МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ

ННІ Економіки та бізнес-освіти

Кафедра Економіки та цифрового бізнесу

Спеціальність 122 «Комп'ютерні науки»

Форма навчання Денна

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

Буйвола Владислава Євгенійовича
(прізвище, ім'я, по батькові здобувача)

на тему Розробка вебсервісу для продажу квитків у кінотеатрі
(повна назва теми)

за матеріалами

(повна назва бази дослідження)

науковий керівник к.т.н. _____ Селезньов М.Є.
(наук. ступінь, вчене звання) *(підпис)* *(прізвище, ініціали)*

Робота допущена до захисту в ЕК
Протокол засідання кафедри
від 12 червня 2026 р. № 15
Завідувач кафедри _____
(підпис)
к.е.н., доцент Радько В.М.
наук. ступень, вчене звання прізвище, ініціали

Кривий Ріг – 2026

ЗАТВЕРДЖЕНО
Наказ Міністерства освіти і науки, молоді та спорту України
29 березня 2012 року № 384
Форма № Н-9.01

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ ЕКОНОМІКИ І ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ЕКОНОМІКИ ТА БІЗНЕС-ОСВІТИ
(повне найменування вищого навчального закладу)

Кафедра економіки та цифрового бізнесу
Освітній ступінь бакалавр
Спеціальність 122 Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри _____ В.М. Радько

"30" березня 2026 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ БАКАЛАВРСЬКУ РОБОТУ ЗДОБУВАЧУ

Буйволу Владиславу Євгенійовичу

1. Тема роботи Розробка вебсервісу для продажу квитків у кінотеатрі

науковий керівник роботи Селезньов Максим Євгенович, к.т.н.
затверджені наказом вищого навчального закладу від «23» березня 2026 р. № 193-ст

2. Строк подання здобувачем роботи 29.05.2026р.

3. Зміст кваліфікаційної бакалаврської роботи, об'єкт, предмет та мета дослідження:
Розділ 1 Теоретичні аспекти розробки вебсервісу для продажу квитків у кінотеатрі
Розділ 2 Проєктування та розробка вебсервісу для продажу квитків у кінотеатрі
Розділ 3 Функціонування та елементи інтерфейсу вебсервісу для продажу квитків у кінотеатрі

Об'єкт дослідження – процес автоматизації продажу квитків у кінотеатрі з використанням сучасних вебтехнологій

Предмет дослідження - методи, технології та архітектурні рішення, що застосовуються при створенні вебсервісу для продажу квитків у кінотеатрі

Мета кваліфікаційної бакалаврської роботи – розробити онлайн-додаток для продажу квитків у кінотеатр, який забезпечує зручний перегляд афіші, вибір сеансу, бронювання місць, оформлення замовлення та адміністрування інформації про покази

4. Дата видачі завдання 03.04.2026р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів роботи	Відмітка керівника про виконання етапів (дата, підпис)
1	Підготовка розділу 1	до 17.04.2026р.	16.04.2026р.
2	Підготовка розділу 2	до 08.05.2026р.	07.05.2026р.
3	Підготовка розділу 3	до 25.05.2026р.	24.05.2026р.
4	Реєстрація завершеної кваліфікаційної роботи	до 29.05.2026р.	28.05.2026р.
5	Отримання відгуку від наукового керівника	04.06.2026р.	04.06.2026р.
6	Отримання зовнішньої рецензії	05.06.2026р.	05.06.2026р.
7	Попередній захист кваліфікаційної роботи на кафедрі	08.06.2026р.	08.06.2026р.
8	Перевірка кваліфікаційної роботи на плагіат	12.06.2026р.	12.06.2026р.
9	Допуск кафедрою кваліфікаційної роботи до захисту	12.06.2026р.	12.06.2026р.
10	Підготовка студента до захисту в ЕК	до 19.06.2026р.	18.06.2026р.

Завдання підготував науковий керівник _____ Селезньов М. Є.

Завдання одержав здобувач _____ Буйвол В. Є.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Робота містить 62 сторінки, 24 рисунка, 41 джерело, 2 таблиці, 2 додатка.

Об’єкт дослідження: процес автоматизації продажу квитків у кінотеатрі з використанням сучасних вебтехнологій.

Предмет дослідження: методи, технології та архітектурні рішення, що застосовуються при створенні вебсервісу для продажу квитків у кінотеатрі

Мета: розробити онлайн-додаток для продажу квитків у кінотеатр, який забезпечує зручний перегляд афіші, вибір сеансу, бронювання місць, оформлення замовлення та адміністрування інформації про покази.

У результаті дослідження було проаналізовано особливості організації продажу квитків у кінотеатрі, розглянуто сучасні підходи до створення веб-додатків та визначено функціональні вимоги до системи. Було спроєктовано структуру онлайн-додатка, розроблено модель бази даних, інтерфейс користувача та логіку взаємодії основних модулів системи. Обрано відповідні засоби та технології розробки, що забезпечують стабільну роботу додатка, зручність використання та можливість подальшого розширення функціоналу. Також реалізовано користувацьку частину системи для перегляду фільмів, вибору місць і придбання квитків, а також адміністративну частину для керування сеансами, фільмами та замовленнями.

Область застосування: результати роботи можуть бути використані в діяльності кінотеатрів, розважальних закладів, сервісів онлайн-бронювання та продажу квитків, а також у навчальному процесі під час вивчення веб-програмування, проєктування інформаційних систем і баз даних.

БАЗА ДАНИХ, БРОНЮВАННЯ МІСЦЬ, ВЕБДОДАТОК, ВЕБІНТЕРФЕЙС, ВЕБСЕРВІС, ІНФОРМАЦІЙНА СИСТЕМА, КІНОТЕАТР, ОНЛАЙН-ДОДАТОК, ПРОДАЖ КВИТКІВ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

SPA - Single Page Application

CRM - Customer Relationship Management

QR-код - Quick Response code

REST API - Representational State Transfer Application Programming Interface

API - Application Programming Interface

UX - User Experience

UI - User Interface

Node.js - Node JavaScript

SQL - Structured Query Language

HTTPS - HyperText Transfer Protocol Secure)

CSS - Cascading Style Sheets

ACID - Atomicity, Consistency, Isolation, Durability

PDF - Portable Document Format

JSON - JavaScript Object Notation

JS - JavaScript

ВСТУП

У сучасних умовах цифровізації сфери послуг особливої актуальності набуває створення онлайн-сервісів, що забезпечують швидку та зручну взаємодію між закладами культури й відвідувачами. Однією з таких сфер є кіноіндустрія, де продаж квитків дедалі частіше здійснюється через веборієнтовані системи. Тому тема кваліфікаційної роботи є актуальною, оскільки пов’язана з розробленням сучасного програмного продукту для автоматизації вибору сеансу, бронювання місць та придбання квитків.

Актуальність теми зумовлена зростанням потреби користувачів у дистанційному отриманні послуг, економії часу та підвищенні зручності обслуговування. Для кінотеатрів впровадження онлайн-додатка сприяє покращенню якості сервісу, зменшенню навантаження на каси, оптимізації продажів і зручному керуванню інформацією про фільми, сеанси та замовлення. Отже, розроблення такого додатка відповідає сучасним тенденціям розвитку інформаційних технологій і практичним потребам ринку.

Проблема автоматизації обліку у розважальній сфері, шляхом створення вебдодатків, проєктування інформаційних систем і баз даних достатньо широко висвітлена у вітчизняних і зарубіжних джерелах. Значна кількість праць присвячена загальним підходам до побудови вебсистем, розробленню інтерфейсів користувача та використанню сучасних технологій програмування. Водночас питання створення цілісного онлайн-додатка саме для продажу квитків у кінотеатр з урахуванням специфіки бронювання місць, структури даних та адміністрування контенту потребує додаткового практичного опрацювання.

Інформаційною базою дослідження є наукові праці з проєктування інформаційних систем, вебпрограмування, баз даних, UX/UI-дизайну, а також технічна документація сучасних мов програмування, фреймворків і систем керування базами даних. Окрему групу джерел становлять приклади існуючих сервісів продажу квитків і бронювання, що дають змогу оцінити практичні підходи до реалізації подібних систем.

Об’єктом дослідження є процес автоматизації продажу квитків у кінотеатрі із застосуванням сучасних інформаційних технологій.

Предметом дослідження є методи, засоби та особливості розроблення онлайн-додатка для продажу квитків у кінотеатр, зокрема його структура, функціональні можливості, організація даних та інтерфейс користувача.

Метою роботи є розробка онлайн-додатка для продажу квитків у кінотеатр, який забезпечує перегляд сеансів, вибір місць, оформлення замовлення та адміністрування інформації про фільми й покази.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– проаналізувати предметну область;
– визначити функціональні вимоги до системи;
– обґрунтувати вибір засобів розробки;
– спроєктувати структуру додатка і базу даних;
– реалізувати основний функціонал користувацької та адміністративної частин;
– провести тестування розробленої системи.

Методика дослідження ґрунтується на використанні аналізу наукових і технічних джерел, системного підходу, методів моделювання, проєктування інформаційних систем, програмної реалізації та тестування. Це дало змогу комплексно дослідити проблему та створити працездатний програмний продукт.

Наукова новизна дослідження полягає у практичному поєднанні підходів до створення систем онлайн-продажу квитків у межах єдиного додатка, адаптованого до потреб кінотеатру. Для кваліфікаційної роботи новизна також полягає у самостійному проєктуванні структури системи, бази даних і реалізації її основного функціоналу.

Практична значущість роботи полягає в тому, що розроблений додаток може бути використаний як основа для впровадження у діяльність кінотеатру або як прототип для подальшого вдосконалення. Його застосування дає змогу підвищити ефективність обслуговування клієнтів, спростити процес купівлі квитків і покращити організацію продажів.

РОЗДІЛ 1
ТЕОРЕТИЧНІ АСПЕКТИ РОЗРОБКИ ВЕБСЕРВІСУ ДЛЯ ПРОДАЖУ КВИТКІВ У КІНОТЕАТРІ

1.1 Сучасні підходи до виконання електронного обліку у розважальній сфері

Електронний облік у розважальній сфері в сучасних умовах уже не зводиться лише до фіксації факту продажу квитка або формування звітності наприкінці робочого дня. Для кінотеатрів, концертних майданчиків, театрів та інших закладів дозвілля він перетворився на безперервний інформаційний процес, у межах якого відображаються всі зміни стану послуги: поява події в афіші, відкриття продажу, вибір місця, резервування, оплата, повернення, скасування, формування електронного квитка та аналітика відвідуваності. Саме така логіка дозволяє говорити не просто про облік як про допоміжну функцію, а про цифрову основу всього сервісного циклу.

У науковій і прикладній літературі останніх років цифрова трансформація сервісних галузей розглядається як перехід від фрагментарних інформаційних операцій до інтегрованих систем, де бізнес-процеси, облік, аналітика та клієнтський сервіс функціонують у єдиному середовищі. Для сфери розваг це особливо важливо, оскільки цінність послуги є часово обмеженою: квиток на конкретний сеанс або подію не може бути реалізований після закінчення часу її проведення, а отже будь-які помилки в обліку безпосередньо впливають на дохід і репутацію закладу [1, 2].

Особливість електронного обліку у кінотеатрі полягає в тому, що облікованим об’єктом є не лише замовлення, а вся система взаємопов’язаних сутностей. До них належать фільм, сеанс, зал, місце, тариф, промокод, користувач, замовлення, платіж, повернення, електронний квиток та службові журнали дій адміністратора (таблиця 1.1). Кожна з цих сутностей змінює свій

стан у процесі експлуатації системи, а тому правильна побудова обліку потребує не стільки простого накопичення даних, скільки чіткого опису життєвого циклу кожного запису.

Таблиця 1.1

Основні об’єкти електронного обліку у кінотеатрі

Об’єкт обліку	Призначення	Ключові дані
Фільм	Базова одиниця афіші та розкладу	назва, жанр, тривалість, формат, вікові обмеження
Сеанс	Конкретний показ фільму	дата, час, зал, ціна, статус продажу
Місце	Одиниця резервування та продажу	ряд, номер, категорія, стан доступності
Замовлення	Фіксація наміру та факту покупки	користувач, сума, статус, часові мітки
Платіж	Підтвердження фінансової операції	спосіб оплати, транзакція, результат
Електронний квиток	Фінальний результат послуги	ідентифікатор, QR-код, статус використання

На практиці це означає, що сучасний вебсервіс продажу квитків має працювати як централізована інформаційна система. Дані про афішу, розклад, доступність місць і транзакції повинні синхронізуватися між публічним сайтом, адміністративною панеллю, касою та, за потреби, мобільним застосунком. За відсутності єдиного сховища або узгоджених правил синхронізації виникають дублювання записів, затримки оновлення станів місць та конфлікти між каналами продажу. Саме тому одним із базових сучасних підходів є формування «єдиного джерела істини», у якому всі канали звертаються до спільної актуальної моделі даних.

Ще одним важливим підходом є подійна логіка фіксації операцій. Для електронного обліку недостатньо знати, що квиток продано. Потрібно зберігати інформацію про те, коли користувач відкрив сторінку сеансу, які місця вибрав, коли було створено тимчасове бронювання, чи завершився таймер оплати, чи

успішно пройшла транзакція, чи було видано електронний квиток, а також чи здійснювалося повернення коштів. Такий підхід дозволяє не лише контролювати стан замовлення, а й аналізувати поведінку користувачів, виявляти вузькі місця в сценарії купівлі та вдосконалювати інтерфейс.

Для розважальної сфери характерна ситуація, коли той самий ресурс може бути одночасно цікавим багатьом користувачам. У кінотеатрі таким ресурсом є місце в залі на конкретний сеанс. Через це електронний облік повинен підтримувати механізм короткострокового резервування. Коли користувач вибирає місця, система на певний проміжок часу переводить їх у стан попереднього бронювання, недоступного для інших клієнтів. Якщо оплата не відбувається вчасно, місця знову повертаються в продаж. У цьому процесі важливі точні часові мітки, контроль конкурентного доступу та чітке визначення станів «вільно», «зарезервовано», «оплачено», «повернено», «заблоковано службово».

Окрема риса сучасного електронного обліку полягає в орієнтації на омніканальність. Користувач може ознайомитися з розкладом на сайті, отримати нагадування в мобільному застосунку, завершити оплату через сторонній платіжний сервіс та пред’явити електронний квиток у вигляді QR-коду при вході до зали. З технічної точки зору це означає, що облік не може існувати відокремлено від сервісного шару. Він має забезпечувати однаково коректну роботу в різних точках контакту з клієнтом, а інформація про замовлення повинна залишатися цілісною незалежно від способу взаємодії.

Суттєве значення в сучасних підходах має аналітична складова. Електронний облік дає змогу збирати не лише обов’язкову транзакційну інформацію, а й дані для управлінських рішень: популярність жанрів, завантаженість залів у різні дні тижня, ефективність акцій, частоту використання промокодів, середній чек, частку мобільних замовлень, конверсію від перегляду сторінки сеансу до завершення покупки. Для менеджменту кінотеатру це означає можливість оптимізувати розклад, варіювати ціни, точніше планувати рекламні кампанії та своєчасно реагувати на зміни попиту.

Практика роботи кінотеатрів показує, що накопичені облікові дані мають цінність не лише для підсумкової статистики, а й для щоденного оперативного керування. На основі поточних показників можна відстежувати швидкість продажу на окремі сеанси, оцінювати ефективність часових слотів, своєчасно відкривати або закривати додаткові покази, а також коригувати рекламну підтримку прем'єр. Отже, сучасний електронний облік дедалі більше наближається до ролі управлінської інформаційної системи, у якій дані не просто зберігаються, а перетворюються на основу для оперативних рішень адміністрації закладу.

Не менш важливою є побудова ролей і зон відповідальності в межах системи. Публічна частина сервісу орієнтована на відвідувача, який переглядає афішу, вибирає сеанс і купує квиток. Водночас службова частина повинна надавати адміністраторам інструменти для створення та редагування фільмів, налаштування залів, запуску продажу, контролю статусів бронювань, обробки повернень і перегляду статистики. Тому сучасний електронний облік у розважальній сфері завжди поєднує фронтальну клієнтську взаємодію з внутрішнім управлінським контуром.

Умовою ефективності такого обліку є також безпека. Інформація про клієнтів, історію замовлень і результати оплати не може оброблятися без механізмів автентифікації, авторизації, журналювання дій та резервного копіювання. Для сервісів, що працюють у режимі постійної доступності, важливе значення мають відмовостійкість, контроль версій даних і захист від помилок персоналу. У протилежному разі навіть технічно справний інтерфейс не забезпечить надійної роботи всієї системи.

Таким чином, сучасні підходи до виконання електронного обліку у розважальній сфері спираються на інтегрованість, централізацію даних, подійний характер операцій, підтримку кількох каналів продажу, аналітичність, розмежування ролей і безпеку. Саме ці принципи мають бути враховані при розробці вебсервісу продажу квитків у кінотеатр, оскільки вони формують не лише технічні вимоги до системи, а й модель її реального використання.

1.2 Огляд існуючих цифрових рішень для електронного обліку у розважальній сфері

Огляд наявних цифрових рішень у розважальній сфері доцільно проводити не лише з точки зору зовнішнього інтерфейсу, а й з позицій того, як саме в них організовано електронний облік подій, місць, замовлень і взаємодію між каналами продажу. Для кваліфікаційної роботи важливо не копіювати конкретний комерційний продукт, а виділити ті функціональні та архітектурні рішення, які вже стали галузевим стандартом і можуть бути адаптовані до власного вебсервісу.

Умовно сучасні цифрові рішення можна поділити на дві великі групи. До першої належать агрегаторні платформи, які об’єднують пропозиції багатьох кінотеатрів, дозволяють шукати сеанси за містом або фільмом та виконують роль єдиного інтерфейсу купівлі для широкої аудиторії. До другої належать власні екосистеми кіномереж, де цифровий сервіс глибше інтегрований з внутрішніми процесами компанії, програмами лояльності, продажем супутніх товарів, бонусами та мобільними додатками.

Платформа Fandango є прикладом сервісу, у якому велика увага приділяється доступності розкладу, швидкому переходу від вибору фільму до вибору конкретного сеансу та інтерактивній схемі залу. Для користувача важливо, що сервіс дозволяє працювати з багатьма кінотеатрами за схожим сценарієм, а для предметної області показовим є те, що навіть у моделі агрегатора зберігається вимога до актуального стану місць і коректного підтвердження бронювання.

Сервіс AMC Theatres демонструє інший підхід. Тут вебсайт і мобільний застосунок є частиною цілісної брендованої екосистеми. Користувач отримує не лише інструмент для купівлі квитків, а й особистий кабінет, цифрові квитки, персоналізовані пропозиції, бонусні програми та інші елементи довготривалої

взаємодії з мережею. Для електронного обліку це означає тісніший зв’язок між транзакційним модулем і CRM-складовою.

Atom Tickets акцентує увагу на мобільному сценарії використання, спрощеній логіці купівлі та соціальній складовій вибору походу в кіно. Для досліджуваної теми цей приклад важливий тим, що в ньому особливо чітко простежується пріоритет мінімальної кількості дій користувача: швидкий вибір сеансу, зрозуміла карта місць, короткий шлях до оплати та цифрове підтвердження замовлення. Така модель особливо релевантна для молодіжної аудиторії, що звикла до mobile-first сервісів.

Для українського ринку репрезентативними прикладами є сервіси Multiplex та Планета Кіно. Вони підтримують повний базовий сценарій купівлі: перегляд афіші, фільтрацію сеансів, вибір місць, оплату та отримання електронного квитка. Водночас локальні сервіси враховують мовні, організаційні та платіжні особливості українського ринку, що робить їх корисним орієнтиром саме для навчального проєкту, який розробляється з урахуванням очікувань вітчизняного користувача [3-7].

Порівняння перелічених рішень (див. таб. 1.2) дозволяє побачити, що ядро галузевого стандарту вже сформувалося. Користувач очікує, що сервіс покаже афішу в зручній формі, дозволить швидко змінювати дату та кінотеатр, наочно відобразить схему залу, дасть змогу вибрати кілька місць без зайвих переходів, прийме оплату онлайн і збереже електронний квиток. Якщо хоча б один із цих елементів реалізовано незручно, загальний користувацький досвід погіршується навіть за наявності широкого функціоналу.

Для електронного обліку особливо показовим є сценарій роботи з місцями. Успішні сервіси візуально розрізняють доступні, вибрані, заброньовані та недоступні місця, а також швидко синхронізують стан між користувачем і сервером. Це знижує ризик конфліктів і підвищує довіру до системи. Якщо ж зміни відображаються із затримкою, користувач може отримати негативний досвід навіть тоді, коли проблема полягає не в самій оплаті, а в некоректному відображенні доступності.

Іншим важливим критерієм є ступінь навантаження інтерфейсу другорядною інформацією. Частина наявних сервісів прагне поєднати в одному сценарії продаж квитків, рекламу фільмів, програми лояльності, продаж напоїв і снеків, пропозиції преміальних карток та інші активності. З точки зору бізнесу це зрозуміло, однак для користувача перевантажений інтерфейс нерідко ускладнює основну дію - швидку купівлю квитка на конкретний сеанс.

Таблиця 1.2

Порівняльна характеристика існуючих цифрових рішень

Сервіс	Вибір місць	Мобільний сценарій	Додаткові можливості	Практичний висновок
Fandango	так	високий	агрегування сеансів, reserved seating	показовий приклад платформи-агрегатора
AMC Theatres	так	високий	digital tickets, loyalty, app ecosystem	сильна інтеграція з брендовою екосистемою
Atom Tickets	так	дуже високий	mobile-first, швидкий сценарій купівлі	орієнтир для мобільного UX
Multiplex	так	високий	українська локалізація, зрозумілий процес купівлі	релевантний орієнтир для локального ринку
Планета Кіно	так	високий	афіша, карта залу, онлайн-оплата	приклад збалансованого сервісу мережі
Узагальнений стандарт	обов’язково	обов’язково	QR-квиток, оплата, синхронізація статусів	база для власного проєктування

Важливо також враховувати післяпродажний етап взаємодії з користувачем. Якісний цифровий сервіс не завершується моментом успішної оплати, а повинен підтримувати зручне отримання квитка, повторний перегляд замовлення, інформування про зміни в розкладі та зрозумілу логіку повернення. Саме ці елементи часто визначають загальне враження від сервісу не менше, ніж власне процес вибору місць. Для кваліфікаційної роботи це означає, що при аналізі аналогів слід звертати увагу не тільки на головну сторінку чи афішу, а й на повний життєвий цикл замовлення після завершення транзакції.

Для власного вебсервісу доцільно зробити висновок, що функціональність має розгортатися навколо основного сценарію, а не відволікати від нього. У центрі проєктування повинні залишатися афіша, сеанс, карта залу, замовлення та електронний квиток. Додаткові елементи - відгуки, рекомендації, програми лояльності, акції - можуть бути реалізовані як розширення, але не повинні руйнувати логіку базової взаємодії.

Отже, аналіз існуючих цифрових рішень показує, що для успішного сервісу продажу квитків вирішальними є простота інтерфейсу, швидкий сценарій вибору місць, стабільна синхронізація станів, підтримка мобільного використання, інтеграція з оплатою та можливість внутрішнього обліку й аналітики. Саме ці характеристики мають бути покладені в основу проєктованого вебсервісу.

1.3 Вибір технологічного стеку та архітектури вебсервісу

Вибір технологічного стеку та архітектури вебсервісу для продажу квитків у кінотеатрі повинен спиратися не на популярність окремих інструментів як таких, а на відповідність вимогам предметної області. Для цього сервісу важливими є швидка реакція інтерфейсу, підтримка сценарію вибору місць, обробка великої кількості однотипних запитів, надійне збереження транзакцій, чітке розмежування ролей користувача й адміністратора, а також можливість подальшого розширення функціоналу без повної переробки системи.

З урахуванням логіки, закладеної в попередньому проєктуванні, доцільно виходити з клієнт-серверної моделі, у межах якої фронтенд відповідає за взаємодію з користувачем, а бекенд - за бізнес-логіку, роботу з базою даних та інтеграцію із зовнішніми сервісами. Такий поділ дозволяє чітко розмежувати відповідальність компонентів, спростити тестування і надалі розвивати окремі частини системи без порушення цілісності всього рішення.

На стороні клієнта найбільш доцільним є застосування підходу SPA, тобто односторінкового вебзастосунку. Для сервісу продажу квитків це важливо тому, що користувач у межах однієї сесії неодноразово переходить між афішею, сторінкою фільму, розкладом, картою залу, кошиком і формою підтвердження замовлення. Якщо після кожної дії сторінка буде повністю перезавантажуватися, інтерфейс сприйматиметься повільним і менш зручним. Компонентний підхід React (рис. 1.1) добре підходить для побудови повторно використовуваних елементів - карток фільмів, блоків сеансів, схем місць, форм авторизації, повідомлень про статус замовлення та інших вузлів інтерфейсу [8].

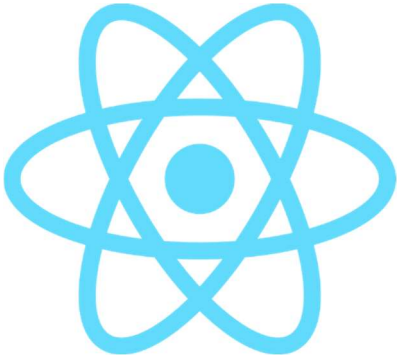

Рис. 1.1. Логотип React - основи компонентного фронтенду

У поєднанні з React доцільно використати сучасний інструментарій збирання проєкту, зокрема Vite, який забезпечує швидкий старт розробки, зрозумілу структуру фронтенд-проєкту та комфортний цикл локального тестування. Для навчального та бакалаврського проєкту це має практичне значення: менше часу витрачається на технічне налаштування середовища, а більше - на реалізацію предметної логіки. Крім того, сучасний фронтенд-стек дає змогу організувати маршрутизацію, керування станом, валідацію форм і взаємодію з API у послідовній і зрозумілій моделі.

Особливо важливим для фронтенду є моделювання станів місць у залі. Інтерфейс повинен показувати, які місця вже зайняті, які доступні, які щойно вибрані користувачем, а які зарезервовані на короткий час. У компонентній архітектурі така поведінка реалізується природно: окремі компоненти

відповідають за відображення рядів, місць, панелі замовлення і підсумкової вартості, а зміни стану передаються через контрольовані оновлення даних. Саме тому обраний підхід добре масштабується від простого макета до повноцінного сервісу.

На серверному рівні раціонально обрати Node.js (його логотип представлений на рис.1.2) у поєднанні з фреймворком Express. Таке рішення дозволяє використовувати JavaScript як на клієнтській, так і на серверній стороні, що прискорює розробку та полегшує підтримку коду. Для вебсервісу продажу квитків перевагою є також подієва й асинхронна модель роботи Node.js, яка добре підходить до обробки великої кількості запитів на перегляд сеансів, авторизацію, створення бронювання та підтвердження оплати. Express, своєю чергою, забезпечує зрозумілу організацію маршрутів, middleware-логіку, обробку помилок і побудову API без зайвої складності [9, 10].

Рис. 1.2. Логотип Node.js - середовища виконання серверної частини

У межах серверної частини доцільно виокремити кілька логічних модулів: робота з афішею та фільмами; керування залами, місцями й сеансами; авторизація та ролі; замовлення і бронювання; платежі; звітність та адміністративні операції. Навіть якщо кваліфікаційна робота реалізується у формі модульного моноліту, така внутрішня декомпозиція дозволяє зберегти архітектурну дисципліну та підготувати систему до подальшого масштабування.

Як спосіб взаємодії між клієнтом і сервером доцільно використовувати REST API. Для предметної області продажу квитків цей підхід є зручним і прозорим, оскільки більшість сутностей природно подаються як ресурси:

фільми, сеанси, місця, бронювання, замовлення, користувачі. Це спрощує як проєктування запитів, так і тестування системи. Наприклад, окремі маршрути можуть повертати розклад за датою, карту доступних місць за конкретним сеансом, створювати тимчасове бронювання або підтверджувати оплату. Така структура API є зрозумілою для фронтенду і добре документується.

Окремої уваги заслуговує спостережуваність системи, тобто наявність журналів, повідомлень про помилки та засобів контролю ключових подій. Для сервісу продажу квитків це принципово, оскільки помилка може виникнути на різних етапах: під час резервування місця, підтвердження оплати, формування електронного квитка або надсилання повідомлення користувачу. Якщо архітектура від самого початку передбачає логування критичних операцій і зрозумілі коди помилок, це суттєво спрощує супровід, діагностику проблем та підвищує надійність сервісу в реальних умовах експлуатації.

Щодо зберігання даних, найбільш обґрунтованим є вибір реляційної бази даних SQLite (його логотип представлений на рис.1.3). Продаж квитків потребує цілісності даних, підтримки транзакцій та надійної роботи зі зв’язками між сутностями. У системі потрібно гарантувати, що два користувачі не куплять одне й те саме місце, що замовлення пов’язане з конкретним користувачем, сеансом і набором місць, а статуси бронювання змінюються послідовно і контрольовано. Реляційна модель добре відповідає такій логіці, а SQLite є доцільним рішенням для розробленого вебсервісу, оскільки підтримує SQL, транзакції ACID, зберігає базу даних в одному файлі та не потребує окремого серверного налаштування [11].

Структура бази даних для вебсервісу кінотеатру має включати щонайменше таблиці користувачів, ролей, фільмів, жанрів, залів, місць, сеансів, тарифів, бронювань, замовлень, платежів і журналів дій. Важливо не лише створити ці таблиці, а й коректно визначити зв’язки між ними, унікальні обмеження, індекси та правила перевірки даних. Наприклад, комбінація «сеанс + місце» повинна контролюватися таким чином, щоб виключити дублювання

продажів, а транзакція створення замовлення повинна або відбутися повністю, або не змінити стан системи взагалі.

Рис. 1.3. Логотип SQLite - системи управління базами даних

З архітектурної точки зору найбільш доцільною є модель модульного моноліту. На відміну від мікросервісної архітектури, вона не вимагає складної інфраструктури розгортання, окремих служб, брокерів повідомлень і мережевої координації між модулями. Водночас модульний моноліт дозволяє логічно відокремити компоненти системи й підтримувати їх у впорядкованому вигляді. Для кваліфікаційної роботи це оптимальний компроміс між архітектурною дисципліною та реалістичністю реалізації.

У такій моделі фронтенд звертається до одного серверного застосунку через набір API-маршрутів, сервер виконує бізнес-правила і взаємодіє з єдиною базою даних. Це спрощує налаштування середовища, деплоймент, локальне тестування і налагодження. Одночасно зберігається можливість у майбутньому винести окремі вузли - наприклад, модуль сповіщень або аналітики - в окремі сервіси, якщо система зростатиме.

Ще одним аргументом на користь обраного стеку є доступність засобів розгортання та тестування. Для React- і Node.js-проєктів існує велика кількість готових інструментів для автоматичної перевірки коду, запуску модульних тестів, збирання застосунку та розміщення його на хмарних або локальних серверах. Це має практичну перевагу: є можливість зосередитися на реалізації прикладної логіки сервісу, не витрачаючи надмірних ресурсів на підтримку

складної інфраструктури. У результаті обрана архітектура відображає сучасні підходи професійної веброзробки.

Окреме місце при виборі архітектури посідає безпека. Вебсервіс для продажу квитків працює з персональними даними, історією замовлень та результатами платіжних операцій, тому питання захисту не може розглядатися як другорядне. На рівні архітектури це означає обов'язкове використання HTTPS, чітку модель автентифікації та авторизації, захист від типових вебуразливостей, контроль доступу до адміністративної панелі, журналювання критичних подій та резервне копіювання даних. Для платіжного сценарію доцільно інтегрувати зовнішній платіжний шлюз, не зберігаючи повні реквізити карток у власній системі [12].

У контексті кінотеатру доцільно відразу розділити ролі звичайного користувача та адміністратора. Користувач повинен мати доступ до афіші, покупки та історії замовлень, а адміністратор - до керування каталогом фільмів, запуску сеансів, редагування цін, перегляду статистики та обробки нестандартних ситуацій. Відповідно, система автентифікації повинна працювати не лише як спосіб входу, а як механізм обмеження операцій за ролями.

Не менш важливим є питання продуктивності. Архітектура повинна коректно поводитися в пікових ситуаціях - наприклад, під час запуску продажу на прем'єру популярного фільму. Для цього потрібно уникати зайвих запитів, кешувати довідкові дані, використовувати пагінацію, а також оптимізувати запити до бази даних, особливо в частині розкладу та карти залу.

Перспективним напрямом розвитку є використання механізмів, близьких до реального часу, для відображення змін стану місць під час одночасного перегляду сеансу кількома користувачами. У базовому варіанті це може бути реалізовано через регулярні запити до сервера, а в подальшому - через WebSocket або подібні інструменти. Важливо забезпечити надійну базову логіку роботи бронювання, а вже потім розширювати систему реального часу.

Підсумовуючи, для розробки вебсервісу продажу квитків у кінотеатр доцільно обрати таку конфігурацію: React + Vite для клієнтської частини, Node.js + Express для серверної логіки, REST API як стиль взаємодії між рівнями та SQLite як основну базу даних. Архітектурно найбільш виправданою є клієнт-серверна модель із реалізацією у вигляді модульного моноліту. Такий стек є достатньо сучасним, узгоджується з вимогами предметної області та створює надійну основу для подальшої реалізації системи.

Висновки до розділу 1

Сучасний вебсервіс продажу квитків у кінотеатр повинен розглядатися не лише як інтерфейс для покупки, а як інтегрована інформаційна система, у якій поєднуються облік, бронювання, оплата, адміністрування й аналітика. Розважальна сфера висуває до такого сервісу особливі вимоги, пов’язані з часовою обмеженістю послуги, високою динамікою попиту та необхідністю точного контролю доступності місць.

Огляд наявних цифрових рішень показав, що ринок уже сформував набір базових очікувань користувача: швидкий перегляд афіші, зручний вибір сеансу, наочна карта залу, короткий шлях до оплати, електронний квиток і коректна робота на мобільних пристроях. Одночасно аналіз виявив, що частина наявних сервісів перевантажує основний сценарій другорядними функціями, тому для власного проєкту доцільно зберігати фокус на ключових операціях купівлі.

На підставі проведеного аналізу обґрунтовано вибір технологічного стеку та архітектури вебсервісу. Для клієнтської частини доцільно використовувати React та Vite, для серверної - Node.js та Express, для зберігання даних - SQLite, а взаємодію між рівнями організувати через REST API. Архітектурно найбільш виправданою є клієнт-серверна модель у формі модульного моноліту, яка забезпечує достатню гнучкість, керованість і можливість подальшого розвитку системи.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБСЕРВІСУ ДЛЯ ПРОДАЖУ КВИТКІВ У КІНОТЕАТРІ

2.1 Проєктування інформаційної моделі та бази даних вебсервісу

Проєктування інформаційної моделі вебсервісу для продажу квитків у кінотеатрі доцільно розпочати з уточнення предметної області шдяхом визначення основних сутностей, ролей користувачів і зв’язків між програмними модулями. При цьому, слід враховувати, що на відміну від статичної сторінки, розроблений ресурс повинен не лише показувати афішу, а й підтримувати стан користувачів, сеансів, місць, замовлень, електронних квитків, бонусів і адміністративних змін.

Логічна модель системи відображає взаємодію двох основних груп користувачів: відвідувача та адміністратора. Відвідувач працює з афішею, каталогом, сторінкою фільму, схемою залу, профілем та електронним квитком. Адміністратор відповідає за службові операції: створення та редагування фільмів, керування сеансами, перегляд замовлень, зміну статусів квитків і перевірку QR-коду [13].

Загальну діаграму прецедентів роботи вебсервісу представлено на рис. 2.1. Діаграма показує, що користувацький сценарій завершується отриманням QR-квитка, а адміністративний сценарій забезпечує контроль каталогу, розкладу та замовлень.

Для деталізації логіки предметної області було побудовано діаграму класів. Вона описує основні об’єкти програмної моделі: користувача, фільм, сеанс, замовлення, місце у замовленні, сесію входу, обране та промокод. Таке подання дає змогу побачити, які дані зберігаються в системі та які зв’язки виникають між ними під час покупки квитка [14].

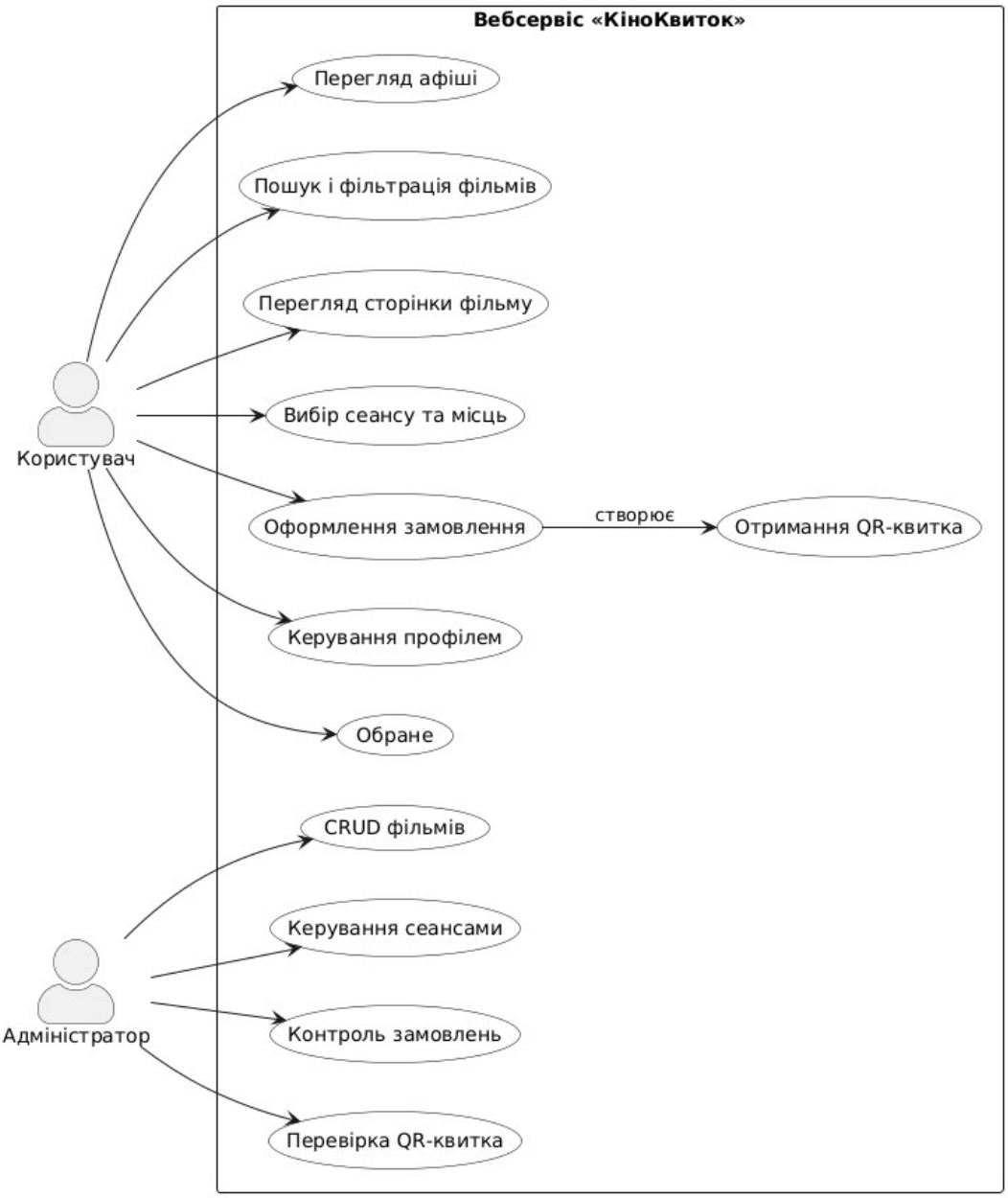

Рис. 2.1. Діаграма прецедентів вебсервісу «КіноКвиток»

На рис. 2.2 показано діаграму класів вебсервісу. Центральними сутностями є User, Movie, Showtime та Order. Користувач може мати багато замовлень, фільм може мати багато сеансів, а одне замовлення може містити кілька місць. Окремі класи Session і Favorite забезпечують підтримку авторизації та персоналізованого списку обраних фільмів.

Окремо було змодельовано послідовність прецеденту оформлення квитка. Цей прецедент є ключовим, оскільки саме він поєднує клієнтську частину,

серверну логіку та базу даних. Користувач обирає фільм, дату, сеанс і місця, після чого клієнтська частина надсилає структурований запит на сервер.

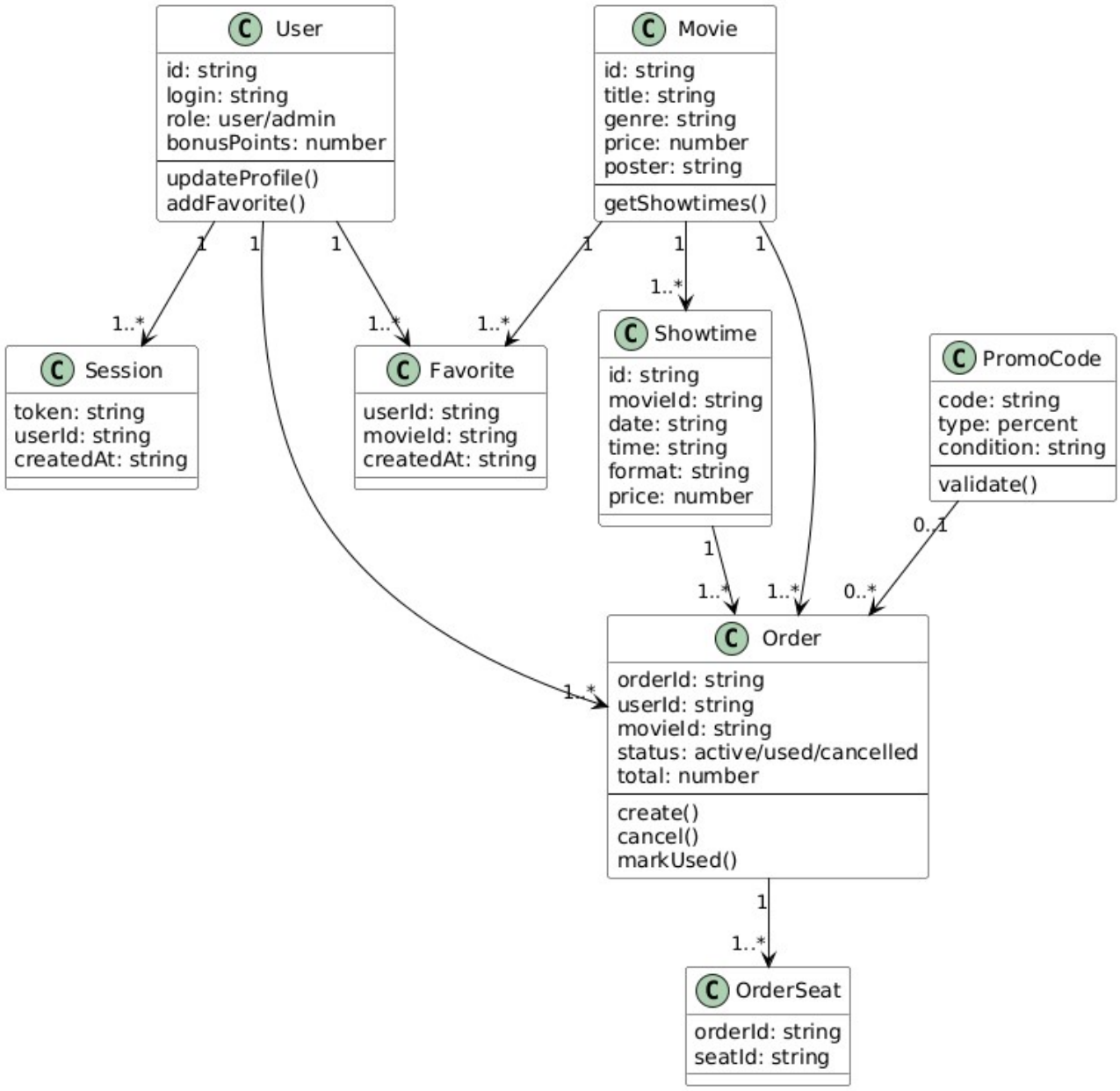

Рис. 2.2. Діаграма класів вебсервісу «КіноКвиток»

Сервер перевіряє доступність місць, створює запис замовлення, зберігає вибрані місця і повертає готовий електронний квиток. Послідовність виконання цього сценарію наведено на рис. 2.3. Вона показує, що основна перевірка

відбувається на сервері, а клієнтський інтерфейс відповідає за зручне відображення результату для користувача.

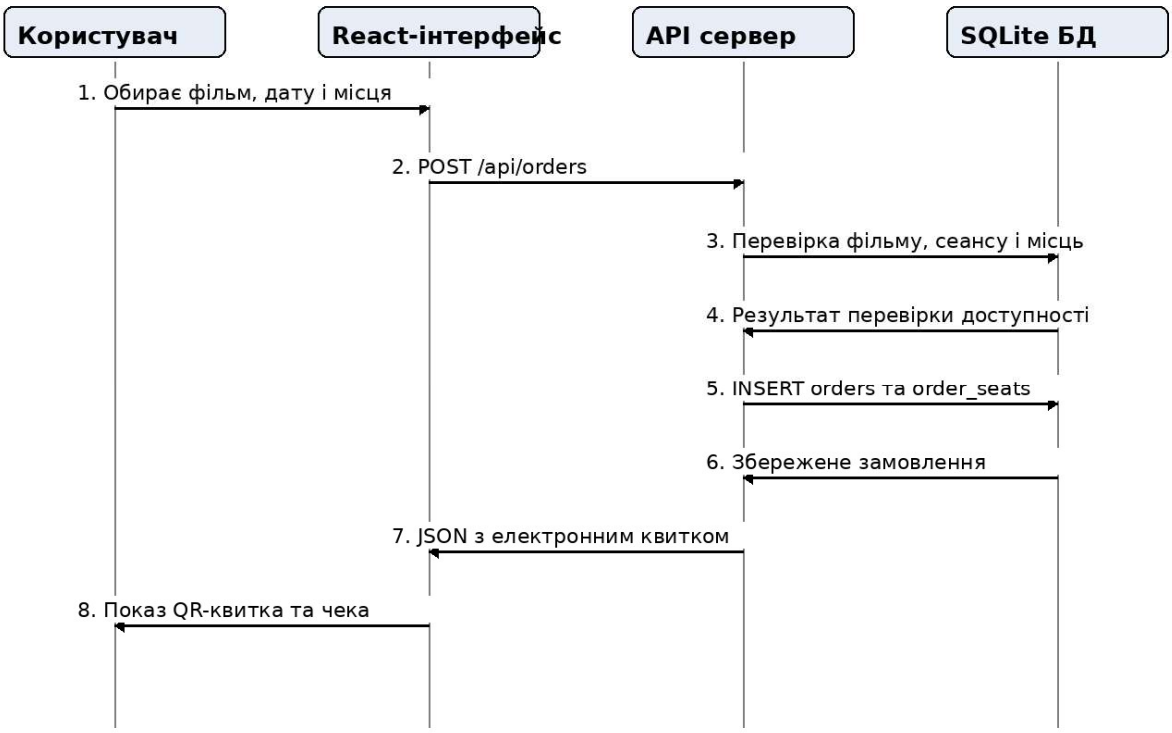

Рис. 2.3. Діаграма послідовності прецеденту оформлення квитка

Фізична модель даних була реалізована у SQLite. База даних містить таблиці users, sessions, movies, showtimes, orders, order_seats і favorites. Такий набір таблиць дозволяє відокремити персональні дані користувачів, довідкові дані фільмів, розклад показів та операційні дані замовлень. Це знижує дублювання інформації і спрощує подальше адміністрування [15].

ER-діаграма бази даних наведена на рис. 2.4. Вона відображає первинні та зовнішні ключі, а також основні зв’язки один-до-багатьох між користувачами, фільмами, сеансами та замовленнями.

Компонентна модель пояснює, як окремі частини проєкту взаємодіють між собою під час запуску вебресурсу. Браузер завантажує зібрану клієнтську

частину React, клієнт надсилає REST-запити на Node.js-сервер, а сервер звертається до SQLite-бази, файлового сховища постерів і службових модулів.

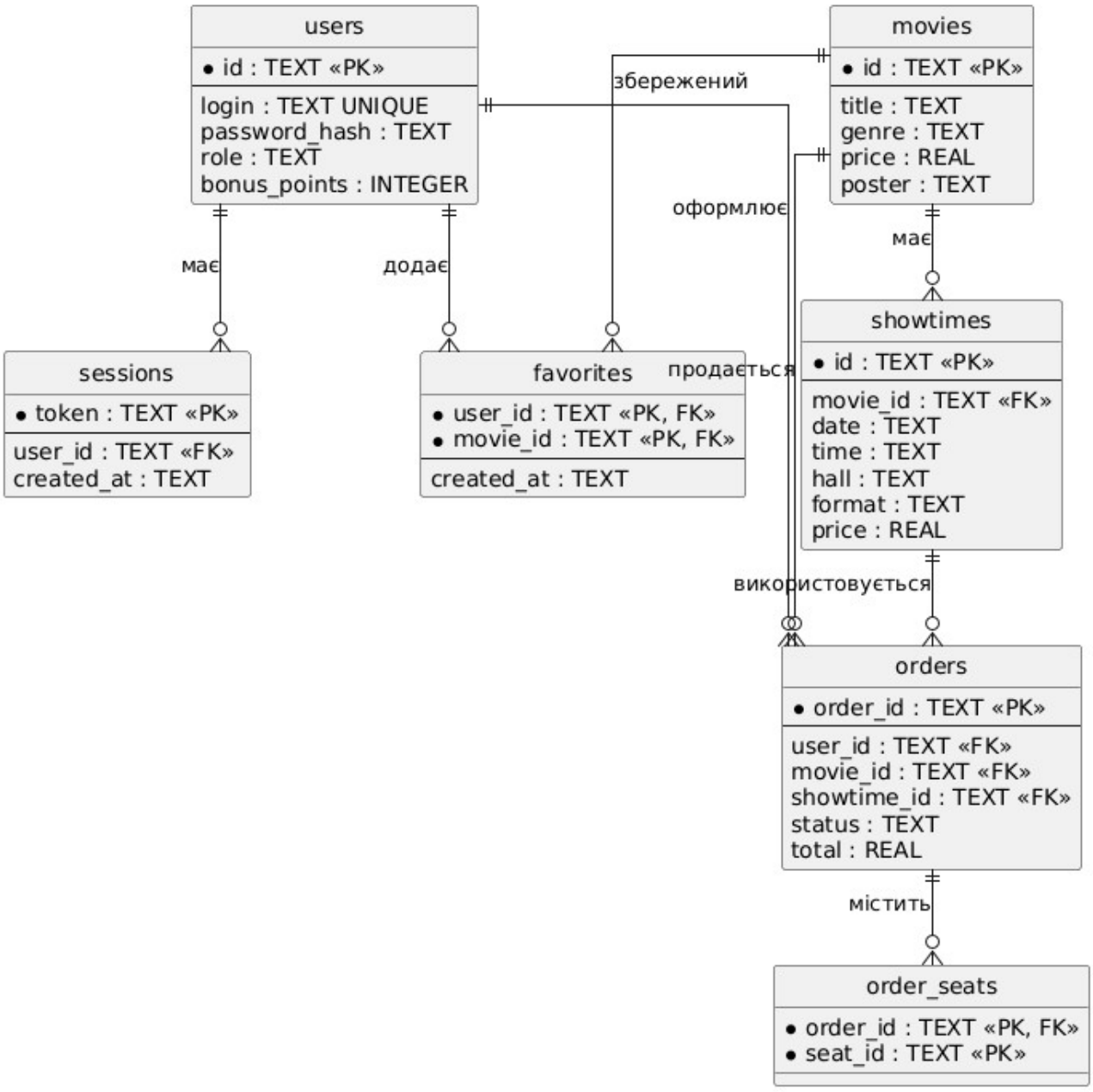

Рис. 2.4. ER-діаграма бази даних вебсервісу «КіноКвиток»

Діаграму компонентів наведено на рис. 2.5. Вона ілюструє розділення відповідальності між інтерфейсом, API, модулями авторизації, замовлень, адміністрування та постійним сховищем даних.

Рис. 2.5. Діаграма компонентів вебсервісу «КіноКвиток»

У результаті проєктування сформовано логічну та фізичну модель вебсервісу. Логічна модель описує ролі, прецеденти та програмні класи, а фізична модель конкретизує структуру бази даних і взаємодію компонентів. Завдяки цьому подальша реалізація виконувалася не як набір окремих сторінок, а як цілісна інформаційна система для продажу квитків у кінотеатрі.

2.2 Опис використовуваних підходів до збору та обробки даних

Збір і обробка даних у вебсервісі організовані навколо основного користувацького сценарію: перегляд афіші, фільтрація фільмів, вибір сеансу, вибір місць, застосування промокоду або бонусів, підтвердження покупки та отримання електронного квитка. Кожний етап супроводжується перевіркою введених даних і оновленням стану інтерфейсу без перезавантаження сторінки.

Такий підхід відповідає логіці SPA, де користувач не переходить між багатьма окремими HTML-сторінками, а працює з єдиним інтерактивним застосунком.

Початкові дані про фільми формуються на сервері під час першого запуску системи. Якщо база ще не містить фільмів, застосунок імпортує початковий набір із seedMovies.json. У цьому наборі зберігаються назви, жанри, тривалість, рік, рейтинг, опис, вікове обмеження, ціна, постер та посилання на трейлер. Надалі ці записи можна змінювати через адміністративну панель, тому файл початкового наповнення виконує роль стартового каталогу, а постійним джерелом актуальних даних стає SQLite-база.

На стороні користувача збір даних починається з фільтрів афіші. Користувач може ввести пошуковий запит, вибрати жанр, вікове обмеження, спосіб сортування та мову інтерфейсу. Пошук виконується за назвою, описом і тематичним мотивом фільму. Фільтр за віковими обмеженнями дає змогу приховати стрічки, що не відповідають обраному рівню. Сортування за новизною, ціною або рейтингом допомагає швидко знайти потрібний варіант.

Мультимовність у проєкті реалізована на рівні клієнтської частини. У React-застосунку визначено словники для української, російської та англійської мов. До них входять не тільки назви кнопок і пунктів меню, а й службові підказки, повідомлення, правила промокодів, назви розділів, тексти сторінок «Про кінотеатр» і «Повернення квитків», а також переклади для карток фільмів, жанрів і описів. Завдяки цьому перемикання мови змінює весь видимий текст інтерфейсу, а не лише окремі елементи навігації.

Обробка даних профілю передбачає кілька рівнів перевірки. Ім’я користувача обмежено двадцятьма символами, email повинен містити латинські символи, знак @ та домен, телефон має відповідати українському формату +380XXXXXXXXX, а аватарка не може перевищувати 10 МБ і повинна бути у форматі PNG, JPG або WEBP. Перевірка реалізована і на клієнтському, і на серверному рівнях. Такий підхід важливий, оскільки клієнтська перевірка підвищує зручність. Серверна перевірка залишається обов’язковою частиною

обробки введення, а клієнтська перевірка використовується для швидкого інформування користувача про помилки форми [16].

Для дати народження використовується календарне поле. Користувач вибирає день, місяць і рік народження, після чого вік автоматично розраховується однією цифрою та відображається поруч у профілі. Така реалізація зменшує кількість помилок, тому що користувач не вводить вік вручну, а система самостійно визначає його на основі поточної дати. HTML-поля вибору дати зручні тим, що нормалізують значення у форматі року, місяця і дня незалежно від локалі браузера [17].

Під час вибору сеансу система обробляє дату, час, формат, базову ціну та список уже зайнятих місць. Для кожного сеансу клієнт запитує у сервера актуальну карту місць. Якщо інший користувач уже купив місце, воно відображається як зайняте і не може бути вибране повторно. Такий механізм особливо важливий для кінотеатру, адже одна й та сама одиниця ресурсу не може бути продана двічі на один сеанс.

На сторінці фільму користувач бачить опис, рейтинг, трейлер, дату, список доступних сеансів, схему зали, VIP-зону, форму покупця, промокод, поле списання бонусів і підсумкову суму. Саме тут відбувається найбільш інтенсивна обробка даних, тому що кожна зміна вибраних місць може змінювати ціну. Якщо серед обраних місць є VIP, для них застосовується підвищений коефіцієнт. Якщо введено промокод, система перевіряє умови його застосування. Якщо користувач списує бонуси, підсумок додатково зменшується в межах допустимого ліміту.

Правила промокодів обробляються сервером, тому їх не можна обійти простим редагуванням інтерфейсу. Для KINO10 сервер перевіряє кількість попередніх замовлень користувача. Для STUDENT перевіряється наявність номера студентського квитка. Для VIP20 підраховується кількість VIP-місць серед вибраних користувачем. Якщо умова не виконана, сервер повертає повідомлення про помилку, а замовлення не створюється. Такий підхід дозволяє підтримувати маркетингові правила надійно та однаково для всіх клієнтів.

Система бонусів також не обмежується візуальним перерахунком. Кількість доступних бонусів береться з профілю користувача, а списання виконується лише після успішного створення замовлення. Якщо квиток скасовано, нараховані за нього бонуси знімаються, а використані бонуси повертаються. У такий спосіб бонусна програма не порушує фінансову логіку замовлення. Вона працює як додатковий шар розрахунку, але зберігає прозорість для електронного чека та адміністративної статистики.

Після підтвердження покупки клієнтська частина надсилає серверу структурований JSON-запит. У ньому передаються ідентифікатор фільму, дата, час, місця, дані покупця, промокод, номер студентського квитка за потреби та кількість бонусів для списання. Сервер перевіряє всі дані, створює замовлення і повертає об’єкт електронного квитка. Для таких операцій у браузері використовується Fetch API, який забезпечує роботу з HTTP-запитами та відповідями без повного перезавантаження сторінки [18].

Окремим результатом обробки даних є електронний квиток. Він містить номер замовлення, фільм, дату, час, зал, місця, суму, покупця, статус, QR-код і електронний чек та інше. QR-код формується на основі даних замовлення, тому може бути використаний для перевірки квитка адміністратором. Також на сторінці передбачено можливість завантаження квитка у форматі PDF.

Система також збирає службові дані для адміністратора. До них належать кількість фільмів, кількість користувачів, кількість замовлень, виручка, продані місця, популярний фільм і список останніх покупок. Ці дані не вводяться вручну, а розраховуються на основі записів у базі. Наприклад, виручка визначається за сумами активних і використаних замовлень, а популярний фільм - за кількістю проданих місць. Завдяки цьому адміністративна панель виконує не тільки функцію редагування, а й функцію оперативного контролю.

Обробка даних у проєкті має подійний характер. Вхід користувача створює сесію, оновлення профілю змінює персональні поля, додавання в обране створює запис у favorites, покупка створює замовлення та місця, скасування змінює статус і повертає бонуси, а перевірка QR-коду дає

адміністратору можливість позначити квиток використаним. Така логіка відповідає реальному життєвому циклу квитка, де важлива не лише фінальна покупка, а й усі проміжні стани.

Значну увагу приділено повідомленням без перезавантаження сторінки. Після успішного входу, збереження профілю, покупки квитка, помилки промокоду або невдалої перевірки введення користувач бачить коротке повідомлення у поточному інтерфейсі. Це підвищує зручність взаємодії, оскільки користувач одразу отримує результат дії. Для сервісу продажу квитків така швидкість реакції є принциповою: користувач повинен розуміти, чи його місця обрані, чи промокод застосовано, чи замовлення створено.

Отже, підходи до збору та обробки даних у вебсервісі поєднують клієнтську інтерактивність, серверну перевірку, транзакційне збереження та постійне оновлення інтерфейсу. Така організація дозволяє підтримувати точність обліку, зменшувати ризик помилок введення і забезпечувати зручний сценарій покупки квитка.

2.3 Реалізація серверної та клієнтської частини вебсервісу

Реалізація вебсервісу для продажу квитків у кінотеатрі виконана на основі клієнт-серверної архітектури. Такий підхід дозволив розділити відповідальність між інтерфейсом користувача, серверною логікою та базою даних. Клієнтська частина відповідає за відображення сторінок, вибір фільму, сеансу та місць у залі, а серверна частина обробляє запити, перевіряє вхідні дані, працює із сесіями користувачів, виконує операції з базою даних і повертає результат у форматі JSON. Завдяки такому поділу структура вебсервісу залишається зрозумілою, а окремі модулі можна змінювати без повної переробки всього проєкту.

У проєкті серверна частина реалізована у файлі server.js. Вона запускає локальний HTTP-сервер, обробляє статичні файли інтерфейсу, відкриває базу даних SQLite через бібліотеку sql.js і формує набір API-маршрутів для роботи з

фільмами, користувачами, замовленнями та адміністративними функціями. Клієнтська частина реалізована у файлі main.jsx і побудована за компонентним принципом React. Стилі інтерфейсу винесено у файл styles.css, що забезпечує єдине оформлення сторінок, карток фільмів, форм, схеми залу, квитків і адміністративних таблиць [19].

Першим етапом роботи серверної частини є ініціалізація бази даних. Під час запуску сервер перевіряє наявність файлу data/cinema.sqlite, відкриває його або створює нову базу, якщо файл ще не існує. У базі формуються таблиці для користувачів, сесій, фільмів, сеансів, замовлень, місць у замовленнях та обраних фільмів. Саме ця частина відповідає за фізичне збереження інформації після перезапуску сайту. Фрагмент коду, який відповідає за створення структур бази даних та підготовку сховища, наведено на рисунку 2.6 [20].

```
function initDb() {
  db.exec(`
    PRAGMA foreign_keys = ON;
    CREATE TABLE IF NOT EXISTS users (id TEXT PRIMARY KEY, login TEXT NOT NULL UNIQUE, password_hash TEXT NOT NULL, role TEXT NOT NULL DEFAULT 'u
    CREATE TABLE IF NOT EXISTS sessions (token TEXT PRIMARY KEY, user_id TEXT NOT NULL REFERENCES users(id) ON DELETE CASCADE, created_at TEXT NO
    CREATE TABLE IF NOT EXISTS movies (id TEXT PRIMARY KEY, title TEXT NOT NULL, genre TEXT NOT NULL, duration TEXT NOT NULL, rating TEXT NOT NUL
    CREATE TABLE IF NOT EXISTS orders (order_id TEXT PRIMARY KEY, user_id TEXT NOT NULL REFERENCES users(id) ON DELETE CASCADE, movie_id TEXT NOT
    CREATE TABLE IF NOT EXISTS order_seats (order_id TEXT NOT NULL REFERENCES orders(order_id) ON DELETE CASCADE, seat_id TEXT NOT NULL, PRIMARY
    CREATE TABLE IF NOT EXISTS favorites (user_id TEXT NOT NULL REFERENCES users(id) ON DELETE CASCADE, movie_id TEXT NOT NULL REFERENCES movies(
    CREATE TABLE IF NOT EXISTS showtimes (id TEXT PRIMARY KEY, movie_id TEXT NOT NULL REFERENCES movies(id) ON DELETE CASCADE, date TEXT NOT NULL
    CREATE INDEX IF NOT EXISTS idx_orders_show ON orders(movie_id, date, time);
    CREATE INDEX IF NOT EXISTS idx_showtimes_movie_date ON showtimes(movie_id, date);
  `);
```

Рис. 2.6. Фрагмент коду, який відповідає за створення таблиць у базі даних

Наведений фрагмент показує, що система зберігає не лише загальні відомості про фільми, а й операційні дані, необхідні для продажу квитків. До таких даних належать сесії користувачів, конкретні сеанси, замовлення і місця, пов’язані з кожним замовленням. Така структура важлива для коректного контролю зайнятості місць, оскільки продане місце має бути пов’язане не лише з фільмом, а й з конкретною датою, часом і сеансом.

Після ініціалізації бази даних важливим елементом серверної частини є автентифікація. У вебсервісі передбачено реєстрацію, вхід, вихід із системи та перевірку поточного користувача. Під час реєстрації сервер приймає логін, пароль і персональні дані, виконує базову перевірку коректності введення,

хешує пароль і створює запис користувача у таблиці users. Під час входу система порівнює введені дані з даними в базі, формує токен сесії та зберігає його в таблиці sessions. Фрагмент коду, що реалізує авторизацію користувача показано на рисунку 2.7 [21].

```
if (req.method === 'POST' && parsed.pathname === '/api/login') {
  const login = String(body.login || '').trim();
  const user = db.prepare('SELECT * FROM users WHERE lower(login)=lower(?) AND password_hash = ?').get(login, hashPassword(String(body.passwo
  if (!user) return json(res, 401, { error: 'Невірний логін або пароль' });
  const token = newToken();
  db.prepare('INSERT INTO sessions (token, user_id, created_at) VALUES (?, ?, ?)').run(token, user.id, new Date().toISOString());
  setSessionCookie(res, token);
  return json(res, 200, userToApi(user));
}
```

Рис. 2.7. Фрагмент коду, що реалізує авторизацію користувача

Застосування сесій дозволяє не передавати логін і пароль під час кожної дії користувача. Після успішного входу браузер отримує cookie із сесійним токеном, а сервер під час захищених запитів визначає користувача за цим токеном. Такий механізм використовується для доступу до профілю, перегляду квитків, додавання фільмів в обране та оформлення замовлення. Окремо перевіряється роль користувача, оскільки адміністративні маршрути повинні бути доступні лише адміністратору [22].

Клієнтська частина вебсервісу організована таким чином, щоб користувач міг пройти основний сценарій без зайвого перезавантаження сторінки. На сторінці фільму користувач обирає дату, сеанс і місця в залі. У компоненті зберігається поточний стан вибраних місць, промокоду, бонусів і підсумкової вартості. Після натискання кнопки купівлі формується об’єкт замовлення, який надсилається на сервер через POST-запит. Фрагмент коду клієнтської частини, що відповідає за підготовку та відправлення замовлення, показано на рисунку 2.8 [23].

У цьому фрагменті відображено зв’язок між інтерфейсом і серверною частиною. Користувач взаємодіє з візуальними елементами, але фактичне створення квитка не відбувається на клієнті. Клієнт лише збирає необхідні дані та передає їх серверу. Такий підхід є коректним, оскільки остаточна перевірка

доступності місць, розрахунок вартості, застосування промокоду та створення записів у базі повинні виконуватися на сервері. Це зменшує ризик некоректного замовлення і запобігає зміні критичних даних лише на стороні браузера.

```
if (req.method === 'POST' && parsed.pathname === '/api/orders') {
  const user = requireUser(req, res); if (!user) return;
  const movie = db.prepare('SELECT * FROM movies WHERE id = ?').get(body.movieId);
  const date = String(body.date || ''); const time = String(body.time || '');
  const seats = Array.isArray(body.seats) ? [...new Set(body.seats.map(String))] : [];
  if (!movie) return json(res, 400, { error: 'Фільм не знайдено' });
  const selectedShowtime = findSelectedShowtime(movie, date, time, String(body.showtimeId || ''));
  if (!isValidDate(date) || !selectedShowtime || seats.length === 0) return json(res, 400, { error: 'Оберіть дату, сеанс і місця' });
  const taken = new Set(db.prepare(`SELECT os.seat_id FROM order_seats os JOIN orders o ON o.order_id = os.order_id WHERE o.movie_id = ? AND
  const conflict = seats.filter(seat => taken.has(seat));
  if (conflict.length) return json(res, 409, { error: 'Деякі місця вже зайняті', conflict });
  const buyer = body.buyer || {};
  const orderId = `KT-${Date.now().toString(36).toUpperCase()}`;
  const buyerName = String(buyer.name || user.name || user.login).trim();
  const buyerEmail = String(buyer.email || user.email || '').trim();
  const buyerPhone = String(buyer.phone || user.phone || '').trim();
  const buyerError = validatePersonData({ name: buyerName, email: buyerEmail, phone: buyerPhone }, true);
  if (buyerError) return json(res, 400, { error: buyerError });
  const createdAt = new Date().toISOString();
  const basePrice = Number(selectedShowtime.price || movie.price);
  const subtotal = seats.reduce((sum, seat) => sum + seatPrice(basePrice, seat), 0);
  const previousOrders = Number(db.prepare('SELECT COUNT(*) AS count FROM orders WHERE user_id = ?').get(user.id).count || 0);
  const promo = promoDiscount(body.promoCode, subtotal, { previousOrders, studentId: body.studentId, vipSeats: seats.filter(isVipSeat).length
  if (promo.error) return json(res, 400, { error: promo.error });
  const amountAfterPromo = Math.max(0, subtotal - promo.discount);
  const bonusRedeem = calcBonusRedemption(user, body.bonusUse, amountAfterPromo);
```

Рис. 2.8. Фрагмент коду клієнтської частини, що відповідає за підготовку та відправлення замовлення

На сервері запит на створення замовлення проходить кілька етапів перевірки. Спочатку визначається поточний користувач, далі перевіряється наявність фільму, правильність вибраної дати й часу, формат сеансу та список місць. Після цього сервер звертається до таблиць orders і order_seats, щоб визначити, чи не були вибрані місця вже продані або використані в іншому активному замовленні. Фрагмент коду серверної частини, що відповідає за перевірку місць і підготовку замовлення наведено на рисунку 2.9 [24].

Перевірка зайнятості місць є однією з ключових частин вебсервісу. Без неї два користувачі могли б одночасно оформити квиток на одне й те саме місце. У реалізованій логіці сервер порівнює новий запит із уже створеними замовленнями та не дозволяє продати місце повторно, якщо воно пов’язане з активним або використаним квитком. Також на цьому етапі перевіряються промокоди, кількість бонусів користувача та тип місця, оскільки VIP-місця мають вищу вартість.

```
if (!movie) return json(res, 400, { error: 'Фільм не знайдено' });
const selectedShowtime = findSelectedShowtime(movie, date, time, String(body.showtimeId || ''));
if (!isValidDate(date) || !selectedShowtime || seats.length === 0) return json(res, 400, { error: 'Оберіть дату, сеанс і місця' });
const taken = new Set(db.prepare(`SELECT os.seat_id FROM order_seats os JOIN orders o ON o.order_id = os.order_id WHERE o.movie_id = ? AND
const conflict = seats.filter(seat => taken.has(seat));
if (conflict.length) return json(res, 409, { error: 'Деякі місця вже зайняті', conflict });
const buyer = body.buyer || {};
const orderId = `KT-${Date.now().toString(36).toUpperCase()}`;
const buyerName = String(buyer.name || user.name || user.login).trim();
const buyerEmail = String(buyer.email || user.email || '').trim();
const buyerPhone = String(buyer.phone || user.phone || '').trim();
const buyerError = validatePersonData({ name: buyerName, email: buyerEmail, phone: buyerPhone }, true);
if (buyerError) return json(res, 400, { error: buyerError });
const createdAt = new Date().toISOString();
const basePrice = Number(selectedShowtime.price || movie.price);
const subtotal = seats.reduce((sum, seat) => sum + seatPrice(basePrice, seat), 0);
const previousOrders = Number(db.prepare('SELECT COUNT(*) AS count FROM orders WHERE user_id = ?').get(user.id).count || 0);
const promo = promoDiscount(body.promoCode, subtotal, { previousOrders, studentId: body.studentId, vipSeats: seats.filter(isVipSeat).length
if (promo.error) return json(res, 400, { error: promo.error });
const amountAfterPromo = Math.max(0, subtotal - promo.discount);
const bonusRedeem = calcBonusRedemption(user, body.bonusUse, amountAfterPromo);
const total = Math.max(0, amountAfterPromo - bonusRedeem.discount);
const bonusPoints = Math.max(0, Math.floor(total / 10));
```

Рис. 2.9. Фрагмент коду серверної частини, що відповідає за перевірку місць і підготовку замовлення

Після успішної перевірки сервер переходить до створення електронного квитка. Для цього використовується транзакційний підхід: спочатку створюється запис у таблиці orders, потім до таблиці order_seats додаються всі вибрані місця, після чого зміни підтверджуються. Якщо на будь-якому етапі виникає помилка, транзакція скасовується і база даних не залишається у частково зміненому стані. Фрагмент коду, що відповідає за транзакційне створення замовлення представлений на рисунку 2.10.

```
try {
  db.exec('BEGIN IMMEDIATE');
  db.prepare(`INSERT INTO orders (order_id,user_id,movie_id,movie_title,showtime_id,date,time,hall,format,subtotal,discount,promo_code,bonu
  const stmt = db.prepare('INSERT INTO order_seats (order_id, seat_id) VALUES (?, ?)');
  for (const seat of seats) stmt.run(orderId, seat);
  db.prepare('UPDATE users SET bonus_points = MAX(0, COALESCE(bonus_points, 0) - ?) + ? WHERE id = ?').run(bonusRedeem.points, bonusPoints,
  db.exec('COMMIT');
} catch (err) { db.exec('ROLLBACK'); return json(res, 500, { error: err.message }); }
return json(res, 200, orderToApi(db.prepare('SELECT * FROM orders WHERE order_id = ?').get(orderId)));
}
```

Рис. 2.10. Фрагмент коду, що відповідає за транзакційне створення замовлення

Транзакційна логіка особливо важлива для системи продажу квитків, оскільки замовлення складається з кількох пов’язаних записів. Якщо створити лише запис замовлення без місць або навпаки, система втратить цілісність.

Використання послідовності BEGIN, INSERT, COMMIT і ROLLBACK дозволяє гарантувати, що квиток буде створений повністю або не буде створений взагалі. Це підвищує надійність вебсервісу та забезпечує коректність даних у базі.

Завершальним елементом реалізації серверної частини є адміністративна панель, яка охоплює не лише перевірку окремого квитка, а весь службовий контур керування вебсервісом. Через адміністративні маршрути сервер повертає списки фільмів, користувачів, замовлень і сеансів, формує статистичні показники, а також забезпечує додавання, редагування та видалення фільмів і керування розкладом показів. Фрагмент коду, який відповідає за CRUD-операції адміністративної панелі, наведено на рисунку 2.11.

```
if (req.method === 'POST' && parsed.pathname === '/api/admin/movies') {
  const user = requireAdmin(req, res); if (!user) return;
  const movie = sanitizeMovie(body); if (movie.error) return json(res, 400, { error: movie.error });
  insertMovie(movie); return json(res, 200, movieRowToApi(db.prepare('SELECT * FROM movies WHERE id = ?').get(movie.id)));
}
if ((params = route(null, info, '/api/admin/movies/:id')) && req.method === 'PUT') {
  const user = requireAdmin(req, res); if (!user) return;
  const existing = db.prepare('SELECT * FROM movies WHERE id = ?').get(params.id);
  if (!existing) return json(res, 404, { error: 'Фільм не знайдено' });
  const movie = sanitizeMovie(body, existing); if (movie.error) return json(res, 400, { error: movie.error });
  updateMovie(params.id, movie); return json(res, 200, movieRowToApi(db.prepare('SELECT * FROM movies WHERE id = ?').get(params.id)));
}
if ((params = route(null, info, '/api/admin/movies/:id')) && req.method === 'DELETE') {
  const user = requireAdmin(req, res); if (!user) return;
  const existing = db.prepare('SELECT * FROM movies WHERE id = ?').get(params.id);
  if (!existing) return json(res, 404, { error: 'Фільм не знайдено' });
```

Рис. 2.11. Фрагмент коду, який відповідає за CRUD-операції адміністративної панелі

Наведений фрагмент показує, що адміністративна панель побудована як набір захищених серверних операцій. Спочатку виконується перевірка ролі користувача, після чого адміністратор отримує доступ до даних для таблиць і статистики, може додавати нові сеанси, оновлювати інформацію про фільми та видаляти записи, які втратили актуальність. Такий підхід дозволяє керувати головним функціоналом вебсервісу без прямого редагування бази даних.

З практичної точки зору адміністративна частина доповнює користувацький сценарій і забезпечує супровід роботи сервісу після його запуску. Якщо користувач працює з афішею, вибором місць і квитками, то адміністратор відповідає за актуальність контенту, розкладу, статусів замовлень

і загальної статистики. Завдяки цьому вебсервіс може використовуватися не лише як демонстраційна сторінка, а як керована інформаційна система для продажу квитків.

Клієнтська частина також підтримує відображення результатів серверних операцій. Якщо замовлення створено успішно, користувач бачить повідомлення про готовий квиток і може перейти до розділу власних квитків. Якщо виникає помилка, наприклад не вибрано місця, користувач не авторизований або місце вже зайняте, інтерфейс показує відповідне повідомлення. Це покращує зрозумілість взаємодії та допомагає користувачу виправити дію без звернення до адміністратора.

Отже, реалізація серверної та клієнтської частин вебсервісу побудована як послідовний процес взаємодії між інтерфейсом, API та базою даних. База даних забезпечує збереження інформації, авторизація визначає права користувача, клієнтська частина формує запит на купівлю, сервер перевіряє його коректність, транзакція створює квиток, а адміністративна частина дає змогу перевірити результат. Така структура відповідає логіці реального вебсервісу продажу квитків у кінотеатрі та створює основу для подальшого розширення функціоналу.

Висновки до розділу 2

Під час проєктування вебсервісу було визначено його основну структуру, ролі користувачів та логіку взаємодії між окремими частинами системи. Створені UML-діаграми дали змогу наочно показати, як відвідувач і адміністратор працюють із вебресурсом, які сутності використовуються в системі, як відбувається оформлення квитка та яким чином клієнтська частина пов’язана із сервером і базою даних. Це дозволило сформувати не окремий набір сторінок, а цілісну модель вебсервісу.

Основою роботи системи є продумана організація даних. У вебсервісі передбачено збереження інформації про користувачів, фільми, сеанси, місця, замовлення, електронні квитки, обрані фільми, промокоди та бонуси. Такий підхід забезпечує логічний зв’язок між діями користувача і записами в базі даних. Особливо важливою є перевірка доступності місць, оскільки вона гарантує, що одне й те саме місце не буде продане кільком користувачам одночасно.

Реалізація серверної та клієнтської частин підтвердила правильність обраної архітектури. Користувач отримує зручний інтерфейс для перегляду афіші, вибору сеансу, місць і оформлення квитка, а сервер забезпечує авторизацію, обробку запитів, перевірку даних, створення замовлень і роботу адміністративної панелі. У результаті було створено функціональну основу вебсервісу, яка може використовуватися для автоматизації продажу квитків у кінотеатрі та подальшого розширення системи.

РОЗДІЛ 3
ФУНКЦІОНУВАННЯ ТА ЕЛЕМЕНТИ ІНТЕРФЕЙСУ ВЕБСЕРВІСУ ДЛЯ ПРОДАЖУ КВИТКІВ У КІНОТЕАТРІ

3.1 Огляд функціоналу розробленого вебсервісу

Розроблений вебсервіс «КіноКвиток» функціонує як цілісний онлайн-додаток для продажу квитків у кінотеатрі. Його призначення полягає у поєднанні кількох взаємопов’язаних процесів: перегляду афіші, ознайомлення з інформацією про фільм, вибору сеансу, вибору місць, розрахунку вартості, оформлення замовлення, отримання електронного квитка та подальшого адміністративного контролю. Функціональність сервісу не обмежується тільки кнопкою купівлі, оскільки реальний процес продажу квитка потребує роботи з користувацькими профілями, історією покупок, статусами замовлень і службовими діями адміністратора.

Функціональна структура побудована за клієнт-серверним принципом. Клієнтська частина відповідає за відображення сторінок, форм, карток фільмів, схем місць і повідомлень користувачу. Серверна частина обробляє запити, перевіряє коректність даних, працює з базою SQLite, створює замовлення, контролює доступ до адміністративних функцій і повертає оновлені дані у форматі JSON. Такий поділ відповідає сучасній практиці побудови вебзастосунків, де інтерфейс і бізнес-логіка мають чітко визначені зони відповідальності [25].

Основним публічним модулем є каталог фільмів. У ньому представлено сорок фільмів з постерами, назвами, жанрами, роком виходу, тривалістю, рейтингом, віковим обмеженням і базовою ціною. Користувач може застосовувати пошук, фільтрувати записи за жанром та віком, а також змінювати сортування за новизною, ціною або рейтингом. Цей блок виконує роль точки входу до сервісу, але не перевантажує користувача зайвими даними.

Картка фільму містить тільки ту інформацію, яка потрібна для попереднього вибору, а детальніший опис відкривається на окремій сторінці.

Детальна сторінка фільму містить постер, жанр, вікове обмеження, рейтинг, опис, мотив, трейлер, блок оцінок глядачів, дату, список доступних сеансів, формат показу, зал, схему місць, форму покупця, промокод, бонусне списання та підсумкову суму. Така концентрація елементів на одній сторінці зроблена для скорочення кількості переходів. Користувач не повинен відкривати окрему сторінку для опису, окрему для сеансів і окрему для місць. Усі основні рішення він приймає в одному сценарії, що відповідає компонентній логіці побудови інтерфейсу [26].

Окремий функціональний блок становить обліковий запис користувача. Система підтримує реєстрацію, вхід, вихід, редагування профілю, завантаження аватарки, перегляд бонусів, історію покупок і список обраних фільмів. Наявність профілю важлива не тільки для зручності користувача, а й для коректного збереження історії замовлень. Через профіль користувач може повторно відкрити електронний квиток, перевірити статус покупки, переглянути нараховані бонуси або змінити контактні дані.

Форма реєстрації містить поля для імені користувача, логіна, електронної пошти, телефону та пароля. Для імені передбачено обмеження довжини, для телефону - формат +380XXXXXXXXX, для email - перевірку на наявність коректної адреси. Такі обмеження зменшують кількість помилкових записів у базі даних і спрощують подальше використання контактної інформації. Реалізація форм спирається на стандартні можливості HTML і клієнтську перевірку введення, що відповідає рекомендаціям щодо валідації вебформ [27].

Крім реєстрації, у системі передбачено відновлення пароля. Ця можливість не є центральною для продажу квитків, але вона суттєво підвищує завершеність сервісу. У реальному використанні користувач може втратити доступ до облікового запису, тому відсутність механізму відновлення призводила б до втрати історії покупок, бонусів та обраних фільмів. У

розробленому проєкті відновлення реалізовано через введення логіна або email та нового пароля.

Вебсервіс також містить інформаційні сторінки. Сторінка «Про кінотеатр» пояснює призначення сервісу, містить контактні дані, відомості про програму лояльності, опис VIP-місць і короткий перелік можливостей. Сторінка правил повернення описує умови скасування квитка до початку сеансу, поведінку системи після початку показу та структуру електронного чека. Ці сторінки не виконують операцію продажу безпосередньо, але підвищують прозорість сервісу і зменшують кількість питань користувача перед покупкою.

Адміністративна частина призначена для службового контролю. Вона дозволяє переглядати статистику, додавати й редагувати фільми, створювати сеанси, переглядати замовлення, змінювати статуси квитків, видаляти записи та контролювати користувачів. Наявність такого модуля робить сервіс двостороннім: з одного боку він обслуговує покупця, а з іншого - надає адміністрації інструменти для підтримання актуальності афіші та обробки нестандартних ситуацій.

Отже, функціонал «КіноКвитка» охоплює повний цикл роботи з квитком: від реєстрації та вибору фільму до створення замовлення, отримання QR-квитка, збереження історії і службового контролю. Кожна функція пов’язана з конкретним етапом взаємодії користувача або адміністратора, тому додаток сприймається як єдина система, а не як набір розрізнених сторінок.

3.2 Опис взаємодії користувача з розробленим вебінтерфейсом

Взаємодія користувача з вебсервісом «КіноКвиток» побудована як послідовний сценарій, у межах якого відвідувач поступово переходить від ознайомлення з ресурсом до вибору фільму, перегляду інформації про нього, вибору сеансу, бронювання місць та оформлення квитка. Така логіка відповідає реальному процесу відвідування кінотеатру, але переносить його в онлайн-

середовище: користувач не звертається до каси, а самостійно виконує всі основні дії через вебінтерфейс. [28]

Перший контакт із системою відбувається через головну сторінку. Її призначення полягає в тому, щоб одразу показати користувачеві тематику сервісу, основні розділи та найкоротший шлях до вибору фільму. У верхній частині інтерфейсу розміщено логотип «КіноКвиток» і навігаційне меню, через яке можна перейти на головну сторінку, до каталогу фільмів, інформації про кінотеатр, правил повернення квитків, входу або реєстрації. Праворуч передбачено перемикач мови та кнопку зміни теми, тому користувач може адаптувати інтерфейс під власні потреби ще до початку покупки. [29]

Центральна частина головної сторінки містить назву сервісу, коротке пояснення його призначення та дві основні кнопки дії. Кнопка «Обрати фільм» переводить користувача безпосередньо до каталогу афіші, а кнопка «Про кінотеатр» відкриває інформаційну сторінку з контактами та описом можливостей сервісу. Отже, вже на першому екрані користувач отримує зрозумілий вибір: одразу перейти до покупки або спочатку ознайомитися з кінотеатром [30]. Зовнішній вигляд головної сторінки наведено на рисунку 3.1.

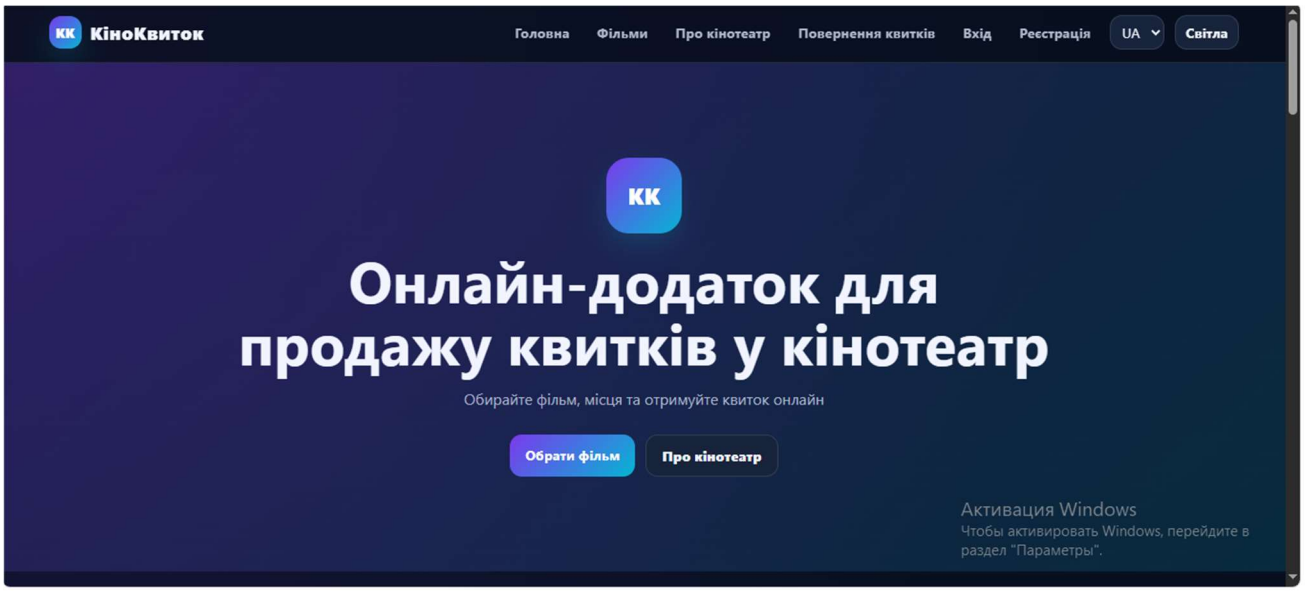

Рис. 3.1. Головна сторінка вебсервісу «КіноКвиток»

Інформаційна сторінка «Про кінотеатр» призначена для користувачів, які хочуть ознайомитися з сервісом перед покупкою. Вона містить короткий опис «КіноКвитка», контакти, графік роботи, програму лояльності та перелік можливостей вебсервісу. Завдяки цьому сторінка виконує не лише довідкову, а й довірчу функцію: користувач бачить, що сервіс має зрозумілу структуру, правила роботи та контактні дані.

Блок контактів містить адресу, графік роботи, телефон та email. Це важливо для ситуацій, коли користувач хоче уточнити інформацію щодо сеансу або звернутися до кінотеатру. Блок програми лояльності пояснює принцип бонусів і VIP-місць, а блок можливостей сервісу підкреслює, що користувач може працювати з профілем, історією покупок, QR-квитками, електронним чеком, промокодами, фільтрами, темою інтерфейсу та адаптацією під мобільні пристрої. Загальний вигляд цієї сторінки наведено на рисунку 3.2.

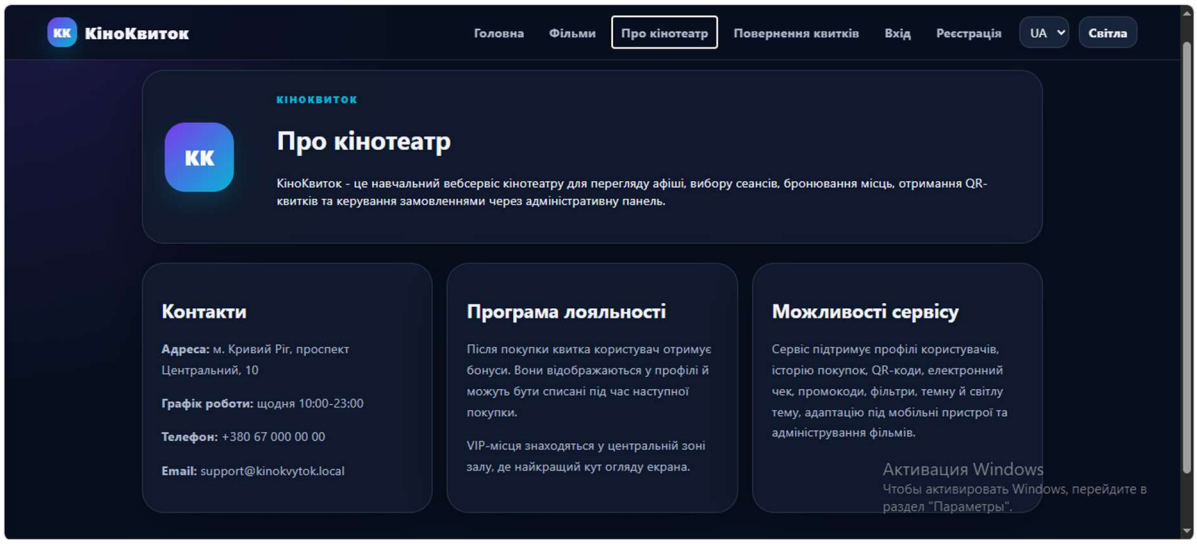

Рис. 3.2. Інформаційна сторінка «Про кінотеатр»

Для повноцінної роботи з сервісом користувач може створити обліковий запис. Реєстрація потрібна для збереження історії покупок, накопичення бонусів, використання профілю та доступу до власних квитків. На сторінці реєстрації користувач вводить ім’я, логін, електронну пошту, телефон і пароль. Поля розміщені у зрозумілій послідовності, тому форма не виглядає

перевантаженою. Після заповнення даних користувач натискає кнопку реєстрації, а система створює новий профіль.

З практичної точки зору реєстрація пов’язує подальші покупки з конкретним користувачем. Це означає, що після входу в акаунт користувач може переглядати свої квитки, бачити історію замовлень, користуватися бонусами та швидше оформлювати наступні покупки. Форма реєстрації також важлива для адміністративної частини, оскільки адміністратор може бачити кількість користувачів і загальну активність сервісу. Вигляд сторінки реєстрації наведено на рисунку 3.3.

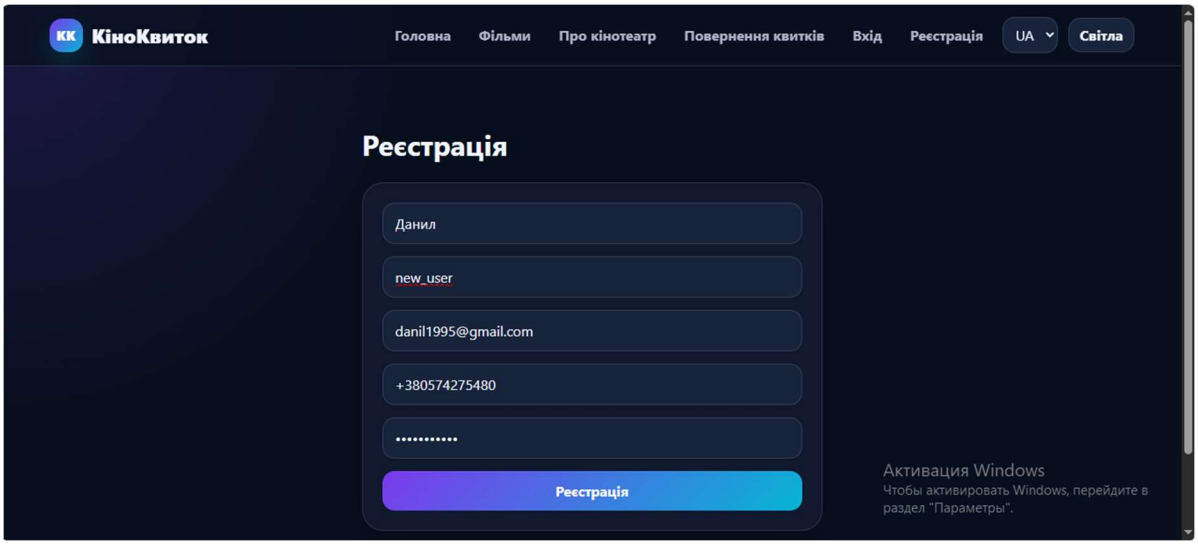

Рис. 3.3. Форма реєстрації користувача

Якщо користувач забув пароль або втратив доступ до облікового запису, він може скористатися сторінкою відновлення пароля. У наведеному інтерфейсі користувач вводить email та новий пароль. Наявність окремої сторінки для цієї операції робить сервіс зручнішим, оскільки користувач не змушений створювати новий акаунт або звертатися до адміністратора для відновлення доступу.

Форма відновлення пароля виконана мінімалістично: користувач бачить лише ті поля, які необхідні для виконання дії [31]. Піктограма біля пароля дозволяє контролювати введення, а кнопка «Змінити пароль» запускає

оновлення облікових даних. Після успішної зміни користувач може повернутися до входу й продовжити роботу з квитками, бонусами та історією покупок. Сторінку відновлення пароля показано на рисунку 3.4.

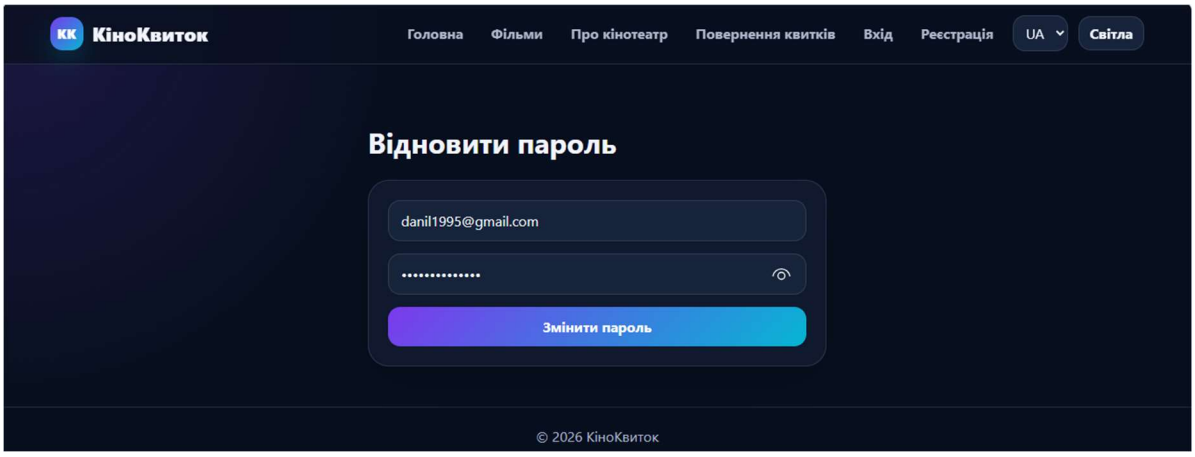

Рис. 3.4. Сторінка відновлення пароля користувача

Після натискання кнопки вибору фільму або переходу через пункт меню користувач потрапляє до каталогу. У цьому розділі представлено афішу фільмів у вигляді карток із постерами. Каталог виконує не лише інформаційну функцію, а й навігаційну, оскільки саме з нього починається вибір конкретного фільму. Користувач може візуально оцінити доступні варіанти за постерами, а також скористатися інструментами пошуку й фільтрації.

У верхній частині каталогу розміщено поле пошуку, випадаючий список жанрів, фільтр за віковими обмеженнями та сортування. Поле пошуку дає змогу швидко знайти фільм за назвою або частиною назви. Фільтр за жанром допомагає звузити список, якщо користувач шукає, наприклад, фантастику, драму, комедію або анімацію. Вікове обмеження дозволяє залишити у списку лише ті фільми, які відповідають потрібній категорії. Сортування за новизною, ціною або рейтингом робить перегляд афіші зручнішим, особливо за наявності великої кількості фільмів.

На кожній картці також передбачено кнопку додавання до обраного. Ця функція корисна для користувача, який ще не готовий купувати квиток, але хоче

зберегти фільм для подальшого перегляду. Для переходу до детальної сторінки достатньо натиснути на картку або постер фільму. Таким чином каталог виконує роль основного вузла, через який користувач обирає напрям подальшої взаємодії з вебсервісом. Приклад сторінки каталогу наведено на рисунку 3.5.

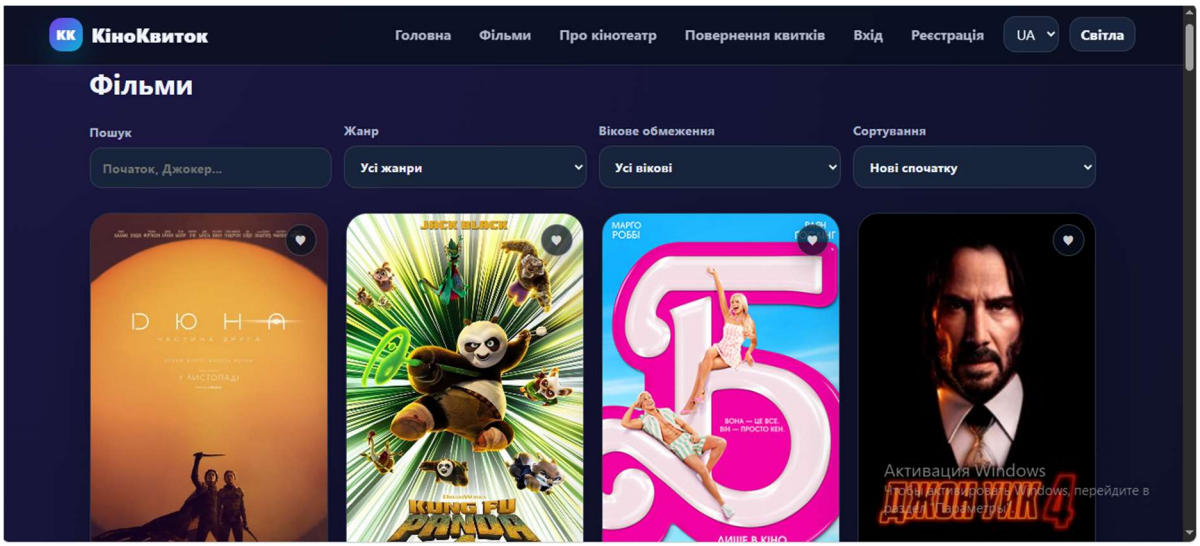

Рис. 3.5. Каталог фільмів із пошуком, фільтрацією та сортуванням

Після вибору фільму відкривається сторінка з детальною інформацією. На ній користувач отримує розширене уявлення про фільм перед оформленням квитка. Важливим елементом є блок трейлера, який дозволяє переглянути відео без переходу на сторонні сторінки. Це зручно, оскільки користувач залишається в межах вебсервісу й одразу після перегляду може перейти до вибору сеансу.

Нижче або поруч із трейлером відображається блок оцінок глядачів. Він містить загальну оцінку та окремі відгуки, що допомагають користувачеві прийняти рішення. У сценарії використання це важливо, тому що користувач може не знати, який фільм обрати, і додаткова інформація у вигляді трейлера, рейтингу та коротких коментарів зменшує невизначеність. Після ознайомлення з цією інформацією користувач переходить до нижньої частини сторінки, де розміщено вибір дати, сеансу та місць. Приклад сторінки фільму з трейлером і оцінками наведено на рисунку 3.6.

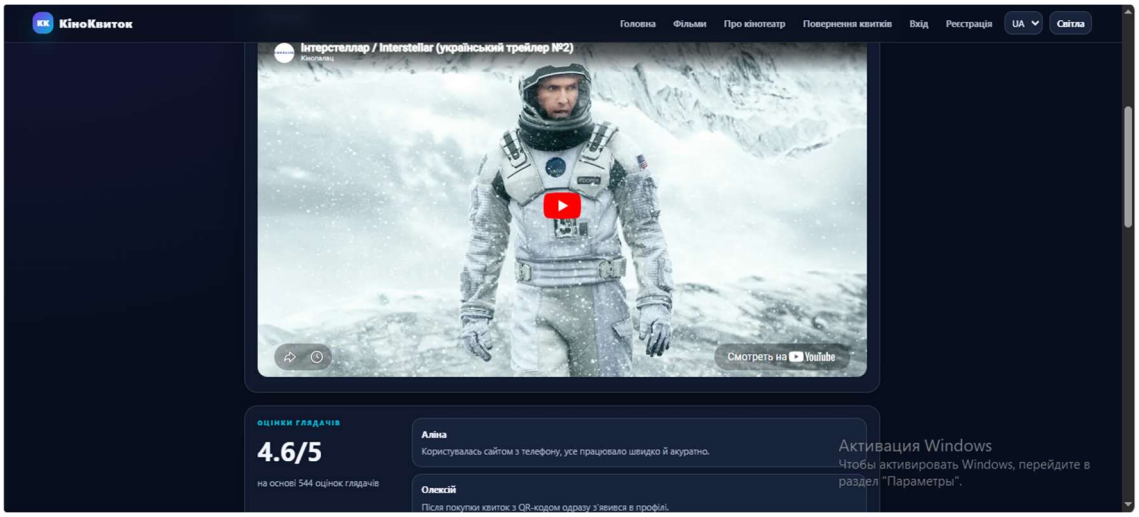

Рис. 3.6 Сторінка фільму з трейлером та оцінками глядачів

Основна дія користувача відбувається на етапі вибору сеансу та місць. Спочатку користувач обирає дату показу, після чого вибирає доступний сеанс із зазначенням часу, формату, залу та ціни. Така структура дозволяє не показувати зайву інформацію одразу, а поступово вести користувача до потрібного рішення. Якщо фільм має кілька сеансів, користувач може порівняти їх і вибрати найзручніший час.

Після вибору сеансу система показує схему залу. Вона оформлена у вигляді рядів і місць, де кожне місце є окремим інтерактивним елементом. Над схемою розміщено позначення екрана, тому користувач розуміє, як місця розташовані відносно нього. Легенда пояснює стани місць: вільні, VIP, обрані та зайняті. VIP-місця виділені окремим кольором і розташовані в центральній зоні, оскільки вони мають кращий огляд і вищу вартість. Під час натискання на місце воно переходить у стан обраного, а підсумкова сума автоматично змінюється.

Під схемою залу розміщено форму покупки квитка. Користувач вводить ім’я, електронну пошту та номер телефону, які потрібні для формування електронного квитка. Додатково доступне поле промокоду, де можна ввести один із підтримуваних кодів, наприклад KINO10, STUDENT або VIP20. Поруч розміщене поле списання бонусів. Сервіс одразу показує проміжну суму,

знижку, бонуси за покупку та кінцеву вартість. Після перевірки всіх даних користувач натискає кнопку «Купити квиток», після чого запит передається на сервер для остаточної перевірки та створення замовлення [32]. Інтерфейс вибору місць і оформлення квитка наведено на рисунку 3.7.

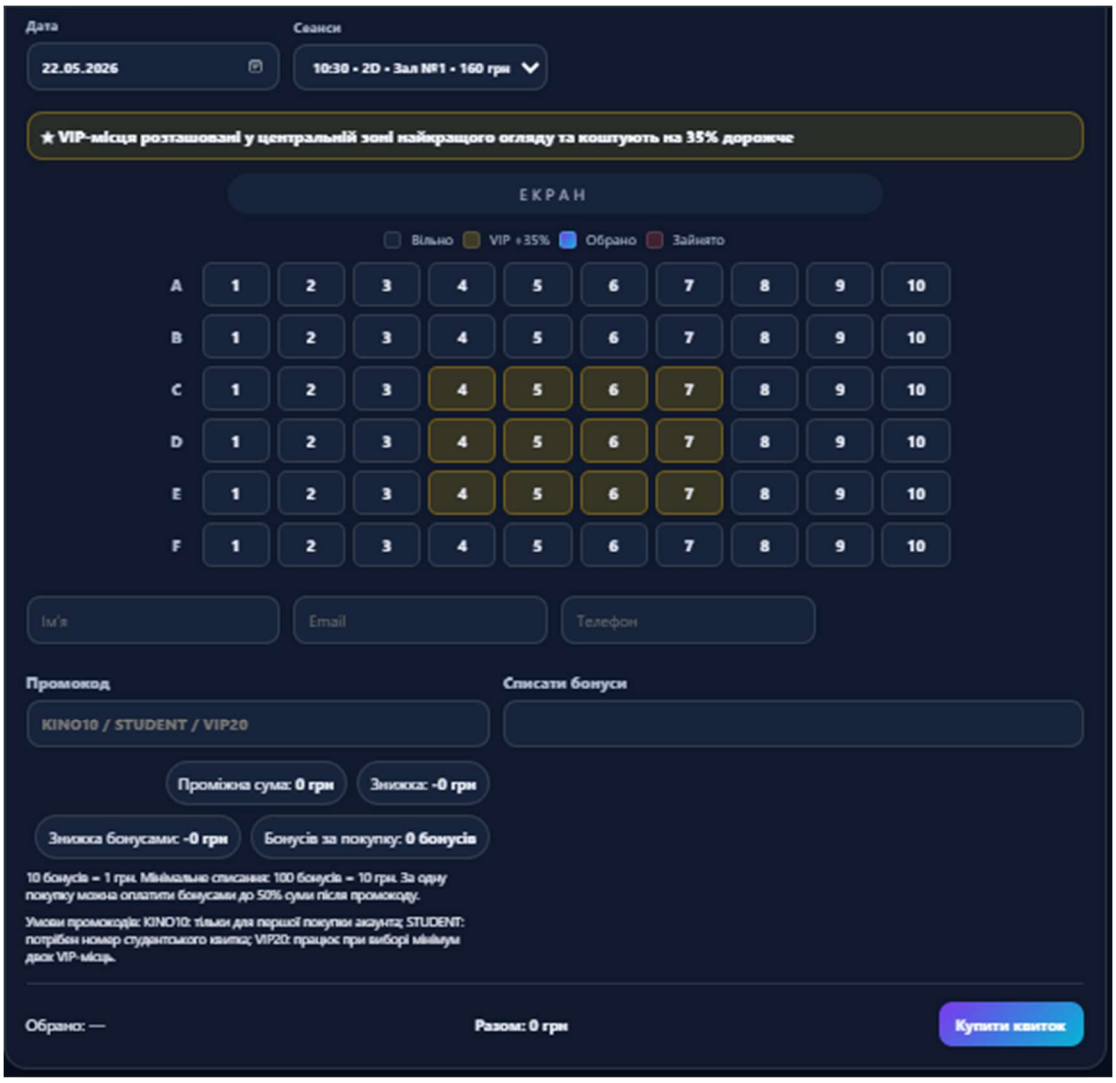

Рис. 3.7. Вибір місць, застосування промокоду та оформлення квитка

Завершальним етапом користувацької взаємодії є перегляд електронного квитка в розділі «Мої квитки». Після успішного оформлення замовлення система зберігає покупку в профілі користувача, тому він може повторно відкрити квиток у будь-який момент без повторного введення даних. У цьому блоці користувач бачить назву фільму, дату й час сеансу, номер залу, формат

показу, вибрані місця та ім’я покупця. Приклад сторінки електронного квитка наведено на рисунку 3.8.

Рис. 3.8. Електронний квиток користувача з QR-кодом та електронним чеком

У правій частині квитка розміщено QR-код, який пов’язаний із конкретним замовленням [33]. Завдяки цьому квиток можна швидко перевірити під час входу до зали або через адміністративну частину вебсервісу. Поруч із QR-кодом відображається сума покупки та ідентифікатор квитка, що дозволяє зіставити електронний запис із даними в базі. Також користувач бачить поточний статус квитка, наприклад активний, використаний, скасований або прострочений. Це допомагає одразу зрозуміти, чи можна використати квиток для відвідування сеансу.

Нижче основної інформації відображається електронний чек. У ньому зазначено номер замовлення, дату покупки, проміжну суму, знижку, бонусну знижку, спосіб оплати та кількість нарахованих бонусів. Такий блок робить покупку прозорою для користувача, оскільки він бачить, з яких складових

сформувалася фінальна вартість квитка. Якщо під час оформлення застосовувався промокод або списувалися бонуси, це також відображається в чеку.

Додатково в інтерфейсі передбачено кнопку «Завантажити PDF». Вона потрібна для того, щоб користувач міг зберегти квиток на пристрій або пред’явити його без постійного відкриття сайту. Таким чином електронний квиток завершує основний сценарій роботи з вебсервісом: користувач обирає фільм, оформлює замовлення, отримує підтвердження покупки та має зручний цифровий документ для подальшого використання.

Окрім користувацьких сторінок, у вебсервісі реалізовано адміністративну панель. Вона доступна тільки користувачу з відповідною роллю та призначена для керування внутрішнім станом системи [34]. На верхньому рівні панель показує статистичні картки: кількість фільмів, користувачів, замовлень, суму виручки, кількість проданих місць і популярний фільм. Такі показники дозволяють адміністратору швидко оцінити стан продажів без ручного аналізу бази даних.

Нижче розміщені інструменти пошуку та фільтрації, а також блок перевірки QR-квитка. Адміністратор може ввести номер квитка або вставити текст із QR-коду та перевірити його статус. Це потрібно під час фактичного відвідування кінотеатру, коли необхідно швидко визначити, чи є квиток активним, використаним або скасованим. Такий механізм поєднує онлайн-продаж із реальною перевіркою квитка перед входом до зали.

Ще одним важливим елементом адміністративної панелі є форма додавання фільму. Адміністратор може ввести назву, жанр, рік, тривалість, рейтинг, ціну, зал, вікове обмеження, посилання на трейлер, шлях до постера, короткий мотив та опис. Після збереження новий фільм з’являється в каталозі для користувачів. Отже, адміністратор керує не лише замовленнями, а й наповненням афіші, що робить систему придатною для постійного оновлення. Вигляд адміністративної панелі наведено на рисунку 3.9.

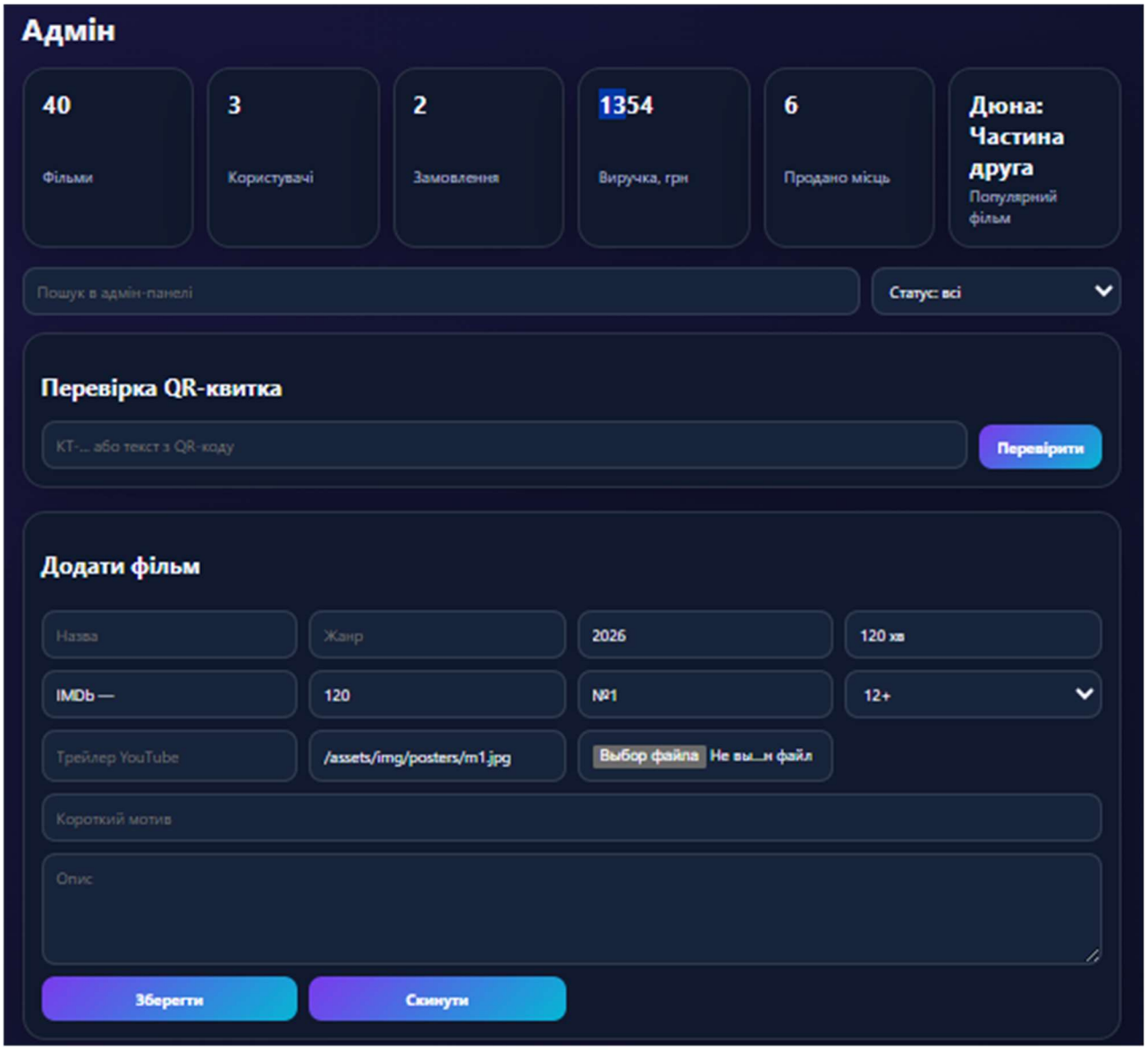

Рис. 3.9. Адміністративна панель керування вебсервісом

Отже, вебінтерфейс «КіноКвитка» побудований так, щоб користувач міг виконати весь шлях покупки квитка без сторонньої допомоги: від відкриття головної сторінки до перегляду каталогу, вибору фільму, ознайомлення з трейлером, вибору місць, застосування знижок і створення квитка. Додаткові сторінки реєстрації, відновлення пароля та інформації про кінотеатр підтримують зручність роботи, а адміністративна панель забезпечує керування контентом, користувачами, замовленнями та статистикою. Завдяки цьому вебсервіс працює як цілісна система взаємодії між відвідувачем і адміністрацією кінотеатру.

3.3 Аналіз ефективності розробленого вебсервісу та можливих проблем у його функціонуванні

Ефективність розробленого вебсервісу можна оцінювати за кількома групами показників: функціональна повнота, швидкість виконання основних сценаріїв, зручність інтерфейсу, надійність збереження даних, безпека доступу, підтримуваність коду та можливість подальшого розвитку. Для сервісу продажу квитків важливо, щоб користувач міг пройти шлях від вибору фільму до квитка без зайвих переходів, а адміністратор мав інструменти для підтримання актуального стану системи.

З погляду функціональної повноти вебсервіс охоплює ключові операції кінотеатру. Він підтримує каталог фільмів, фільтри, сторінку фільму, трейлер, відгуки, вибір сеансу, карту місць, VIP-зону, промокоди, бонуси, електронний квиток, QR-код, профіль, обране, історію покупок, правила повернення, інформаційну сторінку та адміністративну панель. Такий набір функцій дозволяє розглядати систему як робочий прототип, а не лише як статичну афішу.

Використання React забезпечує компонентний підхід до інтерфейсу. Повторювані елементи, зокрема картки фільмів, форми, схеми місць, квитки, блоки статистики та списки замовлень, можуть підтримуватися незалежно. Це зменшує ризик помилок під час оновлення, оскільки зміна компонента відображається у всіх місцях його використання [35]. Керування станом у межах компонентів дозволяє швидко реагувати на дії користувача: вибір місць, введення промокоду, перемикання мови або оновлення профілю.

SPA-логіка позитивно впливає на відчуття швидкості. Після першого завантаження користувач переходить між розділами без повного перезавантаження сторінки, а дані отримуються через API-запити. Для сценарію купівлі це важливо, оскільки користувач часто змінює дату, сеанс,

місця або параметри знижки. Перезавантаження всієї сторінки після кожної дії погіршувало б досвід і могло б призвести до втрати вже введених даних [36].

Серверна частина на Node.js обробляє запити до фільмів, сеансів, місць, профілю, замовлень і адміністративних даних. Використання маршрутизованих API дозволяє відокремити клієнтський інтерфейс від правил обробки даних. Сервер перевіряє сесію, роль користувача, коректність запиту і стан бази. Такий поділ відповідає принципу, за яким клієнт не повинен самостійно приймати остаточні рішення щодо доступу або продажу місць [37].

SQLite виконує роль постійного сховища. Для цього проєкту перевагою є те, що база зберігається у файлі, не потребує окремого серверного встановлення і підтримує транзакційну логіку. Під час створення замовлення важливо, щоб операція виконувалася цілісно: або створюється запис замовлення і пов’язані з ним місця, або зміни не вносяться. Це зменшує ризик появи неповних записів та конфліктів між станами квитків [38].

Окремо слід оцінити службову ефективність. Адміністратор бачить статистику, таблиці фільмів, сеансів, замовлень і користувачів. У таблиці замовлень доступні дії зміни статусу на використаний або скасований, а також видалення запису. Такий підхід дозволяє швидко реагувати на ситуації, коли користувач прийшов до зали, скасував покупку або коли запис потрібно прибрати під час тестування.

На рис. 3.10 показано нижню частину адміністративної панелі з таблицями замовлень і користувачів. Представлене зображення доповнює попередні матеріали, оскільки показує не загальну статистику, а саме інструменти поточного контролю статусів і користувацьких записів.

Попри працездатність вебсервісу, у його функціонуванні можливі проблеми, які слід враховувати при подальшому розвитку. Перше потенційне обмеження пов’язане з одночасним доступом до місць. Якщо кілька користувачів одночасно переглядають один і той самий сеанс, стан місць повинен оновлюватися швидко і надійно. У базовій реалізації сервер перевіряє зайнятість місць під час створення замовлення, однак у майбутньому доцільно

додати короткострокове резервування з таймером та механізм оновлення в реальному часі.

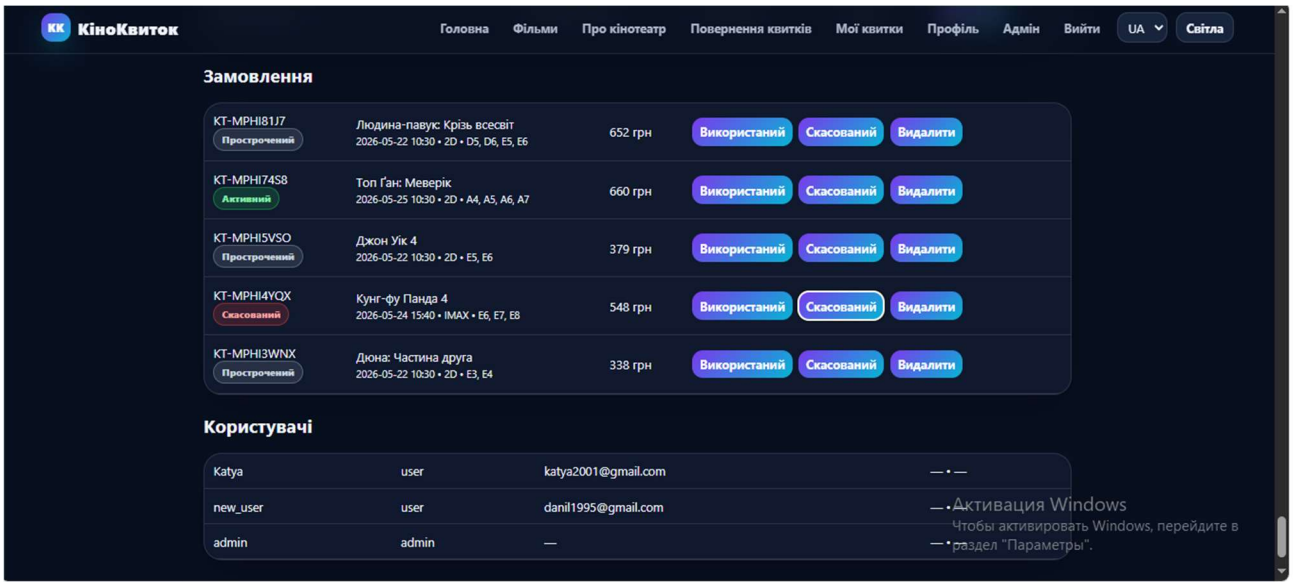

Рис. 3.10. Адміністративні таблиці замовлень і користувачів

Друга можлива складність пов’язана з платежами. У розробленому проєкті оплата моделюється як внутрішній етап створення замовлення, але в реальному впровадженні потрібна інтеграція з платіжним шлюзом. При цьому не слід зберігати повні реквізити банківських карток у власній базі. Система має передавати платіжну операцію спеціалізованому сервісу і отримувати лише підтвердження результату. Такий підхід знижує ризики обробки фінансових даних і відповідає загальним вимогам безпеки платіжного середовища [39].

Третій аспект, що потребує уваги, стосується надійності зовнішніх ресурсів. У проєкті використано постери та трейлери, які можуть завантажуватися з онлайн-джерел. Якщо зовнішнє зображення або відео стане недоступним, картка фільму втратить частину візуальної інформації. Для зменшення цього ризику доцільно зберігати локальні копії постерів або налаштувати резервні зображення. Для трейлерів можна передбачити повідомлення про недоступність відео, щоб сторінка не виглядала несправною [40].

Четвертий можливий недолік пов’язаний з якістю введених даних. Навіть за наявності клієнтської валідації користувач може надіслати некоректний запит напряму до API. Тому сервер повинен повторно перевіряти ім’я, email, телефон, дату, місця, промокод і бонуси. У розробленому сервісі така логіка частково реалізована, але при подальшому розвитку її варто розширити окремими повідомленнями для різних типів помилок.

П’яте питання для подальшого вдосконалення стосується адміністративних дій. Адміністратор має широкі права, тому помилкове видалення фільму, сеансу або замовлення може вплинути на роботу сервісу. Для реальної експлуатації доцільно додати журнал дій, підтвердження критичних операцій, м’яке видалення записів і резервне копіювання бази даних. Це дозволить відновити інформацію у разі помилки персоналу.

З позиції продуктивності важливо контролювати обсяг даних, які передаються на клієнт. Поки каталог містить сорок фільмів, завантаження всього списку є прийнятним. Якщо кількість записів збільшиться до сотень або тисяч, варто додати пагінацію, серверну фільтрацію, кешування довідкових даних і оптимізацію запитів. Для оцінки швидкості роботи у майбутньому можна орієнтуватися на показники Core Web Vitals, які використовуються для аналізу завантаження, інтерактивності та стабільності відображення сторінок [41].

З позиції доступності слід враховувати контрастність, читабельність тексту, зрозумілі підписи форм, адаптивність і можливість користування з клавіатури. У сервісі вже реалізовано темну і світлу тему, адаптивні сітки, підписи кнопок і структуровані форми. У майбутньому варто додати повні текстові альтернативи для всіх зображень, перевірити фокусні стани елементів і протестувати сайт із допоміжними технологіями.

Отже, ефективність розробленого вебсервісу полягає у тому, що він поєднує користувацьку зручність, достатню функціональну повноту, серверну перевірку даних і службове адміністрування. Основні ризики пов’язані не з відсутністю базових функцій, а з майбутнім масштабуванням, реальними

платежами, зовнішніми ресурсами та необхідністю резервного копіювання. Урахування цих ризиків визначає напрями подальшого вдосконалення проєкту.

Висновки до розділу 3

Розроблений вебресурс «КіноКвиток» охоплює повний цикл роботи з квитком: реєстрацію користувача, перегляд каталогу, ознайомлення з фільмом, вибір сеансу, вибір місць, застосування промокоду і бонусів, створення замовлення, збереження квитка та адміністративний контроль.

Взаємодія користувача з інтерфейсом побудована як послідовний сценарій, у якому кожен етап логічно веде до наступного. Додаткові елементи, зокрема трейлер, оцінки глядачів, інформаційна сторінка, програма лояльності, відновлення пароля і профіль, не відволікають від купівлі квитка, а підтримують зручність та довіру до сервісу.

До сильних сторін розробленого вебсервісу належать компонентна побудова інтерфейсу, робота без повного перезавантаження сторінки, централізована обробка даних на сервері, використання SQLite для збереження стану і наявність адміністративних інструментів. До потенційних ризиків належать одночасне бронювання місць, інтеграція реальних платежів, залежність від зовнішніх медіаресурсів, необхідність розширеної серверної валідації та резервного копіювання.

ВИСНОВКИ

У результаті виконання роботи було розроблено вебсервіс «КіноКвиток» для продажу квитків у кінотеатрі. Створений програмний продукт забезпечує перегляд афіші, вибір фільму, сеансу та місць у залі, оформлення замовлення, формування електронного квитка, перегляд історії покупок і адміністрування основних даних системи. Отриманий результат відповідає поставленій меті, оскільки реалізує повний цикл взаємодії користувача з сервісом продажу квитків.

Одержані результати відповідають сучасному рівню наукових і технічних знань у сфері веброзробки. У роботі застосовано клієнт-серверну архітектуру, компонентний підхід до побудови інтерфейсу, REST-взаємодію, серверну перевірку даних і реляційне збереження інформації. Для клієнтської частини використано React і Vite, для серверної частини - Node.js, для зберігання даних - SQLite. Такий стек є доцільним для створення локального робочого прототипу з можливістю подальшого розвитку.

Практичним результатом є робочий програмний прототип, який може бути запущений локально та використаний як основа для подальшого впровадження у діяльність кінотеатру або іншого закладу, пов’язаного з продажем квитків. Система містить користувацьку частину, адміністративну панель, базу даних, механізм авторизації, керування фільмами й сеансами, створення замовлень, QR-квиток, промокоди та бонуси. Ступінь упровадження результатів можна визначити як рівень дослідного програмного зразка.

У межах роботи сформовано практичну методику створення вебсервісу для продажу квитків. Вона охоплює аналіз предметної області, побудову UML-діаграм, проєктування бази даних, реалізацію серверної логіки, створення клієнтського інтерфейсу, перевірку авторизації, CRUD-операцій і сценарію оформлення квитка.

Значимість роботи полягає у практичному поєднанні методів проєктування інформаційних систем, вебпрограмування та організації реляційних даних у єдиному сервісі. При цьому можливий економічний ефект полягає у скороченні часу на придбання квитків, зменшенні навантаження на каси, підвищення зручності обслуговування користувачів і покращення контролю за замовленнями.

Перспективними напрямами розвитку є інтеграція реальної онлайн-оплати, короткострокове резервування місць із таймером, оновлення стану місць у режимі реального часу, розширення системи повідомлень, посилення захисту персональних даних, резервне копіювання бази даних і створення повноцінної мобільної версії.

Розроблений вебсервіс може бути основою для подальшого вдосконалення і практичного впровадження у сфері розважальних послуг.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Turban E., Pollard C., Wood G. Information Technology for Management: Driving Digital Transformation to Increase Local and Global Performance, Growth and Sustainability. 12th ed. Hoboken: Wiley, 2021. 544 p.

2. Chaffey D., Ellis-Chadwick F. Digital Business and E-Commerce Management. 8th ed. Harlow: Pearson, 2022. 640 p.

3. Fandango Support. What is Reserved Seating and How Does it Work? URL: https://support.fandango.com/ .

4. AMC Theatres. Mobile App. URL: https://www.amctheatres.com/mobile/app.

5. Atom Tickets. Help Center. URL: https://www.atomtickets.com/help .

6. Multiplex. Офіційний сайт. URL: https://multiplex.ua/ .

7. Планета Кіно. Офіційний сайт. URL: https://planetakino.ua/ .

8. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol: O'Reilly Media, 2024. 318 p.

9. Brown E. Web Development with Node and Express: Leveraging the JavaScript Stack. 2nd ed. Sebastopol: O'Reilly Media, 2019. 338 p.

10. Jin B., Sahni S., Shevat A. Designing Web APIs: Building APIs That Developers Love. Sebastopol: O'Reilly Media, 2018. 220 p.

11. SQLite. About SQLite. URL: https://sqlite.org/about.html .

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

13. Object Management Group. OMG Unified Modeling Language (OMG UML), Version 2.5.1. URL: https://www.omg.org/spec/UML/2.5.1/ .

14. IBM. What is Unified Modeling Language (UML)? URL: https://www.ibm.com/topics/uml .

15. SQLite. Foreign Key Support. URL: https://www.sqlite.org/foreignkeys.html

16. MDN Web Docs. Using HTML form validation and the Constraint Validation API. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Guides/Constraint_validation .

17. MDN Web Docs. <input type="date">. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Reference/Elements/input/date .

18. MDN Web Docs. Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API .

19. React. Quick Start. URL: https://react.dev/learn .

20. Vite. Getting Started. URL: https://vite.dev/guide/ .

21. Fielding R., Nottingham M., Reschke J. RFC 9110: HTTP Semantics. IETF, 2022. URL: https://www.rfc-editor.org/rfc/rfc9110 .

22. Express. Routing. URL: https://expressjs.com/en/guide/routing/ .

23. MDN Web Docs. Using HTTP cookies. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Guides/Cookies .

24. OWASP Cheat Sheet Series. Authentication Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authentication_Cheat_Sheet.html .

25. Robbins J. Learning Web Design: A Beginner’s Guide to HTML, CSS, JavaScript, and Web Graphics. 5th ed. Sebastopol: O’Reilly Media, 2018. 808 p.

26. React. Describing the UI. URL: https://react.dev/learn/describing-the-ui .

27. MDN Web Docs. Client-side form validation. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Forms/Form_validation .

28. UX247. Usability Principles. URL: https://ux247.com/usability-principles/

29. W3C. Web Content Accessibility Guidelines (WCAG) 2.2. W3C Recommendation, 2024. URL: https://www.w3.org/TR/WCAG22/ .

30. React Router. Routing. URL: https://reactrouter.com/start/framework/routing .

31. MDN Web Docs. Your first form. URL: https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Forms/Your_first_form .

32. MDN Web Docs. Using the Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch .

33. QRCode.js. JavaScript Library for Making QRCode. URL: https://davidshimjs.github.io/qrcodejs/ .

34. OWASP Cheat Sheet Series. Authorization Cheat Sheet. URL: https://cheatsheetseries.owasp.org/cheatsheets/Authorization_Cheat_Sheet.html .

35. React. Managing State. URL: https://react.dev/learn/managing-state .

36. React Router. Home. URL: https://reactrouter.com/ .

37. Express. Routing. URL: https://expressjs.com/en/guide/routing/ .

38. SQLite. Transaction. URL: https://www.sqlite.org/lang_transaction.html .

39. PCI Security Standards Council. PCI DSS Quick Reference Guide. v4.x, 2025. URL: https://www.pcisecuritystandards.org/document_library/ .

40. Vite. Static Asset Handling. URL: https://vite.dev/guide/assets.html .

41. web.dev. Core Web Vitals. URL: https://web.dev/articles/vitals .

Додаток А

Код серверної частини вебсервісу

Конфігурація серверної частини та відкриття бази даних

```
const http = require('http');
const { exec } = require('child_process');
const fs = require('fs');
const path = require('path');
const crypto = require('crypto');
const { URL } = require('url');
const initSqlJs = require('./vendor/sqljs/sql-wasm.js');

const ROOT = __dirname;
const PORT = Number(process.env.PORT || 3000);
const DATA_DIR = path.join(ROOT, 'data');
const DB_FILE = path.join(DATA_DIR, 'cinema.sqlite');
const SECRET = process.env.SESSION_SECRET || 'cinema-local-secret-2026';
const ADMIN_LOGIN = 'admin';
const ADMIN_PASSWORD = 'KinoAdmin2026!';
const SHOWTIMES = ['10:30', '13:00', '15:40', '18:20', '21:00'];
const SEAT_ROWS = 6;
const SEAT_COLS = 10;
let db;

async function openDatabase() {
  const SQL = await initSqlJs({ locateFile: file => path.join(ROOT, 'vendor', 'sqljs', file) });
  const sqlite = fs.existsSync(DB_FILE)
    ? new SQL.Database(fs.readFileSync(DB_FILE))
    : new SQL.Database();

  function saveDatabase() {
    fs.writeFileSync(DB_FILE, Buffer.from(sqlite.export()));
  }

  let transactionDepth = 0;
  function isWrite(sql) {
    const first = String(sql || '').trim().split(/\s+/, 1)[0].toUpperCase();
    return
['INSERT','UPDATE','DELETE','REPLACE','CREATE','DROP','ALTER','PRAGMA','BEGIN','COMMIT','ROLLB
ACK
      '].includes(first);
  }
  function afterWrite(sql) {
    const first = String(sql || '').trim().split(/\s+/, 1)[0].toUpperCase();
    if (first === 'BEGIN') { transactionDepth += 1; return; }
    if (first === 'COMMIT') { transactionDepth = Math.max(0, transactionDepth - 1); saveDatabase(); return;
      }
    if (first === 'ROLLBACK') { transactionDepth = Math.max(0, transactionDepth - 1); saveDatabase();
      return; }
    if (transactionDepth === 0) saveDatabase();
  }

  return {
    exec(sql) { sqlite.run(sql); if (isWrite(sql)) afterWrite(sql); },
    prepare(sql) {
      return {
        run(...args) { const stmt = sqlite.prepare(sql); try { stmt.bind(args); while (stmt.step()) {} if
          (isWrite(sql)) afterWrite(sql); return { changes: sqlite.getRowsModified() }; } finally {
```

```
        stmt.free(); } },
      get(...args) { const stmt = sqlite.prepare(sql); try { stmt.bind(args); return stmt.step() ?
        stmt.getAsObject() : undefined; } finally { stmt.free(); } },
      all(...args) { const stmt = sqlite.prepare(sql); const rows = []; try { stmt.bind(args); while
        (stmt.step()) rows.push(stmt.getAsObject()); return rows; } finally { stmt.free(); } }
    };
  }
 };
}
```

Сесії користувача, ролі та авторизація

```
function hashPassword(password) {
 return crypto.createHash('sha256').update(`${SECRET}:${password}`).digest('hex');
}
function newToken() { return crypto.randomBytes(32).toString('hex'); }
function parseCookies(req) {
 const cookie = req.headers.cookie || '';
 return Object.fromEntries(cookie.split(';').filter(Boolean).map(part => {
  const idx = part.indexOf('=');
  return idx < 0 ? [part.trim(), ''] : [part.slice(0, idx).trim(), decodeURIComponent(part.slice(idx +
   1).trim())];
 }));
}
function setSessionCookie(res, token) {
 res.setHeader('Set-Cookie', `cinema_session=${encodeURIComponent(token)}; Path=/; HttpOnly;
  SameSite=Lax`);
}
function currentUser(req) {
 const token = parseCookies(req).cinema_session;
 if (!token) return null;
 return db.prepare(`SELECT u.id, u.login, u.role, u.name, u.email, u.phone, u.bonus_points
  FROM sessions s JOIN users u ON u.id = s.user_id WHERE s.token = ?`).get(token) || null;
}
function requireUser(req, res) {
 const user = currentUser(req);
 if (!user) { json(res, 401, { error: 'Потрібно увійти в акаунт' }); return null; }
 return user;
}
function requireAdmin(req, res) {
 const user = requireUser(req, res);
 if (!user) return null;
 if (user.role !== 'admin') { json(res, 403, { error: 'Доступ лише для адміністратора' }); return null; }
 return user;
}

if (req.method === 'POST' && parsed.pathname === '/api/login') {
 const login = String(body.login || '').trim();
 const user = db.prepare('SELECT * FROM users WHERE lower(login)=lower(?) AND password_hash = ?')
  .get(login, hashPassword(String(body.password || '')));
 if (!user) return json(res, 401, { error: 'Невірний логін або пароль' });
 const token = newToken();
 db.prepare('INSERT INTO sessions (token, user_id, created_at) VALUES (?, ?, ?)')
  .run(token, user.id, new Date().toISOString());
 setSessionCookie(res, token);
 return json(res, 200, userToApi(user));
}
```

Створення таблиць у SQLite та початкове наповнення

```
function initDb() {
  db.exec(`
    PRAGMA foreign_keys = ON;
    CREATE TABLE IF NOT EXISTS users (
      id TEXT PRIMARY KEY, login TEXT NOT NULL UNIQUE, password_hash TEXT NOT NULL,
      role TEXT NOT NULL DEFAULT 'user', name TEXT DEFAULT '', email TEXT DEFAULT '',
      phone TEXT DEFAULT '', bonus_points INTEGER DEFAULT 0, created_at TEXT NOT NULL
    );
    CREATE TABLE IF NOT EXISTS sessions (
      token TEXT PRIMARY KEY,
      user_id TEXT NOT NULL REFERENCES users(id) ON DELETE CASCADE,
      created_at TEXT NOT NULL
    );
    CREATE TABLE IF NOT EXISTS movies (
      id TEXT PRIMARY KEY, title TEXT NOT NULL, genre TEXT NOT NULL,
      duration TEXT NOT NULL, rating TEXT NOT NULL, price INTEGER NOT NULL,
      hall TEXT NOT NULL, year INTEGER NOT NULL, trailer TEXT DEFAULT '',
      motif TEXT DEFAULT '', desc TEXT NOT NULL, poster TEXT NOT NULL,
      age_limit TEXT DEFAULT '12+', created_at TEXT NOT NULL, updated_at TEXT NOT NULL
    );
    CREATE TABLE IF NOT EXISTS orders (
      order_id TEXT PRIMARY KEY, user_id TEXT NOT NULL REFERENCES users(id) ON DELETE
CASCADE,
      movie_id TEXT NOT NULL, movie_title TEXT NOT NULL, showtime_id TEXT DEFAULT '',
      date TEXT NOT NULL, time TEXT NOT NULL, hall TEXT NOT NULL, format TEXT DEFAULT '2D',
      subtotal INTEGER DEFAULT 0, discount INTEGER DEFAULT 0, promo_code TEXT DEFAULT '',
      bonus_points INTEGER DEFAULT 0, bonus_used INTEGER DEFAULT 0, bonus_discount INTEGER
DEFAULT 0,
      total INTEGER NOT NULL, status TEXT DEFAULT 'active', buyer_name TEXT DEFAULT '',
      buyer_email TEXT DEFAULT '', buyer_phone TEXT DEFAULT '', created_at TEXT NOT NULL
    );
    CREATE TABLE IF NOT EXISTS order_seats (
      order_id TEXT NOT NULL REFERENCES orders(order_id) ON DELETE CASCADE,
      seat_id TEXT NOT NULL, PRIMARY KEY(order_id, seat_id)
    );
    CREATE TABLE IF NOT EXISTS favorites (
      user_id TEXT NOT NULL REFERENCES users(id) ON DELETE CASCADE,
      movie_id TEXT NOT NULL REFERENCES movies(id) ON DELETE CASCADE,
      created_at TEXT NOT NULL, PRIMARY KEY(user_id, movie_id)
    );
    CREATE TABLE IF NOT EXISTS showtimes (
      id TEXT PRIMARY KEY, movie_id TEXT NOT NULL REFERENCES movies(id) ON DELETE CASCADE,
      date TEXT NOT NULL, time TEXT NOT NULL, hall TEXT NOT NULL,
      format TEXT DEFAULT '2D', price INTEGER NOT NULL, created_at TEXT NOT NULL
    );
    CREATE INDEX IF NOT EXISTS idx_orders_show ON orders(movie_id, date, time);
    CREATE INDEX IF NOT EXISTS idx_showtimes_movie_date ON showtimes(movie_id, date);
  `);

  if (!db.prepare('SELECT id FROM users WHERE lower(login)=lower(?)').get(ADMIN_LOGIN)) {
    db.prepare(`INSERT INTO users (id, login, password_hash, role, name, created_at)
      VALUES (?, ?, ?, ?, ?, ?)`).run('admin', ADMIN_LOGIN, hashPassword(ADMIN_PASSWORD),
      'admin', 'Адміністратор', new Date().toISOString());
  }
}
```

Перевірка місць, промокодів і транзакційне створення квитка

```
if (req.method === 'POST' && parsed.pathname === '/api/orders') {
  const user = requireUser(req, res); if (!user) return;
  const movie = db.prepare('SELECT * FROM movies WHERE id = ?').get(body.movieId);
  const date = String(body.date || '');
  const time = String(body.time || '');
  const seats = Array.isArray(body.seats) ? [...new Set(body.seats.map(String))] : [];
  if (!movie) return json(res, 400, { error: 'Фільм не знайдено' });

  const selectedShowtime = findSelectedShowtime(movie, date, time, String(body.showtimeId || ''));
  if (!isValidDate(date) || !selectedShowtime || seats.length === 0) {
    return json(res, 400, { error: 'Оберіть дату, сеанс і місця' });
  }

  const taken = new Set(db.prepare(`SELECT os.seat_id FROM order_seats os
    JOIN orders o ON o.order_id = os.order_id
    WHERE o.movie_id = ? AND o.date = ? AND o.time = ?
    AND COALESCE(o.status, 'active') <> 'cancelled'`)
    .all(movie.id, date, selectedShowtime.time).map(r => r.seat_id));
  const conflict = seats.filter(seat => taken.has(seat));
  if (conflict.length) return json(res, 409, { error: 'Деякі місця вже зайняті', conflict });

  const buyer = body.buyer || {};
  const orderId = `KT-${Date.now().toString(36).toUpperCase()}`;
  const subtotal = seats.reduce((sum, seat) => sum + seatPrice(selectedShowtime.price, seat), 0);
  const promo = promoDiscount(body.promoCode, subtotal, { studentId: body.studentId, vipSeats:
    seats.filter(isVipSeat).length });
  const bonusRedeem = calcBonusRedemption(user, body.bonusUse, subtotal - promo.discount);
  const total = Math.max(0, subtotal - promo.discount - bonusRedeem.discount);

  try {
    db.exec('BEGIN IMMEDIATE');
    db.prepare(`INSERT INTO orders (order_id,user_id,movie_id,movie_title,showtime_id,date,time,
      hall,format,subtotal,discount,promo_code,bonus_used,bonus_discount,total,status,
      buyer_name,buyer_email,buyer_phone,created_at) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
        ?, ?, ?, ?, ?)`)
      .run(orderId, user.id, movie.id, movie.title, selectedShowtime.id, date, selectedShowtime.time,
        selectedShowtime.hall, selectedShowtime.format, subtotal, promo.discount, promo.code,
        bonusRedeem.points, bonusRedeem.discount, total, 'active', buyer.name, buyer.email, buyer.phone,
        new Date().toISOString());
    const stmt = db.prepare('INSERT INTO order_seats (order_id, seat_id) VALUES (?, ?)');
    for (const seat of seats) stmt.run(orderId, seat);
    db.exec('COMMIT');
  } catch (err) {
    db.exec('ROLLBACK');
    return json(res, 500, { error: err.message });
  }
  return json(res, 200, orderToApi(db.prepare('SELECT * FROM orders WHERE order_id = ?').get(orderId)));
}
```

Основні адміністративні маршрути та запуск сервера

```
if (req.method === 'GET' && parsed.pathname === '/api/admin') {
  const user = requireAdmin(req, res); if (!user) return;
  const movies = db.prepare('SELECT * FROM movies ORDER BY year DESC, title
ASC').all().map(movieRowToApi);
```

```
  const users = db.prepare('SELECT id, login, role, name, email, phone, bonus_points, created_at FROM
    users').all().map(userToApi);
  const orders = db.prepare('SELECT * FROM orders ORDER BY created_at DESC').all().map(orderToApi);
  const showtimes = db.prepare('SELECT * FROM showtimes ORDER BY date DESC, time DESC LIMIT
    400').all().map(showtimeRowToApi);
  return json(res, 200, { movies, users, orders, showtimes, stats: buildAdminStats(movies, users, orders)
    });
}

if (req.method === 'GET' && parsed.pathname === '/api/admin/verify') {
  const user = requireAdmin(req, res); if (!user) return;
  const orderId = extractOrderId(query.code || query.orderId || '');
  const order = db.prepare('SELECT * FROM orders WHERE upper(order_id) = ?').get(orderId);
  if (!order) return json(res, 404, { error: 'Квиток не знайдено' });
  return json(res, 200, orderToApi(order));
}

if (req.method === 'POST' && parsed.pathname === '/api/admin/showtimes') {
  const user = requireAdmin(req, res); if (!user) return;
  const movie = db.prepare('SELECT * FROM movies WHERE id = ?').get(String(body.movieId || ''));
  const date = String(body.date || '');
  const time = String(body.time || '').slice(0, 5);
  const format = String(body.format || '2D').trim() || '2D';
  const price = Number(body.price || movie?.price || 120);
  if (!movie || !isValidDate(date) || !/^\d{2}:\d{2}$/.test(time) || price <= 0) {
    return json(res, 400, { error: 'Заповніть фільм, дату, час, формат і ціну сеансу' });
  }
  const id = newId('st');
  db.prepare('INSERT INTO showtimes (id, movie_id, date, time, hall, format, price, created_at) VALUES (?,
    ?, ?, ?, ?, ?, ?, ?)')
    .run(id, movie.id, date, time, movie.hall, format, Math.round(price), new Date().toISOString());
  return json(res, 200, showtimeRowToApi(db.prepare('SELECT * FROM showtimes WHERE id =
?').get(id)));
}

if (req.method === 'POST' && parsed.pathname === '/api/admin/movies') {
  const user = requireAdmin(req, res); if (!user) return;
  const movie = sanitizeMovie(body);
  if (movie.error) return json(res, 400, { error: movie.error });
  insertMovie(movie);
  return json(res, 200, movieRowToApi(db.prepare('SELECT * FROM movies WHERE id = ?').get(movie.id)));
}

async function main() {
  db = await openDatabase();
  initDb();
  const handler = async (req, res) => {
    const parsed = new URL(req.url, `http://${req.headers.host || 'localhost'}`);
    if (parsed.pathname.startsWith('/api/')) return handleApi(req, res, parsed);
    return serveStatic(req, res, parsed);
  };
  http.createServer(handler).listen(PORT, () => console.log(`КіноКвиток запущено:
    http://localhost:${PORT}`));
}
main().catch(err => { console.error('Помилка запуску сайту:', err); process.exit(1); });
```

Додаток Б

Код основного компонента клієнтської частини

API-запити, контекст застосунку та глобальний стан

```
import React, { createContext, useContext, useEffect, useMemo, useState } from 'react';
import { createRoot } from 'react-dom/client';
import { BrowserRouter, Link, Navigate, Route, Routes, useNavigate, useParams } from 'react-router-dom';
import './styles.css';

const AppContext = createContext(null);
const today = () => new Date().toISOString().slice(0, 10);
const seatName = (r, c) => `${String.fromCharCode(64 + r)}${c}`;

async function api(path, options = {}) {
  const res = await fetch(path, {
    credentials: 'include',
    headers: { 'Content-Type': 'application/json', ...(options.headers || {}) },
    ...options
  });
  let data = null;
  try { data = await res.json(); } catch { data = null; }
  if (!res.ok) throw new Error(data?.error || 'Помилка сервера');
  return data;
}

function useApp() { return useContext(AppContext); }

function AppProvider({ children }) {
  const [user, setUser] = useState(null);
  const [movies, setMovies] = useState([]);
  const [homeData, setHomeData] = useState({ popular: [], newest: [], nearest: [] });
  const [favorites, setFavorites] = useState([]);
  const [meta, setMeta] = useState({ genres: [], ageLimits: [], showtimes: [], formats: [], rows: 6, cols:
    10 });
  const [lang, setLang] = useState(localStorage.getItem('lang') || 'uk');
  const [theme, setTheme] = useState(localStorage.getItem('theme') || 'dark');
  const [toast, setToast] = useState(null);
  const t = UI[lang] || UI.uk;

  const notify = (text, type = 'success') => {
    setToast({ text, type, id: Date.now() });
    window.clearTimeout(window.__toastTimer);
    window.__toastTimer = window.setTimeout(() => setToast(null), 3200);
  };
  const refreshUser = async () => setUser(await api('/api/me'));
  const refreshMovies = async () => {
    const rows = await api('/api/movies');
    setMovies(rows);
    try { setHomeData(await api('/api/home')); } catch {}
    return rows;
  };

  useEffect(() => { document.documentElement.dataset.theme = theme; localStorage.setItem('theme',
theme);
    }, [theme]);
  useEffect(() => { localStorage.setItem('lang', lang); }, [lang]);
  useEffect(() => { (async () => {
    const [me, allMovies, m, h] = await Promise.all([api('/api/me'), api('/api/movies'), api('/api/meta'),
```

```
    api('/api/home')]);
   setUser(me); setMovies(allMovies); setMeta(m); setHomeData(h);

 })().catch(err => notify(err.message, 'error')); }, []);

 const value = useMemo(() => ({ user, setUser, movies, homeData, favorites, setFavorites,
   meta, lang, setLang, theme, setTheme, t, notify, refreshUser, refreshMovies }),
   [user, movies, homeData, favorites, meta, lang, theme]);
 return <AppContext.Provider value={value}>{children}{toast && <div className={`toast
   ${toast.type}`}>{toast.text}</div>}</AppContext.Provider>;
}
```

Маршрутизація, навігація та захист сторінок

```
function Layout() {
 const { user, setUser, theme, setTheme, lang, setLang, t, notify } = useApp();
 const [open, setOpen] = useState(false);
 const navigate = useNavigate();
 const logout = async () => {
   await api('/api/logout', { method: 'POST' });
   setUser(null); notify('OK'); navigate('/');
 };
 return <>
   <header className="topbar">
     <Link to="/" className="brand"><span
className="brandIcon">КК</span><span>КіноКвиток</span></Link>
     <button className="menuBtn" onClick={() => setOpen(!open)}>☰</button>
     <nav className={open ? 'nav open' : 'nav'} onClick={() => setOpen(false)}>
       <Link to="/">{t.home}</Link><Link to="/movies">{t.movies}</Link><Link to="/about">{t.about}</Link>
       {user && <Link to="/ticket">{t.tickets}</Link>}{user && <Link to="/profile">{t.profile}</Link>}
       {user?.role === 'admin' && <Link to="/admin">{t.admin}</Link>}
       {!user && <Link to="/login">{t.login}</Link>}{!user && <Link to="/register">{t.register}</Link>}
       {user && <button className="linkButton" onClick={logout}>{t.logout}</button>}
       <select value={lang} onChange={e => setLang(e.target.value)}><option
value="uk">UA</option><option
         value="ru">RU</option><option value="en">EN</option></select>
       <button className="pill" onClick={() => setTheme(theme === 'dark' ? 'light' : 'dark')}>{theme ===
         'dark' ? t.light : t.dark}</button>
     </nav>
   </header>
   <main><Routes>
     <Route path="/" element={<Home />} /><Route path="/movies" element={<Movies />} /><Route
       path="/movie/:id" element={<MoviePage />} />
     <Route path="/ticket" element={<RequireAuth><Tickets /></RequireAuth>} />
     <Route path="/profile" element={<RequireAuth><Profile /></RequireAuth>} />
     <Route path="/login" element={<Login />} /><Route path="/register" element={<Register />} />
     <Route path="/admin" element={<RequireAdmin><Admin /></RequireAdmin>} />
     <Route path="*" element={<Navigate to="/" replace />} />
   </Routes></main>
 </>;
}
function RequireAuth({ children }) { const { user } = useApp(); return user ? children : <Navigate
 to="/login" replace />; }
function RequireAdmin({ children }) { const { user } = useApp(); return user?.role === 'admin' ? children :
 <Navigate to="/" replace />; }
```

Каталог фільмів, пошук і фільтрація

```
function MovieCard({ movie }) {
 const { favorites, lang, t } = useApp();
 const m = trMovie(movie, lang);
```

```
  return <Link className="movieCard" to={`/movie/${movie.id}`}>
    <span className={favorites.includes(movie.id) ? 'heart on' : 'heart'}>♥</span>
    <img src={movie.poster} alt={m.displayTitle} loading="lazy" onError={e => { e.currentTarget.src =
      '/assets/img/posters/m1.jpg'; }} />
    <div className="cardBody"><h3>{m.displayTitle}</h3><p>{movie.year} •
      {movie.duration}</p><b>{movie.price} {t.currency}</b></div>
  </Link>;
}

function Movies() {
  const { movies, meta, t, lang } = useApp();
  const [search, setSearch] = useState('');
  const [genre, setGenre] = useState('');
  const [age, setAge] = useState('');
  const [sort, setSort] = useState('newest');
  const filtered = useMemo(() => {
    let rows = movies.filter(m => {
      const tm = trMovie(m, lang);
      return (!search || `${tm.displayTitle} ${tm.displayGenre}
        ${tm.displayDesc}`.toLowerCase().includes(search.toLowerCase()))
        && (!genre || m.genre === genre)
        && (!age || ageNumber(m.ageLimit) <= ageNumber(age));
    });
    return [...rows].sort((a, b) => sort === 'priceAsc' ? a.price - b.price
      : sort === 'priceDesc' ? b.price - a.price
      : sort === 'rating' ? ratingValue(b.rating) - ratingValue(a.rating)
      : b.year - a.year);
  }, [movies, search, genre, age, sort]);

  return <section className="section page"><h1>{t.movies}</h1>
    <div className="filters">
      <input value={search} onChange={e => setSearch(e.target.value)} />
      <select value={genre} onChange={e => setGenre(e.target.value)}>{meta.genres.map(g => <option
        key={g}>{g}</option>)}</select>
      <select value={age} onChange={e => setAge(e.target.value)}>{['0+','6+','12+','16+','18+'].map(a =>
        <option key={a}>{a}</option>)}</select>
      <select value={sort} onChange={e => setSort(e.target.value)}><option
        value="newest">{t.newest}</option><option value="rating">{t.ratingSort}</option></select>
    </div>
    <div className="movieGrid">{filtered.map(m => <MovieCard key={m.id} movie={m} />)}</div>
  </section>;
}
```

Сторінка фільму, вибір місць і оформлення замовлення

```
function MoviePage() {
  const { id } = useParams();
  const navigate = useNavigate();
  const { movies, meta, user, t, lang, notify, favorites, setFavorites, refreshUser } = useApp();
  const movie = movies.find(m => m.id === id);
  const [date, setDate] = useState(today());
  const [showtimes, setShowtimes] = useState([]);
  const [showtimeId, setShowtimeId] = useState('');
  const [taken, setTaken] = useState([]);
  const [selected, setSelected] = useState([]);
  const [buyer, setBuyer] = useState({ name: '', email: '', phone: '' });

  const [promoCode, setPromoCode] = useState('');
  const [studentId, setStudentId] = useState('');
  const [bonusUse, setBonusUse] = useState('');
```

```
  const selectedShowtime = showtimes.find(s => s.id === showtimeId) || showtimes[0];
  const time = selectedShowtime?.time || meta.showtimes[0] || '18:20';

  useEffect(() => { if (user) setBuyer({ name: user.name || user.login || '', email: user.email || '',
    phone: user.phone || '' }); }, [user]);
  useEffect(() => { if (!movie || !date) return;
    api(`/api/showtimes?movieId=${movie.id}&date=${date}`).then(rows => { setShowtimes(rows);
    setShowtimeId(rows[0]?.id || ''); }).catch(err => notify(err.message, 'error')); }, [movie?.id, date]);
  useEffect(() => { if (!movie || !date || !time) return;
    api(`/api/seats?movieId=${movie.id}&date=${date}&time=${time}`).then(data => { setTaken(data.taken);
    setSelected([]); }).catch(err => notify(err.message, 'error')); }, [movie?.id, date, time]);

  const basePrice = Number(selectedShowtime?.price || movie?.price || 0);
  const subtotal = selected.reduce((sum, seat) => sum + seatPrice(basePrice, seat), 0);
  const promo = promoInfo(promoCode, subtotal, { user, studentId, selected, t });
  const amountAfterPromo = Math.max(0, subtotal - promo.discount);
  const bonus = bonusPreview(user, bonusUse, amountAfterPromo);
  const total = Math.max(0, amountAfterPromo - bonus.discount);

  const buy = async () => {
    if (!user) return notify(t.needLogin, 'error');
    if (!selected.length) return notify(t.noSeats, 'error');
    const err = validateNameEmailPhone(t, buyer, true);
    if (err) return notify(err, 'error');
    try {
      const order = await api('/api/orders', { method: 'POST', body: JSON.stringify({
        movieId: movie.id, showtimeId, date, time, seats: selected, buyer,
        promoCode, studentId, bonusUse: bonus.points
      }) });
      await refreshUser?.();
      notify(t.orderDone);
      navigate(`/ticket/${order.orderId}`);
    } catch (err) { notify(err.message, 'error'); }
  };

  return <section className="section page moviePage">
    {/* трейлер, опис, відгуки, вибір дати, сеансу, місць, промокоду, бонусів і кнопка купівлі */}
    <SeatMap rows={meta.rows} cols={meta.cols} taken={taken} selected={selected}
setSelected={setSelected}
      />
    <button className="primary" onClick={buy}>{t.buy}</button>
  </section>;
}
```

Карта місць, електронний квиток і адміністративна панель

```
function SeatMap({ rows, cols, taken, selected, setSelected }) {
  const { t } = useApp();
  const takenSet = new Set(taken);
  const toggle = (seat) => {
    if (takenSet.has(seat)) return;
    setSelected(prev => prev.includes(seat) ? prev.filter(s => s !== seat) : [...prev, seat]);
  };

  return <div className="seatWrap"><div className="screen">{t.screen}</div>
    {Array.from({ length: rows }).map((_, ri) => <div key={ri} className="seatRow">
      <span className="rowLabel">{String.fromCharCode(65 + ri)}</span>
      {Array.from({ length: cols }).map((_, ci) => {
        const seat = seatName(ri + 1, ci + 1);
        const cls = takenSet.has(seat) ? 'seat taken' : selected.includes(seat) ? 'seat selected' :
```

```
        isVipSeatId(seat) ? 'seat vip' : 'seat';
       return <button key={seat} className={cls} onClick={() => toggle(seat)}>{ci + 1}</button>;
     })}
    </div>)}
  </div>;
}

function Tickets() {
  const { id } = useParams();
  const { t, notify } = useApp();
  const [orders, setOrders] = useState([]);
  const load = async () => setOrders(id ? [await api(`/api/orders/${id}`)] : await api('/api/orders'));
  useEffect(() => { load().catch(err => notify(err.message, 'error')); }, [id]);
  const cancelOrder = async (orderId) => { await api(`/api/orders/${orderId}/cancel`, { method: 'POST' });
    await load(); };
  return <section className="section page"><h1>{t.tickets}</h1><div className="tickets">{orders.map(o
=>
    <Ticket key={o.orderId} order={o} onCancel={cancelOrder} />)}</div></section>;
}

function Ticket({ order, onCancel, adminActions }) {
  const { t, lang, movies } = useApp();
  const orderMovie = trMovie(movies.find(m => m.id === order.movieId) || { title: order.movieTitle }, lang);
  const qrText = `${window.location.origin}/ticket/${order.orderId}|${orderMovie.displayTitle}|${order.date}
    |${order.time}|${order.seats.join(',')}`;
  return <div className="ticket">
    <div className="ticketMain"><h2>{orderMovie.displayTitle}</h2><p>{order.date} •
      {order.time}</p><p>{t.seats}: {order.seats.join(', ')}</p></div>
    <div className="ticketSide"><QrCode text={qrText} /><b>{money(order.total,
      t)}</b><span>{order.orderId}</span></div>
  </div>;
}

function Admin() {
  const { t, lang, notify, refreshMovies, meta } = useApp();
  const [data, setData] = useState({ movies: [], users: [], orders: [], showtimes: [], stats: {} });
  const [form, setForm] = useState({ title: '', genre: '', year: new Date().getFullYear(), price: 120,
    hall: '№1', ageLimit: '12+' });
  const [showForm, setShowForm] = useState({ movieId: '', date: today(), time: '18:20', hall: '№1', format:
    '2D', price: 120 });
  const [verify, setVerify] = useState({ code: '', result: null });
  const load = async () => setData(await api('/api/admin'));
  useEffect(() => { load().catch(err => notify(err.message, 'error')); }, []);

  const save = async e => {
    e.preventDefault();
    await api(form.id ? `/api/admin/movies/${form.id}` : '/api/admin/movies', {
      method: form.id ? 'PUT' : 'POST', body: JSON.stringify(form)
    });
    await load(); await refreshMovies(); notify(t.movieSaved);
  };

  const setOrderStatus = async (id, status) => {
    await api(`/api/orders/${id}/status`, { method: 'PATCH', body: JSON.stringify({ status }) });
    await load(); notify('OK');
  };
  const verifyTicket = async e => {
    e.preventDefault();
    const result = await api(`/api/admin/verify?code=${encodeURIComponent(verify.code)}`);
    setVerify(v => ({ ...v, result }));
```

Продовження додатку Б

```
  };
  return <section className="section page adminPage">
    {/* статистика, перевірка QR-квитка, форма фільму, сеанси, замовлення та користувачі */}
  </section>;
}

function App() { return <BrowserRouter><AppProvider><Layout /></AppProvider></BrowserRouter>; }
createRoot(document.getElementById('root')).render(<App />);
```

ЗГОДА здобувача вищої освіти

Державного університету економіки і технологій про
перевірку кваліфікаційної роботи на прояви
академічного плагіату
та розміщення в Репозитарії Університету

Я, **Буйвол Владислав Євгенійович**,

підтримую політику Державного університету економіки і технологій з академічної доброчесності і відкритого доступу.

Засвідчую, що кваліфікаційна бакалаврська робота

«Розробка вебсервісу для продажу квитків у кінотеатрі»

виконана самостійно та не містить академічного плагіату. Я не надавав і не одержував недозволену допомогу під час підготовки цієї роботи. Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Із чинним Положенням про запобігання та виявлення академічного плагіату в роботах здобувачів вищої освіти Державного університету економіки і технологій ознайомлений. Чітко усвідомлюю, що в разі виявлення у кваліфікаційній роботі порушення норм академічної доброчесності робота не допускається до захисту або оцінюється незадовільно.

Також я поінформований, що відповідно до «Положення про Репозитарій (електронну базу даних) Державного університету економіки і технологій» зазначена робота буде розміщена в Електронному архіві Університету (Репозитарії ДУЕТ). З умовами такого розміщення ознайомлений.

15.06.26 р  Буйвол В. Є.