ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення для онлайн-продажу квитків на благодійні концерти»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олександр ЗАМАЙ
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Олександр ЗАМАЙ

Керівник: В’ячеслав ТРЕЙТЯК
канд.техн.наук

Рецензент:

Київ 2026

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Замаю Олександру Олександровичу

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення для онлайн-продажу квитків на благодійні концерти»
керівник кваліфікаційної роботи канд.техн.наук В’ячеслав ТРЕЙТЯК,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про організацію онлайн-продажу квитків на благодійні концерти, принципи роботи web-застосунків для керування подіями, підходи до обліку квитків, замовлень, користувачів і ролей доступу, методи забезпечення прозорості благодійних внесків за допомогою функції «Паспорт внеску», а також технічна документація щодо використання Java, Spring Boot, Spring Security, JWT, PostgreSQL, React, Vite, REST API, JPA/Hibernate та інструментів тестування програмного забезпечення.

4. Зміст розрахунково-пояснювальної записки:

1. Огляд та аналіз предметної галузі онлайн-продажу квитків на благодійні концерти, способів організації продажу квитків, обліку замовлень, користувачів, подій і благодійних внесків.
2. Аналіз існуючих програмних засобів для онлайн-продажу квитків, організації подій, керування замовленнями та взаємодії з користувачами.
3. Огляд засобів реалізації web-застосунку DobroStage для онлайн-продажу квитків на благодійні концерти.
4. Проєктування архітектури web-застосунку, структури бази даних, ролей користувачів, основних модулів системи та механізму формування «Паспорта внеску».
5. Програмна реалізація та опис функціонування web-застосунку DobroStage.
6. Тестування основних функцій web-застосунку та оцінка відповідності програмного забезпечення поставленим вимогам.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів програмного забезпечення для онлайн-продажу квитків на благодійні концерти.
2. Вимоги до web-застосунку DobroStage.
3. Загальна архітектура програмної реалізації web-застосунку.
4. Діаграма варіантів використання web-застосунку DobroStage.
5. Схема ролей користувачів та прав доступу.
6. Структура бази даних DobroStage.
7. Діаграма компонентів web-застосунку.
8. Діаграма класів серверної частини DobroStage.
9. Діаграми послідовностей.
10. Результати тестування web-застосунку.
11. Апробація результатів дослідження.

6. Дата видачі завдання«23» лютого 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури за темою онлайн-продажу квитків на благодійні концерти	23.02–28.02.2026	
2	Аналіз та дослідження існуючих аналогів програмного забезпечення для онлайн-продажу квитків і організації подій	02.03–08.03.2026	
3	Аналіз предметної галузі благодійних концертів, обліку квитків, замовлень, користувачів і благодійних внесків	09.03–15.03.2026	
4	Огляд засобів реалізації web-застосунку DobroStage	16.03–22.03.2026	
5	Проєктування web-застосунку DobroStage, логічної структури системи, бази даних та UML-діаграм	23.03–05.04.2026	
6	Програмна реалізація серверної та клієнтської частин web-застосунку DobroStage	06.04–25.04.2026	
7	Тестування web-застосунку DobroStage та оформлення результатів тестування	26.04–03.05.2026	
8	Оформлення роботи: вступ, висновки, реферат, перелік посилань та додатки	04.05–10.05.2026	
9	Розробка демонстраційних матеріалів та підготовка до попереднього захисту роботи	11.05–17.05.2026	
10	Попередній захист роботи	18.05–22.05.2026	

Здобувач вищої освіти _____ Олександр ЗАМАЙ
(підпис)

Керівник
кваліфікаційної роботи _____ В’ячеслав ТРЕЙТЯК
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 51 стор., 3 табл., 26 рис., 27 джерела.

Мета роботи – спрощення та автоматизація процесу продажу квитків на благодійні концерти за рахунок розробки веб-застосунку, який забезпечує централізоване керування подіями, квитками, замовленнями та обліком доступних місць.

Об’єкт дослідження – процес онлайн-продажу квитків на благодійні концерти.

Предмет дослідження – засоби, методи та технології реалізації програмного забезпечення для онлайн-продажу квитків на благодійні концерти.

Короткий зміст роботи: у кваліфікаційній роботі проведено аналіз предметної галузі онлайн-продажу квитків на благодійні концерти та визначено основні проблеми, пов’язані з розрізненим зберіганням інформації про події, ручним оформленням замовлень, складністю контролю доступних квитків, ризиком дублювання місць, недостатньою прозорістю благодійного збору та підвищеним навантаженням на організаторів.

Проаналізовано існуючі програмні засоби для онлайн-продажу квитків та організації подій, зокрема Concert.ua, KARABAS, Eventbrite і TicketForEvent. Визначено їхні переваги та недоліки, серед яких універсальність платформ, залежність від сторонніх сервісів, обмежена адаптація під потреби невеликих благодійних концертів і відсутність персоналізованого відображення благодійного результату конкретного користувача.

У роботі сформовано функціональні та нефункціональні вимоги до web-застосунку DobroStage, обґрунтовано вибір засобів реалізації клієнтської частини, серверної частини, бази даних, механізму авторизації та тестування. Для реалізації web-застосунку використано Java, Spring Boot, Spring Security, JWT, Spring Data JPA, Hibernate, PostgreSQL, React, Vite, REST API, Git, IntelliJ IDEA, VS Code та засоби автоматизованого тестування.

Спроєктовано загальну архітектуру системи, логічну структуру web-застосунку, структуру бази даних, діаграму варіантів використання, схему ролей користувачів і прав доступу, діаграму компонентів, діаграму класів серверної частини, діаграму послідовності оформлення замовлення та діаграму послідовності формування «Паспорта внеску». Реалізовано основні модулі системи: авторизацію та реєстрацію користувачів, каталог благодійних концертів, сторінку детального перегляду концерту, вибір типів квитків, оформлення замовлення, сторінку «Мої квитки», профіль користувача, кабінет організатора, адміністративну панель і перегляд благодійного ефекту внеску.

Особливістю розробленої системи є функція «Паспорт внеску», яка дозволяє користувачу бачити не лише факт купівлі квитка, а й практичний благодійний результат свого замовлення. Для цього в системі використовуються спеціальні impact-поля концерту: назва impact-одиниці, вартість однієї impact-одиниці та опис благодійного ефекту. Наприклад, якщо користувач придбав квиток на 1300 грн, а одна impact-одиниця коштує 650 грн, система показує, що внесок підтримав 2 відповідні благодійні одиниці.

КЛЮЧОВІ СЛОВА: DOBROSTAGE, WEB-ЗАСТОСУНОК, ОНЛАЙН-ПРОДАЖ КВИТКІВ, БЛАГОДІЙНИЙ КОНЦЕРТ, ЗАМОВЛЕННЯ, КВИТОК, ПАСПОРТ ВНЕСКУ, IMPACT, SPRING BOOT, REACT, POSTGRESQL, JWT, REST API.

ЗМІСТ

ВСТУП 11
1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ РІШЕНЬ ДЛЯ ОНЛАЙН-ПРОДАЖУ КВИТКІВ НА БЛАГОДІЙНІ КОНЦЕРТИ 15
1.1 Аналіз предметної галузі онлайн-продажу квитків на благодійні концерти 15
1.2 Аналіз способів організації продажу квитків 18
1.3 Огляд та порівняльний аналіз програмних аналогів 20
1.3.1 Сервіс Concert.ua 20
1.3.2 Сервіс KARABAS 21
1.3.3 Сервіс Eventbrite 22
1.3.4 Сервіс TicketForEvent 24
1.3.5 Порівняльний аналіз застосунків-аналогів 25
2 ПРОЄКТУВАННЯ WEB-ЗАСТОСУНКУ DOBROSTAGE ДЛЯ ОНЛАЙН-ПРОДАЖУ КВИТКІВ НА БЛАГОДІЙНІ КОНЦЕРТИ 26
2.1 Визначення вимог до web-застосунку 26
2.2 Огляд засобів програмної реалізації 29
2.3 Проєктування архітектури та бази даних системи 31
2.4 Проєктування основних функціональних модулів DobroStage 34
3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ЗАСТОСУНКУ DOBROSTAGE 38
3.1 Реалізація серверної частини web-застосунку 38
3.2 Реалізація клієнтської частини web-застосунку 42
3.3 Реалізація взаємодії користувачів із системою 46
3.4 Реалізація унікальної функції «Паспорт внеску» 48
4 ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ DOBROSTAGE 50
4.1 Ручне тестування основних функцій системи 50
4.2 Модульне та інтеграційне тестування backend-частини 54
4.3 Тестування frontend-частини 56
4.4 Узагальнення результатів тестування 57
ВИСНОВКИ 60
ПЕРЕЛІК ПОСИЛАНЬ 63
ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ 65
ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ 75

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – інтерфейс прикладного програмування, що використовується для взаємодії між програмними компонентами або зовнішніми сервісами.

REST – архітектурний стиль організації обміну даними між клієнтською та серверною частинами web-застосунку.

HTTP – протокол передавання даних у мережі Інтернет.

HTTPS – захищена версія протоколу HTTP, що забезпечує безпечний обмін даними між клієнтом і сервером.

JSON – текстовий формат обміну даними, який використовується для передавання структурованої інформації між frontend- і backend-частинами системи.

Frontend – клієнтська частина web-застосунку, з якою безпосередньо взаємодіє користувач.

Backend – серверна частина web-застосунку, яка виконує бізнес-логіку, обробляє запити та працює з базою даних.

SPA – односторінковий web-застосунок, у якому взаємодія з користувачем відбувається без повного перезавантаження сторінки.

Java – об’єктно-орієнтована мова програмування, що використовується для реалізації серверної частини web-застосунку.

Spring – фреймворк для створення серверних Java-застосунків.

Spring Boot – інструмент екосистеми Spring, який спрощує створення, налаштування та запуск backend-застосунків.

Spring Security – модуль Spring для налаштування авторизації, автентифікації та захисту web-застосунків.

JWT – JSON Web Token, токен для безпечної передачі інформації про авторизованого користувача між клієнтом і сервером.

Spring Data JPA – модуль Spring для роботи з реляційними базами даних через об’єктну модель.

Hibernate – ORM-фреймворк, який забезпечує зв’язок між Java-класами та таблицями бази даних.

PostgreSQL – об’єктно-реляційна система керування базами даних.

React – бібліотека JavaScript для створення компонентного користувацького інтерфейсу.

Vite – інструмент для швидкого створення, запуску та збирання frontend-застосунків.

JavaScript – мова програмування, що використовується для реалізації логіки клієнтської частини web-застосунку.

CSS – мова стилів, яка використовується для оформлення зовнішнього вигляду web-сторінок.

Git – система контролю версій, що використовується для відстеження змін у програмному коді.

REST API – набір endpoint-ів серверної частини, через які frontend взаємодіє з backend.

Endpoint – окрема адреса API, яка відповідає за виконання певної операції.

Entity – клас серверної частини, який описує сутність предметної галузі та відповідає таблиці бази даних.

Repository – компонент доступу до бази даних, який виконує операції збереження, пошуку, оновлення та видалення даних.

Service – компонент серверної частини, який реалізує бізнес-логіку системи.

Controller – компонент backend-частини, який приймає HTTP-запити та повертає відповіді клієнтській частині.

DTO – об’єкт передачі даних між різними частинами системи.

Паспорт внеску – унікальна функція DobroStage, яка відображає благодійний ефект замовлення користувача.

Impact-одиниця – умовна одиниця благодійного результату, яка показує, який практичний ефект має внесок користувача.

Order – замовлення користувача на купівлю одного або кількох квитків.

TicketType – тип квитка на концерт, який містить назву, опис, ціну, загальну кількість і доступний залишок.

Venue – майданчик проведення концерту.

ВСТУП

У сучасних умовах розвитку цифрових технологій онлайн-сервіси відіграють важливу роль в організації культурних, соціальних і благодійних подій. Благодійні концерти є одним із поширених форматів збору коштів, оскільки поєднують культурну подію з підтримкою суспільно важливої мети. Такі заходи можуть бути спрямовані на допомогу військовим, постраждалим від воєнних дій, медичним ініціативам, волонтерським організаціям, дітям, освітнім або соціальним проєктам [1; 2].

Актуальність теми кваліфікаційної роботи зумовлена необхідністю створення зручного web-застосунку для централізованого онлайн-продажу квитків на благодійні концерти. У реальних умовах інформація про події, замовлення, користувачів, доступні квитки та благодійні внески часто зберігається в різних джерелах: соціальних мережах, месенджерах, електронних таблицях, формах реєстрації або сторонніх сервісах продажу квитків. Така розрізненість ускладнює контроль продажів, підвищує ризик помилок і не завжди забезпечує достатню прозорість благодійного збору [14; 24].

Особливо важливою проблемою є контроль доступних квитків. Якщо продаж здійснюється вручну або через декілька каналів одночасно, організатор може зіткнутися з дублюванням замовлень, неправильним підрахунком доступних місць, затримками в оновленні інформації або втратою даних про покупців. Для благодійних заходів такі проблеми є критичними, оскільки вони можуть впливати на довіру користувачів до організаторів і до самої благодійної ініціативи [2; 3; 4].

У зв’язку з цим актуальною є розробка web-застосунку DobroStage, який об’єднує в одному середовищі каталог благодійних концертів, пошук і фільтрацію подій, вибір квитків, оформлення замовлень, особистий кабінет користувача, кабінет організатора, адміністративну панель і механізм відображення благодійного ефекту внеску. На відміну від універсальних сервісів продажу квитків, DobroStage орієнтований саме на благодійні концерти та потреби

користувачів, які хочуть не лише придбати квиток, а й бачити результат свого внеску [2; 3; 4].

Особливістю системи є функція «Паспорт внеску». Вона дозволяє користувачу побачити, який практичний благодійний ефект створило його замовлення. Наприклад, користувач бачить не лише суму купленого квитка, а й розрахунок у зрозумілих impact-одиницях: кількість теплих наборів, медичних комплектів, продуктових пакетів або інших одиниць допомоги, які були підтримані його внеском. Такий підхід підвищує прозорість благодійної участі та формує більшу довіру до платформи [2; 3; 4].

Мета роботи – спрощення та автоматизація процесу продажу квитків на благодійні концерти за рахунок розробки веб-застосунку, який забезпечує централізоване керування подіями, квитками, замовленнями та обліком доступних місць [2; 3; 4].

Об’єкт дослідження – процес онлайн-продажу квитків на благодійні концерти [2; 3; 4].

Предмет дослідження – засоби, методи та технології реалізації програмного забезпечення для онлайн-продажу квитків на благодійні концерти [2; 3; 4].

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Проаналізувати предметну галузь онлайн-продажу квитків на благодійні концерти та визначити основні проблеми організації продажів, обліку квитків, замовлень і благодійних внесків [2; 3; 4].
2. Дослідити існуючі програмні рішення для продажу квитків та організації подій, порівняти їх функціональні можливості та визначити основні недоліки [6; 7; 8].
3. Сформувати функціональні та нефункціональні вимоги до web-застосунку DobroStage [6; 7; 8].
4. Обґрунтувати вибір технологій для реалізації web-застосунку, зокрема Java, Spring Boot, Spring Security, JWT, PostgreSQL, React, Vite та REST API [23; 24].

5. Спроєктувати архітектуру системи, структуру бази даних, ролі користувачів, основні UML-діаграми та механізм формування «Паспорта внеску» [11; 12; 13].
6. Реалізувати серверну частину для роботи з користувачами, концертами, типами квитків, замовленнями, ролями доступу та impact-показниками [2; 3; 4].
7. Реалізувати клієнтський web-інтерфейс для перегляду каталогу концертів, сторінки події, оформлення замовлень, перегляду профілю, сторінки «Мої квитки», «Паспорта внеску», кабінету організатора та адміністративної панелі [16; 17].
8. Розробити функцію «Паспорт внеску» для розрахунку та відображення благодійного ефекту замовлення користувача [2; 3; 4].
9. Провести тестування основних функцій web-застосунку та оцінити відповідність розробленого програмного забезпечення поставленим вимогам [8; 10; 27].

Методи дослідження. У бакалаврській роботі використовуються такі методи: аналіз предметної галузі, порівняльний аналіз існуючих програмних засобів продажу квитків, об’єктно-орієнтоване проєктування, UML-моделювання, проєктування структури реляційної бази даних, розробка клієнт-серверного web-застосунку, реалізація REST API, тестування програмного забезпечення та аналіз результатів роботи системи [14; 24].

Практична значущість одержаних результатів полягає в тому, що розроблений web-застосунок може бути використаний як основа для організації онлайн-продажу квитків на благодійні концерти. DobroStage дозволяє об’єднати в одному інтерфейсі каталог подій, вибір квитків, оформлення замовлень, кабінет користувача, кабінет організатора, адміністративну панель і перегляд благодійного результату внеску. У майбутньому систему можна розширити додаванням онлайн-оплати, QR-квитків, інтеграції з платіжними сервісами, перевірки квитків на вході, генерації PDF-квитків, розширеної аналітики зборів і звітності для благодійних організацій [16; 17].

1 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ ТА ІСНУЮЧИХ РІШЕНЬ ДЛЯ ОНЛАЙН-ПРОДАЖУ КВИТКІВ НА БЛАГОДІЙНІ КОНЦЕРТИ

1.1 Аналіз предметної галузі онлайн-продажу квитків на благодійні концерти

Благодійні концерти є одним із поширених форматів культурних і соціальних заходів, які поєднують розважальну, інформаційну та суспільно корисну функції. На відміну від звичайних комерційних концертів, головною метою таких подій є не лише організація виступу виконавців, а й збір коштів на певну благодійну ціль. Це може бути підтримка військових, допомога постраждалим від воєнних дій, фінансування медичного лікування, підтримка дитячих, освітніх, культурних або волонтерських ініціатив [1; 2].

У межах такої предметної галузі продаж квитків виконує подвійну роль. З одного боку, квиток є засобом доступу користувача до концерту. З іншого боку, купівля квитка фактично виступає формою благодійного внеску. Саме тому для системи онлайн-продажу квитків на благодійні концерти важливо не лише забезпечити технічну можливість купівлі, а й зробити процес зрозумілим, прозорим і зручним для всіх учасників [16; 17].

Основними учасниками процесу є гість, зареєстрований користувач, організатор та адміністратор. Гість може переглядати каталог концертів, відкривати сторінки подій, шукати заходи за назвою, виконавцем, містом або благодійною метою. Зареєстрований користувач має ширші можливості: він може купувати квитки, переглядати власні замовлення, редагувати профіль та відкривати сторінку «Паспорт внеску», де відображається благодійний ефект його покупки. Організатор відповідає за створення і редагування власних концертів, додавання

типів квитків, перегляд продажів і статистики по своїх подіях. Адміністратор має повний доступ до системи: керує користувачами, ролями, майданчиками, концертами, замовленнями та загальною статистикою платформи [2; 3; 4].

Важливою особливістю предметної галузі є необхідність точного обліку квитків. Кожен концерт має обмежену кількість доступних місць або квитків певної категорії. Якщо облік ведеться вручну або через декілька різних каналів, виникає ризик дублювання замовлень, продажу більшої кількості квитків, ніж доступно, або втрати актуальної інформації про залишок місць. Для благодійних заходів такі помилки є особливо небажаними, оскільки вони можуть впливати не лише на організацію концерту, а й на довіру користувачів до благодійної ініціативи [2; 3; 4].

Окрім обліку квитків, важливою є прозорість благодійного результату. У звичайних сервісах продажу квитків користувач здебільшого бачить лише факт купівлі, назву події, дату, місце проведення та суму замовлення. У межах проєкту DobroStage передбачається додаткова функція — «Паспорт внеску». Вона дозволяє користувачу побачити не просто витрачену суму, а її практичний благодійний ефект. Наприклад, якщо один теплий набір коштує 650 грн, а користувач придбав квиток на 1300 грн, система може показати, що його внесок підтримав 2 теплі набори. Такий підхід підвищує прозорість і робить благодійну участь більш зрозумілою для користувача [2; 3; 4].

Узагальнений процес організації онлайн-продажу квитків на благодійний концерт передбачає створення події, внесення інформації про концерт, публікацію її в каталозі, перегляд користувачем, вибір типу квитка, оформлення замовлення, перевірку доступності квитків, підтвердження покупки, оновлення залишку квитків і формування інформації про благодійний ефект внеску [2; 3; 4].

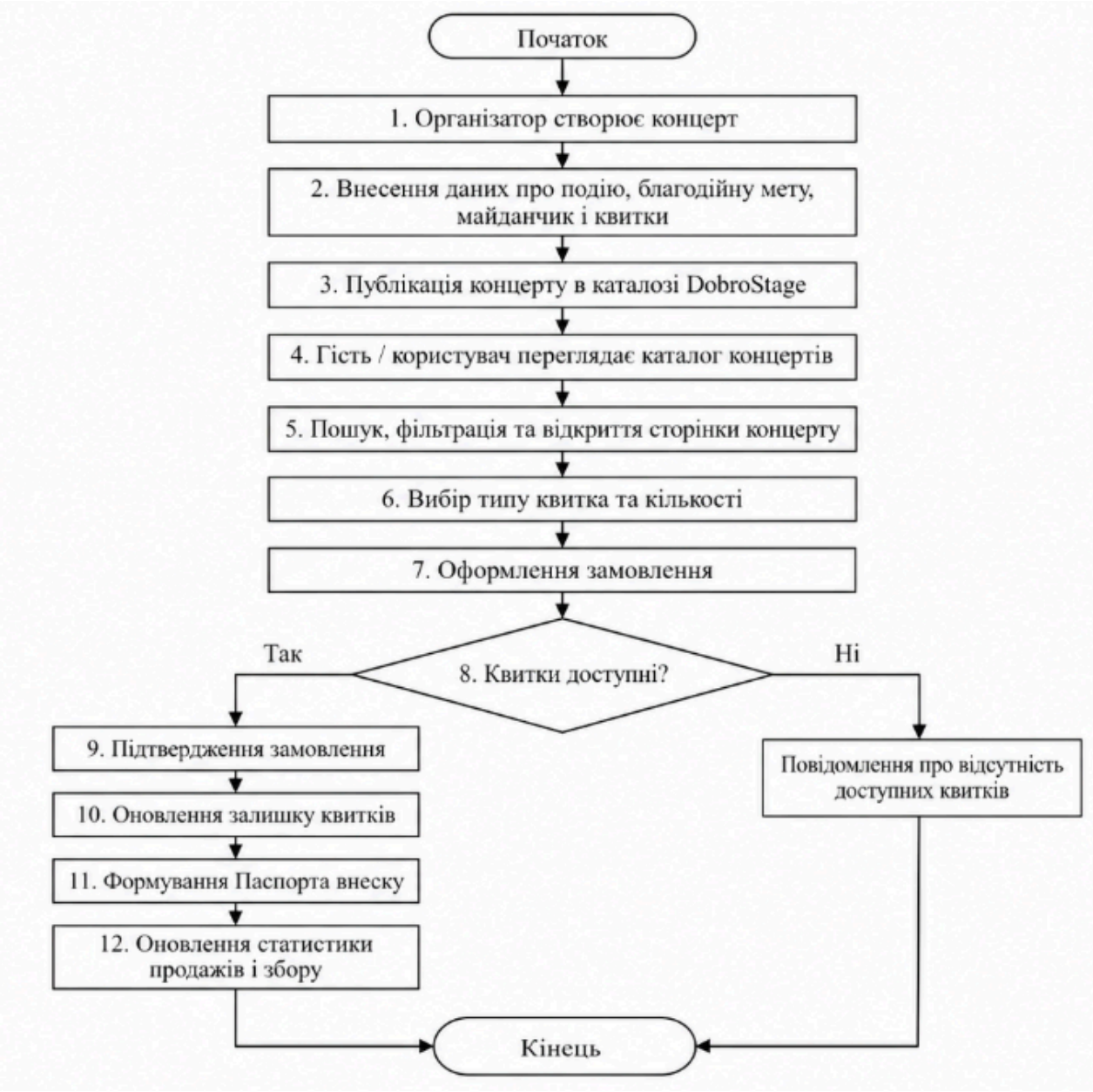


Рис. 1.1 Загальна схема організації онлайн-продажу квитків на благодійний концерт

Таким чином, предметна галузь онлайн-продажу квитків на благодійні концерти потребує програмного забезпечення, яке поєднує функції каталогу подій, системи замовлень, обліку квитків, розмежування прав доступу та відображення благодійного результату. Саме така логіка покладена в основу web-застосунку DobroStage [2; 3; 4].

1.2 Аналіз способів організації продажу квитків

Продаж квитків на благодійні концерти може організовуватися різними способами. Найпростішим варіантом є ручне оформлення замовлень через соціальні мережі, месенджери, телефонні дзвінки або особисті повідомлення. У такому випадку організатор самостійно публікує інформацію про подію, приймає заявки від користувачів, фіксує їх у списках і контролює кількість доступних місць [2; 3; 4].

Перевагою такого способу є простота старту. Організатору не потрібно створювати окрему систему, налаштовувати сайт або працювати з технічно складними інструментами. Достатньо створити публікацію в соціальній мережі або поширити інформацію через месенджери. Такий підхід може бути придатним для дуже малих подій, де кількість учасників невелика, а облік можна вести вручну.

Іншим поширеним способом є використання електронних таблиць, наприклад Excel або Google Sheets. У таблицях можна створити список користувачів, зазначити кількість квитків, суму оплати, статус замовлення та контактні дані. Це вже більш структурований підхід, ніж звичайне листування в месенджерах. Таблиці дозволяють фільтрувати дані, сортувати записи та виконувати прості розрахунки [14; 24].

Проте електронні таблиці також не є повноцінною системою продажу квитків. Дані все одно потрібно вносити вручну, а це створює ризик помилок. Крім того, таблиці не забезпечують зручного інтерфейсу для користувача, не дають можливості автоматично перевіряти доступність квитка перед покупкою та не формують особистий кабінет із замовленнями. Для організатора це може бути прийнятним тимчасовим рішенням, але для масштабування подій та підвищення зручності воно є обмеженим [14; 24].

Окремий варіант — використання готових сервісів для продажу квитків. Такі платформи дозволяють створювати події, продавати квитки, приймати оплати, формувати електронні квитки й переглядати базову статистику. Цей підхід є значно зручнішим за ручний облік, оскільки більшість процесів автоматизована [16; 17].

Недоліком готових платформ є їхня універсальність. Вони зазвичай орієнтовані на різні типи подій: комерційні концерти, вистави, конференції, фестивалі, спортивні заходи. Через це в них може не бути функцій, які потрібні саме для благодійних концертів. Наприклад, не всі сервіси дозволяють показати користувачу конкретний благодійний ефект його внеску або гнучко налаштувати логіку відображення impact-одиниць. Також організатор залежить від правил, комісій, технічних обмежень і внутрішньої структури сторонньої платформи [11; 12; 13].

Основні проблеми ручного або частково автоматизованого обліку квитків і замовлень можна згрупувати за декількома напрямами: розрізнені канали продажу, ручне внесення даних, складність контролю залишку квитків, недостатня прозорість збору, незручність для користувача та підвищене навантаження на організатора [16; 17].



Рис. 1.2 Схема проблем ручного обліку квитків і замовлень

Отже, аналіз способів організації продажу квитків показує, що найефективнішим рішенням для благодійних концертів є спеціалізований web-застосунок. Він повинен поєднувати переваги онлайн-сервісу, простоту користування, автоматичний облік квитків і можливість відображення благодійного результату для користувача [2; 3; 4].

1.3 Огляд та порівняльний аналіз програмних аналогів

Для визначення вимог до web-застосунку DobroStage доцільно розглянути існуючі програмні рішення, які вже використовуються для онлайн-продажу квитків, організації подій або реєстрації учасників. У межах аналізу розглянуто сервіси Concert.ua, KARABAS, Eventbrite та TicketForEvent. Вони мають різне функціональне спрямування, але всі пов’язані з організацією заходів і продажем квитків [19].

1.3.1 Сервіс Concert.ua

Concert.ua є одним із відомих українських сервісів для онлайн-продажу квитків на концерти, театральні вистави, стендап, фестивалі, дитячі події та інші культурні заходи. Платформа має каталог подій, систему пошуку, фільтрацію за категоріями, містом і датою, а також сторінки окремих подій із детальною інформацією [19].

Для користувача такий сервіс є зручним, оскільки дозволяє швидко знайти подію, переглянути інформацію про неї та перейти до купівлі квитка. Concert.ua орієнтований на український ринок, тому його структура, мова інтерфейсу й перелік подій є зрозумілими для локальних користувачів [19].

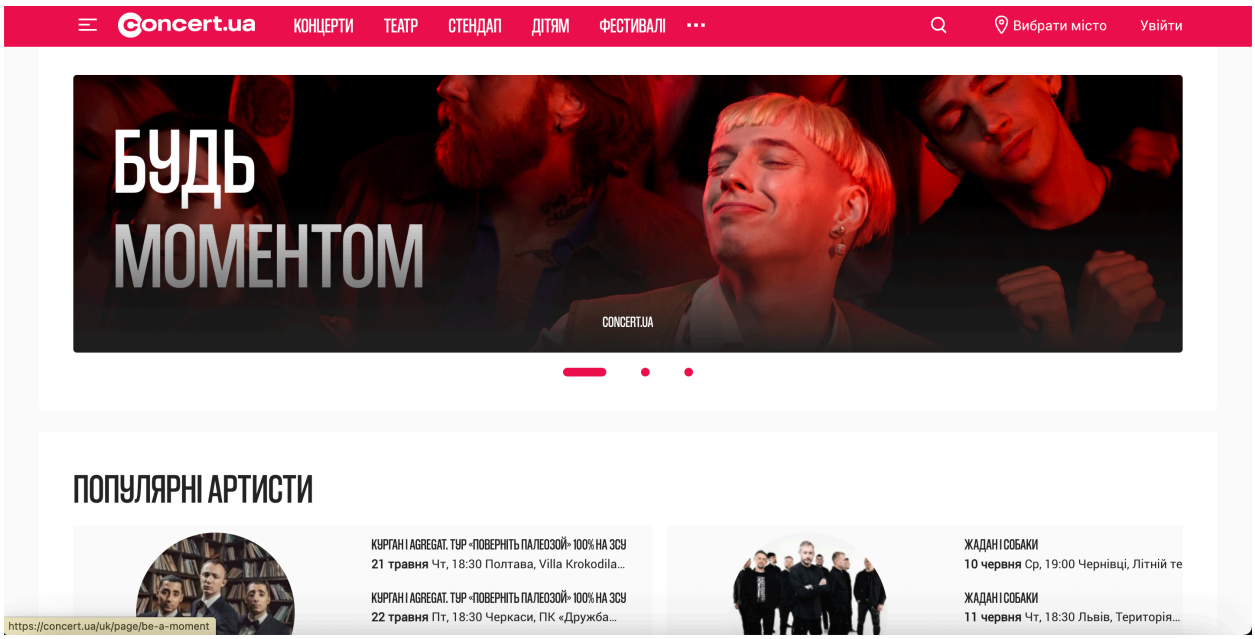


Рис. 1.3 Приклад інтерфейсу сервісу Concert.ua

До переваг Concert.ua можна віднести велику кількість подій, зручну афішу, наявність категорій, зрозумілу сторінку події та можливість онлайн-купівлі квитків. Також важливо, що сервіс працює з різними типами культурних заходів і має сформовану аудиторію [19].

Недоліком для задачі розробки DobroStage є те, що Concert.ua є універсальною платформою. Вона не орієнтована виключно на благодійні концерти та не акцентує увагу на персоналізованому відображенні благодійного ефекту внеску. Для невеликих організаторів благодійних заходів такий сервіс може бути функціонально надмірним або залежним від умов сторонньої платформи [19].

1.3.2 Сервіс KARABAS

KARABAS також є українським сервісом для продажу квитків на концерти, театри, шоу, фестивалі та інші події. Його основна функція полягає в наданні користувачам доступу до афіші заходів і можливості придбати квитки онлайн [20].

Інтерфейс сервісу орієнтований на швидкий пошук події та купівлю квитка. Користувач може переглядати доступні концерти, відкривати сторінку події, бачити основну інформацію про місце проведення, дату та вартість квитків. Це

робить KARABAS корисним прикладом для аналізу стандартної логіки роботи квиткового сервісу [20].

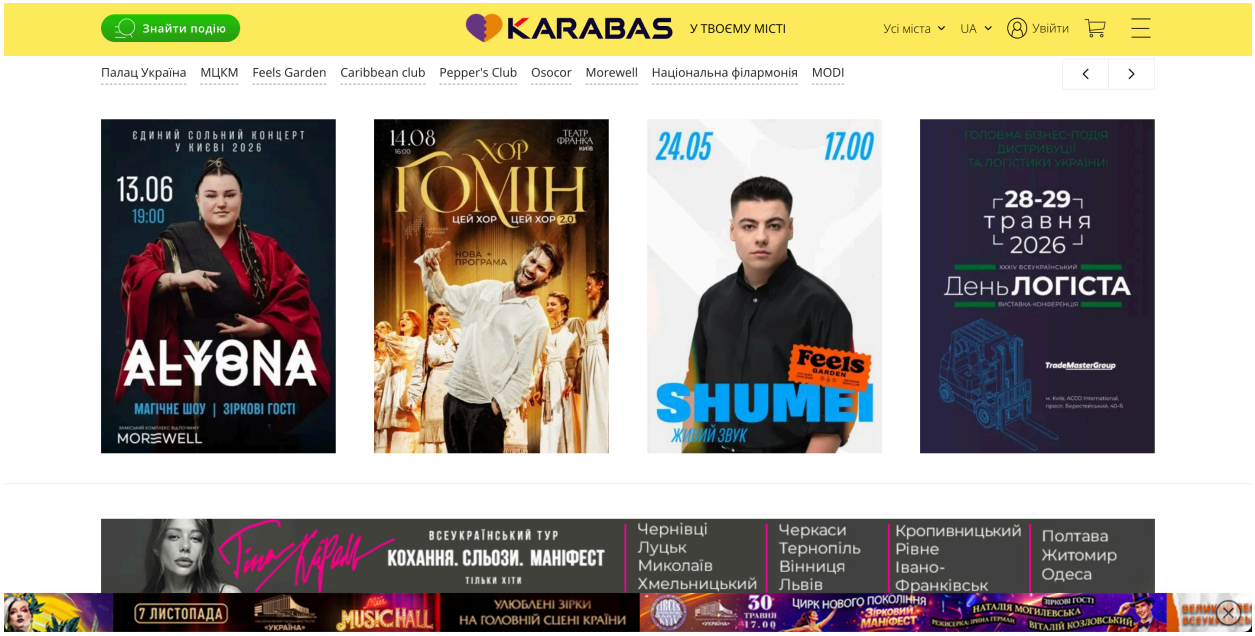


Рис. 1.4 Приклад інтерфейсу сервісу KARABAS

Перевагами KARABAS є адаптація під український ринок, наявність різних категорій подій, можливість онлайн-купівлі квитків і зрозуміла структура сторінок. Сервіс підходить для широкого кола користувачів, які шукають культурні та розважальні заходи [20].

Водночас, як і Concert.ua, KARABAS є загальним сервісом продажу квитків. Він не вирішує специфічну задачу демонстрації благодійного результату конкретного користувача. Також організатор благодійного концерту не має повної свободи у формуванні власної логіки події, ролей, impact-показників і внутрішньої статистики [19].

1.3.3 Сервіс Eventbrite

Eventbrite є міжнародною платформою для створення, просування та продажу квитків на різні події. На відміну від сервісів, які більше орієнтовані на покупців, Eventbrite надає досить широкі можливості саме для організаторів.

Користувач може створити подію, налаштувати типи квитків, опис, дату, місце проведення, реєстрацію учасників і продаж квитків [21].

Eventbrite є корисним аналогом для DobroStage, оскільки показує, як може бути організований кабінет організатора подій. Для благодійних концертів така логіка є важливою, оскільки організатор повинен мати змогу створювати власні події, керувати квитками та переглядати статистику продажів [21].

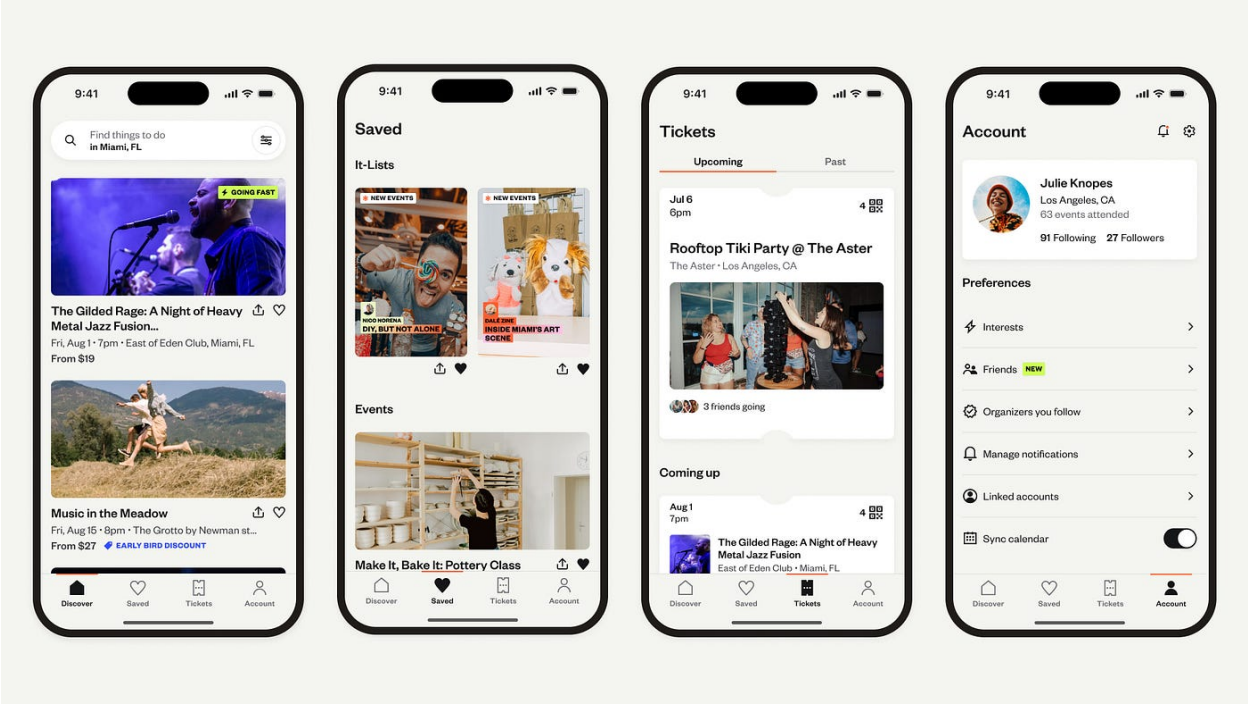


Рис. 1.5 Приклад інтерфейсу сервісу Eventbrite

Перевагами Eventbrite є гнучкість, підтримка різних типів подій, наявність інструментів для організаторів, аналітики та просування. Платформа може використовуватися як для невеликих заходів, так і для масштабних подій [21].

Недоліком є те, що сервіс є міжнародним і не завжди повністю відповідає потребам локального українського ринку. Крім того, Eventbrite має багато функцій, які можуть бути зайвими для вузькоспеціалізованого web-застосунку для благодійних концертів. У DobroStage доцільніше реалізувати меншу, але більш цільову систему, де основна увага приділяється концертам, квиткам, благодійній меті та «Паспорту внеску» [21].

1.3.4 Сервіс TicketForEvent

TicketForEvent — це сервіс для автоматизації реєстрації учасників і продажу електронних квитків на події. Його можна використовувати для конференцій, семінарів, форумів, корпоративних, культурних, спортивних та інших заходів [22].

Особливістю цього сервісу є акцент на реєстрації учасників, роботі зі списками відвідувачів і контролі доступу до події. Для благодійних концертів такі можливості також можуть бути корисними, оскільки організатору важливо бачити список замовлень, кількість проданих квитків і загальну суму збору [2; 3; 4].

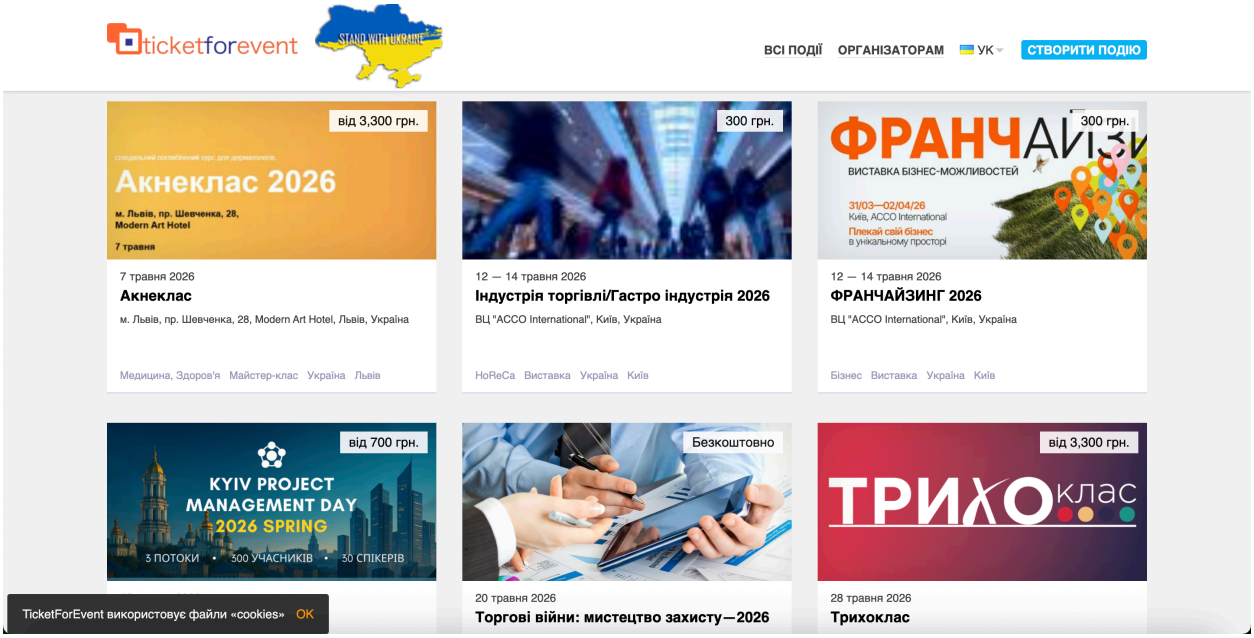


Рис. 1.6 Приклад інтерфейсу сервісу TicketForEvent

Перевагами TicketForEvent є підтримка електронних квитків, реєстрації учасників, приймання оплат і керування списками відвідувачів. Сервіс може бути корисним для подій, де важливо контролювати склад учасників [22].

Недоліком є те, що система більше орієнтована на реєстраційні події, конференції та форуми. Для благодійного концерту важлива не лише реєстрація або факт продажу квитка, а й зручне представлення події, артистів, благодійної мети, прогресу збору та персонального внеску користувача. Саме ці можливості планується реалізувати в DobroStage [16; 17].

1.3.5 Порівняльний аналіз застосунків-аналогів

Для DobroStage важливо врахувати сильні сторони аналогів: зручний каталог подій, просту сторінку концерту, можливість вибору квитків, кабінет користувача та інструменти для організатора. Водночас система повинна мати власну відмінність — «Паспорт внеску», який показує користувачу благодійний результат його замовлення [16; 17].

Таблиця 1.1

Порівняльний аналіз програмних аналогів

Показник	**Concert.ua**	**KARABAS**	**Eventbrite**	**DobroStage**
Підтримка благодійних подій	+	Окремої категорії немає	+	+
Онлайн-купівля квитків	+	+	+	+
Список покупців / учасників	Покупці квитків	Покупці квитків	Учасники події	Покупці + замовлення
Аналітика продажів	Кількість продажів	Кількість продажів	Продажі + реєстрації	Продажі + сума збору
Кабінет організатора	Для партнерів сервісу	Для партнерів сервісу	+	+
Керування користувачами та ролями	-	-	Команда організатора	USER / ORGANIZER / ADMIN
Управління майданчиками	Через заявку сервісу	Через заявку сервісу	Назва й адреса події	+
Налаштування благодійної цілі	Текст в описі	Текст в описі	Текст в описі	Cause + goal_amount
Відображення прогресу збору	-	-	-	+
Impact-етапи благодійної цілі	-	-	-	+
Паспорт внеску після купівлі квитка	-	-	-	+
Пояснення благодійного результату внеску	-	-	-	+
Зручність для невеликих благодійних концертів	Потрібна співпраця з сервісом	Потрібна співпраця з сервісом	Багато зайвих налаштувань	Створено під такі події
Основні переваги	Афіша, український ринок	Пошук подій, український ринок	Створення подій, реєстрації	Ролі, impact, Паспорт внеску

2 ПРОЄКТУВАННЯ WEB-ЗАСТОСУНКУ DOBROSTAGE ДЛЯ ОНЛАЙН-ПРОДАЖУ КВИТКІВ НА БЛАГОДІЙНІ КОНЦЕРТИ

2.1 Визначення вимог до web-застосунку

Проєктування web-застосунку DobroStage передбачає визначення вимог до майбутньої системи, її структури, ролей користувачів, основних модулів та способів взаємодії між клієнтською частиною, серверною логікою і базою даних. Оскільки система призначена для онлайн-продажу квитків на благодійні концерти, вона повинна забезпечувати не лише стандартний процес вибору та купівлі квитка, а й підтримувати прозорий облік благодійного внеску користувача [6; 7; 8].

Основною задачею web-застосунку є створення єдиного середовища, у якому гість може переглядати благодійні концерти, зареєстрований користувач — купувати квитки та переглядати власні замовлення, організатор — керувати своїми подіями, а адміністратор — контролювати роботу всієї платформи. Такий поділ дозволяє сформувати чітку рольову модель доступу та уникнути ситуацій, коли користувач отримує доступ до функцій, які не відповідають його ролі [2; 3; 4].

До основних функціональних вимог web-застосунку DobroStage належать: [6; 7; 8].

- реєстрація та авторизація користувачів у системі; [3; 15; 23].
- перегляд каталогу активних благодійних концертів; [1; 2].
- пошук і фільтрація подій за назвою, виконавцем, містом або благодійною метою; [1; 2].
- перегляд детальної інформації про концерт, включаючи опис, дату, місце проведення, виконавця, ціль збору та доступні типи квитків; [2; 3; 4].
- вибір типу квитка та кількості квитків; [2; 3; 4].
- оформлення замовлення авторизованим користувачем; [2; 3; 4].
- автоматична перевірка доступної кількості квитків перед створенням замовлення; [2; 3; 4].

- оновлення залишку доступних квитків після успішної купівлі; [2; 3; 4].
- перегляд користувачем власних замовлень на сторінці «Мої квитки»; [2; 3; 4].
- перегляд сторінки «Паспорт внеску» для конкретного замовлення; [2; 3; 4].
- редагування профілю користувача;
- створення та редагування концертів організатором;
- керування типами квитків для власних концертів організатора; [2; 3; 4].
- перегляд статистики продажів організатором;
- адміністративне керування концертами, майданчиками, користувачами, ролями та замовленнями [2; 3; 4].

Окремо слід виділити вимоги до унікальної функції «Паспорт внеску». Ця функція повинна розраховувати благодійний ефект замовлення користувача на основі суми покупки та impact-полів, заданих для конкретного концерту. Наприклад, для концерту може бути визначено, що одна impact-одиниця відповідає одному теплому набору вартістю 650 грн. Якщо користувач придбав квитки на 1300 грн, система повинна показати, що його внесок підтримав 2 теплі набори. Такий підхід дозволяє зробити благодійний результат більш зрозумілим і наочним [6; 7; 8].

Нефункціональні вимоги визначають якісні характеристики майбутнього програмного забезпечення. Для DobroStage важливими є швидкодія, зручність, безпека, адаптивність, цілісність даних і масштабованість. Основні сторінки системи повинні відкриватися протягом 2–3 секунд, а підтвердження замовлення має виконуватися не довше ніж за 5 секунд. Процес купівлі квитка повинен бути простим і складатися приблизно з 4–5 основних кроків: вибір концерту, вибір типу квитка, зазначення кількості, підтвердження замовлення та перегляд результату [3; 15; 23].

Для запобігання дублюванню продажів система повинна перевіряти доступність квитків безпосередньо перед оформленням замовлення. Якщо потрібна кількість квитків уже недоступна або концерт має статус скасованого,

система повинна повернути повідомлення про помилку та не створювати некоректне замовлення [2; 3; 4].

Ролі користувачів у системі поділяються на чотири основні групи: гість, користувач, організатор і адміністратор. Гість має доступ лише до відкритої частини сайту. Користувач може купувати квитки, переглядати профіль, замовлення та «Паспорт внеску». Організатор має доступ до власних концертів, типів квитків і статистики своїх подій. Адміністратор має повний доступ до керування платформою [2; 3; 4].

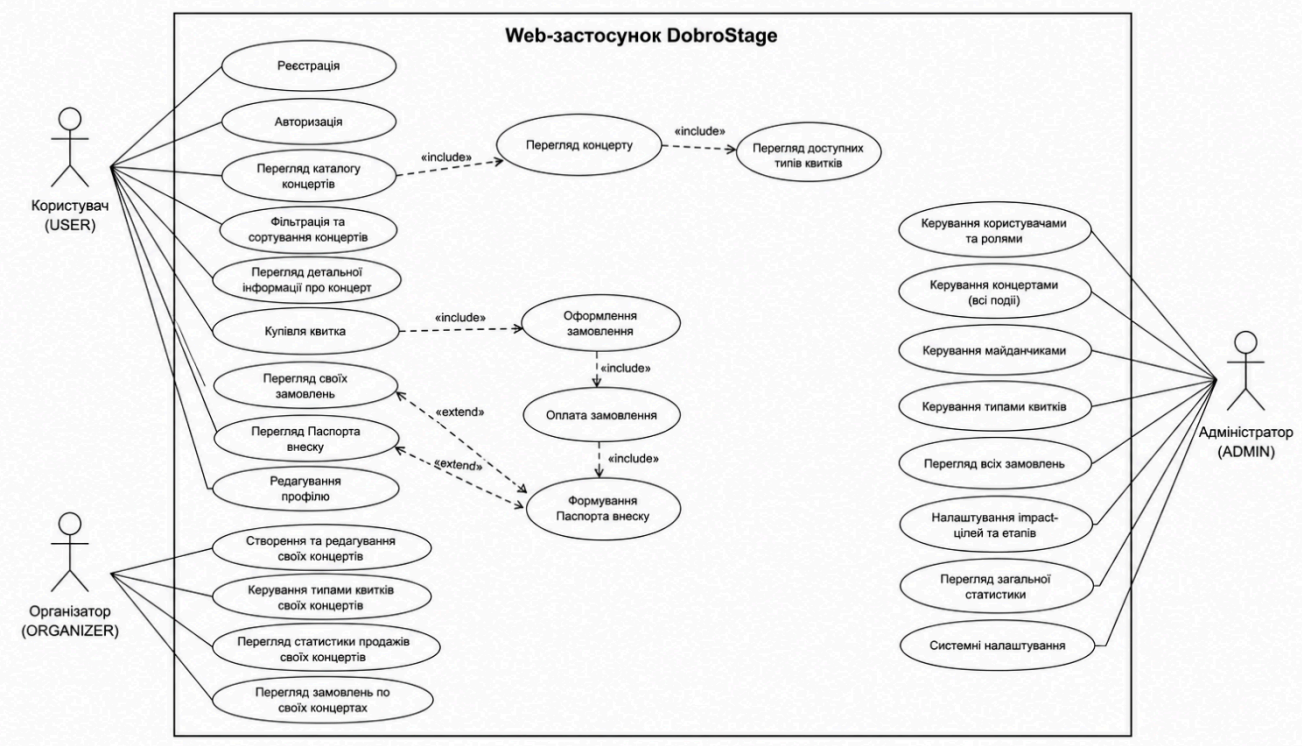


Рис. 2.1 Діаграма варіантів використання web-застосунку DobroStage

Діаграма варіантів використання відображає основні сценарії взаємодії користувачів із системою. Для гостя передбачено перегляд каталогу концертів, пошук і фільтрацію подій, перегляд сторінки концерту, а також перехід до входу або реєстрації. Зареєстрований користувач може оформлювати замовлення, переглядати власні квитки, відкривати «Паспорт внеску» та редагувати профіль. Організатор керує власними концертами, типами квитків і переглядає статистику

продажів. Адміністратор виконує повне керування платформою, включаючи користувачів, ролі, майданчики, концерти та замовлення [2; 3; 4].

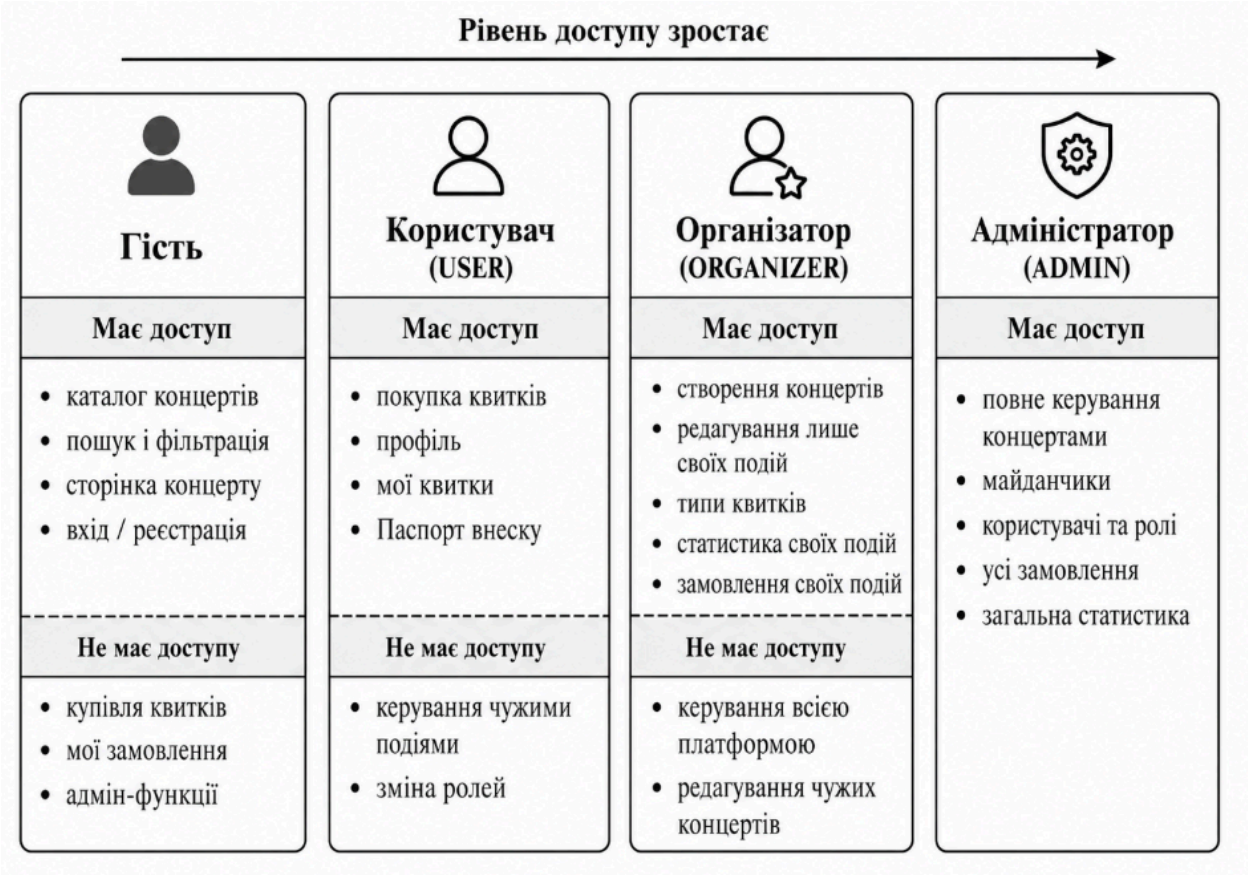


Рис. 2.2 Схема ролей користувачів та прав доступу

Схема ролей і прав доступу дозволяє уточнити, які можливості доступні кожному типу користувача. Така модель є важливою для правильного налаштування авторизації та захисту серверних endpoint-ів. Наприклад, звичайний користувач не повинен мати доступ до адміністративної панелі або зміни ролей інших користувачів, а організатор не повинен редагувати концерти, які були створені іншими організаторами [3; 15; 23].

2.2 Огляд засобів програмної реалізації

Для реалізації web-застосунку DobroStage обрано клієнт-серверну архітектуру. Такий підхід дозволяє розділити систему на клієнтську частину, серверну частину та базу даних. Клієнтська частина відповідає за інтерфейс

користувача, серверна частина — за бізнес-логіку, авторизацію, обробку замовлень і взаємодію з базою даних, а база даних — за зберігання інформації про користувачів, концерти, квитки, замовлення та майданчики [14; 24].

Для реалізації серверної частини використовується мова програмування Java. Вона добре підходить для створення web-застосунків із чіткою структурою, підтримує об’єктно-орієнтований підхід і має розвинену екосистему інструментів для роботи з базами даних, безпекою, REST API та тестуванням. Використання Java дозволяє створити надійну серверну логіку, яка може бути розширена в майбутньому [3; 15; 23].

Основним фреймворком для backend-частини є Spring Boot. Він спрощує створення серверних застосунків, дозволяє швидко налаштувати REST-контролери, сервіси, репозиторії, підключення до бази даних і конфігурацію безпеки. У DobroStage Spring Boot використовується для реалізації API авторизації, каталогу концертів, замовлень, кабінету організатора та адміністративної панелі [23; 24].

Для захисту системи використовується Spring Security разом із JWT-авторизацією. Після входу користувач отримує токен, який надалі передається разом із запитами до захищених endpoint-ів. Це дозволяє серверу визначати, хто саме виконує запит і які права має цей користувач. Такий підхід є зручним для SPA-застосунків, де клієнтська частина активно взаємодіє з REST API [23; 24].

Для роботи з базою даних використовується Spring Data JPA та Hibernate. Ці засоби дозволяють працювати з таблицями бази даних через Java-класи, які описують сутності предметної галузі. Наприклад, сутність User відповідає таблиці користувачів, Concert — таблиці концертів, TicketType — типам квитків, TicketOrder — замовленням, а OrderItem — позиціям замовлення. Такий підхід спрощує роботу з даними та дозволяє зменшити кількість SQL-коду в бізнес-логіці [23; 24].

Як систему керування базами даних обрано PostgreSQL. Вона підходить для зберігання структурованих даних і підтримує зв’язки між таблицями, обмеження цілісності, транзакції та складні запити. Для DobroStage це важливо, оскільки дані

про користувачів, концерти, майданчики, типи квитків і замовлення мають бути логічно пов’язані між собою. Крім того, транзакційність є важливою для процесу оформлення замовлення, оскільки потрібно одночасно створити запис про замовлення, додати позиції замовлення та зменшити кількість доступних квитків [14; 24].

Для клієнтської частини використовується React. Ця бібліотека дозволяє створювати компонентний інтерфейс, де кожна сторінка або функціональний блок може бути представлений окремим компонентом. Наприклад, каталог концертів, картка концерту, форма входу, профіль користувача, таблиця замовлень або блок «Паспорт внеску» можуть бути реалізовані як окремі компоненти. Це спрощує підтримку та подальше розширення інтерфейсу [14; 24].

Для швидкого запуску та збирання frontend-частини використовується Vite. Він забезпечує швидке середовище розробки, підтримує сучасні можливості JavaScript і дозволяє зручно працювати з React-застосунками. Для стилізації інтерфейсу можуть використовуватися CSS або Tailwind CSS залежно від структури проєкту. Основна вимога до інтерфейсу — простота, зрозумілість і адаптивність [25; 26].

Для перевірки API може використовуватися Postman або аналогічний інструмент. Він дозволяє надсилати запити до серверної частини, перевіряти відповіді, тестувати авторизацію, створення замовлень, отримання списку концертів і доступ до адміністративних endpoint-ів. Для контролю версій використовується Git, що дозволяє зберігати історію змін, працювати з окремими гілками та контролювати розвиток програмного коду [3; 15; 23].

2.3 Проєктування архітектури та бази даних системи

Загальна архітектура DobroStage побудована за клієнт-серверним принципом. Користувач працює з web-інтерфейсом, реалізованим на React. Клієнтська частина надсилає HTTP-запити до серверної частини, яка реалізована

на Java Spring Boot. Серверна частина обробляє запити, перевіряє права доступу, виконує бізнес-логіку та звертається до бази даних PostgreSQL [23; 24].

Такий підхід дозволяє розділити відповідальність між основними частинами системи. Frontend відповідає за відображення даних, форми, навігацію та взаємодію користувача з інтерфейсом. Backend відповідає за авторизацію, перевірку ролей, роботу з концертами, замовленнями, типами квитків, користувачами та impact-логікою. База даних відповідає за збереження всієї структурованої інформації [14; 24].

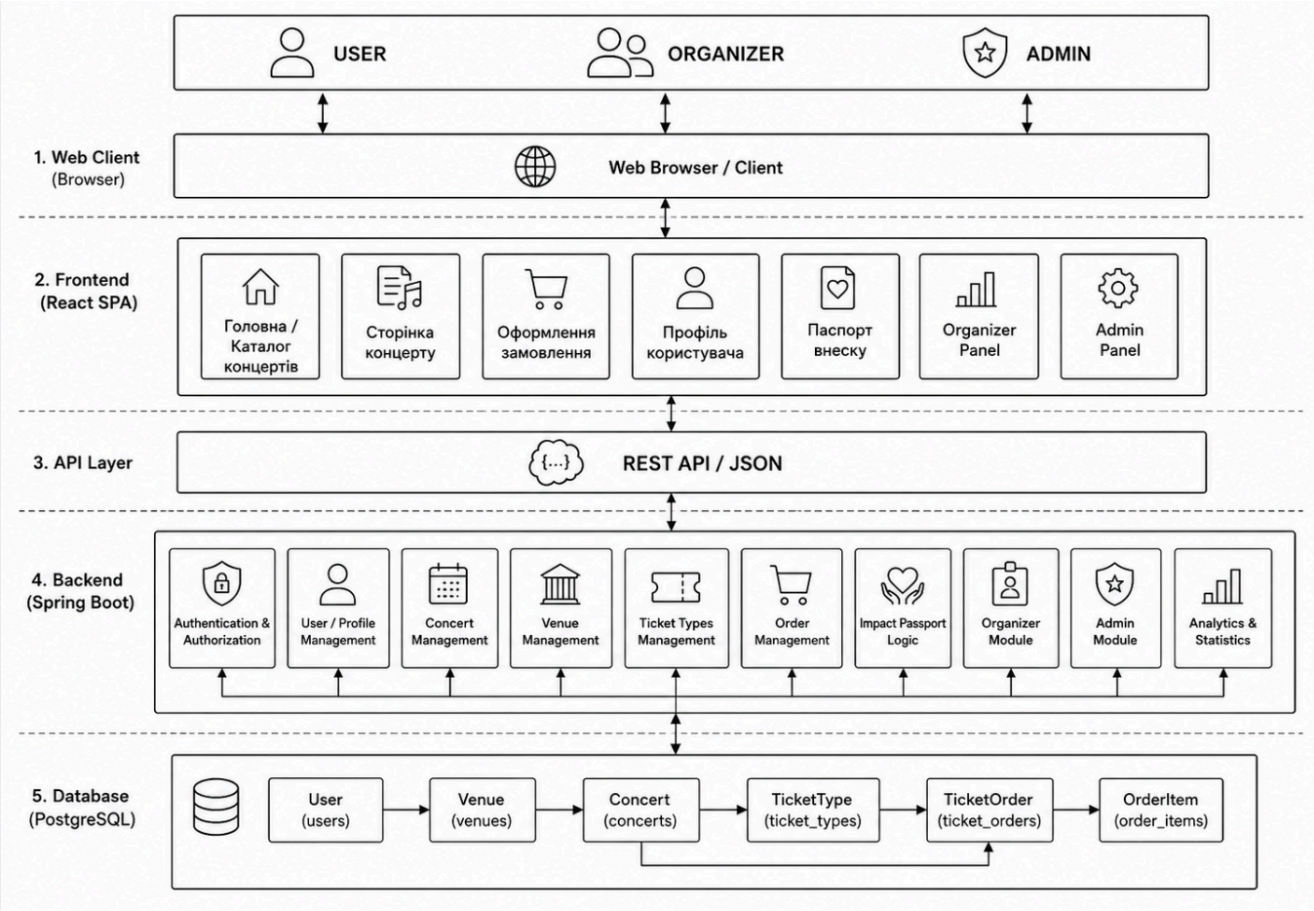


Рис. 2.3 Загальна архітектура web-застосунку DobroStage

У загальній архітектурі можна виділити три основні рівні. Перший рівень — клієнтський, де розміщені сторінки головного каталогу, сторінка концерту, сторінки входу та реєстрації, профіль користувача, сторінка «Мої квитки», «Паспорт внеску», кабінет організатора та адміністративна панель. Другий рівень — серверний, який містить REST API контролери, сервіси бізнес-логіки, модуль

безпеки та рівень доступу до даних. Третій рівень — база даних PostgreSQL, де зберігаються таблиці users, venues, concerts, ticket_types, orders та order_items [14; 24].

Проєктування бази даних є одним із ключових етапів розробки DobroStage, оскільки система повинна коректно зберігати взаємопов’язані дані. Основною таблицею для користувачів є users. Вона містить email, хеш пароля, повне ім’я, телефон, місто, опис профілю, роль і дату створення акаунту. Поле role визначає рівень доступу користувача в системі: USER, ORGANIZER або ADMIN [14; 24].

Таблиця venues зберігає інформацію про майданчики проведення концертів. До її основних полів належать назва майданчика, місто, адреса та місткість. Ці дані використовуються на сторінці концерту та в адміністративній панелі.

Таблиця concerts містить основну інформацію про благодійні концерти. Вона пов’язана з таблицею venues через поле venue_id і з таблицею users через поле organizer_id. Це дозволяє визначити, на якому майданчику проходить концерт і який організатор відповідає за цю подію. Також таблиця concerts містить назву концерту, виконавця, благодійну мету, опис, дату проведення, посилання на постер, цільову суму збору, impact-поля та статус концерту.

Таблиця ticket_types використовується для зберігання типів квитків. Один концерт може мати декілька типів квитків, наприклад «Стандарт», «Підтримка», «Донор» або «Преміум». Для кожного типу квитка зберігається назва, опис, ціна, загальна кількість і доступний залишок.

Таблиця orders зберігає замовлення користувачів. Вона містить унікальний код замовлення, посилання на користувача, статус, загальну суму та дату створення. Таблиця order_items деталізує склад замовлення: який тип квитка був придбаний, у якій кількості та за якою ціною на момент покупки.

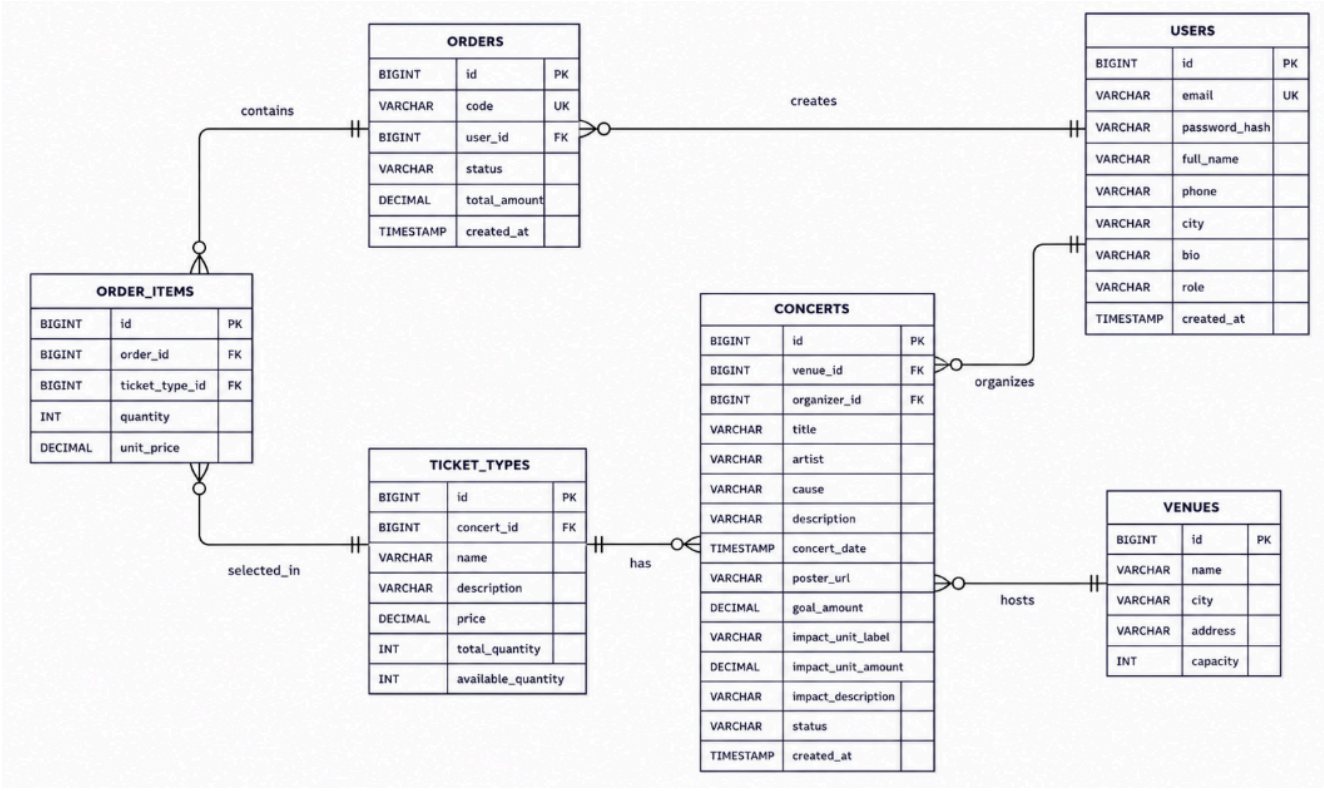


Рис. 2.4 Структура бази даних DobroStage

Зв’язки між таблицями дозволяють забезпечити цілісність даних. Один користувач може мати багато замовлень, один концерт може мати багато типів квитків, один майданчик може використовуватися для багатьох концертів, а одне замовлення може складатися з декількох позицій. Така структура є достатньою для реалізації основного функціоналу системи та може бути розширена в майбутньому, наприклад, додаванням онлайн-оплат, QR-квитків, відгуків або звітів для благодійних організацій [14; 24].

2.4 Проєктування основних функціональних модулів DobroStage

Web-застосунок DobroStage доцільно поділити на декілька основних функціональних модулів. Такий підхід дозволяє логічно розділити функції системи та полегшити подальшу реалізацію, тестування і підтримку програмного продукту [8; 10; 27].

Першим модулем є модуль авторизації та реєстрації. Він відповідає за створення нового акаунту, вхід користувача в систему, перевірку email і пароля, формування JWT-токена та збереження інформації про авторизованого користувача на клієнтській стороні. Цей модуль також є основою для подальшого розмежування прав доступу [3; 15; 23].

Другим модулем є публічний каталог концертів. Він доступний навіть неавторизованим користувачам і дозволяє переглядати активні благодійні концерти. На головній сторінці користувач може бачити картки подій, інформацію про виконавця, місто, дату, благодійну мету, прогрес збору та кількість доступних квитків. Також передбачається пошук і фільтрація за назвою, артистом, містом або метою збору [2; 3; 4].

Третім модулем є сторінка детального перегляду концерту. Вона містить постер, назву, артиста, дату, місце проведення, опис благодійної мети, прогрес збору, організатора, статус концерту та список доступних типів квитків. Саме на цій сторінці користувач вибирає кількість квитків і переходить до оформлення замовлення [2; 3; 4].

Четвертий модуль — модуль замовлень. Він відповідає за створення замовлення, перевірку статусу концерту, перевірку доступної кількості квитків, розрахунок загальної суми, зменшення залишку квитків і збереження позицій замовлення. Цей модуль повинен працювати особливо коректно, оскільки помилки в ньому можуть призвести до некоректного обліку квитків [11; 12; 13].

П’ятий модуль — кабінет користувача. У ньому користувач може переглядати власний профіль, редагувати персональні дані, бачити загальну суму своїх внесків, кількість куплених квитків і список останніх замовлень. Окремо передбачена сторінка «Мої квитки», де відображаються всі замовлення користувача [11; 12; 13].

Шостий модуль — «Паспорт внеску». Він є унікальною особливістю DobroStage. Для формування цієї сторінки система отримує дані про замовлення, користувача, придбані квитки та impact-поля концерту. Далі розраховується кількість impact-одиниць за формулою: [11; 12; 13].

кількість impact-одиниць = сума замовлення / вартість однієї impact-одиниці [2; 3; 4].

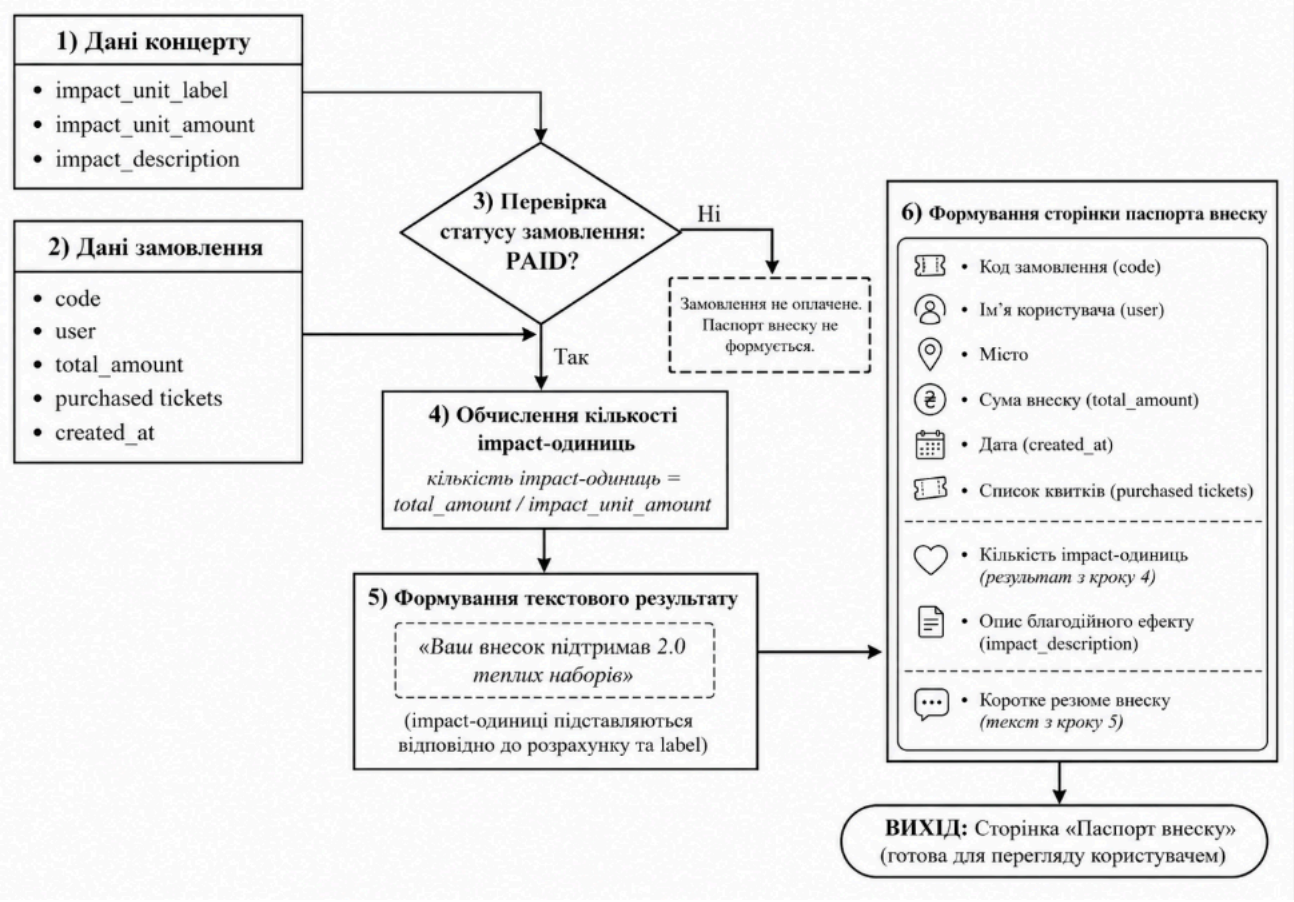


Рис. 2.5 Схема формування “Паспорта внеску”

Після цього система формує текстовий результат, наприклад: «Ваш внесок підтримав 2.0 теплих наборів». Так користувач бачить не лише суму витрачених коштів, а й конкретне пояснення, який благодійний ефект створило його замовлення [2; 3; 4].

Сьомим модулем є кабінет організатора. Він доступний користувачам із роллю ORGANIZER та ADMIN. Організатор може створювати власні концерти,

редагувати лише свої події, додавати та змінювати типи квитків, переглядати статистику своїх концертів і бачити замовлення, пов’язані зі своїми подіями. Важливо, що організатор не має права керувати чужими концертами або змінювати ролі користувачів [2; 3; 4].

Восьмим модулем є адміністративна панель. Вона доступна лише користувачу з роллю ADMIN. Адміністратор може керувати всіма концертами, майданчиками, користувачами, ролями та замовленнями. Також він може призначати організаторів, редагувати будь-які події, керувати типами квитків і переглядати загальну статистику платформи [2; 3; 4].

Окрему увагу під час проєктування було приділено зв’язку між основними модулями системи. Модуль замовлень взаємодіє з каталогом концертів, типами квитків, профілем користувача та функцією «Паспорт внеску», тому саме він є центральним елементом бізнес-логіки DobroStage. Завдяки цьому після оформлення замовлення система не лише фіксує факт купівлі квитків, а й оновлює доступний залишок, зберігає дані про придбані квитки та формує основу для подальшого розрахунку благодійного ефекту [2; 3; 4].

Узагальнюючи, проєктування DobroStage передбачає створення web-застосунку з чіткою клієнт-серверною архітектурою, реляційною базою даних, рольовою моделлю доступу та окремими функціональними модулями. Такий підхід дозволяє забезпечити зручність для користувачів, контроль для організаторів, повне адміністрування для адміністратора та прозоре відображення благодійного ефекту завдяки функції «Паспорт внеску» [14; 24].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ WEB-ЗАСТОСУНКУ DOBROSTAGE

3.1 Реалізація серверної частини web-застосунку

Програмна реалізація web-застосунку DobroStage передбачає створення клієнт-серверної системи, у якій серверна частина відповідає за обробку запитів, роботу з базою даних, авторизацію користувачів, перевірку прав доступу, керування концертами, оформлення замовлень і формування «Паспорта внеску». Серверна частина реалізована з використанням мови програмування Java та фреймворку Spring Boot [23; 24].

Основним класом запуску системи є DobroStageApplication. Він ініціалізує Spring Boot-застосунок, завантажує необхідні компоненти, конфігурації, сервіси, контролери та репозиторії. Після запуску серверна частина починає приймати HTTP-запити від клієнтської частини через REST API [23; 24].

Структура backend-частини побудована за багаторівневим принципом. У системі виділено рівень контролерів, рівень сервісів, рівень репозиторіїв, рівень сутностей та модуль безпеки. Такий підхід дозволяє розділити відповідальність між окремими частинами програми. Контролери приймають запити від frontend-частини, сервіси виконують бізнес-логіку, репозиторії відповідають за доступ до бази даних, а entity-класи описують основні таблиці системи [14; 24].

До основних entity-класів системи належать User, Venue, Concert, TicketType, TicketOrder та OrderItem. Клас User описує користувача системи та містить такі дані, як email, хеш пароля, повне ім’я, номер телефону, місто, опис профілю та роль. Роль користувача визначає його права в системі: звичайний користувач може купувати квитки, організатор — керувати власними подіями, адміністратор — керувати всією платформою [3; 15; 23].

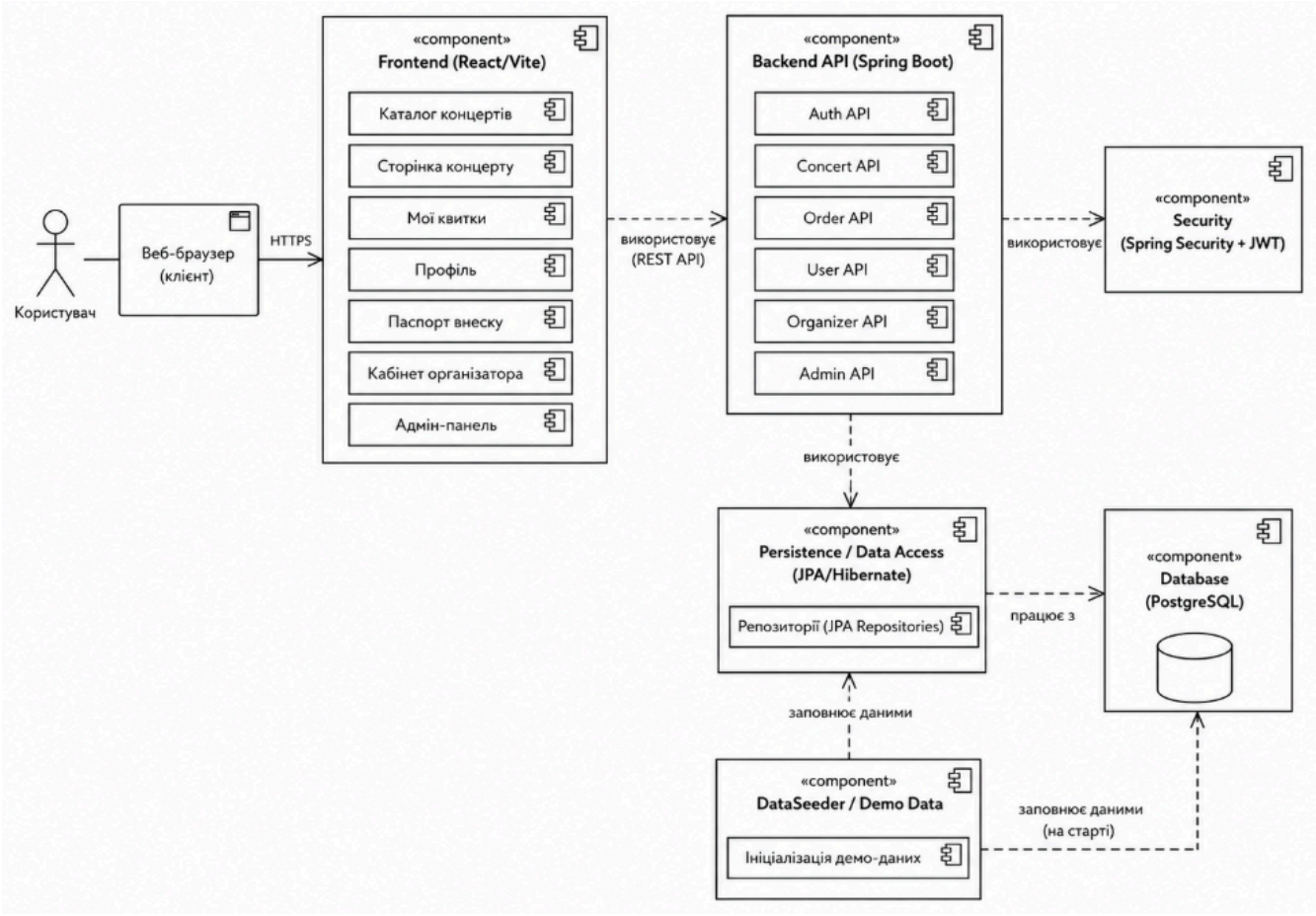


Рис. 3.1 Діаграма компонентів web-застосунку DobroStage

Клас Venue відповідає за майданчики проведення концертів. Він містить назву майданчика, місто, адресу та місткість. Клас Concert описує благодійний концерт: назву, виконавця, благодійну мету, опис, дату проведення, постер, цільову суму збору, статус, організатора та майданчик. Також у цьому класі зберігаються impact-поля, які використовуються для формування «Паспорта внеску» [1; 2].

Клас TicketType відповідає за типи квитків конкретного концерту. Один концерт може мати декілька типів квитків, наприклад стандартний, донорський або преміальний. Для кожного типу квитка зберігається назва, опис, ціна, загальна кількість і доступний залишок. Клас TicketOrder описує замовлення користувача, а OrderItem — окрему позицію в замовленні, тобто конкретний тип квитка та його кількість [2; 3; 4].

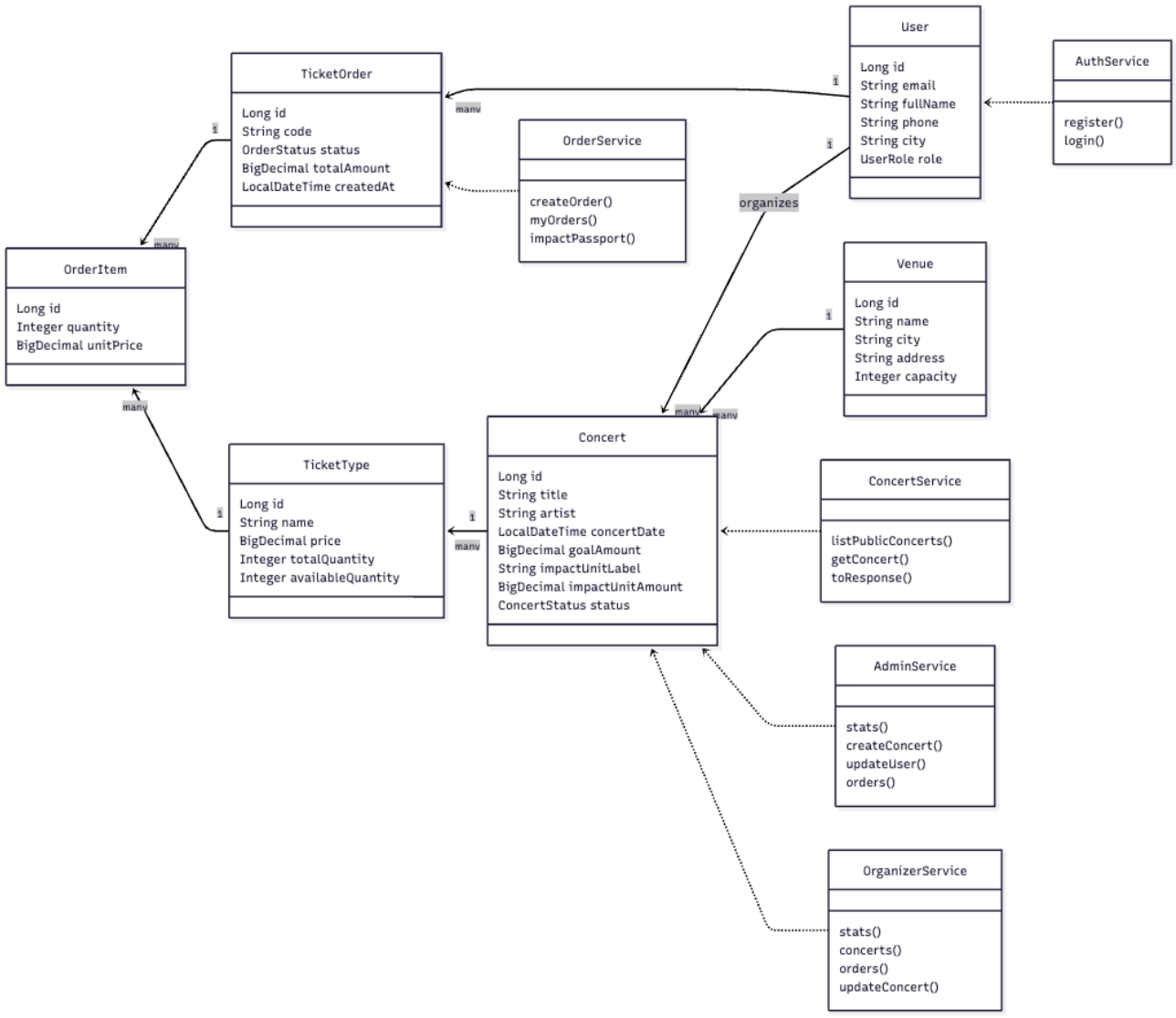


Рис. 3.2 Діаграма класів серверної частини DobroStage

Для роботи з користувачами, концертами, замовленнями та адміністративними функціями реалізовано набір сервісних класів. AuthService відповідає за реєстрацію та вхід користувача. Під час реєстрації система перевіряє унікальність email, хешує пароль і створює нового користувача з базовою роллю. Під час входу система перевіряє облікові дані та формує JWT-токен, який використовується для подальшої авторизації [3; 15; 23].

UserService реалізує роботу з профілем користувача. Він дозволяє отримати поточного користувача, сформувати відповідь для frontend-частини, оновити персональні дані, а також отримати список користувачів для адміністративної

панелі. Для адміністратора передбачена можливість змінювати дані користувача та його роль [25].

ConcertService відповідає за роботу з концертами. Він забезпечує отримання публічного списку активних подій, перегляд повного списку концертів для адміністратора, отримання детальної інформації про конкретний концерт і перетворення entity-об’єктів у DTO-відповіді для клієнтської частини. Використання DTO дозволяє передавати frontend-частині тільки ті дані, які необхідні для відображення інтерфейсу [25].

OrderService є одним із ключових сервісів системи. Він відповідає за створення замовлення, перегляд замовлень користувача, отримання всіх замовлень для адміністратора, перегляд замовлень організатора та формування «Паспорта внеску». Під час створення замовлення сервіс перевіряє статус концерту, доступність потрібної кількості квитків, обчислює загальну суму, зменшує залишок доступних квитків і зберігає замовлення в базі даних [2; 3; 4].

OrganizerService реалізує функції кабінету організатора. Організатор може переглядати статистику тільки своїх подій, створювати та редагувати власні концерти, додавати типи квитків і переглядати замовлення, пов’язані з його концертами. Це дозволяє обмежити доступ організатора лише до тих даних, за які він відповідає [2; 3; 4].

AdminService відповідає за адміністративну логіку. Адміністратор може переглядати загальну статистику платформи, керувати користувачами, концертами, майданчиками та замовленнями. На відміну від організатора, адміністратор має повний доступ до всіх даних системи [2; 3; 4].

Для взаємодії з клієнтською частиною реалізовано REST-контролери. AuthController обробляє запити реєстрації та входу. ConcertController відповідає за публічний каталог концертів і сторінку конкретної події. VenueController надає список майданчиків. UserController реалізує роботу з профілем користувача. OrderController відповідає за створення замовлень, перегляд власних квитків і відкриття «Паспорта внеску». OrganizerController надає API для кабінету організатора, а AdminController — API для адміністративної панелі [23; 25].

Захист системи реалізовано за допомогою Spring Security та JWT. Клас JwtTokenProvider створює та перевіряє токени, а JwtAuthenticationFilter обробляє вхідні HTTP-запити, перевіряє наявність токена та встановлює авторизованого користувача в контекст безпеки. Клас SecurityConfig визначає правила доступу до endpoint-ів. Публічні сторінки каталогу концертів доступні всім користувачам, а створення замовлення, перегляд профілю, кабінет організатора та адміністративна панель доступні лише користувачам із відповідними ролями [23; 24].

3.2 Реалізація клієнтської частини web-застосунку

Клієнтська частина DobroStage реалізована як web-інтерфейс, з яким безпосередньо взаємодіє користувач. Основною задачею frontend-частини є зручне відображення концертів, форм, замовлень, профілю, статистики та адміністративних даних. Для реалізації клієнтської частини використовується React та Vite [25; 26].

React дозволяє будувати інтерфейс із незалежних компонентів. Завдяки цьому окремі частини сторінок можуть бути реалізовані як самостійні блоки: навігаційна панель, картка концерту, форма входу, форма реєстрації, таблиця замовлень, блок вибору квитків, інформаційний блок «Паспорт внеску», панель організатора та адміністративні вкладки [14; 24].

Головна сторінка web-застосунку виконує роль каталогу благодійних концертів. На ній розміщується hero-блок проєкту, пошуковий рядок, фільтри за містом, максимальною ціною, сортуванням і список активних подій. Кожна картка концерту містить назву, виконавця, місто, дату, благодійну мету, прогрес збору та кількість доступних квитків. З цієї сторінки користувач може перейти до детального перегляду концерту або одразу почати процес купівлі [2; 3; 4].

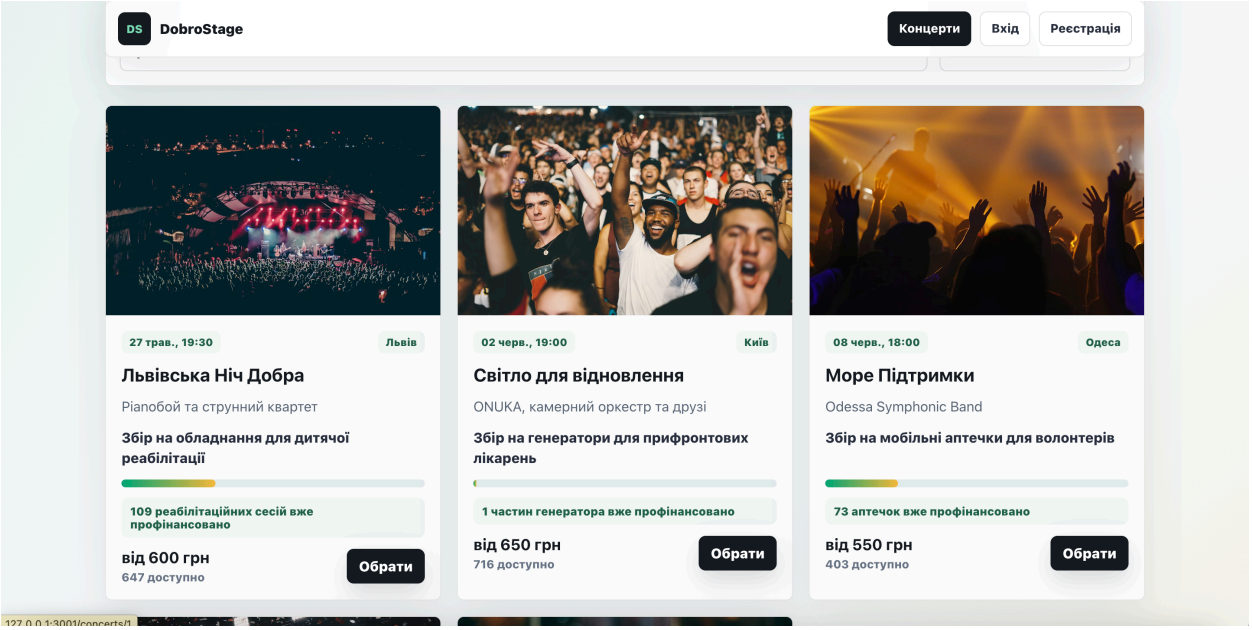


Рис. 3.3 Головна сторінка з каталогом концертів

Сторінка детального перегляду концерту містить повну інформацію про подію. На ній відображається постер, назва концерту, виконавець, дата, час, майданчик, опис благодійної мети, прогрес збору, організатор, статус концерту та список доступних типів квитків. Користувач може вибрати потрібний тип квитка, зазначити кількість і перейти до оформлення замовлення [2; 3; 4].

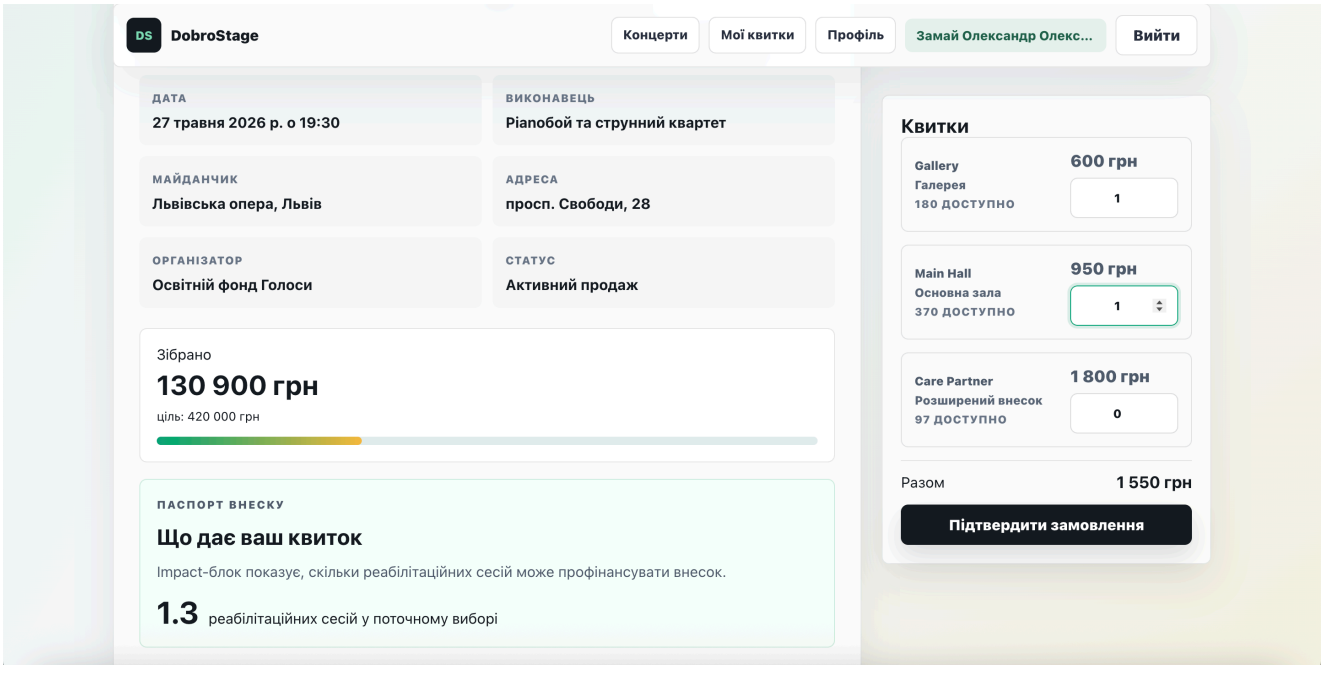


Рис. 3.4 Сторінка детального перегляду концерту

Сторінки входу та реєстрації забезпечують авторизацію користувача в системі. Під час входу користувач вводить email і пароль. Після успішної авторизації frontend отримує JWT-токен, зберігає його та використовує для подальших запитів до захищених API. Сторінка реєстрації дозволяє створити новий обліковий запис, вказавши повне ім’я, email і пароль [25].

Сторінка «Мої квитки» доступна лише авторизованим користувачам. Вона містить список замовлень користувача, код замовлення, дату, суму, назву концерту, майданчик, придбані квитки, кількість, статус і кнопку переходу до «Паспорта внеску». Завдяки цьому користувач може швидко переглянути історію власних покупок [2; 3; 4].

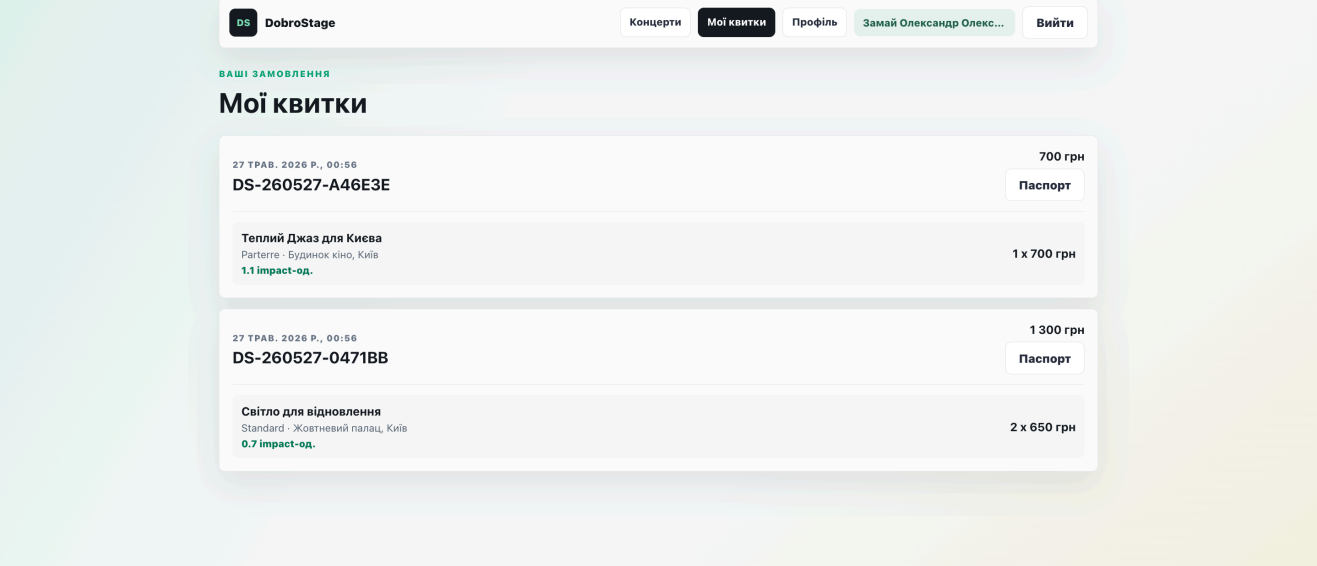


Рис. 3.5 Сторінка “Мої квитки”

Сторінка «Паспорт внеску» є унікальною для проєкту DobroStage. Вона показує користувачу код замовлення, дату, суму, список куплених квитків, розраховану кількість impact-одиниць, опис благодійного ефекту та текстове резюме внеску. Ця сторінка пояснює користувачу, який конкретний результат створив його внесок [2; 3; 4].

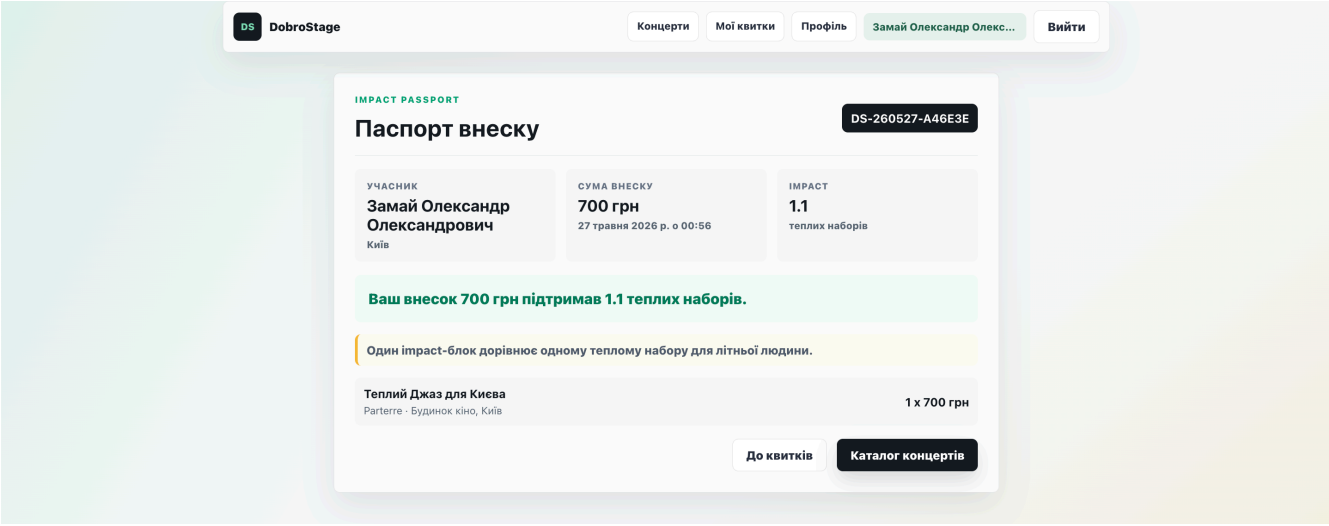


Рис. 3.6 Сторінка “Паспорт внеску”

Кабінет організатора доступний користувачам із роллю ORGANIZER та ADMIN. У ньому відображається статистика організатора, кількість його концертів, кількість активних подій, продані квитки, сума зібраних коштів, список власних концертів, форма створення або редагування концерту, керування типами квитків і останні замовлення по власних подіях [2; 3; 4].

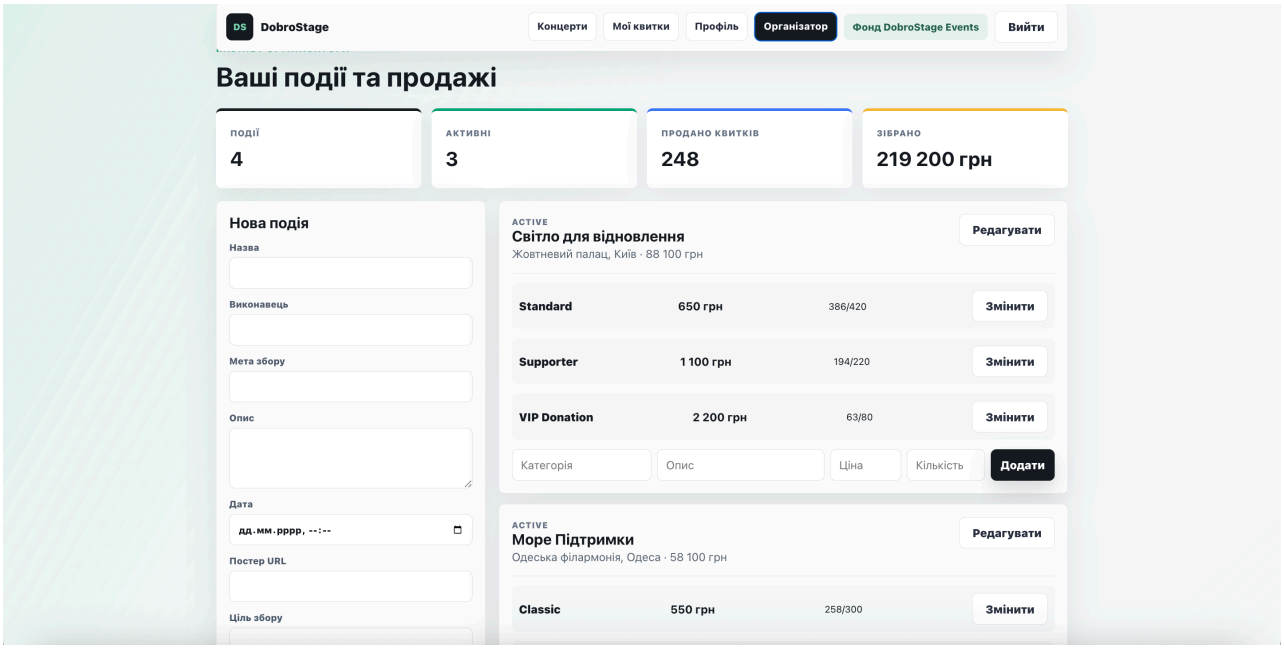


Рис. 3.7 Кабінет організатора

Адміністративна панель доступна лише користувачам із роллю ADMIN. Вона містить вкладки для керування концертами, майданчиками, користувачами та

замовленнями. Адміністратор може створювати й редагувати концерти, призначати організаторів, задавати impact-поля, керувати типами квитків, створювати майданчики, змінювати ролі користувачів і переглядати всі замовлення системи [2; 3; 4].

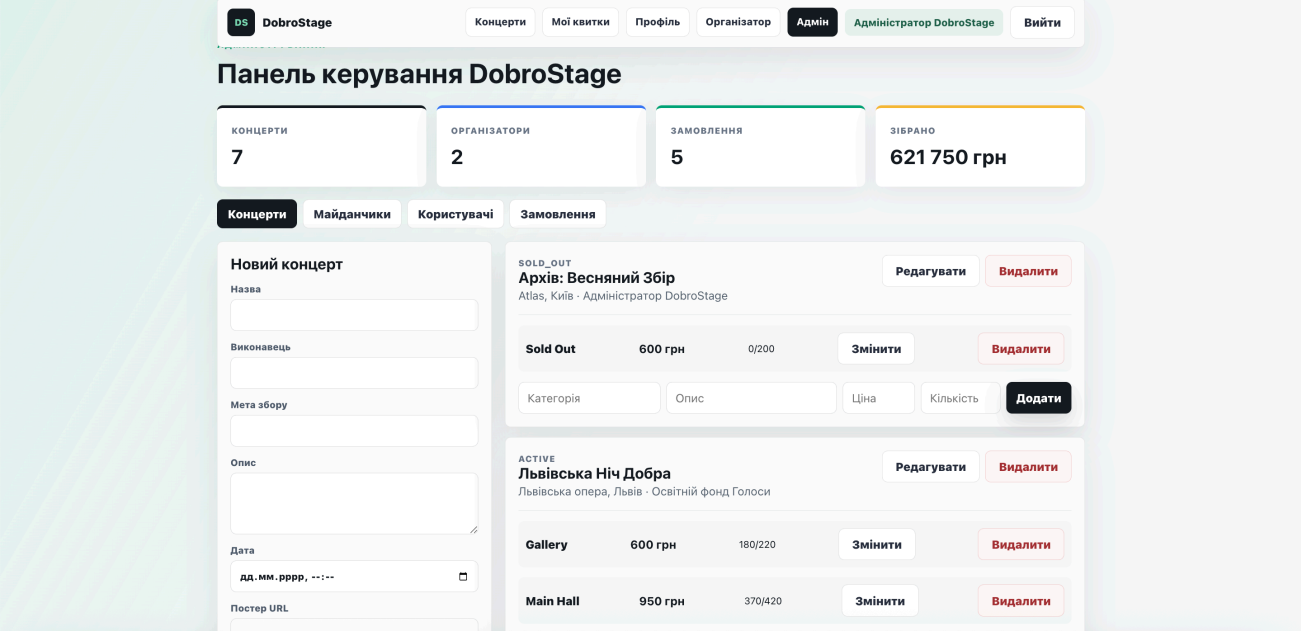


Рис. 3.8 Адміністративна панель

3.3 Реалізація взаємодії користувачів із системою

У web-застосунку DobroStage реалізовано декілька основних сценаріїв взаємодії користувачів із системою. Кожен сценарій залежить від ролі користувача та його прав доступу [8; 18].

Гість може взаємодіяти лише з відкритою частиною системи. Він має можливість переглядати каталог благодійних концертів, використовувати пошук і фільтрацію, відкривати сторінку конкретного концерту, переглядати опис події, дату, місто, майданчик, благодійну мету, прогрес збору та доступні квитки. Якщо гість хоче оформити замовлення, система спрямовує його до сторінки входу або реєстрації [2; 3; 4].

Зареєстрований користувач має доступ до повного сценарію купівлі квитків. Спочатку він обирає концерт у каталозі, відкриває сторінку події, вибирає тип

квитка та кількість. Після цього frontend надсилає запит до backend-частини на створення замовлення. Сервер перевіряє, чи концерт активний, чи доступна потрібна кількість квитків, обчислює суму замовлення, зменшує залишок квитків і зберігає замовлення в базі даних. Після успішного оформлення користувач може переглянути замовлення на сторінці «Мої квитки» та відкрити «Паспорт внеску» [25].

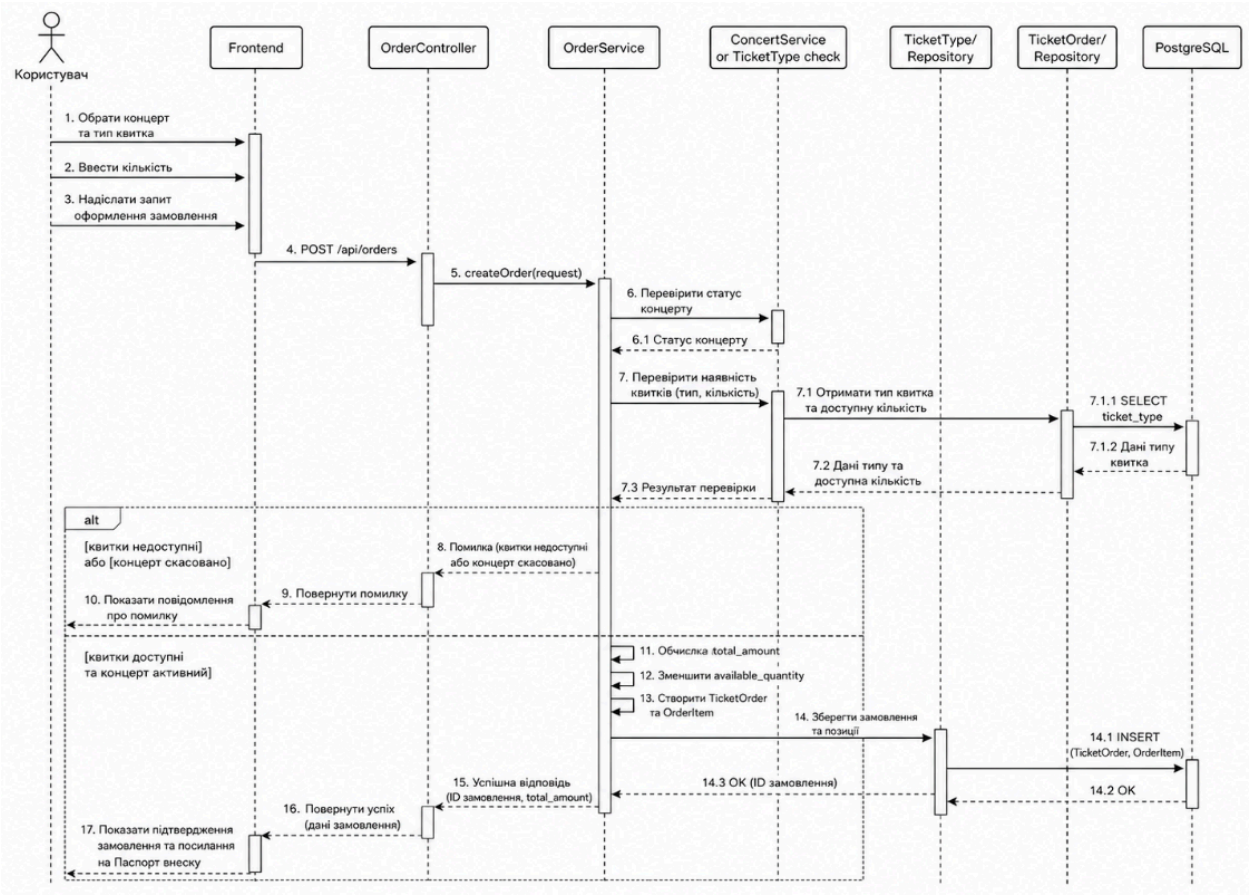


Рис. 3.9 Діаграма послідовності оформлення замовлення на квитки

Організатор взаємодіє із системою через кабінет організатора. Після входу він може переглянути статистику власних подій, створити новий концерт, редагувати інформацію про свої концерти, додавати або змінювати типи квитків, а також переглядати замовлення, пов’язані з його подіями. Усі запити організатора перевіряються на backend-рівні, щоб він не міг змінювати чужі концерти або переглядати дані, які не належать до його подій [2; 3; 4].

Адміністратор має доступ до повного керування системою. Він може працювати з усіма концертами, майданчиками, користувачами, ролями та замовленнями. На рівні backend-частини доступ до адміністративних endpoint-ів дозволений лише користувачам із роллю ADMIN. Це запобігає несанкціонованому доступу до критичних функцій системи [23; 25].

Frontend-частина взаємодіє з backend-частиною через REST API. Для публічних сторінок, таких як каталог концертів або сторінка події, запити можуть виконуватися без токена. Для захищених сторінок, наприклад профілю, моїх квитків, кабінету організатора або адміністративної панелі, frontend додає JWT-токен до заголовка запиту. Якщо токен відсутній або недійсний, користувач перенаправляється на сторінку входу [25].

3.4 Реалізація унікальної функції «Паспорт внеску»

Функція «Паспорт внеску» є ключовою відмінністю DobroStage від звичайних сервісів продажу квитків. Її мета полягає в тому, щоб користувач бачив не лише факт купівлі квитка, а й зрозумілий благодійний результат свого внеску [2; 3; 4].

Для реалізації цієї функції в таблиці концертів передбачено спеціальні поля: impact_unit_label, impact_unit_amount та impact_description. Поле impact_unit_label визначає назву одиниці благодійного результату, наприклад «теплих наборів», «медичних комплектів» або «продуктових пакетів». Поле impact_unit_amount визначає вартість однієї impact-одиниці. Поле impact_description містить пояснення, на що саме спрямовується благодійний внесок [14; 24].

Після оформлення замовлення користувач може відкрити сторінку «Паспорт внеску». Frontend надсилає запит до endpoint-а GET /api/orders/{id}/impact. Backend перевіряє, чи має користувач право переглядати це замовлення. Якщо запит виконує звичайний користувач, він може переглядати лише власні замовлення. Організатор може переглядати замовлення, пов’язані з його концертами, а адміністратор — усі замовлення системи [25].

Після перевірки доступу OrderService отримує дані замовлення, користувача, придбані квитки та impact-поля концерту. Далі система перевіряє, чи має замовлення статус PAID. Якщо замовлення не оплачене або скасоване, «Паспорт внеску» не формується. Якщо замовлення активне, виконується розрахунок кількості impact-одиниць [2; 3; 4].

Наприклад, якщо користувач придбав квиток на суму 1300 грн, а одна impact-одиниця дорівнює 650 грн, система розраховує 2 impact-одиниці. Після цього формується текстовий результат: «Ваш внесок підтримав 2.0 теплих наборів» [1; 2].

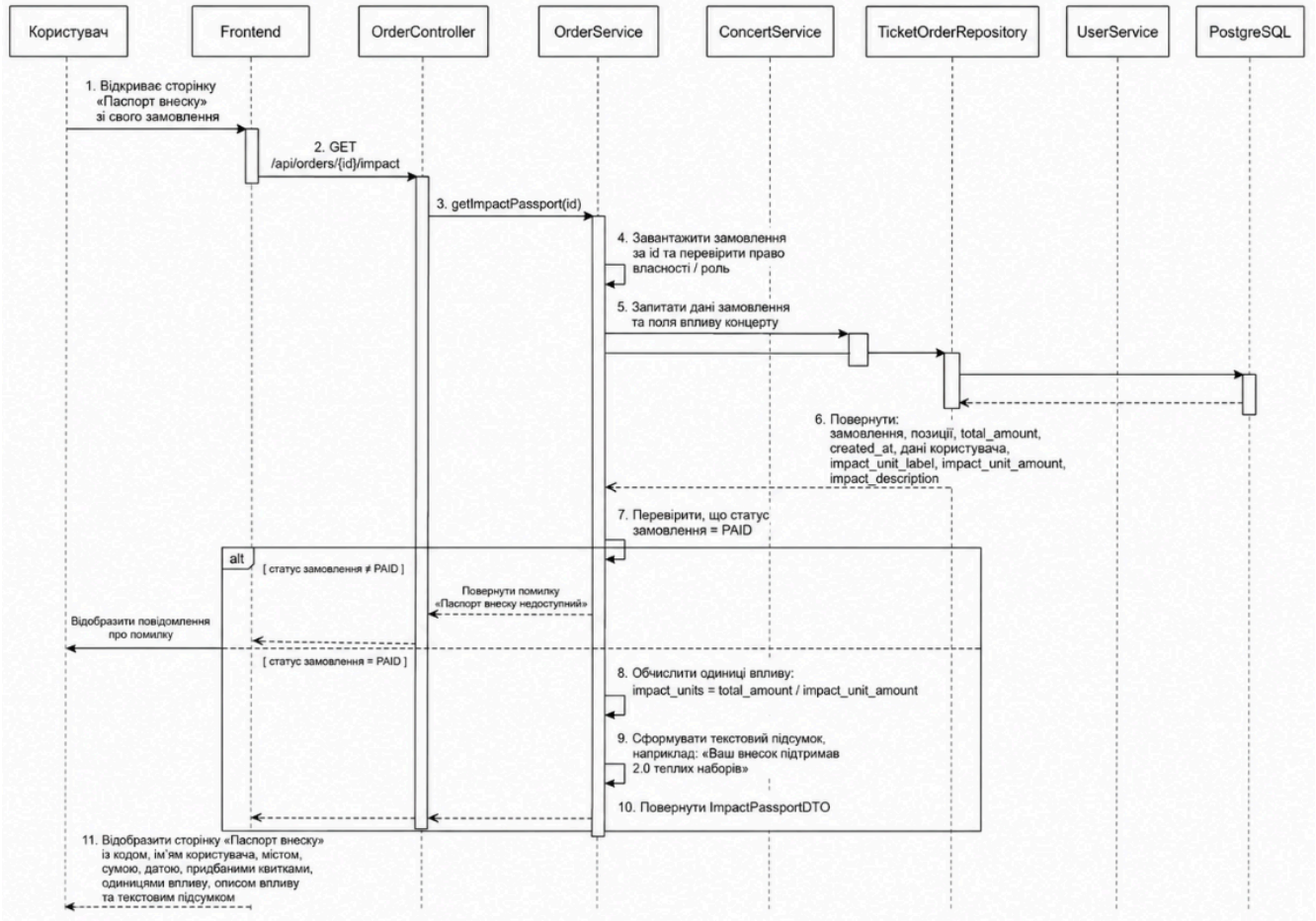


Рис. 3.10 Діаграма послідовності формування “Паспорта внеску”

4 ТЕСТУВАННЯ WEB-ЗАСТОСУНКУ DOBROSTAGE

4.1 Ручне тестування основних функцій системи

Тестування web-застосунку DobroStage виконувалося з метою перевірки коректності роботи основних функцій системи, відповідності реалізованого програмного забезпечення поставленим вимогам, правильності обробки користувацьких дій, захисту сторінок за ролями доступу та коректного збереження даних у базі даних [8; 10; 27].

Оскільки система призначена для онлайн-продажу квитків на благодійні концерти, основну увагу під час тестування було приділено таким процесам: реєстрації та авторизації користувача, перегляду каталогу концертів, відкриттю сторінки події, оформленню замовлення, зменшенню кількості доступних квитків, перегляду сторінки «Мої квитки», формуванню «Паспорта внеску», роботі кабінету організатора та адміністративної панелі [3; 15; 23].

Ручне тестування проводилося шляхом послідовного виконання типових сценаріїв роботи користувачів. Для цього використовувалися демо-акаунти з різними ролями: USER, ORGANIZER та ADMIN. Такий підхід дозволив перевірити, чи правильно система розмежовує права доступу та чи не дозволяє користувачам виконувати дії, які не відповідають їхній ролі [8; 10; 27].

Першим було перевірено сценарій реєстрації нового користувача. На сторінці реєстрації вводилися повне ім’я, email і пароль. Після відправлення форми система створювала нового користувача, зберігала його в базі даних і дозволяла виконати вхід у систему. Додатково було перевірено обробку ситуації, коли користувач намагається зареєструвати email, який уже існує в системі. У такому випадку система повинна повернути повідомлення про помилку та не створювати дубльований акаунт [3; 15; 23].

Другим тестовим сценарієм була авторизація користувача. Під час входу перевірялося, чи коректно система обробляє правильні та неправильні облікові

дані. У разі правильного email і пароля backend формував JWT-токен, а frontend зберігав його для подальшого доступу до захищених сторінок. Якщо пароль був неправильним, користувач отримував повідомлення про помилку, а доступ до системи не надавався [25].

Також було протестовано перегляд публічної частини системи. Гість, тобто неавторизований користувач, може відкривати головну сторінку, переглядати каталог благодійних концертів, використовувати пошук і фільтрацію, а також відкривати сторінку конкретного концерту. При цьому гість не має доступу до сторінок «Мої квитки», профілю, кабінету організатора та адміністративної панелі [8; 10; 27].

Окремо перевірялося оформлення замовлення на квитки. Авторизований користувач обирав концерт, відкривав сторінку події, вибирав тип квитка, зазначав кількість і підтверджував замовлення. Після цього система перевіряла доступність квитків, створювала замовлення, додавала позиції замовлення та зменшувала доступну кількість квитків. Це є одним із найважливіших тестів, оскільки саме він перевіряє коректність основної бізнес-логіки системи [8; 10; 27].

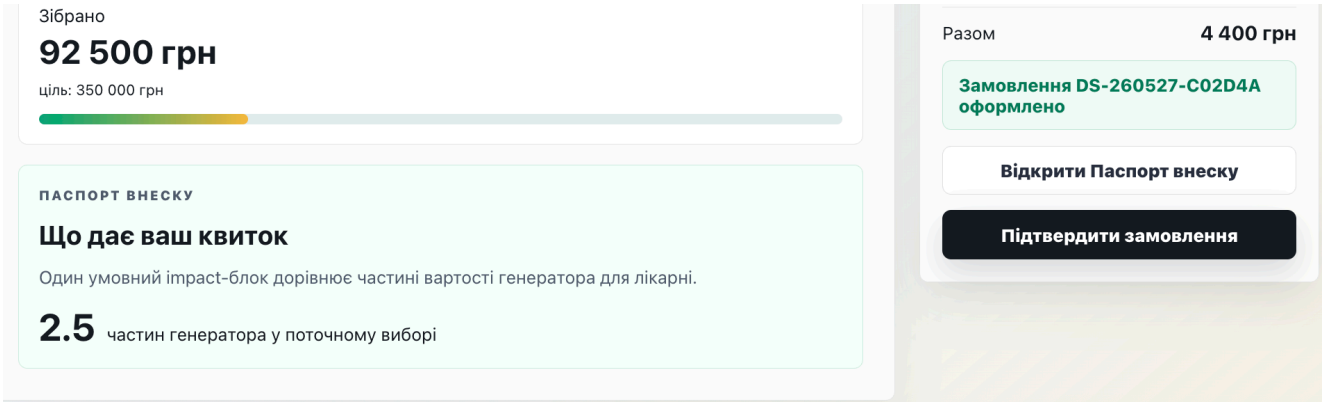


Рис. 4.1 Результат тестування створення замовлення на квитки

Було перевірено і негативний сценарій: спробу придбати більше квитків, ніж доступно в системі. У цьому випадку замовлення не повинно створюватися, а користувач має отримати повідомлення про недостатню кількість доступних квитків. Також перевірялася спроба купівлі квитків на концерт зі статусом

CANCELLED або SOLD_OUT. У таких випадках система повинна заборонити оформлення замовлення [2; 3; 4].

Після створення замовлення перевірялася сторінка «Мої квитки». На ній повинні відображатися всі замовлення авторизованого користувача: код замовлення, дата створення, сума, назва концерту, майданчик, кількість квитків і статус замовлення. Також має бути доступне посилання на сторінку «Паспорт внеску» [2; 3; 4].

Важливою частиною ручного тестування була перевірка функції «Паспорт внеску». Для цього відкривалося конкретне замовлення, після чого система повинна була показати суму замовлення, список придбаних квитків, назву impact-одиниці, розраховану кількість impact-одиниць і текстовий опис благодійного ефекту. Наприклад, якщо користувач придбав квитки на суму 1300 грн, а одна impact-одиниця дорівнює 650 грн, система повинна показати результат 2.0 impact-одиниці [8; 10; 27].

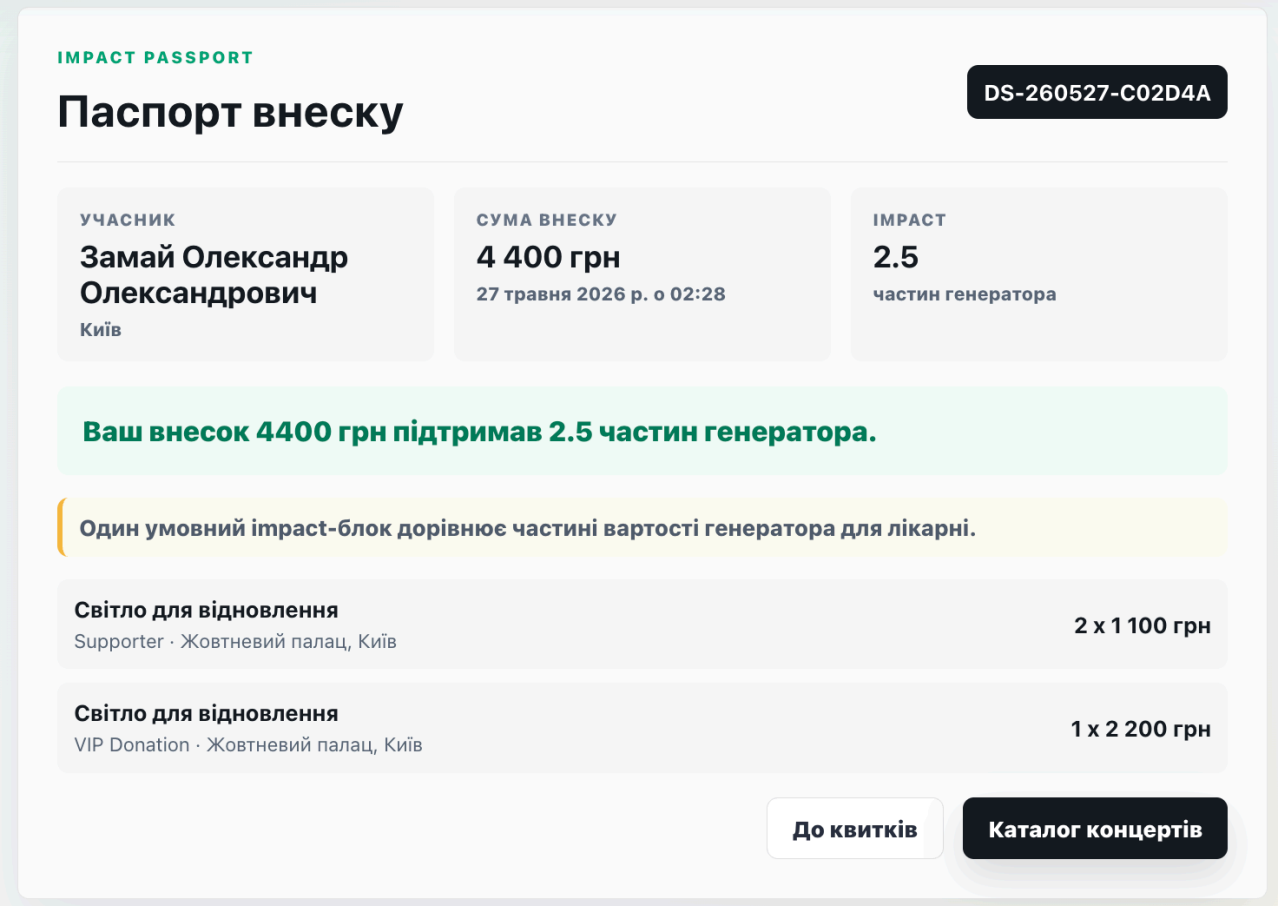


Рис. 4.2 Результат тестування формування “Паспорта внеску”

Також перевірялася робота кабінету організатора. Користувач із роллю ORGANIZER повинен мати доступ до сторінки організатора, переглядати статистику своїх концертів, створювати нові події, редагувати тільки власні концерти, додавати типи квитків і переглядати замовлення, пов’язані з його подіями. Водночас організатор не повинен мати можливість змінювати концерти інших організаторів або керувати ролями користувачів [2; 3; 4].

Окремо тестувалася адміністративна панель. Користувач із роллю ADMIN повинен мати доступ до всіх адміністративних функцій: керування концертами, майданчиками, користувачами, ролями та замовленнями. Було перевірено зміну ролі користувача, створення та редагування майданчика, перегляд усіх замовлень і редагування будь-якого концерту незалежно від організатора [8; 10; 27].

Результати ручного тестування основних функцій наведено в таблиці 4.1 [14; 24].

Таблиця 4.1

Результати ручного тестування web-застосунку DobroStage

№	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
1	Реєстрація нового користувача	Новий акаунт створюється, користувач може увійти в систему	Користувача створено, вхід виконано успішно	Пройдено
2	Реєстрація з уже існуючим email	Система повертає повідомлення про помилку	Повідомлення про помилку відображено	Пройдено
3	Вхід із правильними даними	Користувач отримує доступ до системи	JWT-токен створено, користувача авторизовано	Пройдено
4	Вхід із неправильним паролем	Доступ не надається, виводиться помилка	Помилка відображена, вхід заблоковано	Пройдено
5	Перегляд каталогу концертів гостем	Каталог доступний без авторизації	Список концертів відображено	Пройдено

Продовження таблиці 4.1

Результати ручного тестування web-застосунку DobroStage

№	Тестовий сценарій	Очікуваний результат	Фактичний результат	Статус
6	Оформлення замовлення користувачем	Замовлення створюється, залишок квитків зменшується	Замовлення створено, кількість квитків оновлено	Пройдено
7	Купівля більшої кількості квитків, ніж доступно	Система забороняє створення замовлення	Замовлення не створено, помилка відображена	Пройдено
8	Перегляд сторінки «Мої квитки»	Відображаються замовлення поточного користувача	Дані замовлень відображено коректно	Пройдено
9	Формування «Паспорта внеску»	Система розраховує impact-одиниці	Impact-результат сформовано коректно	Пройдено
10	Доступ USER до адмін-панелі	Доступ має бути заборонено	Користувача перенаправлено / доступ заблоковано	Пройдено
11	Доступ ORGANIZER до власних концертів	Організатор бачить і редагує тільки свої події	Доступ обмежено власними подіями	Пройдено
12	Доступ ADMIN до керування системою	Адміністратор має повний доступ	Адмін-функції доступні	Пройдено

4.2 Модульне та інтеграційне тестування backend-частини

Окрім ручної перевірки, для backend-частини DobroStage було передбачено модульне та інтеграційне тестування. Модульні тести дозволяють перевірити окремі сервіси системи ізольовано, а інтеграційні тести — перевірити роботу API, взаємодію контролерів, сервісів, бази даних і механізмів безпеки [3; 15; 23].

Базовим тестом є DobroStageApplicationTests.java. Він перевіряє, що Spring Boot-застосунок успішно запускає контекст. Це smoke-тест, який дозволяє виявити проблеми з конфігурацією, залежностями, JPA-сутностями, репозиторіями, сервісами або security-налаштуваннями ще до запуску повної системи [23; 24].

Для перевірки авторизації використовується AuthServiceTest.java. Цей набір тестів перевіряє реєстрацію нового користувача, вхід існуючого користувача, генерацію JWT-токена, повернення ролі користувача та імені користувача. Також перевіряється помилка під час спроби зареєструвати email, який уже існує в системі [3; 15; 23].

Для перевірки логіки замовлень використовується OrderServiceTest.java. У цьому тесті перевіряється створення замовлення, зменшення кількості доступних квитків, правильність розрахунку суми замовлення, заборона покупки квитка на скасований концерт і формування «Паспорта внеску». Саме цей тест є особливо важливим, тому що модуль замовлень відповідає за основний бізнес-процес системи [8; 10; 27].

OrganizerServiceTest.java перевіряє логіку роботи кабінету організатора. У тестах перевіряється, що статистика рахується тільки по концертах поточного організатора, а організатор не може змінювати типи квитків концерту, який належить іншому організатору. Це дозволяє підтвердити правильність рольового обмеження доступу [8; 10; 27].

Інтеграційне тестування API виконується за допомогою DobroStageApiIntegrationTest.java. Такі тести перевіряють роботу системи ближче до реального сценарію використання. Вони охоплюють публічний список концертів, заборону створення замовлення гостем, створення замовлення авторизованим користувачем, відкриття «Паспорта внеску», заборону покупки скасованого концерту, доступ організатора тільки до своїх концертів і замовлень, заборону звичайному користувачу заходити в organizer/admin API, а також адміністративну зміну ролі користувача та перегляд усіх замовлень [23; 25].

Для підготовки тестових даних використовується допоміжний клас TestDataFactory.java. Він створює тестових користувачів, майданчики, концерти, типи квитків і замовлення. Завдяки цьому тести залишаються зрозумілими, а однакові фрагменти створення даних не дублюються в різних тестових класах [8; 10; 27].

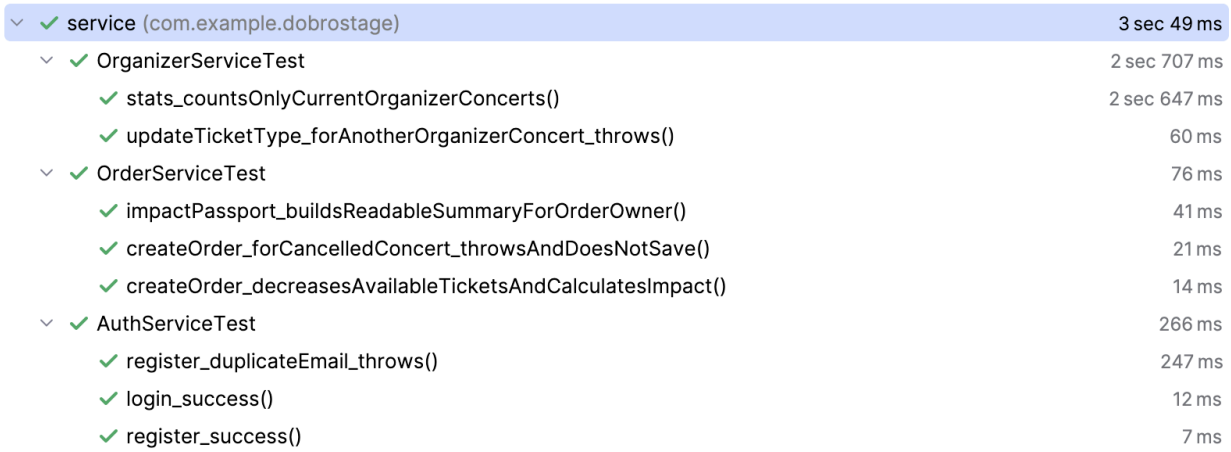


Рис. 4.3 Результат запуску backend-тестів

За результатами автоматизованого тестування backend-частини перевірено 8 тестів. Усі тести виконано успішно, що підтверджує коректність базової серверної логіки, авторизації, роботи з ролями, створення замовлень, обмеження доступу та формування «Паспорта внеску» [3; 15; 23].

4.3 Тестування frontend-частини

Тестування frontend-частини DobroStage виконувалося для перевірки роботи API-клієнта, захищених маршрутів, обробки помилок і доступу до сторінок залежно від стану авторизації та ролі користувача. Оскільки клієнтська частина є основним інтерфейсом взаємодії користувача із системою, її коректна робота є важливою для загальної якості web-застосунку [25].

Для тестування API-клієнта використовується файл client.test.js. У ньому перевіряється, що JWT-токен додається до захищених запитів і не додається до публічних запитів, таких як /auth/login та /auth/register. Також перевіряється, що у випадку відповіді з кодом 401 викликається подія розлогінення користувача. Це потрібно для того, щоб frontend міг коректно реагувати на ситуації, коли токен недійсний або термін його дії завершився [25].

Окремо перевіряється функція getErrorMessage, яка відповідає за витягування зрозумілого тексту помилки з відповіді backend-частини. Вона повинна коректно працювати з різними форматами відповіді, наприклад detail, title

або message. Це важливо для зручності користувача, оскільки система повинна показувати не технічний об’єкт помилки, а зрозуміле повідомлення [16; 17].

Для тестування захищених маршрутів використовується файл PrivateRoute.test.jsx. У ньому перевіряється поведінка системи під час синхронізації авторизації, редирект гостя на сторінку входу, доступ авторизованого користувача до дозволеної сторінки, редирект користувача без потрібної ролі та доступ користувача з відповідною роллю. Наприклад, користувач із роллю ADMIN повинен мати доступ до адміністративної панелі, а користувач із роллю USER — ні [3; 15; 23].

Також використовується файл setup.js, який підключає @testing-library/jest-dom. Це дозволяє застосовувати зручні перевірки React-компонентів, наприклад перевіряти, чи елемент присутній у документі [25].

```
✓ src/components/PrivateRoute.test.jsx (5 tests) 49ms

Test Files  2 passed (2)
     Tests  9 passed (9)
  Start at  02:36:38
  Duration  1.58s (transform 142ms, setup 256ms, import 314ms, tests 67ms, environment 2.08s)
```

Рис. 4.4 Результат запуску frontend-тестів

За результатами frontend-тестування перевірено 9 тестів. Усі тести виконано успішно, що підтверджує коректну роботу API-клієнта, обробку токена, реакцію на 401-помилки, захист маршрутів і перевірку доступу за ролями [25].

4.4 Узагальнення результатів тестування

Результати тестування підтверджують, що web-застосунок DobroStage відповідає основним функціональним і нефункціональним вимогам. Система дозволяє користувачам переглядати каталог благодійних концертів, відкривати сторінки подій, оформлювати замовлення, переглядати власні квитки та відкривати «Паспорт внеску». Організатор може керувати власними концертами, типами квитків і переглядати статистику своїх подій. Адміністратор має повний доступ до

керування користувачами, ролями, майданчиками, концертами та замовленнями [8; 10; 27].

Особливу увагу було приділено перевірці логіки замовлень. Тестування показало, що система коректно перевіряє доступність квитків, не дозволяє купувати квитки на скасований концерт, зменшує доступний залишок після створення замовлення та формує «Паспорт внеску» на основі impact-полів концерту [8; 10; 27].

Також було перевірено рольову модель доступу. Гість не може створити замовлення або перейти до захищених сторінок. Користувач не має доступу до кабінету організатора та адміністративної панелі. Організатор може працювати лише зі своїми концертами. Адміністратор має повний доступ до керування системою [2; 3; 4].

Таблиця 4.2

Результати автоматизованого тестування backend та frontend

№	Частина системи	Назва тесту	Що перевірялося	Результат
1	Backend	DobroStageApplicationTests	Запуск Spring Boot-контексту	Успішно
2	Backend	AuthServiceTest	Реєстрація, вхід, JWT, помилка дублювання email	Успішно
3	Backend	OrderServiceTest	Створення замовлення, залишок квитків, сума, «Паспорт внеску»	Успішно
4	Backend	OrganizerServiceTest	Статистика організатора, заборона доступу до чужих концертів	Успішно

Продовження таблиці 4.2

Результати автоматизованого тестування backend та frontend

№	Частина системи	Назва тесту	Що перевірялося	Результат
5	Backend	DobroStageApiIntegrationTest	REST API, ролі, замовлення, admin/organizer-доступ	Успішно
6	Frontend	client.test.js	JWT у запитах, 401, повідомлення про помилки	Успішно
7	Frontend	PrivateRoute.test.jsx	Захищені маршрути та перевірка ролей	Успішно
8	Frontend	setup.js	Підключення тестового середовища React	Успішно

Загалом backend-тестування охопило 16 тестів, а frontend-тестування — 9 тестів. Обидва набори тестів виконуються успішно.

Таким чином, проведене тестування підтверджує працездатність основних модулів DobroStage. Система коректно виконує ключові сценарії роботи, забезпечує контроль доступу за ролями, підтримує створення замовлень, оновлення доступних квитків і формування благодійного результату через «Паспорт внеску». Отримані результати дозволяють зробити висновок, що розроблений web-застосунок може бути використаний як основа для подальшого розвитку системи онлайн-продажу квитків на благодійні концерти [8; 10; 27].

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було розроблено web-застосунок DobroStage для онлайн-продажу квитків на благодійні концерти. Розроблена система дозволяє централізувати інформацію про благодійні події, забезпечити перегляд концертів, вибір квитків, оформлення замовлень, облік доступних місць, керування ролями користувачів та відображення благодійного результату через функцію «Паспорт внеску» [2; 3; 4].

У першому розділі було проведено аналіз предметної галузі онлайн-продажу квитків на благодійні концерти. Визначено, що основними проблемами цієї сфери є розрізнене зберігання інформації про події, ручне оформлення замовлень, складність контролю кількості доступних квитків, ризик дублювання місць, недостатня прозорість благодійного збору та відсутність зручного інструменту для відображення благодійного ефекту внеску користувача. Також було розглянуто основні способи організації продажу квитків, серед яких ручний облік через соціальні мережі та месенджери, використання електронних таблиць і готових онлайн-сервісів [14; 24].

Було проаналізовано існуючі програмні аналоги, зокрема Concert.ua, KARABAS, Eventbrite та TicketForEvent. У результаті порівняння встановлено, що ці сервіси мають корисний функціонал для організації подій і продажу квитків, однак здебільшого є універсальними платформами та не повністю враховують специфіку благодійних концертів. Зокрема, у них відсутня або обмежено реалізована можливість персоналізованого відображення благодійного результату конкретного замовлення. Це обґрунтовує доцільність розробки окремого web-застосунку DobroStage [19].

У другому розділі було виконано проєктування web-застосунку. Сформовано функціональні та нефункціональні вимоги до системи, визначено основні ролі користувачів: гість, користувач, організатор та адміністратор. Для кожної ролі визначено доступні сценарії роботи. Гість може переглядати каталог концертів, користувач — купувати квитки та переглядати власні замовлення, організатор —

керувати власними концертами та типами квитків, адміністратор — керувати всією платформою. Також було обґрунтовано вибір технологій реалізації: Java, Spring Boot, Spring Security, JWT, Spring Data JPA, Hibernate, PostgreSQL, React, Vite, REST API та Git [23; 24].

У процесі проєктування було розроблено загальну архітектуру системи, структуру бази даних, діаграму варіантів використання, схему ролей користувачів і прав доступу, а також схему формування «Паспорта внеску». База даних містить основні сутності системи: користувачів, майданчики, концерти, типи квитків, замовлення та позиції замовлень. Така структура дозволяє зберігати всі необхідні дані для роботи платформи та забезпечує логічні зв’язки між концертами, користувачами, квитками й замовленнями [14; 24].

У третьому розділі було описано програмну реалізацію web-застосунку DobroStage. Серверна частина реалізована на основі Spring Boot і містить entity-класи, репозиторії, сервіси, контролери та модуль безпеки. Було реалізовано роботу з користувачами, концертами, майданчиками, типами квитків, замовленнями, ролями доступу та impact-показниками. Для захисту системи використано JWT-авторизацію, що дозволяє розмежовувати доступ до сторінок і API залежно від ролі користувача [23; 24].

Клієнтська частина реалізована з використанням React. Було створено основні сторінки web-застосунку: головну сторінку з каталогом благодійних концертів, сторінку детального перегляду концерту, сторінки входу та реєстрації, сторінку «Мої квитки», профіль користувача, сторінку «Паспорт внеску», кабінет організатора та адміністративну панель. Інтерфейс забезпечує зручну взаємодію користувача з системою та підтримує основні сценарії роботи платформи [25].

Особливістю розробленого web-застосунку є функція «Паспорт внеску». Вона дозволяє користувачу побачити не лише суму замовлення, а й конкретний благодійний результат його покупки. Для цього система використовує impact-поля концерту: назву impact-одиниці, вартість однієї impact-одиниці та опис благодійного ефекту. На основі цих даних виконується розрахунок, після чого

користувач отримує зрозуміле пояснення свого внеску, наприклад кількість підтриманих теплих наборів або інших одиниць допомоги [2; 3; 4].

У четвертому розділі було проведено тестування web-застосунку. Під час ручного тестування перевірено реєстрацію та авторизацію користувача, перегляд каталогу концертів, відкриття сторінки події, оформлення замовлення, оновлення залишку доступних квитків, перегляд сторінки «Мої квитки», формування «Паспорта внеску», роботу кабінету організатора та адміністративної панелі. Також перевірено негативні сценарії, зокрема спробу купівлі більшої кількості квитків, ніж доступно, спробу оформлення замовлення гостем і спробу доступу користувача до заборонених сторінок [3; 15; 23].

Було виконано модульне та інтеграційне тестування backend-частини. Перевірено запуск Spring Boot-контексту, роботу сервісу авторизації, створення замовлень, зменшення залишку квитків, формування «Паспорта внеску», обмеження доступу організатора до чужих концертів і роботу REST API. Також проведено тестування frontend-частини, зокрема API-клієнта, обробки JWT-токена, реакції на помилки авторизації та захищених маршрутів [23; 24].

Робота пройшла апробацію. За її результатами було опубліковано наступні тези доповідей:

1. Замай О.О., Трінтіна Н.А. Розробка системи для онлайн-оплати послуг. Матеріали ІІ Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.457-459.

2. Замай О.О., Трейтяк В.В. Особливості розробки веб-застосунку для онлайн-продажу квитків на благодійні концерти. Матеріали Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація», 1–3 травня 2026 року, Київ, ДУІКТ. Збірник тез. *(подано до друку).*

ПЕРЕЛІК ПОСИЛАНЬ

1. Про благодійну діяльність та благодійні організації : Закон України від 05.07.2012 р. № 5073-VI. URL: https://zakon.rada.gov.ua/laws/show/5073-17

2. Про електронну комерцію : Закон України від 03.09.2015 р. № 675-VIII. URL: https://zakon.rada.gov.ua/laws/show/675-19

3. Про захист персональних даних : Закон України від 01.06.2010 р. № 2297-VI. URL: https://zakon.rada.gov.ua/laws/show/2297-17

4. Про електронні документи та електронний документообіг : Закон України від 22.05.2003 р. № 851-IV. URL: https://zakon.rada.gov.ua/laws/show/851-15

5. Про електронні довірчі послуги : Закон України від 05.10.2017 р. № 2155-VIII. URL: https://zakon.rada.gov.ua/laws/show/2155-19

6. ДСТУ ISO/IEC/IEEE 29148:2018. Інженерія систем і програмного забезпечення. Процеси життєвого циклу. Інженерія вимог. Київ : ДП «УкрНДНЦ», 2018. 104 с.

7. ДСТУ ISO/IEC 25010:2016. Інженерія систем і програмного забезпечення. Вимоги до якості систем і програмного забезпечення та оцінювання. Моделі якості системи та програмного продукту. Київ : ДП «УкрНДНЦ», 2016. 56 с.

8. Bourque P., Fairley R. E. Guide to the Software Engineering Body of Knowledge (SWEBOK Guide). Version 4.0. IEEE Computer Society, 2024. 423 p.

9. Sommerville I. Engineering Software Products: An Introduction to Modern Software Engineering. 2nd ed. London : Pearson, 2021. 352 p.

10. Pressman R. S., Maxim B. R. Software Engineering: A Practitioner’s Approach. 9th ed. New York : McGraw-Hill Education, 2020. 672 p.

11. Bass L., Clements P., Kazman R. Software Architecture in Practice. 4th ed. Boston : Addison-Wesley Professional, 2021. 464 p.

12. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O’Reilly Media, 2021. 616 p.

13. Richardson C. Microservices Patterns: With examples in Java. Shelter Island : Manning Publications, 2018. 520 p.

14. Kleppmann M. Designing Data-Intensive Applications. Sebastopol : O'Reilly Media, 2017. 616 p.

15. OWASP Foundation. OWASP Top 10: The Ten Most Critical Web Application Security Risks. 2021. URL: https://owasp.org/www-project-top-ten/

16. Nielsen J. 10 Usability Heuristics for User Interface Design. Nielsen Norman Group. 2024. URL: https://www.nngroup.com/articles/ten-usability-heuristics/

17. Muzyka B. User Flows. Як ця техніка допомагає в роботі над проєктами. DOU. 29.12.2020. URL: https://dou.ua/lenta/columns/user-flows/

18. Дія.Бізнес. Цифровізація бізнесу: інструменти та можливості для підприємців. URL: https://business.diia.gov.ua/

19. Concert.ua — афіша подій та онлайн-продаж квитків в Україні. URL: https://concert.ua/uk

20. KARABAS — сервіс продажу квитків на концерти та події. URL: https://karabas.com

21. Eventbrite. Sell Tickets Online With Eventbrite. URL: https://www.eventbrite.com/organizer/features/sell-tickets/

22. TicketForEvent. Event registration and e-ticketing software. URL: https://ticketforevent.com/en/

23. Spring Boot Reference Documentation. VMware, Inc. URL: https://docs.spring.io/spring-boot/reference/

24. PostgreSQL Documentation. The PostgreSQL Global Development Group. URL: https://www.postgresql.org/docs/

25. React Documentation. Meta Open Source. URL: https://react.dev/

26. Vite Documentation. URL: https://vite.dev/guide/

27. Postman Learning Center. Writing tests. URL: https://learning.postman.com/docs/tests-and-scripts/write-scripts/test-scripts/

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ



ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ



Розробка програмного забезпечення для онлайн-продажу квитків на благодійні концерти

Виконав студент 4 курсу
групи ПД-41
Замай Олександр Олександрович
Керівник роботи
канд.техн.наук, зав.каф. Трейтяк В'ячеслав Віталійович

Київ – 2026

ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Проаналізувати предметну галузь онлайн-продажу квитків на благодійні концерти.
2. Дослідити існуючі програмні засоби для продажу квитків та керування подіями.
3. Визначити функціональні та нефункціональні вимоги до web-застосунку DobroStage.
4. Обґрунтувати вибір технологій реалізації.
5. Спроєктувати архітектуру системи, базу даних та ролі користувачів.
6. Реалізувати каталог концертів, сторінку події, купівлю квитків і профіль користувача.
7. Реалізувати кабінет організатора та адміністративну панель.
8. Провести тестування функціональності, рольового доступу та процесу оформлення замовлення.

МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи: спрощення та автоматизація процесу продажу квитків на благодійні концерти за рахунок розробки веб-застосунку, який забезпечує централізоване керування подіями, квитками, замовленнями та обліком доступних місць.

Об'єкт дослідження: процес онлайн-продажу квитків на благодійні концерти.

Предмет дослідження: засоби, методи та технології реалізації програмного забезпечення для онлайн-продажу квитків на благодійні концерти.

АНАЛІЗ АНАЛОГІВ

Показник	Concert.ua	KARABAS	Eventbrite	DobroStage
Підтримка благодійних подій	+	Окремої категорії немає	+	+
Онлайн-купівля квитків	+	+	+	+
Список покупців / учасників	Покупці квитків	Покупці квитків	Учасники події	Покупці + замовлення
Аналітика продажів	Кількість продажів	Кількість продажів	Продажі + реєстрації	Продажі + сума збору
Кабінет організатора	Для партнерів сервісу	Для партнерів сервісу	+	+
Керування користувачами та ролями	-	-	Команда організатора	USER / ORGANIZER / ADMIN
Управління майданчиками	Через заявку сервісу	Через заявку сервісу	Назва й адреса події	+
Налаштування благодійної цілі	Текст в описі	Текст в описі	Текст в описі	Cause + goal_amount
Відображення прогресу збору	-	-	-	+
Impact-етапи благодійної цілі	-	-	-	+
Паспорт внеску після купівлі квитка	-	-	-	+
Пояснення благодійного результату внеску	-	-	-	+
Зручність для невеликих благодійних концертів	Потрібна співпраця з сервісом	Потрібна співпраця з сервісом	Багато зайвих налаштувань	Створено під такі події
Основні переваги	Афіша, український ринок	Пошук подій, український ринок	Створення подій, реєстрації	Ролі, impact, Паспорт внеску

ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Функціональні вимоги
– Реєстрація та авторизація користувачів.
– Рольовий доступ
– Перегляд каталогу благодійних концертів.
– Пошук, фільтрація та сортування подій.
– Купівля квитків і формування замовлення.
– Перегляд профілю користувача та історії замовлень.
– Кабінет організатора для керування власними концертами.
– Адміністративна панель для керування користувачами, подіями, майданчиками та замовленнями.
– Формування Паспорта внеску після купівлі квитка.

Нефункціональні вимоги
– Час завантаження основних сторінок: до 2–3 с.
– Обробка основних запитів API: до 1–2 с.
– Підтримка одночасної роботи: не менше 20 користувачів.
– Обсяг бази даних: не менше 5 000 замовлень і 1 000 подій.
– Основні дії користувача: до 3–5 кроків.
– Підтримка браузерів: Chrome, Firefox, Microsoft Edge.
– Захищене зберігання паролів.

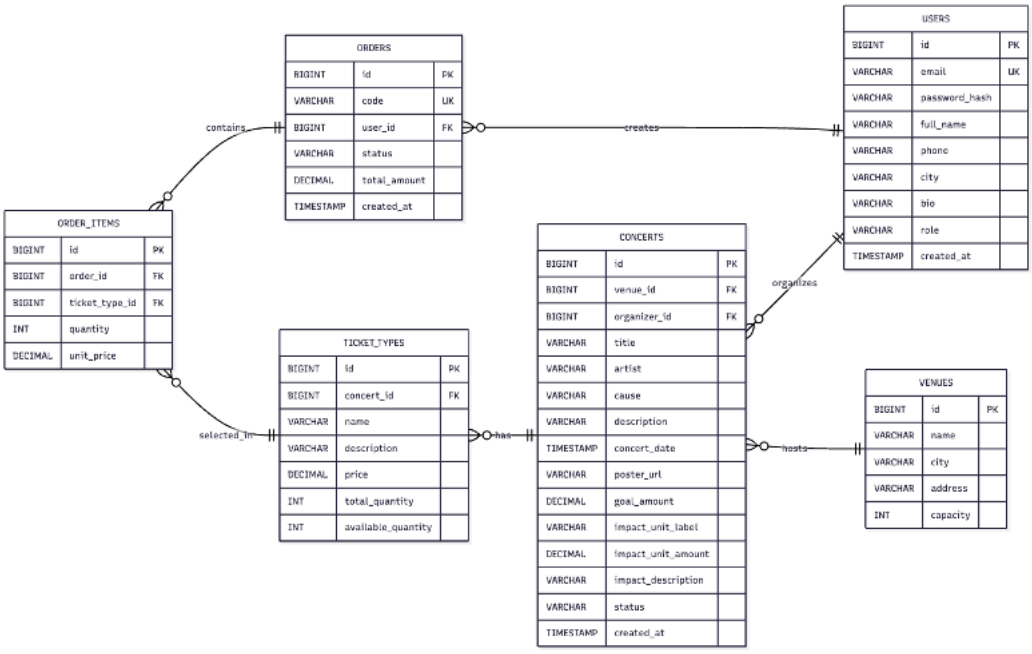
5

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ



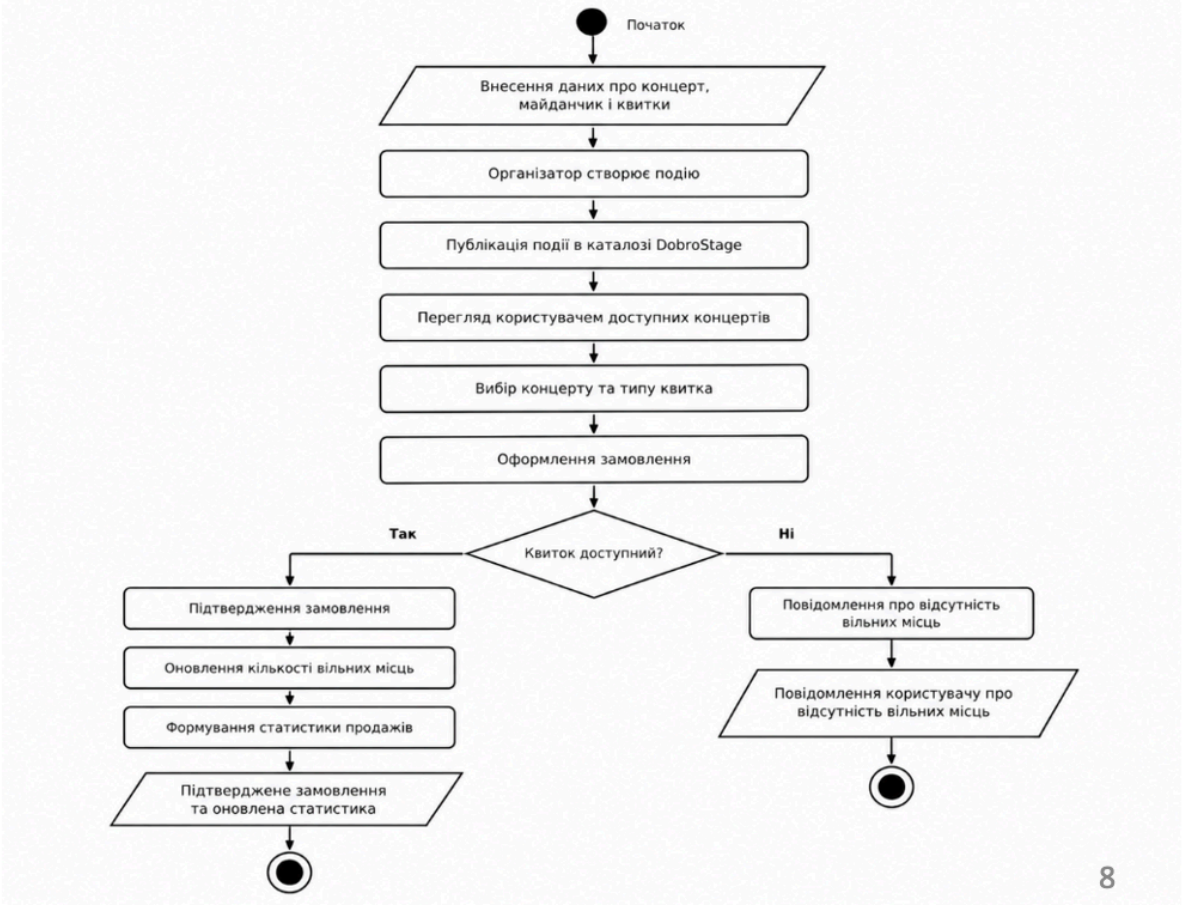
6

СХЕМА БАЗИ ДАНИХ



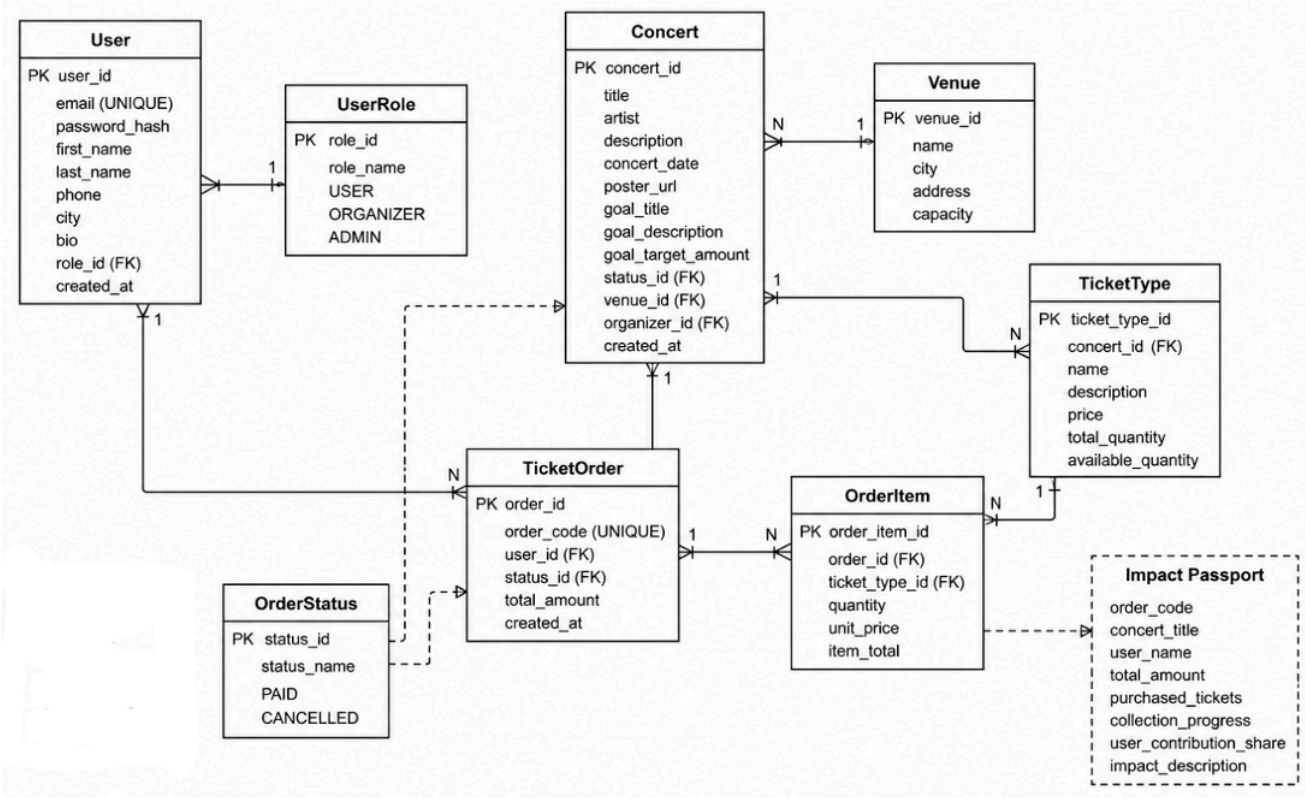
7

ЗАГАЛЬНА СХЕМА БІЗНЕС-ПРОЦЕСУ



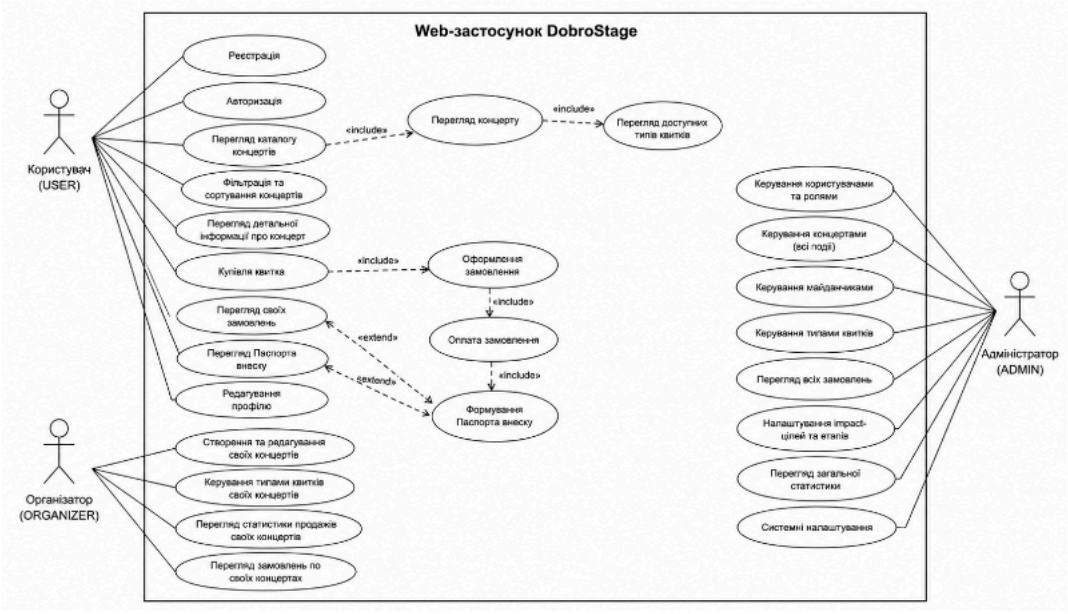
8

ОСНОВНІ СУТНОСТІ СИСТЕМИ



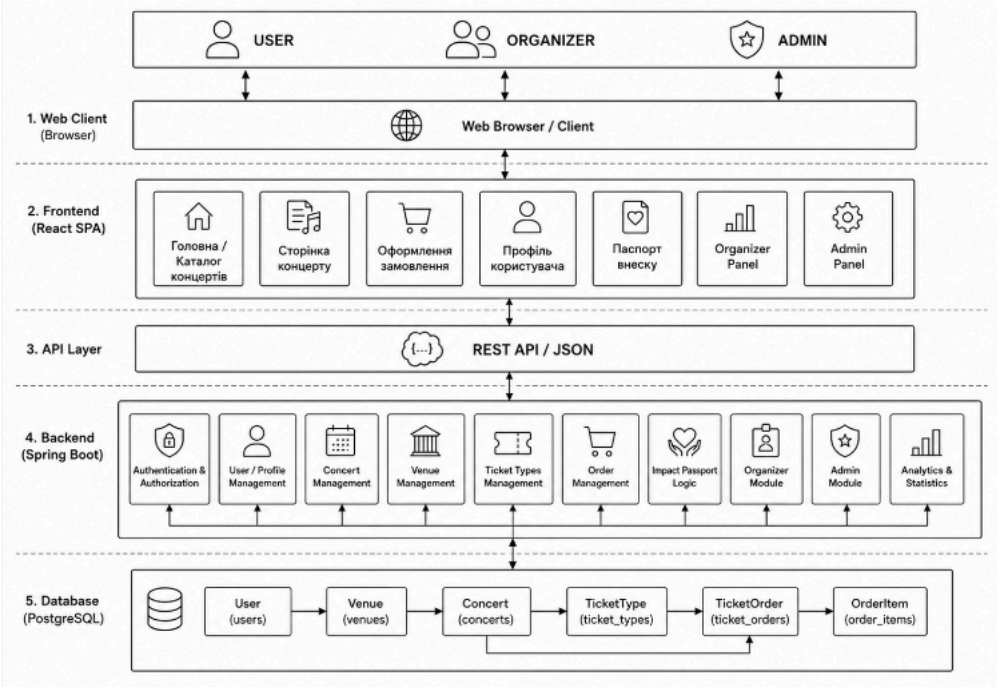
9

ДІАГРАМА ВЗАЄМОДІЇ КОРИСТУВАЧІВ ІЗ СИСТЕМОЮ



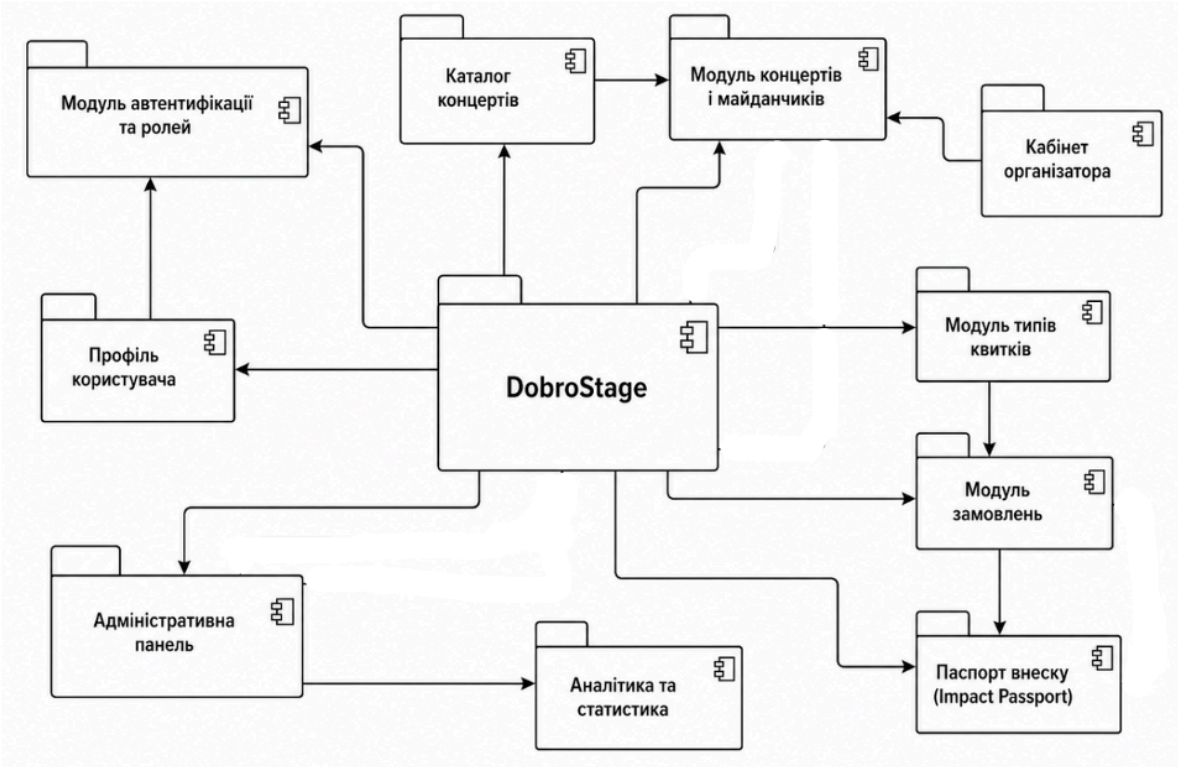
10

ЗАГАЛЬНА АРХІТЕКТУРА



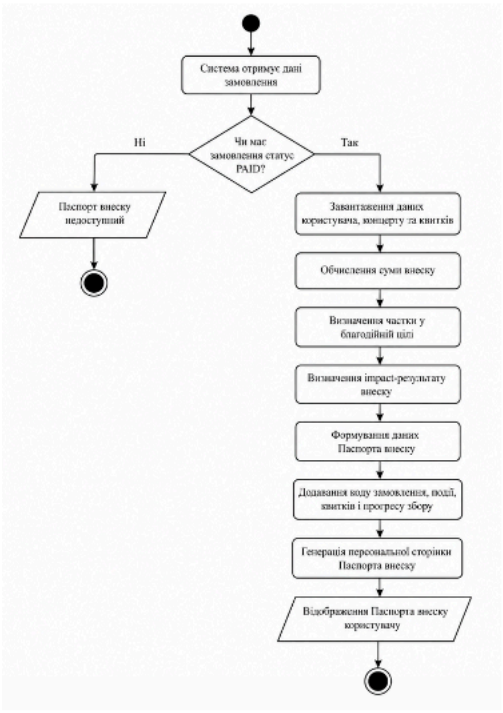
11

МОДУЛЬНА СТРУКТУРА СИСТЕМИ

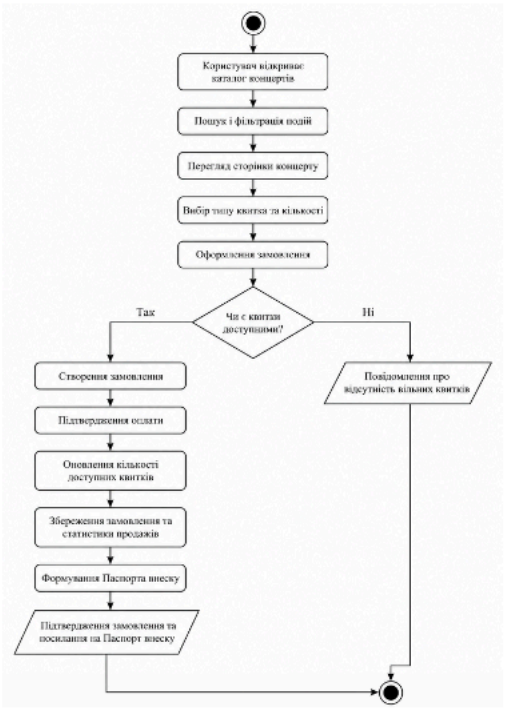


12

ДІАГРАМА ДІЯЛЬНОСТІ



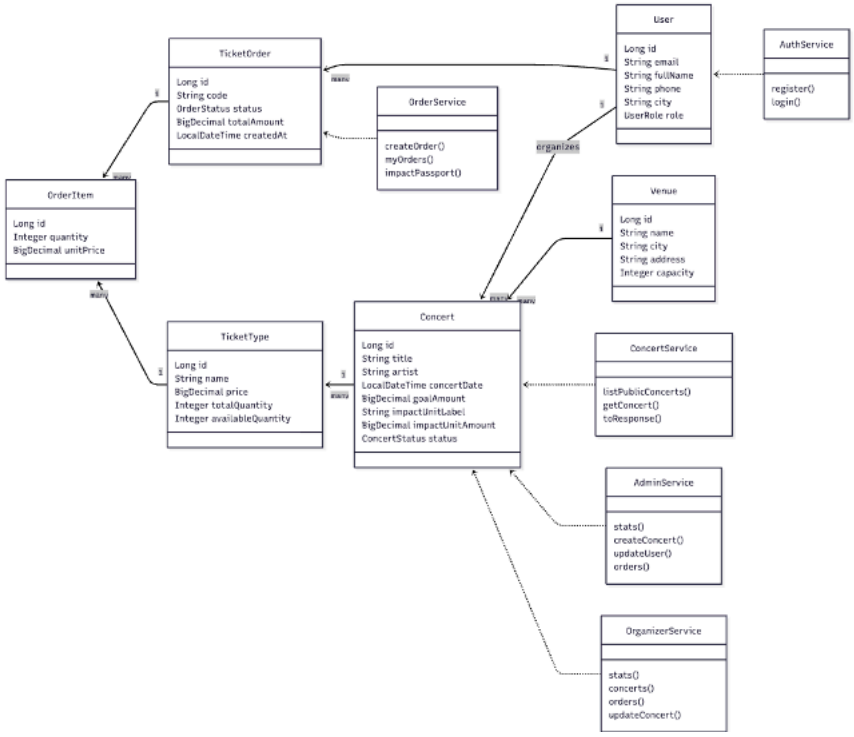
Процес купівлі квитка користувачем



Формування Паспорта внеску

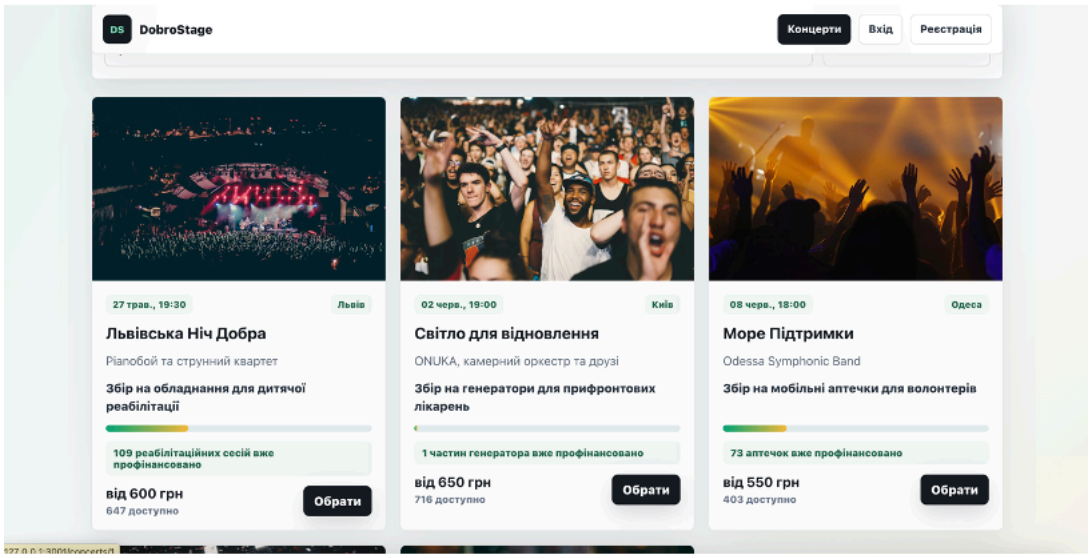
13

ДІАГРАМА КЛАСІВ ОСНОВНОЇ ЛОГІКИ



14

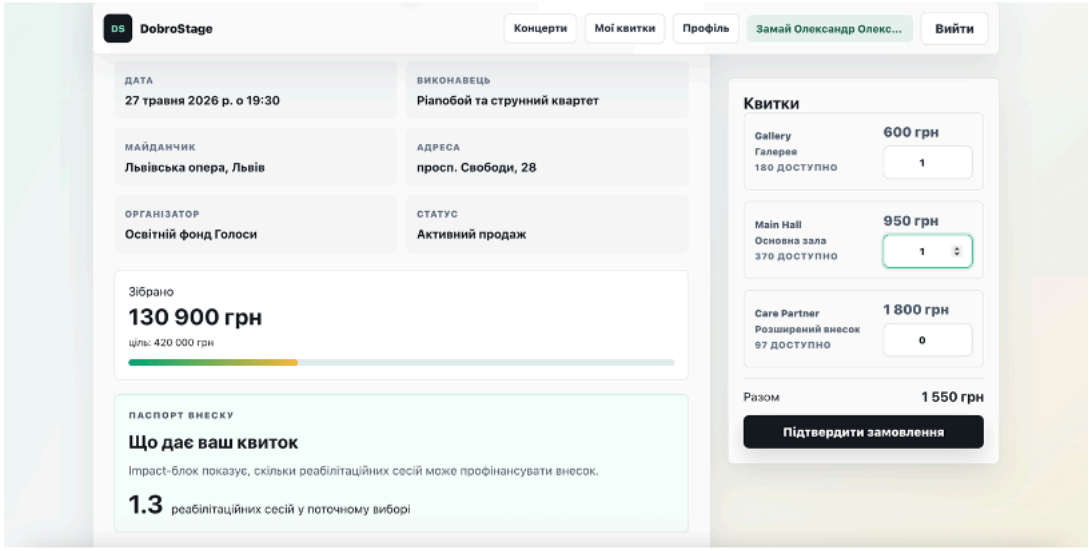
ЕКРАННІ ФОРМИ



Головна сторінка з подіями

15

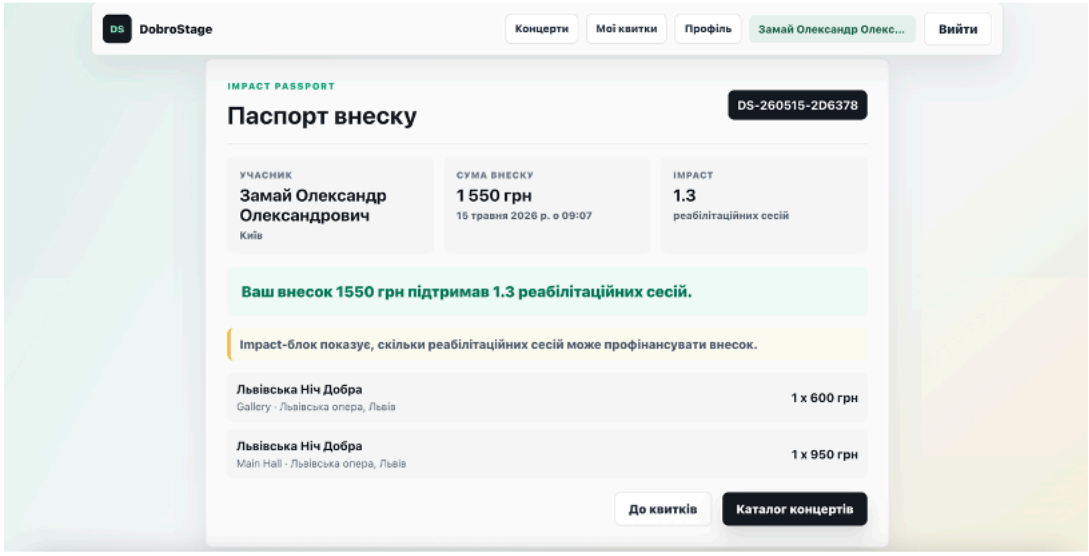
ЕКРАННІ ФОРМИ



Сторінка події

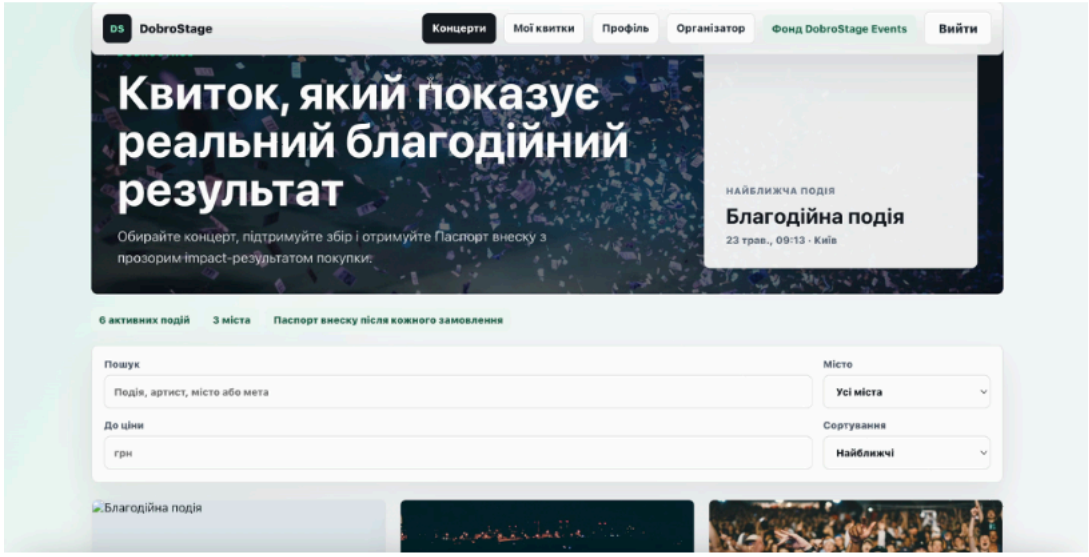
16

ЕКРАННІ ФОРМИ



Сторінка паспорту внеску

ДЕМОНСТРАЦІЯ РОБОТИ ПРОГРАМИ



АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Тези доповідей:

1. Замай О.О., Трінтіна Н.А. Розробка системи для онлайн-оплати послуг. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.457-459.

2. Замай О.О., Трейтяк В.В. Особливості розробки веб-застосунку для онлайн-продажу квитків на благодійні концерти. Матеріали Всеукраїнська науково-технічна конференція «Технології цифрового розвитку: програмні системи та децентралізація», 1–3 травня 2026 року, Київ, ДУІКТ. Збірник тез. *(подано до друку)*.

ВИСНОВКИ

1. Проаналізовано предметну галузь онлайн-продажу квитків на благодійні концерти.
2. Досліджено існуючі програмні засоби для продажу квитків та керування подіями.
3. Визначено функціональні та нефункціональні вимоги до web-застосунку DobroStage.
4. Обґрунтовано вибір технологій реалізації.
5. Спроєктовано архітектуру системи, базу даних, ролі користувачів та UML-діаграми.
6. Реалізовано каталог концертів, сторінку події, купівлю квитків і профіль користувача.
7. Реалізовано кабінет організатора та адміністративну панель.
8. Проведено тестування функціональності, рольового доступу та процесу оформлення замовлення.

ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

Listing 1 – User Entity Class

```
@Entity
@Table(name = "users")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class User {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true, length =
255)
    private String email;

    @Column(name = "password_hash", nullable = false)
    private String passwordHash;

    @Column(name = "full_name", length = 255)
    private String fullName;

    @Column(length = 40)
    private String phone;

    @Column(length = 120)
    private String city;

    @Column(length = 700)
    private String bio;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 20)
    @Builder.Default
    private UserRole role = UserRole.USER;

    @CreationTimestamp
    @Column(name = "created_at", updatable = false)
    private LocalDateTime createdAt;
}
```

Listing 2 – Concert Entity Class

```
@Entity
@Table(name = "concerts")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class Concert {

    @Id
     @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;


    @Column(nullable = false, length = 180)
    private String title;

    @Column(nullable = false, length = 140)
    private String artist;

    @Column(nullable = false, length = 180)
    private String cause;

    @Column(nullable = false, length = 1800)
    private String description;

    @Column(name = "concert_date", nullable = false)
    private LocalDateTime concertDate;

    @Column(name = "poster_url", length = 700)
    private String posterUrl;

    @Column(name = "goal_amount", nullable = false,
precision = 12, scale = 2)
    private BigDecimal goalAmount;

    @Column(name = "impact_unit_label", length = 180)
    private String impactUnitLabel;

    @Column(name = "impact_unit_amount", precision =
10, scale = 2)
    private BigDecimal impactUnitAmount;

    @Column(name = "impact_description", length =
700)
    private String impactDescription;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 20)
    @Builder.Default
    private ConcertStatus status =
ConcertStatus.ACTIVE;

    @ManyToOne(fetch = FetchType.LAZY, optional =
false)
    @JoinColumn(name = "venue_id", nullable = false)
    private Venue venue;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "organizer_id")
    private User organizer;

    @OneToMany(mappedBy = "concert", cascade =
CascadeType.ALL, orphanRemoval = true)
    @Builder.Default
    private List<TicketType> ticketTypes = new
ArrayList<>();
```

```
}
```

Listing 3 – Ticket Type Entity Class

```
@Entity
@Table(name = "ticket_types")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class TicketType {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY, optional =
false)
    @JoinColumn(name = "concert_id", nullable = false)
    private Concert concert;

    @Column(nullable = false, length = 90)
    private String name;

    @Column(length = 500)
    private String description;

    @Column(nullable = false, precision = 10, scale = 2)
    private BigDecimal price;

    @Column(name = "total_quantity", nullable = false)
    private Integer totalQuantity;

    @Column(name = "available_quantity", nullable =
false)
    private Integer availableQuantity;
}
```

Listing 4 – Ticket Order Entity Class

```
@Entity
@Table(name = "orders")
@Getter
@Setter
@NoArgsConstructor
@AllArgsConstructor
@Builder
public class TicketOrder {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    @Column(nullable = false, unique = true, length = 32)
    private String code;

    @ManyToOne(fetch = FetchType.LAZY, optional =
false)
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    @Enumerated(EnumType.STRING)
    @Column(nullable = false, length = 20)
    @Builder.Default
    private OrderStatus status = OrderStatus.PAID;

    @Column(name = "total_amount", nullable = false,
precision = 12, scale = 2)
    private BigDecimal totalAmount;

    @OneToMany(mappedBy = "order", cascade =
CascadeType.ALL, orphanRemoval = true)
    @Builder.Default
    private List<OrderItem> items = new ArrayList<>();
}
```

Listing 5 – Authentication Service Class

```
@Service
@RequiredArgsConstructor
public class AuthService {

    private final UserRepository userRepository;
    private final PasswordEncoder passwordEncoder;
    private final JwtTokenProvider tokenProvider;
    private final AuthenticationManager
authenticationManager;

    @Transactional
    public AuthResponse register(RegisterRequest
request) {
        if
(userRepository.existsByEmail(request.getEmail())) {
            throw new ConflictException("Email вже
використовується: " + request.getEmail());
        }

        User user = User.builder()
                .email(request.getEmail())

.passwordHash(passwordEncoder.encode(request.getPas
sword()))
                .fullName(request.getFullName())
                .build();

        userRepository.save(user);

        String token =
tokenProvider.generateToken(user.getEmail());
        return new AuthResponse(
                token,
                user.getEmail(),
                user.getFullName(),
                user.getPhone(),
                user.getCity(),
                user.getBio(),
                user.getRole().name()
        );
    }

    public AuthResponse login(LoginRequest request) {
        authenticationManager.authenticate(
                new
UsernamePasswordAuthenticationToken(request.getEma
il(), request.getPassword())
        );
```

```
        User user =
userRepository.findByEmail(request.getEmail()).orElseT
hrow();
        String token =
tokenProvider.generateToken(user.getEmail());

        return new AuthResponse(
                token,
                user.getEmail(),
                user.getFullName(),
                user.getPhone(),
                user.getCity(),
                user.getBio(),
                user.getRole().name()
        );
    }
}
```

Listing 6 – Order Service Core Logic Class

```
@Service
@RequiredArgsConstructor
public class OrderService {

    private final UserService userService;
    private final TicketTypeRepository
ticketTypeRepository;
    private final TicketOrderRepository orderRepository;

    @Transactional
    public OrderResponse
createOrder(CreateOrderRequest request) {
        User user = userService.currentUser();

        TicketOrder order = TicketOrder.builder()
                .user(user)
                .code(generateCode())
                .totalAmount(BigDecimal.ZERO)
                .build();

        BigDecimal total = BigDecimal.ZERO;

        for (OrderItemRequest itemRequest :
request.getItems()) {
            TicketType ticketType = ticketTypeRepository

.findByIdForUpdate(itemRequest.getTicketTypeId())
                    .orElseThrow(() -> new
ResourceNotFoundException("Тип квитка не
знайдено"));

            int quantity = itemRequest.getQuantity();

            if (ticketType.getConcert().getStatus() !=
ConcertStatus.ACTIVE) {
                throw new ConflictException("Квитки на цей
концерт зараз недоступні");
            }

            if (ticketType.getAvailableQuantity() < quantity)
{
                throw new ConflictException("Недостатньо
квитків у категорії " + ticketType.getName());
            }

ticketType.setAvailableQuantity(ticketType.getAvailable
Quantity() - quantity);

            BigDecimal lineTotal =
ticketType.getPrice().multiply(BigDecimal.valueOf(qua
ntity));
            total = total.add(lineTotal);

            OrderItem item = OrderItem.builder()
                    .order(order)
                    .ticketType(ticketType)
                    .quantity(quantity)
                    .unitPrice(ticketType.getPrice())
                    .build();

            order.getItems().add(item);
        }

        order.setTotalAmount(total);
        return toResponse(orderRepository.save(order));
    }

    private String generateCode() {
        String date =
LocalDate.now().format(DateTimeFormatter.ofPattern("
yyMMdd"));
        return "DS-" + date + "-" + UUID.randomUUID()
                .toString()
                .substring(0, 6)
                .toUpperCase(Locale.ROOT);
    }
}
```

Listing 7 – JWT Token Provider Class

```
@Component
public class JwtTokenProvider {

    @Value("${app.jwt.secret}")
    private String jwtSecret;

    @Value("${app.jwt.expiration-ms}")
    private long jwtExpirationMs;

    private SecretKey key() {
        return
Keys.hmacShaKeyFor(jwtSecret.getBytes(StandardChar
sets.UTF_8));
    }

    public String generateToken(String email) {
        Date now = new Date();

        return Jwts.builder()
                .subject(email)
                .issuedAt(now)
                .expiration(new Date(now.getTime() +
jwtExpirationMs))
                .signWith(key())
                .compact();
    }
```

```
    public String getEmailFromToken(String token) {
        return Jwts.parser()
                .verifyWith(key())
                .build()
                .parseSignedClaims(token)
                .getPayload()
                .getSubject();
    }

    public boolean validateToken(String token) {
        try {

Jwts.parser().verifyWith(key()).build().parseSignedClai
ms(token);
            return true;
        } catch (JwtException | IllegalArgumentException
e) {
            return false;
        }
    }
}
```

Listing 8 – Security Configuration Class

```
@Configuration
@EnableWebSecurity
@EnableMethodSecurity
@RequiredArgsConstructor
public class SecurityConfig {

    private final JwtAuthenticationFilter jwtAuthFilter;
    private final UserDetailsServiceImpl
userDetailsService;

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity
http) throws Exception {
        http
                .csrf(AbstractHttpConfigurer::disable)
                .cors(cors ->
cors.configurationSource(corsConfigurationSource()))
                .sessionManagement(session ->

session.sessionCreationPolicy(SessionCreationPolicy.ST
ATELESS)
                )
                .authorizeHttpRequests(auth -> auth
                        .requestMatchers(
                                "/api/auth/**",
                                "/api/concerts/**",
                                "/api-docs/**",
                                "/swagger-ui/**",
                                "/swagger-ui.html"
                        ).permitAll()
                        .anyRequest().authenticated()
                )

.authenticationProvider(authenticationProvider())
                .addFilterBefore(jwtAuthFilter,
UsernamePasswordAuthenticationFilter.class);

        return http.build();
    }

    @Bean
    public DaoAuthenticationProvider
authenticationProvider() {
        DaoAuthenticationProvider provider = new
DaoAuthenticationProvider(userDetailsService);
        provider.setPasswordEncoder(passwordEncoder());
        return provider;
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Listing 9 – Order Controller Class

```
@RestController
@RequestMapping("/api/orders")
@RequiredArgsConstructor
@Tag(name = "Замовлення")
public class OrderController {

    private final OrderService orderService;

    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public OrderResponse create(@Valid @RequestBody
CreateOrderRequest request) {
        return orderService.createOrder(request);
    }

    @GetMapping("/my")
    public List<OrderResponse> myOrders() {
        return orderService.myOrders();
    }

    @GetMapping("/{id}")
    public OrderResponse get(@PathVariable Long id) {
        return orderService.getOrder(id);
    }

    @GetMapping("/{id}/impact")
    public ImpactPassportResponse
impact(@PathVariable Long id) {
        return orderService.impactPassport(id);
    }
}
```

Listing 10 – Admin Controller Class

```
@RestController
@RequestMapping("/api/admin")
@RequiredArgsConstructor
@PreAuthorize("hasRole('ADMIN')")
@Tag(name = "Адміністрування")
public class AdminController {

    private final AdminService adminService;
    private final ConcertService concertService;

    @GetMapping("/stats")
    public AdminStatsResponse stats() {
        return adminService.stats();
```

```
    }

    @GetMapping("/users")
    public List<UserResponse> users() {
        return adminService.users();
    }

    @PutMapping("/users/{id}")
    public UserResponse updateUser(
            @PathVariable Long id,
            @Valid @RequestBody AdminUserRequest
request
    ) {
        return adminService.updateUser(id, request);
    }

    @GetMapping("/orders")
    public List<OrderResponse> orders() {
        return adminService.orders();
    }

    @GetMapping("/concerts")
    public List<ConcertResponse> concerts() {
        return concertService.listAllConcerts();
    }

    @PostMapping("/concerts")
    @ResponseStatus(HttpStatus.CREATED)
    public ConcertResponse createConcert(@Valid
@RequestBody AdminConcertRequest request) {
        return adminService.createConcert(request);
    }

    @PutMapping("/concerts/{id}")
    public ConcertResponse updateConcert(
            @PathVariable Long id,
            @Valid @RequestBody AdminConcertRequest
request
    ) {
        return adminService.updateConcert(id, request);
    }
}
```

Listing 11 – Organizer Controller Class

```
@RestController
@RequestMapping("/api/organizer")
@RequiredArgsConstructor
@PreAuthorize("hasAnyRole('ORGANIZER','ADMIN')
")
@Tag(name = "Кабінет організатора")
public class OrganizerController {

    private final OrganizerService organizerService;

    @GetMapping("/stats")
    public OrganizerStatsResponse stats() {
        return organizerService.stats();
    }

    @GetMapping("/concerts")
    public List<ConcertResponse> concerts() {
        return organizerService.concerts();
    }

    @PostMapping("/concerts")
    @ResponseStatus(HttpStatus.CREATED)
    public ConcertResponse createConcert(@Valid
@RequestBody AdminConcertRequest request) {
        return organizerService.createConcert(request);
    }

    @PutMapping("/concerts/{id}")
    public ConcertResponse updateConcert(
            @PathVariable Long id,
            @Valid @RequestBody AdminConcertRequest
request
    ) {
        return organizerService.updateConcert(id, request);
    }

    @GetMapping("/orders")
    public List<OrderResponse> orders() {
        return organizerService.orders();
    }
}
```