

Таблиця 2.3 – Результати тестування web-сайту «Віраж»

№	Сценарій	Очікуваний результат	Результат	Дата	Примітка
1	2	3	4	5	6
1	Головна та /events	Події з БД	Виконано	20.05.2026	4 події після seed
2	Картка події	Опис, постер, кнопка до схеми	Виконано	20.05.2026	-

Продовження таблиці 2.3

1	2	3	4	5	6
3	Вибір місць, масштаб	До 8 місць, sold недоступні	Виконано	21.05.2026	Мобільний емулятор
4	Hold + checkout гість	VRZ-номер, confirmed	Виконано	21.05.2026	Hold 15 хв
5	Завантаження PDF	QR, місця, сума	Виконано	21.05.2026	Transliterate в PDF
6	Реєстрація + бронювання	userId у Booking	Виконано	22.05.2026	-
7	Скасування в профілі	cancelled, місця available	Виконано	22.05.2026	-
8	Спроба sold місця	HTTP 409, UA повідомлення	Виконано	22.05.2026	-
9	Відгук /reviews	Запис у Review	Виконано	23.05.2026	-
10	Чат + відповідь адмін.	SupportChatMessage	Виконано	23.05.2026	Polling 3 с
11	Admin: подія + постер	Файл у public/posters	Виконано	24.05.2026	Потрібно «Зберегти»
12	Admin: відповідь на відгук	adminReply на сайті	Виконано	24.05.2026	-
13	User на /admin	redirect profile?denied=admin	Виконано	24.05.2026	middleware
14	npm run build	Без помилок	Виконано	25.05.2026	next build
15	Перезапуск сервера	Дані в SQLite збережені	Виконано	25.05.2026	Броні, чати, відгуки

Усі сценарії таблиці 2.3 виконано успішно. Прототип відповідає критеріям прийому з п. 1.2.7: функції ТЗ реалізовані, подвійний продаж місця запобігається транзакцією, PDF і ролі працюють коректно, повідомлення про помилки українською. Виявлені в ході розробки колізії (застарілі held, відображення постера після збереження форми) усунуто до фіксації результатів тестування.

Аналіз результатів. Найбільш критичним визнано ланцюг hold → checkout → bookings: саме він перевіряє цілісність даних EventSeat і Booking та запобігання повторному продажу одного місця. Окремо підтверджено коректність розмежування ролей (сценарії 13 і 6), роботу адміністративних операцій із файлами постерів і відповідями на відгуки (сценарії 11–12), а також збереження

даних у SQLite після перезапуску сервера (сценарій 15). Успішне виконання npm run build (сценарій 14) свідчить про відсутність помилок типізації та збірки перед демонстрацією на захисті [10].

Обмеження прототипу, підтверджені тестами: онлайн-оплата не підключена; чат не є WebSocket real-time; SQLite не розрахований на високе навантаження без міграції на PostgreSQL [11], [16]. Ці обмеження узгоджені з розділом 1 і не перешкоджають захисту кваліфікаційної роботи.

Подальші кроки щодо експлуатації сайту - розміщення на хостингу, наповнення контентом, налаштування SEO та рекомендації з просування - описуються в розділі 3 пояснювальної записки. На цьому етапі технічний і робочий проєкт веб-сайту «Віраж» можна вважати завершеним у межах демонстраційного прототипу.

3 СПЕЦІАЛЬНИЙ РОЗДІЛ

Третій розділ пояснювальної записки присвячено практичним питанням експлуатації веб-сайту бронювання квитків для концертного залу «Віраж». У ньому розглядаються умови функціонування застосунку в мережі Internet, підтримка його працездатності, наповнення контентом і заходи щодо залучення відвідувачів.

Матеріал розділу спирається на вимоги технічного завдання та узгоджується з проєктними рішеннями, сформульованими в попередніх розділах записки. Окремо визначаються вимоги до програмно-апаратного середовища, порядок розгортання сайту, робота з базою даних і адміністративною частиною, а також підходи до пошукової оптимізації й подальшого супроводу ресурсу [2], [16].

3.1 Інструкція з розміщення сайту в Internet

Мета підрозділу - описати вимоги до середовища виконання, порядок локального розгортання для розробки та демонстрації на захисті, варіанти публікації в мережі Internet і налаштування конфігураційного файлу .env.

Вимоги до програмно-апаратного середовища. Для коректної роботи застосунку потрібне середовище виконання Node.js версії 20 або новішої. Підтримуються 64-розрядні ОС Microsoft Windows 10/11 та Linux (Ubuntu 22.04 LTS і новіші). Мінімальний обсяг оперативної пам’яті - 4 ГБ; для команди npm run build комфортніше 8 ГБ. Вільне місце на диску - не менше 2 ГБ (каталог node_modules, файл бази даних prisma/dev.db, завантажені постери в public/posters/).

На стороні відвідувача (клієнта) підтримуються браузери Google Chrome, Mozilla Firefox, Microsoft Edge останніх версій. Мінімальна ширина екрана для зручного бронювання - 360 px; оптимально - від 1280 px. Для production-розміщення додатково потрібні: постійне підключення до мережі Internet,

зареєстроване доменне ім’я, TLS-сертифікат, регламент резервного копіювання файлу бази даних [11], [16]. Узагальнені вимоги наведено в таблиці 3.1.

Таблиця 3.1 – Мінімальні вимоги до середовища розгортання

Компонент	**Мінімум (локально)**	**Рекомендація (production)**
Node.js	20.x	20.x LTS
Оперативна пам'ять	4 ГБ	8 ГБ і більше
Вільне місце на диску	2 ГБ	20 ГБ та резервні копії
Веббраузер	Google Chrome, Mozilla Firefox, Microsoft Edge	Остання стабільна версія
Протокол доступу	HTTP (localhost)	HTTPS із власним доменом
Система керування БД	SQLite (dev.db)	SQLite або PostgreSQL

Режими розгортання. Веб-сайт «Віраж» можна розгорнути двома способами, обидва є коректними для експлуатації прототипу.

Перший спосіб - локальне розгортання на робочій станції розробника (персональний комп’ютер або ноутбук). Сайт запускається командою npm run dev або npm run start і доступний за адресою http://localhost:3000 (або іншим локальним портом). Такий режим використовують під час розробки, тестування та демонстрації на захисті кваліфікаційної роботи, коли комісія переглядає робочий застосунок безпосередньо на машині автора.

Другий спосіб - розміщення на хостингу в мережі Internet. Після публікації сайт стає доступним за публічною адресою https://virazh-production.up.railway.app, і відвідувачі з будь-якого місця можуть відкрити афішу, обрати місця та оформити бронювання через браузер. Цей режим застосовують після захисту кваліфікаційної роботи або при підготовці до реальної експлуатації залу.

Обидва режими використовують той самий вихідний код з репозиторію virazh і ті самі команди npm (таблиця 3.2); відрізняються лише середовище запуску і значення NEXTAUTH_URL у файлі .env.

Локальне розгортання на робочій станції. Послідовність підготовки така [3].

Крок 1. Встановити Node.js 20+ з офіційного дистрибутива; у терміналі перевірити версії командами node -v і npm -v.

Крок 2. Скопіювати каталог проєкту на локальний диск. Рекомендується уникати надто довгих шляхів із спецсимволами, якщо на Windows виникають помилки збірки.

Крок 3. У корені virazh виконати npm install - встановлюються залежності Next.js, React, Prisma, NextAuth, pdf-lib, zod та інші пакети з package-lock.json.

Крок 4. Скопіювати файл .env.example у .env; для локального режиму задати NEXTAUTH_URL=http://localhost:3000 (детальніше - таблиця 3.3).

Крок 5. Виконати npm run db:setup - створюються таблиці (prisma db push) і початкові дані (seed: зал «Віраж», 420 місць, чотири події, відгуки, demo@virazh.local і admin@virazh-hall.ua / virazh123).

Крок 6. Запустити npm run dev. За замовчуванням сайт відкривається за адресою http://localhost:3000. Якщо порт 3000 зайнятий, Next.js пропонує 3001 - тоді NEXTAUTH_URL у .env оновлюють відповідно.

Крок 7. Перевірити сценарій: /events → подія → схема місць → бронювання → PDF; вхід admin@virazh-hall.ua на /admin.

Для перевірки production-збірки на тій самій машині: npm run build, потім npm run start (без hot reload). Основні команди npm для обох режимів узагальнено в таблиці 3.2.

Таблиця 3.2 – Основні команди npm для розгортання та супроводу

Команда	**Призначення**	**Коли виконувати**
1	2	3
npm install	Встановлення залежностей	Перший запуск, після git pull
npm run db:setup	Створення БД і початкове наповнення даними (seed)	Перший запуск, після зміни schema.prisma
npm run dev	Запуск сервера розробки на порту 3000	Щоденна розробка, демонстрація та захист кваліфікаційної роботи
npm run build	Створення production-збірки проекту	Перед деплоєм, для перевірки типів і збірки

Продовження таблиці 3.2

1	2	3
npm run start	Запуск зібраної версії за стосунку	Локальне тестування production-версії
npm run lint	Перевірка коду за правилами ESLint	Перед здачею проєкту, після ре факторингу
npm run db:push	Оновлення схеми бази даних	Після зміни моделей Prisma
npm run db:seed	Повторне наповнення бази тестовими даними	За потреби відновлення демонстраційного стану

Розміщення на хостингу в мережі Internet. Після успішного тестування (таблиця 2.2, розділ 2) той самий застосунок публікують у мережі Internet, щоб він був доступний відвідувачам за постійним посиланням (URL), а не лише на localhost. Для стеку Next.js 16 застосовують один із поширених варіантів [3], [16].

Розміщення на хостингу в мережі Internet. Після перевірки працездатності застосунку в локальному режимі вихідний код репозиторію virazh підключено до сервісу Railway (деплой з GitHub). Платформа автоматично виконує збірку (npm run build із генерацією Prisma Client) і запуск застосунку.

Для збереження файлу бази даних SQLite між перезапусками створено том (Volume) з монтуванням у каталог /data; у змінній середовища DATABASE_URL вказано file:/data/dev.db.

У панелі Railway задано обов’язкові змінні: AUTH_SECRET, NEXTAUTH_URL (повна публічна адреса з https://) та DATABASE_URL. Після першого успішного деплою виконано початкове наповнення бази командою npm run db:seed. Оновлення версії сайту здійснюється через git push у гілку main - Railway перезбирає застосунок автоматично.

Публічна адреса сайту: https://virazh-production.up.railway.app. Перевірено доступність головної сторінки, API /api/events, цикл бронювання, генерацію PDF-квитка та вхід адміністратора в /admin.

На хостингу обов’язково: унікальний AUTH_SECRET; NEXTAUTH_URL має збігатися з фактичною публічною адресою; пароль адміністратора замінюють на складний замість демонстраційного virazh123 [6], [9].

Вибір режиму. Для захисту кваліфікаційної роботи достатньо локального розгортання (localhost). Для демонстрації роботи сайту комісії та відвідувачам у мережі Internet застосовано мережевий режим на платформі Railway. Обидва варіанти використовують один і той самий вихідний код репозиторію virazh.

Налаштування змінних середовища. Конфігурація застосунку винесена у файл .env у корені репозиторію (зразок - .env.example). Next.js завантажує ці змінні при старті; секрети не зберігаються у вихідному коді, що відповідає рекомендаціям безпечної розробки [9].

Обов'язкові параметри: DATABASE_URL - рядок підключення Prisma (для SQLite: file:./dev.db, файл prisma/dev.db) [5]; AUTH_SECRET - секретний ключ для підпису сесій NextAuth; NEXTAUTH_URL - повна базова URL застосунку (http://localhost:3000 локально; https://домен у production). У мережевому режимі на Railway значення NEXTAUTH_URL відповідає публічній адресі застосунку, наприклад https://virazh-production.up.railway.app Невідповідність NEXTAUTH_URL і фактичної адреси призводить до помилок входу [6].

Опційні параметри розширюють функціонал: GOOGLE_CLIENT_ID і GOOGLE_CLIENT_SECRET - кнопка «Увійти через Google»; SMTP_HOST, SMTP_PORT, SMTP_SECURE, SMTP_USER, SMTP_PASS, EMAIL_FROM - відправка PDF-квитка на email після бронювання; OPENAI_API_KEY і OPENAI_MODEL - інтелектуальні відповіді у чаті підтримки (без ключа працює вбудований сценарний асистент). Повний перелік змінних наведено в таблиці 3.3.

Таблиця 3.3 – Змінні середовища файлу .env

Змінна	**Обов'язковість**	**Приклад значення**	**Призначення**
1	2	3	4
DATABASE_URL	Так	file:./dev.db	Підключення Prisma до бази даних SQLite
AUTH_SECRET	Так	випадковий рядок Base64	Захист сесій та токенів NextAuth
NEXTAUTH_URL	Так	http://localhost:3000	Базова URL-адреса за стосунку

Продовження таблиці 3.3

1	2	3	4
GOOGLE_CLIENT_ID	Ні	значення з Google Cloud	Ідентифікатор клієнта OAuth Google
GOOGLE_CLIENT_SECRET	Ні	значення з Google Cloud	Секретний ключ OAuth Google
SMTP_HOST	Ні	smtp.gmail.com	Адреса SMTP-сервера для надсилання листів
SMTP_PORT	Ні	587	Порт SMTP-сервера
SMTP_USER / SMTP_PASS	Ні	облікові дані пошти	Автентифікація на SMTP-сервері
EMAIL_FROM	Ні	Віраж <noreply@example.com>	Адреса відправника електронних листів
OPENAI_API_KEY	Ні	sk-...	Доступ до API OpenAI для AI-функцій чату підтримки

3.2 Інструкція з обслуговування та наповнення сайту

Мета підрозділу - описати порядок супроводу бази даних SQLite, наповнення сайту інформаційним контентом (афіша, статичні сторінки) та щоденну експлуатацію адміністративної панелі персоналом залу.

Обслуговування бази даних. У прототипі всі персистентні дані зберігаються в одному файлі prisma/dev.db (формат SQLite). Prisma ORM виконує запити з серверних маршрутів app/api і серверних компонентів; клієнтський JavaScript безпосередньо до файлу БД не звертається [5], [11].

Структура даних відповідає schema.prisma (розділ 2): Hall, Seat (420 записів), Event, EventSeat, Booking, User, Account, Session, Review, SupportChat, SupportChatMessage. Координати та ціни місць генеруються програмно функцією buildHallSeatTemplate() у модулі lib/seats.ts під час seed; повторне редагування координат уручну не передбачено - зміни вносять у код і виконують npm run

db:push.

Початкове наповнення виконує команда npm run db:seed (або npm run db:setup при першому запуску). Скрипт prisma/seed.ts створює: один зал id=virazh; усі 420 місць; чотири демонстраційні події з lib/mock-data.ts; по 12 проданих місць на подію для ілюстрації статусу sold; три відгуки; облікові записи demo@virazh.local (role user) і admin@virazh-hall.ua (role admin). Повторний запуск seed очищає таблиці перед insert - на production перед цим обов'язково роблять резервну копію dev.db.

Резервне копіювання. Достатньо періодично копіювати файл prisma/dev.db і каталог public/posters/ на зовнішній носій або хмарне сховище. Відновлення: зупинити Node-процес, замінити файл бази, запустити npm run start. Частота копіювання залежить від інтенсивності бронювань; мінімум - перед кожним оновленням версії сайту або зміною схеми БД.

Оновлення лише структури таблиць без повного перезапису контенту: npm run db:push. Перспектива масштабування: при зростанні навантаження SQLite замінюють на PostgreSQL (зміна provider у schema.prisma, міграція, оновлення DATABASE_URL) [11], [16].

Наповнення сайту контентом. Контент поділяють на динамічний (зберігається в SQLite, керується через адмін-панель) і статичний (зашитий у файли коду, оновлюється розробником при випуску нової версії сайту).

Динамічний контент - насамперед афіша подій. Кожна подія містить поля: назва (title), виконавець (artist), опис (description), дата й час (date, time), категорія (category), мінімальна ціна (minPrice), кількість вільних місць (availableSeats), URL постера (imageUrl). При створенні події сервер автоматично генерує 420 записів EventSeat зі статусом available. Постер завантажують через компонент PosterUpload: файл JPEG, PNG, WebP або GIF розміром до 5 МБ надсилається на POST /api/admin/upload і зберігається в public/posters/ з унікальним іменем; отриманий шлях підставляється в imageUrl події.

Рекомендований порядок наповнення афіши після відкриття залу або для оновлення демонстраційного наповнення:

1) Увійти як admin@virazh-hall.ua на сторінку /admin/events.

2) Заповнити форму CreateEventForm: усі обов’язкові поля, завантажити постер.

3) Натиснути «Створити подію» - перевірити появу картки на публічній афіші /events.

4) Відкрити /events/[id]/seats - переконатися, що схема показує вільні місця.

5) За потреби відредагувати подію через EventEditor (PATCH /api/admin/events/[id]); після зміни зображення постера обов’язково натиснути «Зберегти» у формі.

Бронювання та відгуки наповнюються діями відвідувачів; адміністратор лише модерує записи в /admin/bookings та /admin/reviews.

Статичний контент оновлюється в репозиторії:

– сторінка /faq - масив faqItems у lib/mock-data.ts (питання-відповіді про бронювання, оплату в касі, повернення);

– сторінка /about - опис залу, місткість 420 місць, компонент HallMapPreview;

– сторінка /contacts - адреса вул. Театральна, 9, м. Тернопіль; вбудована карта Google Maps;

– сторінка /reviews - публічний список із таблиці Review; початкові три записи створює seed; нові відгуки надходять через ReviewForm (POST /api/reviews);

– головна сторінка / - блок «Найближчі події» отримує записи Event із БД серверним запитом.

У поточній версії прототипу афіша містить чотири події з початкового наповнення seed; подальші події додаються адміністратором через /admin/events.

Процес створення події наведено на рисунку 3.1.

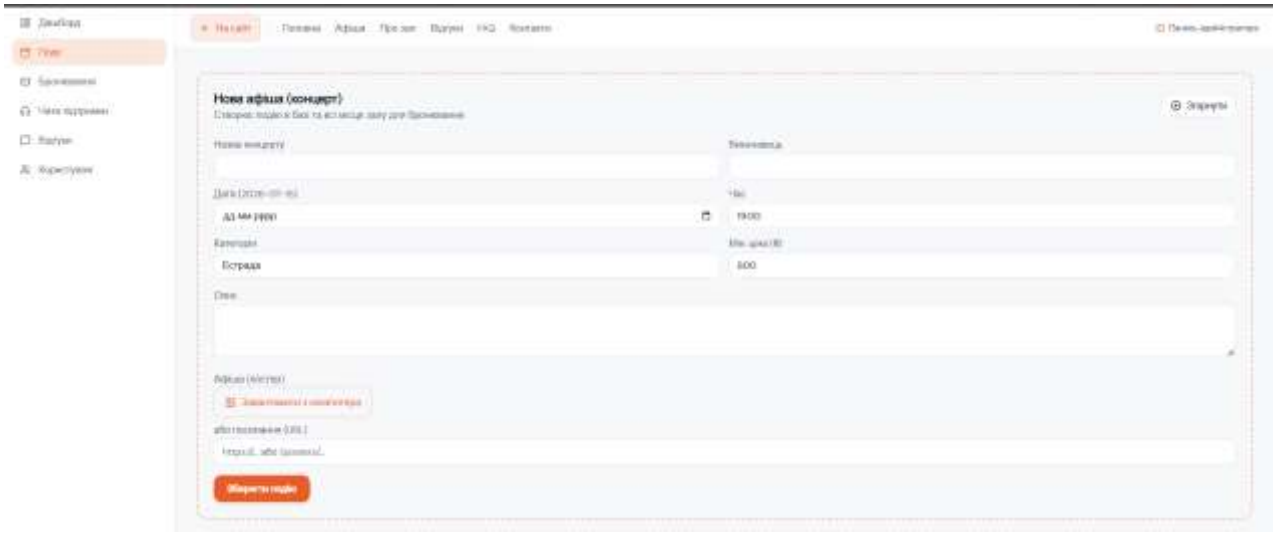


Рисунок 3.1 – Наповнення афіши: форма створення події в адмін-панелі

Експлуатація адміністративної панелі. Адміністративна зона доступна за префіксом /admin після автентифікації користувача з role=admin. Middleware і AdminLayout дублюють перевірку ролі; звичайний користувач (role user) перенаправляється на /profile?denied=admin (перевірено сценарієм 13, таблиця 2.2).

Дашборд /admin відображає зведену статистику: кількість подій, бронювань, відкритих чатів підтримки, відгуків (запити Prisma count на сервері). Використовується для щоденного моніторингу завантаженості афіши та звернень відвідувачів.

Сторінка /admin/events - створення, редагування та видалення подій; завантаження постерів. Після зміни постера через PosterUpload обов’язково натиснути «Зберегти» у формі EventEditor - інакше URL зображення не оновиться в записі Event (типова помилка, усунена під час тестування).

Сторінка /admin/bookings - таблиця замовлень з номером VRZ-…, назвою події, ім’ям гостя або користувача, сумою, статусом confirmed/cancelled. AdminBookingActions дозволяє скасувати бронь - місця повертаються в статус available.

Сторінка /admin/chats - список діалогів SupportChat; перегляд історії SupportChatMessage; поле відповіді адміністратора; закриття чату. Відповіді з’являються у віджеті SupportChat на публічному сайті після наступного циклу

polling (інтервал 3 с).

Сторінка /admin/reviews - редагування тексту відгуку, додавання adminReply (публічна відповідь закладу), видалення спаму.

Сторінка /admin/users - перегляд email, імені, ролі, дати реєстрації. Зміна паролів через інтерфейс не реалізована.

Функції адмін-панелі покривають щоденні операції персоналу залу: моніторинг статистики на дашборді; облік бронювань і скасувань; відповіді у чатах підтримки; модерація відгуків; ведення афіші подій. Поле availableSeats у записі Event відображає кількість вільних місць згідно з даними EventSeat у БД.

3.3 Інструкція з популяризації та підтримки сайту

Мета підрозділу - описати реалізовані та заплановані заходи пошукової оптимізації (SEO), канали популяризації сайту та порядок технічного супроводу.

Оптимізація для пошукових систем (SEO). Пошукова оптимізація забезпечує індексацію сторінок афіші та інформації про зал «Віраж» у м. Тернопіль пошуковими системами за запитами на кшталт «концертний зал Тернопіль», «афіша концертів» [7].

У прототипі реалізовано: атрибут lang="uk" у кореневому layout.tsx; глобальні metadata (title «Віраж - Концертний зал | Бронювання квитків», description з ключовими словами); семантична структура HTML (header, main, footer); серверний рендеринг сторінок Next.js App Router - текст афіші передається в HTML при першому запиті; адаптивна верстка Tailwind CSS; україномовний контент інтерфейсу та описів подій [3], [13].

Для мережевої версії сайту передбачено додаткові заходи: файл robots.txt (індексація публічних сторінок, виключення /admin і /api); sitemap.xml з URL подій /events/[id]; унікальні meta для карток подій (generateMetadata); Open Graph теги з постером події; реєстрація в Google Search Console; мікророзмітка Schema.org Event (JSON-LD); Чекліст SEO-заходів наведено в таблиці 3.4.

Таблиця 3.4 – Чекліст SEO для сайту «Віраж»

Захід / SEO-елемент	**У локальній версії**	**У мережевій версії (хостинг)**
lang="uk", базовий title / description	Реалізовано	Реалізовано
Унікальні meta для /events/[id]	Базовий рівень	generateMetadata за подією
sitemap.xml	Не застосовується	Генерація через app/sitemap.ts
robots.txt	Не застосовується	Виключення /admin, /api
HTTPS	HTTP (localhost)	TLS на домені
Google Search Console	-	Підключення домену
OG-теги	Глобальні metadata	og:image = постер події
Schema.org Event	-	JSON-LD на картці події
Формат постерів	JPEG, PNG, GIF GIF	WebP, ширина ≤ 1200 px

Популяризація сайту. Заходи популяризації спрямовані на інформування глядачів про афішу залу «Віраж» та сценарій онлайн-бронювання з оплатою в касі до початку події.

Цифрові канали включають: посилання на сайт у профілях закладу в соціальних мережах (Facebook, Instagram, Telegram) з UTM-мітками; публікацію анонсів із прямим URL на /events/[id]; розміщення матеріалів на регіональних культурних порталах; QR-код на друкованій афіші та плакатах, що веде на сторінку події або загальну афішу; email-інформування через SMTP-модуль після бронювання.

Офлайн-канали включають: інформаційний стенд у фойє з описом кроків бронювання (відповідає схемі на рисунку 2.5); вказівники на веб-ресурс у рекламних матеріалах закладу; участь майданчика в міських культурних заходах із посиланням на сайт.

Технічна підтримка та супровід. Експлуатація сайту включає такі операції:

– періодичне оновлення залежностей npm і перевірку npm audit;

– контроль доступності (запит GET /api/events, моніторинг uptime на хостингу);

– аналіз логів сервера при збоях бронювання, генерації PDF або

відправки email;

– керування обліковими записами адміністратора (зміна пароля, роль admin у таблиці User);

– резервне копіювання prisma/dev.db і каталогу public/posters/;

– оновлення схеми БД через npm run db:push при зміні моделей Prisma.

Перспектива масштабування бази даних. У поточній версії прототипу застосовано SQLite - файлову СУБД, яка не вимагає окремого серверного процесу і спрощує розгортання на одній машині або VPS із постійним диском [11]. Усі запити до даних виконуються через Prisma ORM; прикладний код у маршрутах app/api і серверних компонентах не прив'язаний жорстко до конкретного драйвера - змінюється лише рядок DATABASE_URL і поле provider у файлі schema.prisma (з sqlite на postgresql). Після зміни provider виконують prisma migrate для перенесення структури таблиць; дані з dev.db імпортують у PostgreSQL відповідними засобами міграції. Такий підхід зберігає модель Hall, Seat, Event, EventSeat, Booking та інші сутності без переписування бізнес-логіки hold, транзакційного бронювання та генерації PDF.

Перехід на PostgreSQL доцільний при одночасній роботі великої кількості відвідувачів на етапі бронювання: реляційна СУБД краще обробляє паралельні транзакції оновлення EventSeat, ніж один файл SQLite на високому навантаженні [5], [11]. Для концертного залу на 420 місць із піковими продажами на відкритті репертуару мережева версія сайту на VPS із PostgreSQL забезпечує стабільніший облік місць, ніж serverless-хостинг із тимчасовим файловим сховищем.

Перспектива підключення онлайн-оплати. У прототипі поле paymentNote таблиці Booking має значення pay_at_venue; квиток підтверджує бронь, а оплата відбувається в касі залу - це зафіксовано в технічному завданні (розділ 1). Архітектура API POST /api/bookings і сторінки checkout побудована так, що після успішного створення запису Booking можна додати окремий крок оплати через платіжний шлюз (LiqPay, Portmone або інший) без зміни схеми місць EventSeat: статус sold виставляється після підтвердження платежу замість миттєвого підтвердження при submit форми [16]. Поле status у Booking уже підтримує

розширення (confirmed, cancelled); додатковий статус pending_payment може вводитися при інтеграції шлюзу.

Підсумок спеціального розділу. Матеріали п. 3.1–3.3 описують повний цикл експлуатації сайту «Віраж»: розміщення в локальному або мережевому режимі, обслуговування SQLite, наповнення афіши через адмін-панель, SEO та канали популяризації, регламент технічного супроводу. Реалізований прототип відповідає вимогам технічного завдання щодо інструкцій з розміщення та наповнення; обмеження поточної версії (файлова БД, відсутність онлайн-оплати, polling у чаті) узгоджені з розділом 1 і підтверджені тестуванням (таблиця 2.2). Економічне обґрунтування розробки та вимоги охорони праці програміста при роботі над сайтом наведені в розділах 4 і 5 пояснювальної записки.

4 ЕКОНОМІЧНИЙ РОЗДІЛ

Метою економічної частини кваліфікаційної роботи є проведення розрахунків, спрямованих на визначення економічної ефективності розробки веб-сайту бронювання квитків для концертного залу «Віраж» та обґрунтування доцільності його подальшого використання після введення залу в експлуатацію.

Об’єктом розробки є програмний продукт - full-stack веб-застосунок на базі Next.js 16, React 19, TypeScript, Prisma ORM і SQLite, що забезпечує афішу подій, інтерактивну схему залу на 420 місць, бронювання з генерацією PDF-квитка, особистий кабінет, чат підтримки, відгуки та адміністративну панель. Розрахунки виконано за методикою консультанта з економіки; натуральні показники трудовитрат узгоджені з п. 1.2.5 технічного завдання.

Розрахунок вартості розробки виконується у такій послідовності:

– визначити стадії технологічного процесу та трудомісткість операцій;

– розрахувати витрати на оплату праці та відрахування на соціальні заходи;

– обчислити витрати на електроенергію;

– нарахувати амортизаційні відрахування на використане обладнання;

– визначити накладні витрати;

– скласти кошторис і визначити собівартість науково-дослідної роботи (НДР);

– розрахувати ціну робіт;

– визначити економічну ефективність і термін окупності для майбутнього оператора залу.

4.1 Визначення стадій технологічного процесу та загальної тривалості проведення НДР

У підрозділі розглянуто основні етапи технологічного процесу розробки сайту «Віраж». Процес охоплює аналітичну, проєктну, програмну, тестову та

документальну стадії життєвого циклу, описані в п. 1.2.6. Для визначення загальної тривалості робіт дані витрат часу по операціях зведено в таблицю 4.1.

Таблиця 4.1 – Середній час виконання робіт за стадіями технологічного процесу розробки сайту «Віраж»

№ п/п	Назва операції (стадії)	Виконавець	Середній час виконання операції, год.
1	Планування та аналіз	Кер. проєкту Pm	8
		Інженер (l1)	8
2	Розробка технічного завдання	Кер. проєкту (Pm)	7
		Інженер (l1)	4
3	Дизайн інтерфейсу	Інженер (l1)	16
		Інженер (l2)	24
4	Розробка функціоналу	Інженер (l1)	42
5	Тестування та відладка	Тестувальник	8
6	Документування	Інженер (l1)	2
7	Розгортання та підтримка	Інженер (l2)	16
8	Управління проєктом	Кер. проєкту (Pm)	24
Разом			159

Сумарний час виконання технологічного процесу становить 159 годин.

4.2 Визначення витрат на оплату праці та відрахувань на соціальні заходи

У даному підрозділі проводиться аналіз і розрахунок витрат, пов'язаних з оплатою праці та відрахуваннями на соціальні заходи, що необхідні для розробки сайту «Віраж»

Розмір заробітної плати залежить від складності та умов виконуваної

роботи, професійно-ділових якостей працівника, результатів його праці та діяльності підприємства.

Основна заробітна плата розраховується за формулою:

$$З_{осн.} = Т_с * К_г \quad (4.1)$$

де: T_c – тарифна ставка, грн. (приймаємо для керівника проєкту (Pm) – 385 грн./год, інженера (l2) – 255 грн./год.), інженера (l1) – 105 грн./год., тестувальник – 90 грн./год.; $K_г$ – кількість відпрацьованих годин.

Отже, основна заробітна плата для:

Керівника проекту (Pm) $З_{осн} = 39 * 385 = 15\,015$ грн.

Інженера (l2) $З_{осн} = 40 * 255 = 10\,200$ грн.

Інженера (l1) $З_{осн} = 72 * 105 = 7\,560$ грн.

Тестувальник $З_{осн} = 8 * 90 = 720$ грн.

Сумарна основна заробітна плата становить

$З_{осн} = 33\,495$ грн.

Додаткова заробітна плата становить 10 – 15 % від суми основної заробітної плати.

$$З_{дод.} = З_{осн.} \cdot К_{допл.} \quad (4.2)$$

де: $К_{допл.}$ – коефіцієнт додаткових виплат працівникам.

Отже додаткова заробітна плата по категоріях працівників становить:

Керівника проекту $З_{дод2} = 33\,495 * 0{,}1 = 3\,350$ грн.

Загальна додаткова заробітна плата становить:

$З_{дод} = 3\,350$ грн.

Звідси загальні витрати на оплату праці ($B_{о.п.}$) визначаються за формулою:

$$В_{о.п.} = З_{осн.} + З_{дод.} \quad (4.3)$$

$$В_{о.п} = 33\,495 + 3\,350 = 36\,845 \text{ грн.}$$

Єдиний соціальний внесок (ЄСВ – 22%) визначається за формулою:

$$В_{єсв} = Воп * 0{,}22 \quad (4.4)$$

$$В_{єсв} = 36\,845 * 0{,}22 = 8\,106 \text{ грн.}$$

Проведені розрахунки витрат на оплату праці наведено у таблиці 4.2.

Таблиця 4.2 – Зведені розрахунки витрат на оплату праці

№ п/п	Категорія працівників	Основна заробітна плата, грн.			Додаткова заробітна плата, грн.	ЄСВ, грн.	Всього витрати на оплату праці, грн. 6 = 3+4+5
		Тарифна ставка, грн.	К-сть годин	Фактично нарах. зарплати, грн.			
		1	2	3	4	5	6
1	Кер. проєкту (Pm)	385	39	15 015	1 502	3 632	20 149
2	Інженера (l2)	255	40	10 200	1 020	2 468	13 688
3	Інженера (l1)	105	72	7 560	756	1 829	10 145
4	Тестувальник	90	8	720	72	177	969
Разом				33 495	3 350	8 106	44 951

Отже, загальні витрати на оплату праці становлять 44 951 грн.

4.3 Розрахунок витрат на електроенергію

Розрахуємо вартість електроенергії. Затрати на електроенергію 1-ці обладнання визначаються за формулою:

$$З_в = W * T * S \quad (4.5)$$

де: W – необхідна потужність, кВт; T – кількість годин роботи обладнання;
S – вартість кіловат-години електроенергії (приймаємо 15,94 грн).

В нашій системі є 1 ноутбук. Витрати на електроенергію для цього пристрою обчислимо окремо, взявши за основу, що час роботи обладнання обчислюється в залежності від виконуваних робіт (згідно табл. 4.1) і споживані потужності наступні: ноутбук – 0,30 кВт/год.

$$З_{ек} = 0{,}30*159*15{,}94 = 760 \text{ грн.}$$

Витрати на електроенергію становлять 760 грн

4.4 Розрахунок суми амортизаційних відрахувань вебсайту «Віраж»

Характерною особливістю застосування основних фондів у процесі виробництва є їх відновлення. Для відновлення засобів праці у натуральному виразі необхідне їх відшкодування у вартісній формі, яке здійснюється шляхом амортизації.

Амортизація – це процес перенесення вартості основних фондів на вартість новоствореної продукції з метою їх повного відновлення

Комп'ютери та оргтехніка належать до четвертої групи основних фондів.

Амортизація на них нараховується лише в випадку, якщо мінімально допустимі строки їх корисного використання 2 роки. Для визначення амортизаційних відрахувань застосовуємо формулу:

$$A = \frac{Б_В * Н_А}{100\%} * T, \quad (4.6)$$

де: А – амортизаційні відрахування за звітний період, грн.;

$Б_В$ – балансова вартість групи основних фондів на початок звітного періоду, грн.;

$Н_А$ – норма амортизації, 0,04 %.

Оскільки для написання програми та її тестування використовується один ноутбук, вартістю 38900,00 грн., то сума амортизаційних відрахувань становитиме:

$$A = \frac{38\,900 * 0{,}04}{150} * 159 = 1\,649 \text{ грн.}$$

4.5 Обчислення накладних витрат

Накладні витрати пов’язані з обслуговуванням виробництва, утриманням апарату управління підприємства (фірми) та створення необхідних умов праці.

В залежності від організаційно-правової форми діяльності господарюючого суб’єкта, накладні витрати можуть становити 20–60 % від суми основної та додаткової заробітної плати працівників.

$$Н_в = В_{о.п.} * \ 0{,}2..0{,}6 \quad (4.7)$$

де: $Н_В$ – накладні витрати.

$$Н_В = 36\ 845 * 0{,}2 = 14\ 738 \text{ грн.}$$

4.6 Складання кошторису витрат та визначення собівартості веб-сайту «Віраж»

Для складення кошторису витрат та визначення собівартості, результати проведених вище розрахунків зведемо у таблиці 4.4.

Таблиця 4.4 - Кошторис витрат на розробку сайту «Віраж»

№	Зміст витрат	Сума, грн.	В % до загальної суми
1.	Витрати на оплату праці	44 951	73
2.	Витрати на електроенергію	706	0,3
3.	Амортизаційні відрахування	1 649	2,7
4.	Накладні витрати	14 738	24
5.	Собівартість	62 044	100

Собівартість ($С_В$) НДР розрахуємо за формулою:

$$Св = Во.п + Вс.з + Зе + А + Нв \quad (4.8)$$

Отже, собівартість дорівнює $С_В = 62\ 044$ грн.

4.7 Розрахунок ціни робіт

Розрахунок ціни науково-дослідної роботи включає в себе урахування різноманітних факторів, таких як рівень рентабельності, собівартість та податкова ставка.

Ціну робіт можна визначити за формулою:

$$Ц = Св * (1 + Р_{рен}) * (1 + ПДВ) \quad (4.9)$$

де: Св – собівартість; $Р_{рен.}$ – рівень рентабельності; ПДВ – ставка податку на додану вартість.

$$Ц = 62\ 044*(1+0,3)*(1+0,2) = 96\ 788 \text{ грн.}$$

4.8 Визначення економічної ефективності і терміну окупності капітальних вкладень

Економічний ефект

Для визначення ефективності продукту від впровадження сайту «Віраж» після відкриття залу розраховують чисту теперішню вартість (ЧТВ) і термін окупності (Ток).

$$ЧТВ = -Св + \sum_{i=1}^{t} \frac{Г_П}{(1+i)^t}, \quad (4.10)$$

де: Св – собівартість розробки; $Г_п$ – грошовий потік за t – ий рік; t – відповідний рік проекту; і – величина дисконтної ставки (10…15%).

$$ЧТВ = -62\ 044 + \frac{34\ 744}{(1+0,1)^1} + \frac{34\ 744}{(1+0,1)^2} + \frac{34\ 744}{(1+0,1)^3} = 24\ 377 \text{ грн}$$

Якщо ЧТВ ≥ 0, то проект може бути рекомендований до впровадження.

Термін окупності визначається за формулою:

$$T_{ОК} = T_{ПВ} + \frac{Н_В}{Г_{ПР}} \quad (4.11)$$

де: $Т_{ПВ}$ – період до повного відшкодування витрат, років; $Н_В$ – невідшкодовані витрати на початок року, грн.; $Г_{ПР}$ – грошовий потік на початок року, грн.

$$Т_{ок} = 2 + \frac{1745}{34\ 744} = 2,1 \text{ р.}$$

Всі дані внесемо в зведену таблицю 4.5.

Таблиця 4.5 – Техніко-економічні показники проєкту «Віраж»

№ п/п	Показник	Значення
1	2	3
1.	Собівартість, грн.	62 044
2.	Плановий прибуток або грошовий потік, грн.	34 744

Продовження таблиці 4.5

1	2	3
3.	Ціна, грн.	96 788
4.	Чиста теперішня вартість, грн.	24 377
5.	Термін окупності, рік	2,1

Прибутковість проєкту та термін окупності свідчать про його фінансову ефективність та здатність повернути капітальні вкладення протягом 2,1 року. Отже, на основі отриманих показників можна зробити висновок, що розробка веб-сайту бронювання квитків для концертного залу «Віраж» є доцільною з економічної точки зору.

5 ОХОРОНА ПРАЦІ, ТЕХНІКА БЕЗПЕКИ ТА ЕКОЛОГІЧНІ ВИМОГИ

Розробка веб-сайту бронювання квитків для концертного залу «Віраж» (далі - сайт «Віраж») виконувалась автором кваліфікаційної роботи самостійно в домашніх умовах на персональній робочій станції. Характер робіт - інтелектуальна діяльність з проєктування інтерфейсу, написання програмного коду на TypeScript, налаштування бази даних SQLite, тестування та підготовка документації. Фізичне навантаження мінімальне; основні ризики пов’язані з тривалою роботою за комп’ютером, використанням електрообладнання та мережевого підключення.

Метою розділу є систематизація вимог охорони праці, техніки безпеки та екології, які застосовувались під час виконання кваліфікаційної роботи, а також опис заходів, що забезпечують безпечні умови праці розробника програмного забезпечення та мінімізують негативний вплив на навколишнє середовище.

Об’єктом розгляду є робоче місце програміста - автора сайту «Віраж». Предметом - небезпечні й шкідливі виробничі фактори, правила безпечної експлуатації ПЕОМ, вимоги електро- та пожежної безпеки, відповідальність за порушення законодавства про охорону праці, вимоги до вентиляції приміщення, а також екологічні аспекти розробки й майбутньої експлуатації веб-застосунку.

Структура розділу: п. 5.1 - нормативна база, характеристика умов праці та відповідальність за порушення; п. 5.2 - опис робочого місця та систем вентиляції; п. 5.3 - аналіз небезпечних і шкідливих факторів; п. 5.4 - техніка безпеки при роботі з ПЕОМ; п. 5.5 - електробезпека та пожежна безпека; п. 5.6 - екологічні вимоги; п. 5.7 - перша допомога та дії при нещасних випадках.

5.1 Нормативно-правова база та характеристика умов праці розробника

Діяльність з розробки програмного забезпечення регулюється законодавством України про охорону праці, охорону навколишнього природного

середовища та санітарно-гігієнічними нормативами щодо роботи з відеодисплейними терміналами (ВДТ). Основні нормативні документи: Закон України «Про охорону праці» [20]; Закон України «Про охорону навколишнього природного середовища» [21]; ДСН 3.3.6.042-99 «Гігієнічні вимоги до роботи з відеодисплейними терміналами» [22]; ДНАОП 0.00-1.28-18 «Правила охорони праці під час роботи з обчислювальною технікою» [23]; ДСТУ 12.0.003-2015 щодо класифікації небезпечних і шкідливих виробничих факторів [24]; Правила технічної експлуатації електроустановок споживачів та Правила пожежної безпеки в Україні.

Клас умов праці за основними шкідливими факторами для роботи програміста за ПЕОМ за умови дотримання перерв, освітлення та ергономіки робочого місця зазвичай відноситься до класу 3.2 (допустимі умови праці). Роботи з розробки сайту «Віраж» виконувались протягом 159 годин (розділ 4, таблиця 4.1) у режимі окремих сеансів по 2–4 години з перервами, без нічних змін, роботи на висоті, з важкими вантажами чи шкідливими речовинами. Основні види навантаження: розумове (проєктування логіки бронювання, моделі Prisma), зорове (верстка, схема залу на 420 місць), помірне нервово-емоційне (дедлайни етапів, усунення помилок збірки Next.js).

Відповідальність за безпеку на домашньому робочому місці під час самостійної розробки несе виконавець - студент: дотримуватись правил техніки безпеки, не допускати порушень, що можуть призвести до нещасного випадку. Керівник кваліфікаційної роботи здійснює методичний контроль, а не щоденний нагляд за робочим місцем.

Закон України «Про охорону праці» визначає коло суб’єктів відповідальності за стан охорони праці. Роботодавець зобов’язаний забезпечити безпечні умови праці, проводити інструктажі, забезпечити засобами захисту, не допускати до роботи осіб, що не пройшли навчання (ст. 14, 17, 41 Закону). Працівник зобов’язаний дотримуватись вимог охорони праці, правил безпечної експлуатації обладнання, проходити навчання та інструктажі (ст. 35, 53 Закону). У контексті кваліфікаційної роботи роль «працівника» виконує студент-розробник;

роль «роботодавця» формально не застосовується до домашнього робочого місця, однак принципи особистої відповідальності за дотримання правил безпеки залишаються обов’язковими.

За порушення законодавства про охорону праці передбачена адміністративна та кримінальна відповідальність. Адміністративна відповідальність (ст. 258 Кодексу України про адміністративні правопорушення) застосовується до посадових осіб і громадян за порушення вимог охорони праці, якщо вони не містять ознак кримінального правопорушення: наприклад, допуск працівника до роботи без навчання, невиконання розпоряджень органів державного нагляду, порушення правил безпечної експлуатації обладнання. Штрафи встановлюються судом або уповноваженими органами у межах, визначених законом.

Кримінальна відповідальність (ст. 271 Кримінального кодексу України) настає за порушення вимог охорони праці, якщо воно спричинило потерпілому тяжкі наслідки - тілесні ушкодження, професійне захворювання або смерть. Відповідальність несуть керівники підприємств, установ, організацій та інші посадові особи, які зобов’язані були забезпечити дотримання правил безпеки. Для самостійної навчальної розробки вдома кримінальна відповідальність малоймовірна, але знання цих норм необхідне для майбутньої професійної діяльності програміста в штаті ІТ-компанії або при експлуатації серверної інфраструктури сайту «Віраж».

Цивільно-правова відповідальність може настати за завдану шкоду здоров’ю або майну внаслідок порушення правил безпеки (наприклад, пожежа через перевантаження розетки). У таких випадках винна особа відшкодовує збитки відповідно до Цивільного кодексу України. Підсумовуючи, під час розробки сайту «Віраж» виконавець дотримувався вимог охорони праці в межах домашнього робочого місця; порушень, що могли б спричинити адміністративну чи кримінальну відповідальність, не допускалось.

5.2 Опис та оснащення робочого місця розробника

Робоче місце - закріплена за виконавцем зона з ПЕОМ, периферією, меблями та підключенням до мережі 220 В і Internet, де виконувались операції з розробки сайту «Віраж». Розташоване в окремій кімнаті (домашній кабінет) м. Тернопіль; площа зони робочого столу - не менше 6 м², висота стелі - не нижче 2,5 м, що відповідає санітарним вимогам до об’єму повітря на одного працівника за ПЕОМ.

Основне ПЕОМ - ноутбук ASUS TUF Gaming F15 FX506LH: екран 15,6", 1920×1080, IPS, anti-glare; процесор Intel Core i5-10300H, ОЗП 8 ГБ DDR4, SSD 512 ГБ NVMe, GPU NVIDIA GTX 1650 4 ГБ; зарядний адаптер 150 Вт. Програмне середовище: Windows 11, Node.js 20+, npm, Git, Visual Studio Code, браузери Chrome/Firefox/Edge. Локальна база - файл SQLite (prisma/dev.db). Вартість ноутбука 42 500 грн урахована в розділі 4. Організація меблів: стіл з твердою поверхнею, офісний стілець з регулюванням висоти; відстань від очей до екрана 50–60 см; зовнішня USB-миша; настільна LED-лампа зліва (для правші); підключення через мережевий фільтр. На ноутбуці виконувались усі стадії розробки: API, PDF-квитки, NextAuth, seed, тестування бронювання на 420 місць.

Призначення систем вентиляції на робочому місці розробника - забезпечити нормальний газовий склад повітря, відведення надлишкового тепла від ПЕОМ і підтримання параметрів мікроклімату, необхідних для тривалої роботи за ВДТ. За ДСН 3.3.6.042-99 у приміщеннях з ВДТ температура повітря в холодний період має бути 22–24 °С, у теплий - 23–25 °С; відносна вологість 40–60 %; необхідна кратність повітрообміну забезпечується природною та, за потреби, штучною вентиляцією.

На домашньому робочому місці застосовується природна вентиляція через вікно (провітрювання кожні 1–2 години роботи) та загальна вентиляція житлового приміщення (приплив через вікна, витяжка на кухні/у ванній кімнаті). Окремої примусової системи вентиляції або кондиціонування в кабінеті не встановлено; підтримання температури 20–22 °С здійснюється опаленням у холодний період і

провітрюванням у теплий. Під час npm run build, коли система охолодження ноутбука FX506LH працює на підвищених обертах, додатково провітрювалось приміщення для відведення тепла від корпусу ПЕОМ.

Кваліфікація (класифікація) систем вентиляції за призначенням для розглядуваного робочого місця: природна вентиляція - загальна обмінна, без механічного підсилення; призначення - підтримання свіжості повітря та зниження концентрації CO_2 при тривалій роботі однієї особи. Локальна вентиляція ПЕОМ - вбудована система охолодження ноутбука (кулери CPU/GPU, вентиляційні отвори корпусу); призначення - відведення тепла від компонентів, запобігання перегріву; кваліфікація - відповідно до паспортних даних виробника (рабоча температура CPU до 90–95 °C, throttling при перевищенні). Вимоги до організації вентиляції: не закривати вентиляційні решітки корпусу, не працювати з ноутбуком на м’якій поверхні, залишати не менше 0,5 м вільного простору позаду корпусу для циркуляції повітря; ноутбук розміщувався на підставці з відкритими отворами.

5.3 Аналіз небезпечних і шкідливих виробничих факторів

За ДСТУ 12.0.003-2015 до небезпечних і шкідливих виробничих факторів відносять фізичні, хімічні, біологічні та психофізіологічні чинники, що можуть спричинити професійне захворювання або нещасний випадок. На робочому місці розробника сайту «Віраж» домінують фактори, пов’язані з тривалою роботою за ПЕОМ.

Небезпечні фактори (можуть безпосередньо спричинити травму):

– напруга в мережі електропостачання та несправність кабелів - ризик ураження електричним струмом або загоряння;

– перегрів блоку живлення, процесора або зарядного пристрою ноутбука при тривалому npm run build - ризик опіків при дотику до гарячих поверхонь;

– пожежна небезпека через перевантаження розетки (ПК, монітор, роутер, зарядні пристрої в одній групі);

– підключення до неперевірених USB-носіїв або завантаження сумнівних

npm-пакетів - ризик компрометації системи (косвенно впливає на безпеку даних проєкту).

Шкідливі фактори (впливають на здоров'я при тривалому або неконтрольованому впливі):

– випромінювання монітора (електромагнітне поле, бликовість, мерехтіння) втома органів зору, синдром комп’ютерного зору;

– неправильна поза при тривалому наборі коду, напруження м’язів шиї, плечового пояса, зап’ясть (ризик RSI);

– монотонність та зорове навантаження при роботі зі схемою залу на 420 місць, психофізіологічна втома;

– шум від системи охолодження ПК при піковому навантаженні (production build) незначний, але тривалий фоновий шум;

– інформаційне навантаження (одночасна робота з кодом, документацією, тестами) стрес, зниження концентрації при відсутності перерв.

Хімічні та біологічні фактори на даному робочому місці практично відсутні: розробка веб-сайту не передбачає контакту зі шкідливими речовинами, пилом виробництва чи мікроорганізмами. Виключення - можливий пил у приміщенні, що впливає на чистоту повітря; рекомендується провітрювання кімнати кожні 1–2 години роботи.

Зв’язок факторів із етапами розробки сайту «Віраж» (таблиця 4.1):

– на етапі дизайну інтерфейсу (40 год) - домінує зорове навантаження, робота з кольорами секторів залу;

– на етапі розробки функціоналу (42 год) - тривалий набір коду, npm run dev, налаштування Prisma, зорове та статичне навантаження на шию і спину;

– на етапі тестування (8 год) повторювані дії миші на схемі 420 місць, ризик перевтоми зап’ястя;

– на етапі розгортання (16 год) підвищене енергоспоживання ПК, npm run build, перегрів, шум вентиляторів.

Зведений аналіз факторів, їх джерел і заходів запобігання наведено в таблиці 5.1

Таблиця 5.1 – Небезпечні та шкідливі фактори на робочому місці розробника

№	Фактор	Джерело	Можливі наслідки	Заходи запобігання
1	Електрична напруга	Мережа 220 В, блок живлення ПК	Ураження електричним струмом, пожежа	Заземлення, використання ДБЖ, не перевантажувати розетки, не працювати з відкритим корпусом під напругою
2	Перегрів обладнання	CPU/GPU ноутбука ASUS TUF Gaming F15 FX506LH під час виконання build-процесів	Опіки, зниження продуктивності (throttling)	Використання підставки з відкритими вентиляційними отворами, регулярні перерви під час тривалих збірок
3	Випромінювання та блики екрана	Екран ноутбука	Втома очей, погіршення зору, головний біль	Відстань до екрана 50–60 см, антибликове покриття, перерви 10–15 хв кожні 2 години роботи
4	Невідповідна робоча поза	Тривала робота за комп'ютером	Біль у спині та шиї, синдром повторюваних навантажень (RSI)	Регулювання висоти стільця, перерви для розминки, розташування клавіатури на рівні ліктів
5	Пожежна небезпека	Електропроводка, велика кількість підключених пристроїв	Загоряння обладнання або приміщення	Вимикати комп'ютер за тривалої відсутності, заборона куріння, наявність вогнегасника
6	Інформаційна перевтома	Програмний код, тестування, робота з документацією	Зниження концентрації уваги, збільшення кількості помилок у коді	Планування робочих сесій, чергування видів діяльності, достатній відпочинок і сон

5.4 Техніка безпеки під час роботи з ПЕОМ та програмним забезпеченням

Під час розробки сайту «Віраж» дотримувались правил охорони праці при роботі з обчислювальною технікою (ДНАОП 0.00-1.28-18) та гігієнічних норм для ВДТ (ДСН 3.3.6.042-99).

Перед початком роботи:

– перевірити справність кабелів живлення, відсутність пошкодження ізоляції;

– переконатися в достатньому освітленні робочої зони (не менше 300–500 лк на робочому столі);

– налаштувати яскравість і контраст монітора; увімкнути режим «нічної теми» в IDE за бажанням для зменшення навантаження на зір;

– переконатися в наявності резервної копії проєкту (Git) та файлу prisma/dev.db.

Під час роботи:

– тривалість безперервної роботи за ПЕОМ не більше 2 годин, далі перерва 10–15 хв (встань, розминка, відведення погляду вдалину);

– загальна тривалість роботи за ВДТ на добу при самостійній розробці не перевищувала 6–8 год із урахуванням перерв;

– заборонено вставляти сторонні USB-накопичувачі без перевірки антивірусом;

– залежності проєкту встановлювати лише з офіційного реєстру npm (package.json); не використовувати піратське або неліцензоване ПЗ;

– при роботі з персональними даними в прототипі (користувачі, бронювання)не зберігати реальні паролі в незахищених файлах; .env не комітити в Git (розділ 3);

– під час npm run build, коли процесор і система охолодження ноутбука FX506LH навантажені, не закривати вентиляційні отвори корпусу і не працювати з ноутбуком на м’якій поверхні (ковдра, подушка).

Після завершення роботи:

– коректно завершити сеанс (зупинити npm run dev, зберегти файли);

– за можливості перевести ноутбук у режим сну або вимкнути його для зменшення споживання електроенергії;

– переконатися, що вогнегасник і засоби першої допомоги доступні в

домашньому кабінеті.

Особливості роботи саме над сайтом «Віраж»:

– тестування інтерактивної схеми залу (420 місць) вимагає тривалого введення миші - рекомендовано періодично змінювати положення руки та використовувати клавіатурні скорочення в IDE;

– генерація PDF-квитків і QR-кодів на локальному сервері перевірялась у контрольованому середовищі без передачі тестових даних третім особам;

– адмін-панель (admin@virazh-hall.ua) використовувалась лише в локальній або захищеній мережі; паролі демо-облікових записів змінюються перед production-розгортанням (розділ 3).

Режим праці та перерви при розробці сайту «Віраж» наведено в таблиці 5.2; рекомендовані вправи під час перерви - у таблиці 5.3.

Таблиця 5.2 – Рекомендований режим праці за ПЕОМ під час розробки

Етап робочого дня	**Тривалість**	**Зміст роботи**	**Перерва після завершення**
Ранковий сеанс	2 год	Розробка програмного коду, робота з Prisma та API	15 хв (розминка, провітрювання приміщення)
Денний сеанс	2 год	Розробка інтерфейсу користувача, робота зі схемою залу	15 хв (вправи для очей, відпочинок від екрана)
Вечірній сеанс	1,5–2 год	Тестування програмного забезпечення, підготовка документації	10 хв відпочинку
Максимальна тривалість роботи на добу	6–8 год	Виконання всіх етапів розробки з урахуванням регламентованих перерв	-

Таблиця 5.3 – Вправи під час регламентованої перерви (10–15 хв)

№	**Вправа**	**Мета**
1	2	3
1	Відведення погляду у вікно на відстань 6–10 м протягом 20 с	Зняття спазму акомодації та зменшення втоми очей
2	Оберти плечима та нахили голови	Зняття напруження м'язів шиї та плечового пояса

1	2	3
3	Розгинання та згинання зап'ястя	Профілактика синдрому повторюваних навантажень (RSI)
4	Ходіння по кімнаті протягом 2–3 хвилин	Поліпшення кровообігу та зменшення статичного навантаження
5	Провітрювання приміщення	Забезпечення надходження свіжого повітря та зниження концентрації CO_2

Дотримання режиму таблиці 5.2 та вправ таблиці 5.3 застосовувалось під час усіх 159 годин розробки.

5.5 Електробезпека та пожежна безпека

Робоча станція розробника (ноутбук FX506LH + зарядний адаптер 150 Вт) підключена до однофазної мережі 220 В / 50 Гц. Споживана потужність у режимі розробки (npm run dev, VS Code, браузер) оцінюється приблизно 0,15–0,20 кВт; при npm run build - до 0,30 кВт (узгоджено з розрахунком електроенергії в п. 4.3, амортизація ноутбука 38 900 грн). Загальна потужність групи розетки не повинна перевищувати допустимого навантаження домашньої проводки.

Вимоги електробезпеки:

– підключати обладнання лише до справних розеток з заземленням (контакт «земля»);

– не допускати мокрого приміщення біля ПК та розеток;

– не ремонтувати блок живлення та системний блок під напругою;

– використовувати мережевий фільтр або ДБЖ з захистом від перенапруги;

– при поодинокій роботі вдома - мати у доступі номер екстрених служб (101, 103, 104).

Вимоги пожежної безпеки:

– не залишати увімкненим ПК без нагляду на тривалий час при високому навантаженні (build);

– тримати навколо системного блока мінімум 0,5 м вільного простору

для циркуляції повітря;

– не розміщувати на системному блоці папір, тканини, легкозаймисті предмети;

– мати в кімнаті первинний засіб пожежогасіння (порошковий вогнегасник ОП-2 або аналог) та знати порядок його застосування;

– заборонено куріння та використання відкритого вогню в зоні розташування ПЕОМ.

При виникненні запаху горілого пластику або іскіння - негайно відключити ПК від мережі, за можливості перервати живлення автоматом, повідомити дорослих / сусідів, викликати ДСН за потреби. При ураженні електричним струмом - відключити джерело напруги, не торкатися потерпілого голими руками до ізоляції, викликати швидку допомогу.

5.6 Екологічні вимоги при розробці та експлуатації сайту «Віраж»

Розробка програмного забезпечення за своєю природою належить до маловідходних видів діяльності: основний «продукт» - цифровий код і база даних, що не генерує промислових стоків і газопилових викидів на робочому місці. Водночас екологічний аспект кваліфікаційної роботи включає енергоспоживання, утворення електронних відходів, вплив майбутньої експлуатації сайту та відповідальне поводження з даними.

Енергоспоживання. За 159 годин розробки спожито близько 47,7 кВт·год, що у грошовому виразі становить 760 грн при тарифі 15,94 грн/кВт·год. Для зменшення енергоспоживання застосовувались: вимкнення монітора при перервах, режим енергозбереження ОС, завершення процесів npm після завершення сеансу, уникнення одночасного запуску кількох важких build-процесів.

Утворення відходів. Паперова документація мінімізована: пояснювальна записка готується в електронному вигляді з друком лише необхідного примірника. Тонерні картриджі принтера (якщо використовується) підлягають

збору на утилізацію через пункти прийому відходів у м. Тернопіль. При заміні ноутбука, кабелів, периферії - передача на спеціалізований збір електронних відходів згідно Закону України «Про управління відходами»; не допускається скидання техніки на звичайні полігони.

Екологічний ефект функціонування сайту «Віраж» після запуску залу:

– електронне бронювання з PDF-квитком зменшує потребу в друку паперових квитків і черг у касі - економія паперу та палив на доставку глядачів «навмання»;

– онлайн-афіша знижує обсяг друкованих афіш і рекламних листівок;

– централізована SQLite/PostgreSQL замість роздроблених паперових журналів спрощує облік без дублювання документів.

Обмеження та заходи при production-розгортанні (розділ 3):

– вибір хостингу з дата-центром, що використовує відновлювані джерела енергії або енергоефективне обладнання (Vercel, зелений хостинг VPS);

– оптимізація зображень постерів для зменшення трафіку та навантаження на сервери;

– регулярне видалення застарілих логів і резервних копій, що не потрібні для роботи;

– захист персональних даних користувачів (хешування паролів bcrypt, HTTPS) - запобігання «цифровому сміттю» з витоків і повторної обробки скомпрометованих баз.

Зведені екологічні заходи наведено в таблиці 5.4.

Таблиця 5.4 – Екологічні заходи при розробці та експлуатації сайту «Віраж»

№	Напрям	Захід	Очікуваний ефект
1	2	3	4
1	Енергозбереження	Регламентовані перерви, вимкнення ПК після завершення роботи, використання режимів енергозбереження	Зниження споживання електроенергії на 10–15 % порівняно з неконтрольованим режимом роботи

Продовження таблиці 5.4

1	2	3	4
2	Управління відходами	Використання електронного документообігу, належна утилізація картриджів та застарілої комп'ютерної техніки	Мінімізація утворення твердих побутових та електронних відходів
3	Транспортна оптимізація	Використання онлайн-бронювання квитків замість особистого відвідування каси	Зменшення викидів CO_2 від транспортних засобів відвідувачів
4	Економія паперу	Використання електронних PDF-квитків замість друкованих (за згодою користувача)	Скорочення витрат паперу на кожне бронювання
5	Оптимізація серверних ресурсів	Оптимізація зображень, використання SQLite/PostgreSQL без дублювання даних	Зменшення використання дискового простору та енергоспоживання центрів обробки даних
6	Управління даними	Використання HTTPS, регулярне резервне копіювання, видалення тестових і застарілих записів	Запобігання накопиченню надлишкових даних та підвищення ефективності обробки інформації

Орієнтовний порівняльний ефект електронного та паперового обігу квитків для залу на 420 місць (за повного заповнення одного концерту) наведено в таблиці 5.5. Розрахунок умовний: один паперовий квиток - близько 5 г паперу; PDF-квиток на екрані смартфона не потребує друку; афіша формату А3 - близько 10 г паперу на примірник. Використання електронних квитків дозволяє значно скоротити споживання. Крім того, зменшується кількість відходів після проведення заходів, а користувачі отримують можливість швидкого доступу до квитка через мобільний пристрій. Перехід до електронного документообігу також сприяє зниженню негативного впливу на навколишнє середовище та відповідає сучасним принципам цифровізації культурно-мистецької сфери. Порівняння паперового та електронного обігу наведено в таблиці 5.5

Таблиця 5.5 – Умовне порівняння паперового та електронного обігу

Показник	**Паперовий облік**	**Сайт «Віраж»**	**Економія / Екологічний ефект**
Квитки	420 × 5 г ≈ 2,1 кг паперу	PDF-квитки на смартфоні або електронній пошті	Економія до 2,1 кг паперу на один захід
Афіші	50 × 10 г = 0,5 кг паперу	Сторінка події на вебсайті та в соціальних мережах	Економія до 0,5 кг паперу
Журнал бронювань	Паперовий журнал близько 100 г на подію	Електронне зберігання даних у SQLite/PostgreSQL	Практично повна відмова від використання паперу
Поїздки глядачів	Часті додаткові поїздки для отримання інформації або купівлі квитків	Онлайн-перегляд інформації та бронювання	Зменшення викидів CO_2 від транспортних засобів

5.7 Перша допомога та дії при нещасних випадках

Нещасні випадки на робочому місці програміста найчастіше пов’язані з ураженням електричним струмом, опіками від перегрітого обладнання, пожежами через несправну проводку або різким погіршенням зору/самопочуття після тривалої роботи за ВДТ. Під час 159 годин розробки сайту «Віраж» нещасних випадків не зафіксовано; нижче наведено алгоритм дій, яким керувався виконавець.

При ураженні електричним струмом: негайно відключити джерело напруги (вимкнути з розетки, зняти автомат); не торкатися потерпілого, поки він під напругою; викликати швидку допомогу (103); при відсутності дихання серцево-легенева реанімація до приїзду медиків.

При опіках від перегрітого блоку живлення: охолодити уражену ділянку прохолодною водою 10–15 хв; не проколювати пухирі; при опіках II–III ступеня звернутися до лікаря.

При пожежі: повідомити про пожежу (101); евакуюватися; гасити

первинними засобами (вогнегасник ОП-2), якщо вогонь невеликий і не небезпечно; не гасити водою живий ПК під напругою.

При різкому погіршенні зору, головокрутінні, болю в шиї: припинити роботу за ПЕОМ; зробити перерву не менше 15–30 хв; при стійких симптомах - звернутися до лікаря-офтальмолога або терапевта.

На робочому місці рекомендовано мати аптечку першої допомоги (бинт, антисептик, пластир), вогнегасник та записані телефони екстрених служб 101, 103, 104.

Крім того, працівник повинен знати місце розташування найближчого евакуаційного виходу та вміти користуватися первинними засобами пожежогасіння. У разі виникнення аварійної ситуації необхідно зберігати спокій, діяти відповідно до інструкцій з охорони праці та не наражати себе на додаткову небезпеку. Регулярне дотримання правил безпеки, контроль технічного стану обладнання та своєчасне усунення виявлених несправностей дозволяють значно знизити ризик виникнення нещасних випадків під час роботи над програмним забезпеченням. Особливу увагу слід приділяти профілактиці професійної втоми, оскільки перевтома може призводити до зниження концентрації уваги, збільшення кількості помилок у роботі та погіршення загального стану здоров'я. Таким чином, виконання вимог охорони праці є важливою складовою безпечної та ефективної розробки інформаційної системи «Віраж».

ВИСНОВКИ

У кваліфікаційній роботі виконано розробку веб-сайту бронювання квитків для концертного залу «Віраж» у м. Тернопіль. Створено full-stack застосунок на базі Next.js, React, TypeScript, Prisma ORM і SQLite, який забезпечує публічну афішу подій, інтерактивну схему залу на 420 місць, оформлення бронювання, генерацію PDF-квитка з QR-кодом, особистий кабінет користувача, чат підтримки, відгуки та адміністративну панель. Прототип є демонстраційно-експлуатаційним рішенням на період до капітального відкриття залу й працює як локально на робочій станції розробника, так і в мережі Internet на хмарній платформі Railway.

На етапі аналітичного дослідження розглянуто діяльність білетних платформ і сайтів культурних майданчиків. Обґрунтовано доцільність власного internet-ресурсу для залу «Віраж»: він дозволяє зберегти контроль над брендом, правилами відвідування та моделлю продажу квитків, не залежати від комісій сторонніх агрегаторів і реалізувати інтерактивну схему залу під конкретну конфігурацію на 420 місць. На основі проведеного аналізу сформовано технічне завдання з визначенням функціональних і нефункціональних вимог, структури сторінок, вимог до інтерфейсу, безпеки та документації. У завданні зафіксовано обмеження поточної версії: оплата в касі залу без онлайн-платіжного шлюзу, файлова база SQLite та демонстраційні облікові записи для перевірки системи.

У технічному та робочому проєкті спроєктовано архітектуру застосунку з виділенням публічної частини, особистого кабінету, адміністративної панелі та серверних API-маршрутів. Побудовано логічну модель даних, що охоплює користувачів, події, місця, бронювання, чати та відгуки. Реалізовано клієнтську частину з інтерактивною схемою залу, адаптивною верстkою публічних сторінок і автентифікацією через NextAuth. На серверному рівні забезпечено обробку бронювань, генерацію PDF-квитка, керування подіями, модерацію відгуків і роботу чату підтримки. Запобігання подвійному продажу одного місця реалізовано за допомогою тимчасового утримання місць і транзакцій бази даних.

Функціональне та інтеграційне тестування за контрольними сценаріями підтвердило працездатність основних функцій прототипу.

У спеціальному розділі підготовлено інструкції з експлуатації сайту: визначено вимоги до програмно-апаратного середовища, описано локальне та мережеве розгортання, налаштування змінних середовища, обслуговування бази даних SQLite, наповнення афіші через адмін-панель, заходи пошукової оптимізації та популяризації ресурсу. Сайт опубліковано на платформі Railway і доступний за публічною адресою в мережі Internet, що дозволяє демонструвати його роботу комісії та відвідувачам без обмеження локальним комп'ютером.

Економічне обґрунтування розробки виконано для проєктної команди з трудомісткістю 159 годин. За результатами розрахунків собівартість науково-дослідної роботи становить 62 044 грн, договірна ціна - 96 788 грн. Для майбутнього оператора залу розраховано річний грошовий потік 34 744 грн, чисту теперішню вартість 24 377 грн і термін окупності 2,1 року. Отримані показники, наведені в таблиці 4.5, підтверджують доцільність створення власного каналу бронювання після введення залу в експлуатацію.

У розділі з охорони праці проаналізовано умови роботи розробника на домашньому робочому місці в м. Тернопіль, визначено небезпечні та шкідливі фактори при роботі за відеодисплейним терміналом, заходи з техніки безпеки, електро- та пожежної безпеки, а також екологічні аспекти енергоспоживання та електронізації квиткового обігу.

Метою кваліфікаційної роботи було створення цілісного веб-сайту залу «Віраж» із повним сценарієм для глядача та адміністратора - ця мета досягнута. Практичне значення результатів полягає в тому, що розроблений сайт є діючим програмним продуктом, який може бути представлений замовникам майбутнього концертного залу як модель цифрового сервісу. Вихідний код структуровано за каталогами app, components, lib і prisma, що спрощує подальший супровід, а інструкції з розділу 3 дозволяють розгорнути та підтримувати прототип на початковому етапі експлуатації.

Обмеження поточної версії свідомо зафіксовані в технічному завданні:

відсутність онлайн-оплати банківською карткою, залежність від файлової SQLite при високому паралельному навантаженні та оновлення чату підтримки за принципом polling замість WebSocket. Ці обмеження не заважають демонстрації основного функціоналу, але визначають напрями подальшої модернізації - підключення платіжного шлюзу, міграцію на PostgreSQL, розширення аналітики для адміністрації залу та вдосконалення мобільної доступності ресурсу.

Таким чином, у кваліфікаційній роботі створено веб-сайт концертного залу «Віраж», який відповідає вимогам технічного завдання, пройшов тестування, має інструкції з експлуатації, економічно обґрунтований для оператора залу та розроблений з дотриманням вимог охорони праці й екології. Результати роботи мають практичну цінність для майбутнього впровадження цифрового бронювання в концертному залі «Віраж» у м. Тернопіль..

ПЕРЕЛІК ПОСИЛАНЬ

1) ДСТУ 3008:2015. Інформація та документація. Звіти у сфері науки і техніки. Структура та правила оформлювання. К. : ДП «УкрНДНЦ», 2016. 17 с.

2) Марціяш Г.Я., Слободян Р.О. Методичні вказівки до виконання кваліфікаційної роботи для здобувачів фахової передвищої освіти спеціальності 122 «Комп'ютерні науки» : навч. посіб. Тернопіль : ВСП «ТФК ТНТУ ім. І. Пулюя», 2026. 48 с.

3) Next.js Documentation : вебсайт. URL: https://nextjs.org/docs (дата звернення: 15.03.2026).

4) React : офіційна документація : вебсайт. URL: https://react.dev (дата звернення: 15.03.2026).

5) Prisma Documentation : вебсайт. URL: https://www.prisma.io/docs (дата звернення: 15.03.2026).

6) Auth.js (NextAuth.js) Documentation : вебсайт. URL: https://authjs.dev (дата звернення: 15.03.2026).

7) Concert.ua : офіційний вебсайт сервісу продажу квитків : вебсайт. URL:. https://concert.ua (дата звернення: 15.03.2026).

8) Ticketmaster : офіційний вебсайт міжнародної білетної платформи : вебсайт. URL: https://www.ticketmaster.com (дата звернення: 15.03.2026).

9) OWASP Top Ten : перелік основних ризиків безпеки веб-застосунків : вебсайт. URL: https://owasp.org/www-project-top-ten/ (дата звернення: 15.03.2026).

10) IEEE Std 830-1998. IEEE Recommended Practice for Software Requirements Specifications : вебсайт. URL: https://standards.ieee.org/standard/830-1998.html (дата звернення: 15.03.2026).

11) SQLite Documentation : вебсайт. URL: https://www.sqlite.org/docs.html (дата звернення: 15.03.2026).

12) TypeScript Handbook:вебсайт. URL: https://www.typescriptlang.org/docs/ (дата звернення: 15.03.2026).

13) Tailwind CSS Documentation: вебсайт. URL: https://tailwindcss.com/docs

(дата звернення: 15.03.2026).

14) pdf-lib : бібліотека генерації PDF-документів : вебсайт. URL: https://pdf-lib.js.org/ (дата звернення: 15.03.2026).

15) Web Content Accessibility Guidelines (WCAG) 2.1 : вебсайт. URL: https://www.w3.org/TR/WCAG21/ (дата звернення: 15.03.2026).

16) Vercel Documentation : вебсайт. URL: https://vercel.com/docs (дата звернення: 15.03.2026).

17) bcryptjs : документація npm-пакета : вебсайт. URL: https://www.npmjs.com/package/bcryptjs (дата звернення: 15.03.2026).

18) Karabas Live : офіційний вебсайт сервісу продажу квитків : вебсайт. URL: https://www.karabas.com (дата звернення: 15.03.2026).

19) ДСТУ 8302:2015. Інформація та документація. Бібліографічне посилання. Загальні положення та правила складання. К. : ДП «УкрНДНЦ», 2016. 16 с.

20) Закон України «Про охорону праці» : вебсайт. URL: https://zakon.rada.gov.ua/laws/show/2694-12 (дата звернення: 05.06.2026).

21) Закон України «Про охорону навколишнього природного середовища» : вебсайт. URL: https://zakon.rada.gov.ua/laws/show/1624-14 (дата звернення: 05.06.2026).

22) ДСН 3.3.6.042-99. Гігієнічні вимоги до роботи з відеодисплейними терміналами : вебсайт. URL: https://zakon.rada.gov.ua/go/va042282-99 (дата звернення: 05.06.2026).

23) ДНАОП 0.00-1.28-18. Правила охорони праці під час роботи з обчислювальною технікою : вебсайт. URL: https://zakon.rada.gov.ua/go/z0293-10 (дата звернення: 05.06.2026).

24) ДСТУ 12.0.003-2015. Система стандартів з безпеки праці. Безпека праці. Основні положення. К. : ДП «УкрНДНЦ», 2015. 8 с.

ДОДАТКИ

Додаток А. Лістинг файлу route.ts

```
import { NextResponse } from "next/server";
import { z } from "zod";
import { auth } from "@/auth";
import {
  MAX_SEATS_MESSAGE,
  MAX_SEATS_PER_BOOKING,
} from "@/lib/booking-limits";
import { prisma } from "@/lib/prisma";
import { formatDate } from "@/lib/format";
import { resolveUserId } from "@/lib/resolve-user-id";
import { generateTicketPdf } from "@/lib/ticket-pdf";
import { sendTicketEmail } from "@/lib/email";
import type { SeatSection } from "@/lib/types";

const schema = z.object({
  eventId: z.string(),
  seatIds: z.array(z.string()).min(1).max(MAX_SEATS_PER_BOOKING),
  guestName: z.string().min(2),
  guestEmail: z.string().email(),
  guestPhone: z.string().optional(),
});

export async function POST(req: Request) {
  try {
    let userId: string | null = null;
    try {
      const session = await auth();
      userId = await resolveUserId(session);
    } catch (authErr) {
      console.warn("[booking] auth skipped:", authErr);
```

```
    }

    let body: unknown;
    try {
      body = await req.json();
    } catch {
      return NextResponse.json({ error: "Невірний формат запиту" }, { status: 400 });
    }

    const parsed = schema.safeParse(body);
    if (!parsed.success) {
      const tooMany = Array.isArray((body as { seatIds?: unknown })?.seatIds)
        && (body as { seatIds: unknown[] }).seatIds.length > MAX_SEATS_PER_BOOKING;
      return NextResponse.json(
        { error: tooMany ? MAX_SEATS_MESSAGE : "Перевірте ім'я, email і місця" },
        { status: 400 },
      );
    }

    const { eventId, seatIds, guestName, guestEmail, guestPhone } = parsed.data;
    const email = guestEmail.toLowerCase();

    const event = await prisma.event.findUnique({ where: { id: eventId } });
    if (!event) {
      return NextResponse.json({ error: "Подію не знайдено" }, { status: 404 });
    }
```

```
const seatRows = await prisma.seat.findMany({
  where: { id: { in: seatIds } },
});
if (seatRows.length !== seatIds.length) {
  return NextResponse.json(
    { error: "Деякі місця не знайдено в залі. Оновіть схему." },
    { status: 400 },
  );
}

const total = seatRows.reduce((s, seat) => s + seat.basePrice, 0);
const bookingId = `VRZ-${Date.now().toString(36).toUpperCase()}`;

await prisma.$transaction(async (tx) => {
  const eventSeats = await tx.eventSeat.findMany({
    where: { eventId, seatId: { in: seatIds } },
  });

  if (eventSeats.length !== seatIds.length) {
    throw new Error("NOT_FOUND");
  }

  const now = new Date();
  for (const es of eventSeats) {
    if (es.status === "sold") throw new Error("SOLD");
    if (
      es.status === "held" &&
      es.heldUntil &&
      es.heldUntil < now
    ) {
      continue;
    }
    if (es.status === "held" && es.heldUntil && es.heldUntil >= now) {
      continue;
```

```
      }
    }

    await tx.eventSeat.updateMany({
      where: { eventId, seatId: { in: seatIds } },
      data: { status: "sold", heldUntil: null },
    });

    await tx.booking.create({
      data: {
        id: bookingId,
        eventId,
        userId: userId ?? null,
        guestName,
        guestEmail: email,
        guestPhone: guestPhone?.trim() || null,
        total,
        status: "confirmed",
        paymentNote: "pay_at_venue",
        seats: JSON.stringify(seatIds),
      },
    });
  });

  let pdfBytes: Uint8Array;
  try {
    pdfBytes = await generateTicketPdf({
      bookingId,
      eventTitle: event.title,
      artist: event.artist,
      date: formatDate(event.date),
      time: event.time,
      guestName,
      guestEmail: email,
```

```
        guestPhone: guestPhone?.trim() || null,
        total,
        seats: seatRows.map((s) => ({
          section: s.section as SeatSection,
          row: s.row,
          number: s.number,
          price: s.basePrice,
        })),
      });
    } catch (pdfErr) {
      console.error("[booking] PDF error:", pdfErr);
      return NextResponse.json(
        {
          error: "Бронь створено, але PDF не згенеровано. Завантажте з
профілю.",
          bookingId,
          total,
          emailSent: false,
        },
        { status: 201 },
      );
    }

    const mail = await sendTicketEmail({
      to: email,
      subject: `Квиток «Віраж» - ${event.title}`,
      html: `
      <div style="font-family:sans-serif;max-width:480px">
        <h2 style="color:#e85d2a">Концертний зал «Віраж»</h2>
        <p>Вітаємо, <strong>${guestName}</strong>!</p>
        <p>Бронювання <strong>${bookingId}</strong> підтверджено.</p>
        <p><strong>${event.title}</strong><br/>${formatDate(event.date)}  о
${event.time}</p>
        <p>PDF-квиток у вкладенні. Оплата в касі залу до початку події.</p>
```

```
      </div>
    `,
      pdfBytes,
      bookingId,
    });

    return NextResponse.json({
      bookingId,
      total,
      emailSent: mail.sent,
      devTicketUrl: mail.devPath ?? `/api/bookings/${bookingId}/ticket`,
    });
  } catch (e) {
    const msg = e instanceof Error ? e.message : "";
    console.error("[booking] POST error:", e);
    if (msg === "SOLD") {
      return NextResponse.json(
        { error: "Місця вже зайняті. Оновіть схему та оберіть інші." },
        { status: 409 },
      );
    }
    if (msg === "NOT_FOUND") {
      return NextResponse.json(
        { error: "Місця не знайдено. Запустіть npm run db:setup" },
        { status: 400 },
      );
    }
    const hint =
      process.env.NODE_ENV === "development" && e instanceof Error
        ? e.message.slice(0, 200)
        : undefined;
    return NextResponse.json(
      {
        error:
```

```
          "Помилка сервера. Зупиніть старий dev-сервер, виконайте npm run
db:setup і npm run dev на http://localhost:3000",
        hint,
      },
      { status: 500 },
    );
  }
}

export async function GET() {
  const session = await auth();
  const userId = await resolveUserId(session);
  if (!userId) {
    return NextResponse.json({ error: "Unauthorized" }, { status: 401 });
  }

  const bookings = await prisma.booking.findMany({
    where: { userId },
    include: { event: true },
    orderBy: { createdAt: "desc" },
  });

  return NextResponse.json({ bookings });
}
```

Додаток Б. Лістинг файлу HallMap.tsx

```
"use client";

import { useCallback, useEffect, useRef, useState } from "react";
import { HallMapSvg } from "@/components/seating/HallMapSvg";
import { Loader2, Minus, Plus, RotateCcw } from "lucide-react";
import type { Seat } from "@/lib/types";
import { VIEW_H, VIEW_W } from "@/lib/seats";
import { formatPrice } from "@/lib/format";
import { formatSeatLabel } from "@/lib/format-seat";
import {
  MAX_SEATS_MESSAGE,
  MAX_SEATS_PER_BOOKING,
} from "@/lib/booking-limits";

const MIN_SCALE = 0.5;
const MAX_SCALE = 2.2;

interface HallMapProps {
  eventId: string;
  onSelectionChange?: (seats: Seat[]) => void;
}

export function HallMap({ eventId, onSelectionChange }: HallMapProps) {
  const [seats, setSeats] = useState<Seat[]>([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState<string | null>(null);
  const [selectedIds, setSelectedIds] = useState<Set<string>>(new Set());
  const [hoveredId, setHoveredId] = useState<string | null>(null);
  const [scale, setScale] = useState(0.72);
  const [pan, setPan] = useState({ x: 0, y: 0 });
  const [fitDone, setFitDone] = useState(false);
  const [limitHint, setLimitHint] = useState<string | null>(null);
```

```
  const drag = useRef<{ active: boolean; sx: number; sy: number; px: number; py: number }>({
    active: false,
    sx: 0,
    sy: 0,
    px: 0,
    py: 0,
  });
  const viewportRef = useRef<HTMLDivElement>(null);
  const onSelectionChangeRef = useRef(onSelectionChange);

  useEffect(() => {
    onSelectionChangeRef.current = onSelectionChange;
  }, [onSelectionChange]);

  const loadSeats = useCallback(async () => {
    setLoading(true);
    setError(null);
    try {
      const res = await fetch(`/api/events/${eventId}/seats`);
      if (!res.ok) throw new Error("Помилка завантаження");
      const data = (await res.json()) as { seats: Seat[] };
      setSeats(data.seats);
    } catch {
      setError("Не вдалося завантажити схему. Перевірте, чи запущена база (npm run db:seed).");
    } finally {
      setLoading(false);
    }
  }, [eventId]);

  useEffect(() => {
    loadSeats();
  }, [loadSeats]);
```

```
  useEffect(() => {
    if (loading || fitDone || !viewportRef.current) return;
    const el = viewportRef.current;
    const pad = 32;
    const fit = Math.min(
      (el.clientWidth - pad) / VIEW_W,
      (el.clientHeight - pad) / VIEW_H,
      1,
    );
    setScale(Math.max(MIN_SCALE, fit * 0.96));
    setFitDone(true);
  }, [loading, fitDone]);

  useEffect(() => {
    setFitDone(false);
  }, [eventId]);

  const selected = seats.filter((s) => selectedIds.has(s.id));

  // Повідомляємо батьківський компонент лише коли змінюється вибір (Set),
не новий масив `selected` щоразу
  useEffect(() => {
    const picked = seats.filter((s) => selectedIds.has(s.id));
    onSelectionChangeRef.current?.(picked);
  }, [selectedIds, seats]);

  const toggleSeat = (id: string) => {
    const seat = seats.find((s) => s.id === id);
    if (!seat || seat.status === "sold" || seat.status === "held") return;

    setSelectedIds((prev) => {
      const next = new Set(prev);
      if (next.has(id)) {
```

```
        next.delete(id);
        setLimitHint(null);
      } else if (next.size >= MAX_SEATS_PER_BOOKING) {
        setLimitHint(MAX_SEATS_MESSAGE);
      } else {
        next.add(id);
        setLimitHint(null);
      }
      return next;
    });
  };

  const zoomBy = (delta: number) => {
    setScale((s) => Math.min(MAX_SCALE, Math.max(MIN_SCALE, s + delta)));
  };

  const resetView = () => {
    const el = viewportRef.current;
    if (el) {
      const pad = 32;
      const fit = Math.min(
        (el.clientWidth - pad) / VIEW_W,
        (el.clientHeight - pad) / VIEW_H,
        1,
      );
      setScale(Math.max(MIN_SCALE, fit * 0.96));
    } else {
      setScale(0.72);
    }
    setPan({ x: 0, y: 0 });
  };

  const onPointerDown = (e: React.PointerEvent) => {
    if ((e.target as HTMLElement).closest("[data-seat]")) return;
```

```
    drag.current = {
      active: true,
      sx: e.clientX,
      sy: e.clientY,
      px: pan.x,
      py: pan.y,
    };
    (e.currentTarget as HTMLElement).setPointerCapture(e.pointerId);
  };

  const onPointerMove = (e: React.PointerEvent) => {
    if (!drag.current.active) return;
    setPan({
      x: drag.current.px + (e.clientX - drag.current.sx),
      y: drag.current.py + (e.clientY - drag.current.sy),
    });
  };

  const onPointerUp = () => {
    drag.current.active = false;
  };

  const onWheel = (e: React.WheelEvent) => {
    e.preventDefault();
    zoomBy(e.deltaY > 0 ? -0.12 : 0.12);
  };

  if (loading) {
    return (
      <div className="flex h-[480px] items-center justify-center rounded-xl
border border-card-border bg-card">
        <Loader2 className="h-8 w-8 animate-spin text-success" />
      </div>
    );
```

```
  }

  if (error) {
    return (
      <div className="rounded-xl border border-danger/30 bg-danger/10 p-6
text-center text-sm text-danger">
        {error}
        <button
          type="button"
          onClick={loadSeats}
          className="mt-3 block w-full text-accent underline"
        >
          Спробувати знову
        </button>
      </div>
    );
  }

  return (
    <div className="w-full space-y-3">
      <div  className="flex  flex-wrap  items-center  justify-between  gap-2
rounded-t-xl border border-b-0 border-card-border bg-card px-3 py-2">
        <p className="text-xs text-muted">
          Прокрутіть коліщатком або кнопками для масштабу · перетягніть
схему
        </p>
        <div className="flex gap-1">
          <button
            type="button"
            onClick={() => zoomBy(-0.2)}
            className="rounded-lg  border  border-card-border  bg-background
p-2 hover:bg-card"
            aria-label="Зменшити"
          >
```

```
            <Minus className="h-4 w-4 text-foreground" />
          </button>
          <span  className="flex  min-w-[3rem]  items-center  justify-center
text-xs font-medium text-foreground">
            {Math.round(scale * 100)}%
          </span>
          <button
            type="button"
            onClick={() => zoomBy(0.2)}
            className="rounded-lg  border  border-card-border  bg-background
p-2 hover:bg-card"
            aria-label="Збільшити"
          >
            <Plus className="h-4 w-4 text-foreground" />
          </button>
          <button
            type="button"
            onClick={resetView}
            className="rounded-lg  border  border-card-border  bg-background
p-2 hover:bg-card"
            aria-label="Скинути"
          >
            <RotateCcw className="h-4 w-4 text-foreground" />
          </button>
        </div>
      </div>

      <div
        ref={viewportRef}
        className="relative h-[min(78vh,620px)] cursor-grab overflow-hidden
rounded-b-xl border border-card-border bg-card active:cursor-grabbing"
        onWheel={onWheel}
        onPointerDown={onPointerDown}
        onPointerMove={onPointerMove}
```

```
        onPointerUp={onPointerUp}
        onPointerLeave={onPointerUp}
      >
        <div
          className="absolute left-1/2 top-1/2 origin-center transition-transform duration-75"
          style={{
            transform: `translate(calc(-50% + ${pan.x}px), calc(-50% + ${pan.y}px)) scale(${scale})`,
          }}
        >
          <HallMapSvg
            seats={seats}
            selectedIds={selectedIds}
            hoveredId={hoveredId}
            interactive
            onSeatClick={toggleSeat}
            onSeatHover={setHoveredId}
          />
        </div>
      </div>

      {limitHint && (
        <p className="rounded-lg border border-danger/30 bg-danger/10 px-3 py-2 text-xs text-danger">
          {limitHint}
        </p>
      )}

      {selected.length > 0 && (
        <div className="rounded-xl border border-accent/30 bg-card p-4 shadow-sm">
          <p className="text-sm font-medium text-accent">
            Обрано: {selected.length} / {MAX_SEATS_PER_BOOKING}
```

```
          </p>
          <ul  className="mt-2  max-h-28  space-y-1  overflow-y-auto  text-sm
text-muted">
            {selected.map((s) => (
              <li key={s.id}>
                {formatSeatLabel(s.id)} - {formatPrice(s.price)}
              </li>
            ))}
          </ul>
        </div>
      )}
    </div>
  );
}
```

Додаток В. Лістинг файлу ticket-pdf.ts

```
import QRCode from "qrcode";
import { PDFDocument, rgb, StandardFonts } from "pdf-lib";
import { SECTION_LABELS } from "@/lib/seats";
import type { SeatSection } from "@/lib/types";
import { toPdfSafe } from "@/lib/transliterate";

export interface TicketData {
  bookingId: string;
  eventTitle: string;
  artist: string;
  date: string;
  time: string;
  guestName: string;
  guestEmail: string;
  guestPhone?: string | null;
  total: number;
  seats: { section: SeatSection; row: number; number: number; price: number
}[];
}

export     async     function     generateTicketPdf(data:     TicketData):
Promise<Uint8Array> {
  const doc = await PDFDocument.create();
  const page = doc.addPage([420, 620]);
  const font = await doc.embedFont(StandardFonts.Helvetica);
  const fontBold = await doc.embedFont(StandardFonts.HelveticaBold);

  const draw = (
    text: string,
    x: number,
    y: number,
    size: number,
```

```
  bold = false,
) => {
  page.drawText(toPdfSafe(text), {
    x,
    y,
    size,
    font: bold ? fontBold : font,
    color: rgb(0.12, 0.14, 0.18),
  });
};

page.drawRectangle({
  x: 0,
  y: 545,
  width: 420,
  height: 75,
  color: rgb(0.91, 0.36, 0.16),
});
draw("VIRAZH CONCERT HALL", 36, 580, 13, true);
draw("Elektronnij kvytok / E-ticket", 36, 562, 9);

const qrPayload = JSON.stringify({
  id: data.bookingId,
  event: data.eventTitle,
  guest: data.guestName,
  total: data.total,
});
const qrPng = await QRCode.toBuffer(qrPayload, {
  width: 200,
  margin: 1,
  errorCorrectionLevel: "M",
});
const qrImage = await doc.embedPng(qrPng);
page.drawImage(qrImage, { x: 280, y: 60, width: 110, height: 110 });
```

```
  draw(`Booking ID: ${data.bookingId}`, 36, 520, 10);
  draw(data.eventTitle, 36, 498, 14, true);
  draw(data.artist, 36, 480, 10);
  draw(`${data.date}  ${data.time}`, 36, 462, 10);
  draw(`Guest: ${data.guestName}`, 36, 438, 11, true);
  if (data.guestPhone) draw(`Phone: ${data.guestPhone}`, 36, 422, 10);
  draw(data.guestEmail, 36, 406, 9);

  draw(`Tickets: ${data.seats.length}`, 36, 378, 11, true);
  let y = 360;
  for (const s of data.seats) {
    const section = SECTION_LABELS[s.section] ?? s.section;
    draw(
      `${toPdfSafe(section)}, row ${s.row}, seat ${s.number}  -  ${s.price}
UAH`,
      36,
      y,
      9,
    );
    y -= 16;
  }

  draw(`Total: ${data.total} UAH`, 36, y - 8, 12, true);
  draw("Pay at Virazh box office before the event", 36, y - 28, 8);
  draw("Show this PDF or QR code at the entrance", 36, y - 42, 8);

  return doc.save();
}
```

Додаток Г. Лістинг файлу seed.ts

```
import { PrismaClient } from "@prisma/client";
import bcrypt from "bcryptjs";
import { buildHallSeatTemplate } from "../src/lib/seats";
import { events, reviews } from "../src/lib/mock-data";

const prisma = new PrismaClient();

async function main() {
  await prisma.booking.deleteMany();
  await prisma.session.deleteMany();
  await prisma.account.deleteMany();
  await prisma.eventSeat.deleteMany();
  await prisma.event.deleteMany();
  await prisma.seat.deleteMany();
  await prisma.supportChatMessage.deleteMany();
  await prisma.supportChat.deleteMany();
  await prisma.review.deleteMany();
  await prisma.user.deleteMany();
  await prisma.hall.deleteMany();

  await prisma.hall.create({
    data: { id: "virazh", name: "Віраж", city: "Тернопіль" },
  });

  const template = buildHallSeatTemplate();
  await prisma.seat.createMany({
    data: template.map((s) => ({
      id: s.id,
      hallId: "virazh",
      section: s.section,
      row: s.row,
      number: s.number,
```

```
      x: s.x,
      y: s.y,
      basePrice: s.price,
    })),
  });

  for (const e of events) {
    await prisma.event.create({
      data: {
        id: e.id,
        title: e.title,
        artist: e.artist,
        description: e.description,
        date: e.date,
        time: e.time,
        category: e.category,
        minPrice: e.minPrice,
        availableSeats: e.availableSeats,
        imageUrl: e.imageUrl ?? null,
        hallId: "virazh",
      },
    });

    const eventSeats = template.map((s, i) => ({
      eventId: e.id,
      seatId: s.id,
      status: i < 12 ? "sold" : "available",
      heldUntil: null,
    }));

    await prisma.eventSeat.createMany({ data: eventSeats });
  }

  await prisma.review.createMany({
```

```
    data: reviews.map((r) => ({
      id: r.id,
      name: r.name,
      role: r.role,
      text: r.text,
      rating: r.rating,
    })),
  });

  const demoHash = await bcrypt.hash("virazh123", 10);
  await prisma.user.create({
    data: {
      email: "demo@virazh.local",
      passwordHash: demoHash,
      name: "Демо Користувач",
      role: "user",
      phone: "+380 67 000 00 01",
    },
  });
  await prisma.user.create({
    data: {
      email: "admin@virazh-hall.ua",
      passwordHash: demoHash,
      name: "Адмін Віраж",
      role: "admin",
      phone: "+380 44 000 00 01",
    },
  });

  console.log(
    `Seeded hall, ${template.length} seats, ${events.length} events,
${reviews.length} reviews.\n` +
      `  Demo user: demo@virazh.local / virazh123\n` +
      `  Admin:     admin@virazh-hall.ua / virazh123 → /admin`,
```

```
  );
}

main()
  .then(() => prisma.$disconnect())
  .catch((e) => {
    console.error(e);
    prisma.$disconnect();
    process.exit(1);
  });
```

Додаток Д. Лістинг файлу AdminSidebar.tsx

```
"use client";

import Link from "next/link";
import { usePathname } from "next/navigation";
import { useEffect, useState } from "react";
import {
  LayoutDashboard,
  Calendar,
  Ticket,
  Users,
  MessageSquare,
  Headphones,
} from "lucide-react";

const links = [
  { href: "/admin", label: "Дашборд", icon: LayoutDashboard },
  { href: "/admin/events", label: "Події", icon: Calendar },
  { href: "/admin/bookings", label: "Бронювання", icon: Ticket },
  { href: "/admin/chats", label: "Чати підтримки", icon: Headphones },
  { href: "/admin/reviews", label: "Відгуки", icon: MessageSquare },
  { href: "/admin/users", label: "Користувачі", icon: Users },
];

export function AdminSidebar() {
  const pathname = usePathname();
  const [openChats, setOpenChats] = useState(0);

  useEffect(() => {
    fetch("/api/admin/chats?status=open")
      .then((r) => r.json())
      .then((d: { openCount?: number }) => setOpenChats(d.openCount ?? 0))
      .catch(() => {});
```

```
    const t = setInterval(() => {
      fetch("/api/admin/chats?status=open")
        .then((r) => r.json())
        .then((d: { openCount?: number }) => setOpenChats(d.openCount ??
0))
        .catch(() => {});
    }, 15000);
    return () => clearInterval(t);
  }, []);

  return (
    <aside className="flex w-64 shrink-0 flex-col border-r border-card-
border bg-card">
      <div className="border-b border-card-border p-5">
        <p    className="font-display    text-lg    font-semibold    text-
accent">Віраж</p>
        <p className="text-xs text-muted">Панель адміністратора</p>
      </div>
      <nav className="flex-1 space-y-1 p-3">
        {links.map(({ href, label, icon: Icon }) => {
          const active =
            href === "/admin"
              ? pathname === "/admin"
              : pathname.startsWith(href);
          const badge = href === "/admin/chats" && openChats > 0;
          return (
            <Link
              key={href}
              href={href}
              className={`flex items-center justify-between gap-3 rounded-
xl px-3 py-2.5 text-sm transition ${
                active
                  ? "bg-accent/15 text-accent"
                  : "text-muted hover:bg-white/5 hover:text-foreground"
```

```
              }`}
            >
              <span className="flex items-center gap-3">
                <Icon className="h-4 w-4" />
                {label}
              </span>
              {badge && (
                <span  className="rounded-full  bg-accent  px-2  py-0.5  text-
[10px] font-bold text-white">
                  {openChats}
                </span>
              )}
            </Link>
          );
        })}
      </nav>
    </aside>
  );
}
```

Додаток Е. Лістинг файлу middleware.ts

```
import NextAuth from "next-auth";
import { NextResponse } from "next/server";
import { authConfig } from "@/auth.config";

const { auth } = NextAuth(authConfig);

export default auth((req) => {
  const { pathname } = req.nextUrl;

  const isAdmin =
    pathname === "/admin" || pathname.startsWith("/admin/");
  if (!isAdmin) {
    return NextResponse.next();
  }

  if (!req.auth?.user) {
    const url = new URL("/login", req.url);
    url.searchParams.set("callbackUrl", pathname);
    return NextResponse.redirect(url);
  }

  if (req.auth.user.role !== "admin") {
    return      NextResponse.redirect(new      URL("/profile?denied=admin",
req.url));
  }

  return NextResponse.next();
});

export const config = {
  matcher: ["/admin", "/admin/:path*"],
};
```

Додаток К. Лістинг файлу page.tsx

```
"use client";

import Link from "next/link";
import { useRouter, useSearchParams } from "next/navigation";
import { Suspense, useEffect, useState } from "react";
import { useSession } from "next-auth/react";
import { CheckCircle2, Info, Loader2, Ticket } from "lucide-react";
import { CheckoutStepper } from "@/components/checkout/CheckoutStepper";
import { BookingPageShell } from "@/components/booking/BookingPageShell";
import { Button } from "@/components/ui/Button";
import { formatDate, formatPrice } from "@/lib/format";
import { formatSeatLabel } from "@/lib/format-seat";
import type { Event, Seat } from "@/lib/types";
import {
  MAX_SEATS_MESSAGE,
  MAX_SEATS_PER_BOOKING,
} from "@/lib/booking-limits";

function CheckoutContent() {
  const router = useRouter();
  const { data: session } = useSession();
  const params = useSearchParams();
  const eventId = params.get("event") ?? "1";
  const seatsParam = params.get("seats") ?? "";
  const seatIds = seatsParam.split(",").filter(Boolean);
  const [event, setEvent] = useState<Event | null>(null);
  const [seats, setSeats] = useState<Seat[]>([]);
  const [loading, setLoading] = useState(true);
  const [submitting, setSubmitting] = useState(false);
  const [error, setError] = useState<string | null>(null);

  useEffect(() => {
```

```
    async function load() {
      const [eventRes, seatsRes] = await Promise.all([
        fetch(`/api/events/${eventId}`),
        fetch(`/api/events/${eventId}/seats`),
      ]);
      if (eventRes.ok) {
        const d = (await eventRes.json()) as { event: Event };
        setEvent(d.event);
      }
      if (seatsRes.ok) {
        const data = (await seatsRes.json()) as { seats: Seat[] };
        setSeats(data.seats.filter((s) => seatIds.includes(s.id)));
      }
      setLoading(false);
    }
    void load();
  }, [eventId, seatsParam]);

  const total = seats.reduce((sum, s) => sum + s.price, 0);

  return (
    <BookingPageShell>
      <CheckoutStepper current={2} />
      <h1 className="font-display text-3xl font-semibold sm:text-4xl">
        Підтвердження броні
      </h1>
      <p className="mt-2 text-lg text-muted">{event?.title}</p>

      <div  className="mt-8  grid  gap-8  lg:grid-cols-[1fr_320px]  lg:items-
start">
        <div className="space-y-6">
          <div className="flex gap-3 rounded-xl border border-accent/20 bg-
accent-soft p-4 text-sm">
            <Info className="h-5 w-5 shrink-0 text-accent" />
```

```
            <div>
              <p className="font-semibold text-foreground">
                Без онлайн-оплати карткою
              </p>
              <p className="mt-2 leading-relaxed text-muted">
                Після підтвердження надішлемо PDF-квиток на email. Оплату
можна
                зробити в касі залу «Віраж» у Тернополі до початку
концерту.
              </p>
            </div>
          </div>

          <form
            className="space-y-6"
            onSubmit={async (e) => {
              e.preventDefault();
              if (seatIds.length > MAX_SEATS_PER_BOOKING) {
                setError(MAX_SEATS_MESSAGE);
                return;
              }
              setSubmitting(true);
              setError(null);
              const fd = new FormData(e.currentTarget);
              try {
                const res = await fetch("/api/bookings", {
                  method: "POST",
                  headers: { "Content-Type": "application/json" },
                  body: JSON.stringify({
                    eventId,
                    seatIds,
                    guestName: fd.get("name"),
                    guestEmail: fd.get("email"),
                    guestPhone: fd.get("phone") || undefined,
```

```
                }),
              });
              const data = (await res.json()) as {
                bookingId?: string;
                error?: string;
                hint?: string;
                emailSent?: boolean;
                devTicketUrl?: string;
              };
              if (!res.ok) {
                setError(
                  data.hint
                    ? `${data.error ?? "Помилка"} (${data.hint})`
                    : (data.error ?? "Помилка бронювання"),
                );
                return;
              }
              const q = new URLSearchParams({
                event: eventId,
                seats: seatIds.join(","),
                booking: data.bookingId ?? "",
                emailSent: String(data.emailSent ?? false),
                devTicket: data.devTicketUrl ?? "",
              });
              router.push(`/booking/success?${q.toString()}`);
            } catch {
              setError("Немає з'єднання з сервером");
            } finally {
              setSubmitting(false);
            }
          }}
        >
          <fieldset className="space-y-4 rounded-2xl border border-card-
border bg-card p-6 shadow-sm">
```

```
              <legend className="px-2 text-sm font-semibold text-accent">
                Контактні дані
              </legend>
              <input
                name="name"
                required
                defaultValue={session?.user?.name ?? ""}
                placeholder="Ім'я та прізвище"
                className="w-full rounded-xl border border-card-border px-4
py-3  text-sm  outline-none  focus:border-accent  focus:ring-2  focus:ring-
accent/20"
              />
              <input
                name="email"
                required
                type="email"
                defaultValue={session?.user?.email ?? ""}
                placeholder="Email для квитка"
                className="w-full rounded-xl border border-card-border px-4
py-3  text-sm  outline-none  focus:border-accent  focus:ring-2  focus:ring-
accent/20"
              />
              <input
                name="phone"
                placeholder="Телефон"
                className="w-full rounded-xl border border-card-border px-4
py-3  text-sm  outline-none  focus:border-accent  focus:ring-2  focus:ring-
accent/20"
              />
              {!session?.user && (
                <p className="text-xs text-muted">
                  <Link        href="/login"        className="text-accent
hover:underline">
                    Увійдіть
```

```
                </Link>{" "}
                - квитки з&amp;apos;являться в кабінеті
              </p>
            )}
          </fieldset>

          <label  className="flex  items-start  gap-3  rounded-xl  border
border-card-border bg-card p-4 text-sm text-muted">
            <input
              type="checkbox"
              required
              className="mt-1 h-4 w-4 accent-accent"
            />
            Погоджуюсь з умовами бронювання залу «Віраж»
          </label>

          {error && (
            <p className="rounded-lg bg-red-50 p-3 text-sm text-danger">
              {error}
            </p>
          )}

          <Button
            type="submit"
            size="lg"
            className="w-full sm:w-auto sm:min-w-[280px]"
            disabled={submitting || seats.length === 0}
          >
            {submitting ? (
              <Loader2 className="h-5 w-5 animate-spin" />
            ) : (
              <>
                <CheckCircle2 className="h-5 w-5" />
                Підтвердити бронювання
```

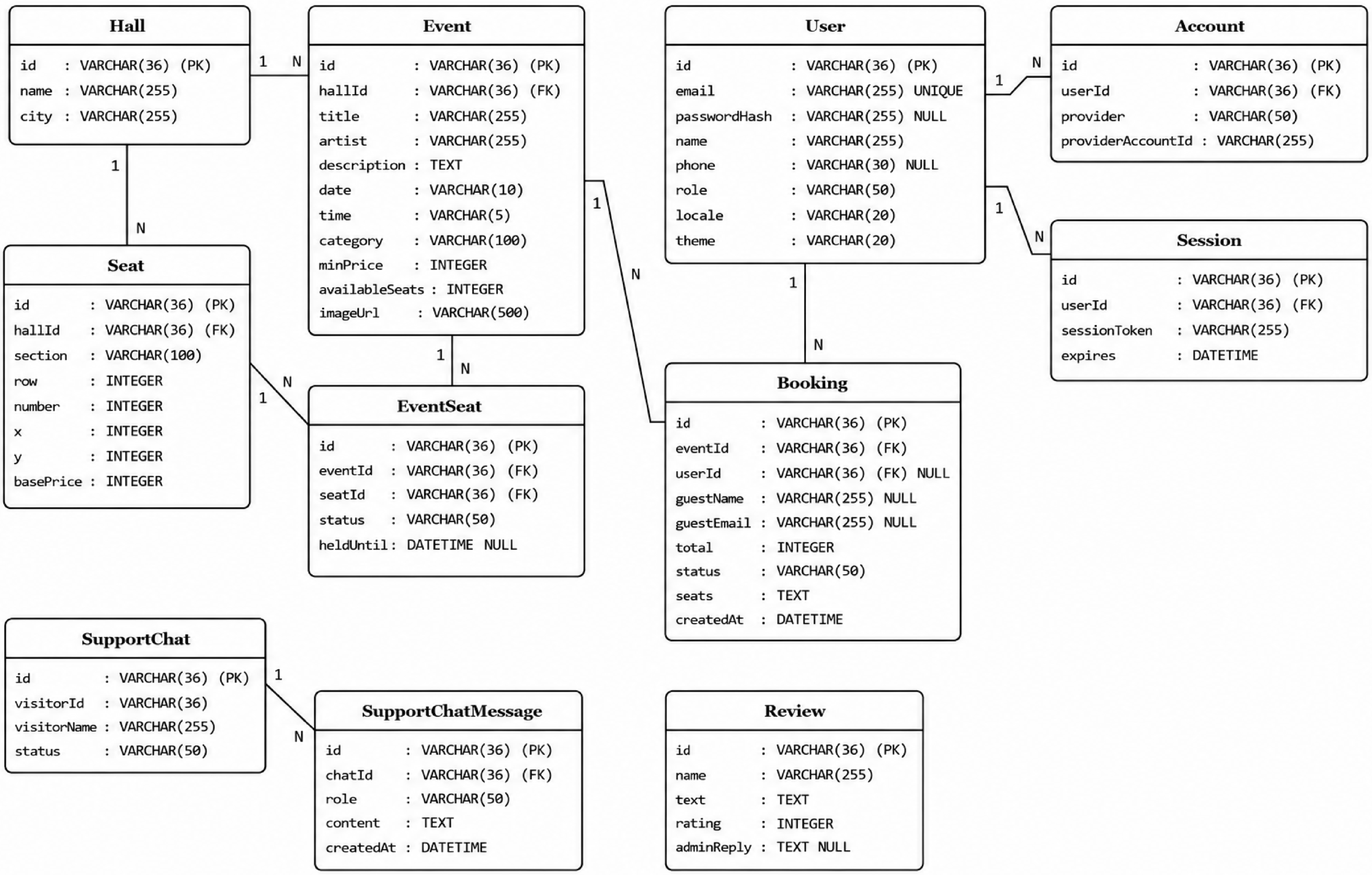
```
                </>
              )}
            </Button>
          </form>
        </div>

        <aside className="rounded-2xl border border-card-border bg-card p-6
shadow-lg lg:sticky lg:top-24">
          <h2 className="flex items-center gap-2 font-semibold">
            <Ticket className="h-5 w-5 text-accent" />
            Ваші місця
          </h2>
          {event && (
            <p className="mt-2 text-sm text-muted">
              {formatDate(event.date)} · {event.time}
            </p>
          )}
          {loading ? (
            <Loader2 className="mt-6 h-8 w-8 animate-spin text-accent" />
          ) : seats.length === 0 ? (
            <p className="mt-4 text-sm text-muted">
              <Link    href={`/events/${eventId}/seats`}    className="text-
accent">
                Оберіть місця
              </Link>
            </p>
          ) : (
            <ul className="mt-4 space-y-3 border-t border-card-border pt-4
text-sm">
              {seats.map((s) => (
                <li key={s.id} className="flex justify-between gap-2">
                  <span className="text-muted">
                    {formatSeatLabel(s.id)}
                  </span>
```

```
                  <span                                    className="font-
medium">{formatPrice(s.price)}</span>
                </li>
              ))}
            </ul>
          )}
          <div  className="mt-6  flex  justify-between  border-t  border-card-
border pt-4 text-lg font-semibold">
            <span>До сплати в касі</span>
            <span className="text-accent">{formatPrice(total)}</span>
          </div>
        </aside>
      </div>
    </BookingPageShell>
  );
}

export default function CheckoutPage() {
  return (
    <Suspense
      fallback={
        <p className="p-12 text-center text-muted">Завантаження…</p>
      }
    >
      <CheckoutContent />
    </Suspense>
  );
}
```

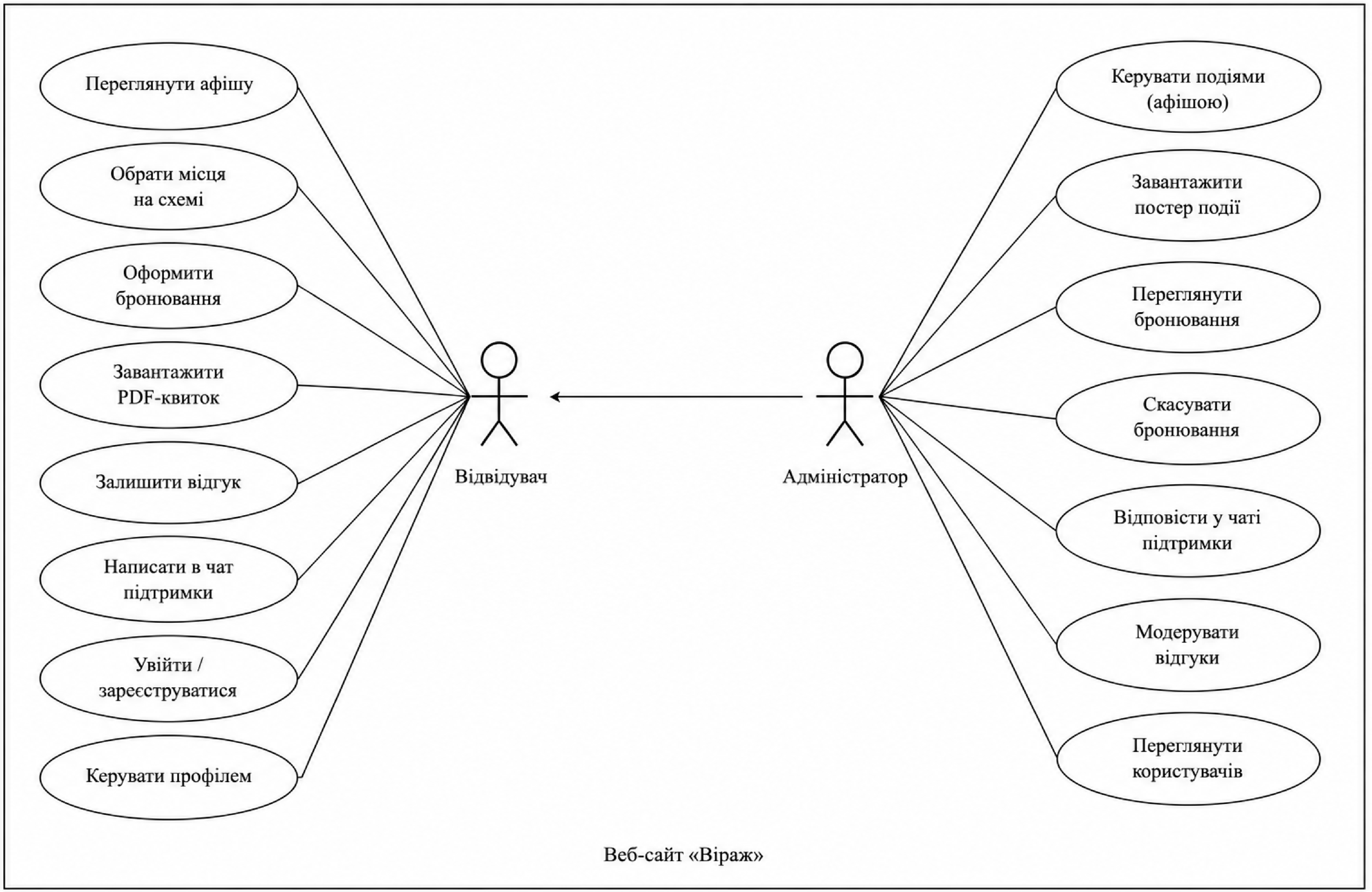
ER-діаграма бази даних



Изм.	Лист	№ докум.	Подп.	Дата	2026.КВР.122.423.07.00.00 БД
Разраб.		Кравець А.В.			Розробка вебсайту для бронювання квитків для концертного залу «Віраж»
Пров.		Болобаш Р.Ю.			ER-діаграма бази даних
Т.контр.					Лит. / Масса / Масштаб
Рецензент.					Лист / Листов 1
Н.контр.		Приймак В.А.			ВСП ТФК ТНТУ КН-423 м. Тернопіль
Утв.					

Копировал Формат А1

UML-діаграма варіантів використання

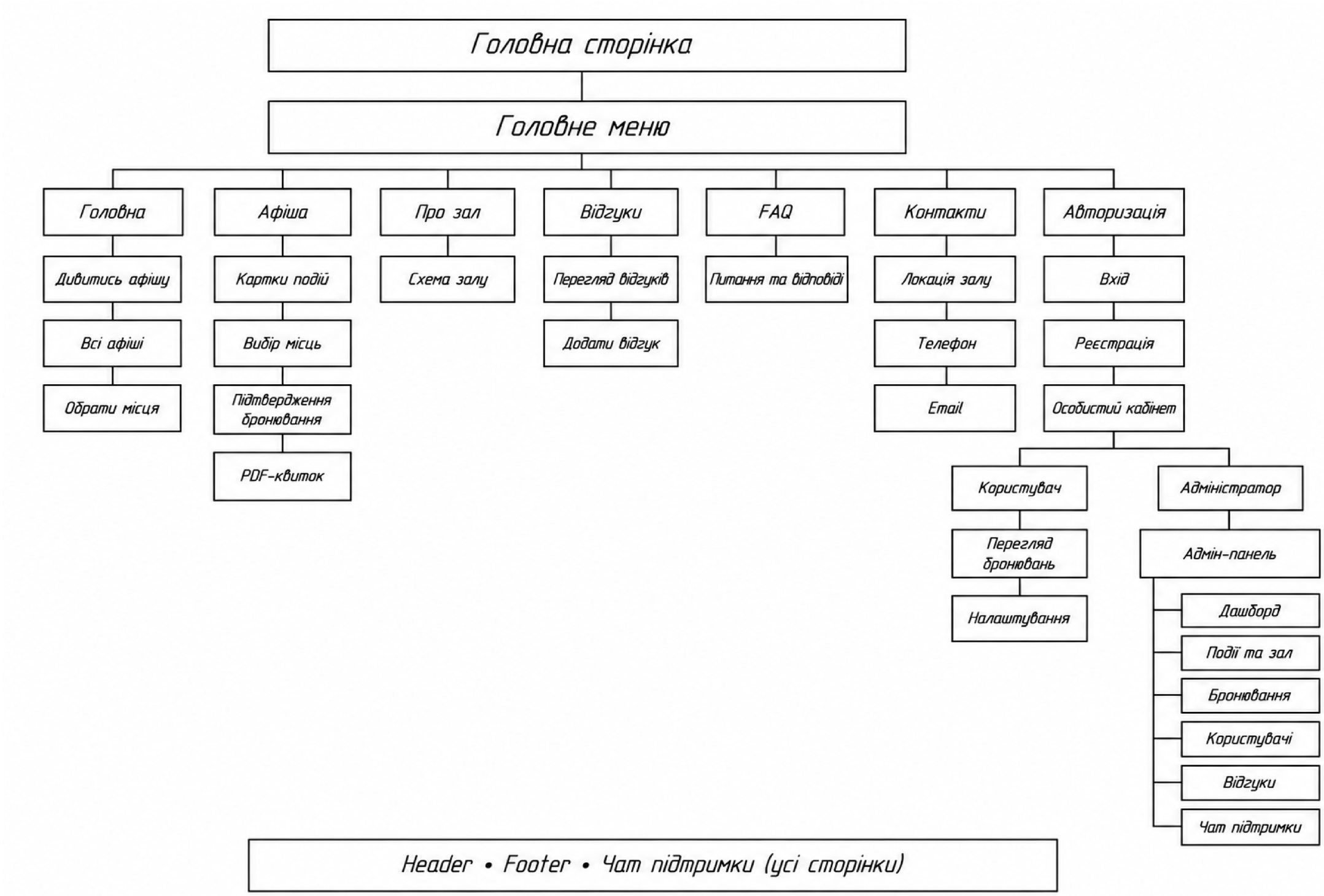


					2026.КВР.122.423.07.00.00 ДВ	
Изм.	Лист	№ докум.	Подп.	Дата	Розробка вебсайту для бронювання квитків для концертного залу «Віраж»	Лит. / Масса / Масштаб
Разраб.		Кравець А.В.				
Пров.		Болюбаш Р.Ю.			UML-діаграма варіантів використання	Лист / Листов 1
Т.контр.						
Рецензент						ВСП ТФК ТНТУ КН-423
Н.контр.		Приймак В.А.				м. Тернопіль
Утв.						

Копировал Формат А1

Перв. примен. | Справ. № | Подп. и дата | Инв. № подл. | Взам. инв. № | Инв. № дубл. | Подп. и дата

Структурна схема веб-сайту «Віраж»



Изм.	Лист	№ докум.	Подп.	Дата	2026.КВР.122.423.07.00.00 СС	
Разраб.		Кравець А.В.			Розробка вебсайту для бронювання квитків для концертного залу «Віраж»	Лит. / Масса / Масштаб
Пров.		Білодаш Р.Ю.			Схема структурна	Лист / Листов 1
Т.контр.						
Рецензент.						ВСП ТФК ТНТУ КН-423
Н.контр.		Приймак В.А.				м. Тернопіль
Утв.						

Копировал Формат А1

Перед. примен. | Справ. № | Инв. № подл. | Подп. и дата | Взам. инв. № | Инв. № дубл. | Подп. и дата

Таблиця техніко-економічних показників

№	Показник	Одиниці вимірювання	Значення
1	Платформа	–	Next.js 16 / React 19
2	Інтерфейс	–	Веб-сайт
3	Мова програмування	–	TypeScript
4	База даних	–	SQLite
5	Трудомісткість розробки	год	159
6	Собівартість	грн	62 044
7	Плановий прибуток	грн	37 520
8	Ціна	грн	96 788
9	Чиста теперішня вартість	грн	24 377
10	Термін окупності	рік	2,1