

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

«Вебсистема управління мережею кінотеатрів»

Виконав студент групи ІПЗ-22-1
ЗАКУСИЛО Володимир Вячеславович

Керівник роботи:
ЧИЖМОТРЯ Олексій Володимирович

Рецензент:
ВЛАСЕНКО Олег Васильович

Житомир – 2026

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

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна Вакалюк

«27» лютого 2026 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ЗАКУСИЛО Володимир Вячеславович**

Керівник роботи: **ЧИЖМОТРЯ Олексій Володимирович**

Тема роботи: **«Вебсистема управління мережею кінотеатрів»**,

затверджена Наказом закладу вищої освіти від **«27» лютого 2026 р., №92/с**

Вихідні дані для роботи: архітектура вебсистеми для автоматизації внутрішньої операційної діяльності та збору бізнес-аналітики мережі кінотеатрів, що поєднує централізоване реляційне сховище, серверні конвеєри рольового розмежування доступу й валідації часових колізій сеансів.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Чижмотря О.В.	02.03.2026р.	02.03.2026р
2	Чижмотря О.В.	02.04.2026р	02.04.2026р
3	Чижмотря О.В.	02.05.2026р	02.05.2026р

РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи освітнього ступеня «бакалавр» на тему «Вебсистема управління мережею кінотеатрів» складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. Текстова частина викладена на 66 сторінках друкованого тексту основного змісту. Список використаних джерел містить 30 найменувань і займає 2 сторінки. Матеріали роботи містять 2 додатки, розташовані на 12 сторінках. У тексті пояснювальної записки наведено 17 таблиць та 30 ілюстрацій. Загальний повний обсяг кваліфікаційної роботи становить 80 сторінок.

Метою випускної кваліфікаційної роботи є теоретичне обґрунтування, архітектурне проєктування, алгоритмічна оптимізація та практична розробка централізованої вебсистеми управління мережею кінотеатрів. Система автоматизує такі адміністративні процеси, як формування репертуару, контроль колізій сеансів, розмежування доступу та інтегрує підсистему онлайн-продажів як джерело формування зведеної бізнес-аналітики мережі.

Під час розробки застосовано методи об'єктно-орієнтованого аналізу, принципи Clean Architecture та патерн CQRS. Серверну частину реалізовано на базі .NET Core, СКБД PostgreSQL та SignalR. Клієнтську частину створено за допомогою React, а платіжний функціонал інтегровано через шлюз Stripe.

Розроблений програмний продукт може бути впроваджений в комерційних мережах кінотеатрів з метою мінімізації операційних витрат на адміністрування репертуару.

КЛЮЧОВІ СЛОВА: ВЕБСИСТЕМА, УПРАВЛІННЯ МЕРЕЖЕЮ, РЕПЕРТУАРНИЙ ПЛАН, РОЛЬОВИЙ ДОСТУП, ОНЛАЙН-БРОНЮВАННЯ, СЕАНС, СХЕМА ЗАЛУ, СКБД POSTGRESQL, ASP.NET CORE, REACT, SIGNALR, STRIPE, АРХІТЕКТУРА.

					ІПЗ.КР.Б-121-26-ПЗ			
Змн.	Арк.	№ докум.	Підпис	Дата	Вебсистема управління мережею кінотеатрів	Лім.	Арк.	Аркуші
Розроб.		Закусило В.В.						
Керівник		Чижмоторя О.В.					3	80
Рецензент		Власенко О.В.				Житомирська політехніка, група ІПЗ-22-1		
Зав. каф.		Вакалюк Т.А.						

ABSTRACT

The explanatory note to the bachelor's qualification thesis on the topic "Web System for Cinema Network Management" consists of an introduction, three chapters, conclusions, a list of references, and appendices. The textual part is presented on 66 pages of the main printed text. The list of references contains 30 items and occupies 2 pages. The materials of the thesis include 2 appendices located on 12 pages. There are 17 tables and 30 illustrations provided within the text of the explanatory note. The total volume of the qualification thesis is 80 pages.

The purpose of the qualification thesis is the theoretical substantiation, architectural design, algorithmic optimization, and practical development of a centralized web system for cinema network management. The system automates such administrative processes as repertoire formation, session conflict control, role-based access control, and integrates an online sales subsystem as a source for generating consolidated business analytics for the network.

During the development process, methods of object-oriented analysis, Clean Architecture principles, and the CQRS pattern were applied. The server side is implemented based on .NET Core, PostgreSQL DBMS, and SignalR. The client side is created using the React library, and the payment functionality is integrated via the Stripe gateway.

KEYWORDS: WEB SYSTEM, CINEMA NETWORK MANAGEMENT, REPERTOIRE PLAN, ROLE-BASED ACCESS CONTROL, ONLINE BOOKING, SESSION, SEAT MAP, POSTGRESQL DBMS, ASP.NET CORE, REACT, SIGNALR, STRIPE, ARCHITECTURE.

					ИПЗ.КР.Б-121-26-ПЗ	Арк.
						4

ЗМІСТ

РЕФЕРАТ	3
ABSTRACT	4
ВСТУП	6
РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ УПРАВЛІННЯ МЕРЕЖЕЮ КІНОТЕАТРІВ.....	8
1.1. Постановка задачі.....	8
1.2. Порівняльний аналіз наявних аналогів	10
1.3. Вибір архітектури вебсистеми	15
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	19
Висновки до першого розділу.....	22
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБСИСТЕМИ	24
2.1. Специфікація вимог та моделювання варіантів використання	24
2.2. Розробка структури бази даних	29
2.3. Проєктування та реалізація алгоритмів роботи системи	35
2.4. Програмна реалізація та архітектура взаємодії компонентів вебсистеми	38
Висновки до другого розділу	41
РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ КОМПОНЕНТІВ ВЕБСИСТЕМИ	43
3.1. Порядок встановлення та налаштування параметрів системи	43
3.2. Архітектура програмного комплексу та його розгортання	46
3.3. Тестування роботи програмного комплексу	58
Висновки до третього розділу.....	63
ВИСНОВКИ.....	65
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	67
ДОДАТКИ.....	69

ВСТУП

Актуальність теми дослідження безпосередньо зумовлена стрімкою цифровізацією процесів обслуговування клієнтів і внутрішнього операційного менеджменту підприємств кінопрокату. Сучасні мережі кінотеатрів є складними організаційно-технічними структурами. Ефективність їхнього функціонування залежить від оперативності, узгодженості та надійності обробки динамічних масивів даних. Вони включають процеси формування єдиного репертуарного плану, синхронізації розкладу кіносеансів між багатьма географічно розподіленими філіями, контролю рольового розмежування доступу персоналу, моніторингу заповнюваності залів, гнучкого ціноутворення залежно від категорій місць, а також збору бізнес-аналітики для прийняття управлінських рішень. Використання застарілих локально орієнтованих засобів автоматизації викликає десинхронізацію інформаційного обміну, що суттєво ускладнює централізований контроль з боку головного офісу компанії, унеможлиблює збір аналітики та знижує рівень лояльності кінцевих користувачів. Особливої інженерної актуальності набуває побудова централізованих архітектурних рішень, які дозволяють координувати роботу географічно розподілених філій як єдиного бізнес-комплексу, усуваючи часові колізії сеансів та конфлікти паралельного конкурентного бронювання ресурсів сховища.

Метою випускної кваліфікаційної роботи є теоретичне обґрунтування, архітектурне проектування та практична реалізація централізованої вебсистеми управління мережею кінотеатрів. Система автоматизує такі адміністративні процеси, як формування репертуару, контроль колізій сеансів, розмежування доступу та інтегрує підсистему онлайн-продажів як первинне джерело формування зведеної бізнес-аналітики мережі.

Для досягнення поставленої мети передбачено вирішення таких завдань:

- Проаналізувати предметну область та дослідити особливості бізнес-процесів у сфері кінопрокату.
- Провести аналіз наявних аналогів програмного продукту.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						6

- Сформулювати та формалізувати функціональні й нефункціональні вимоги до вебсистеми.
- Спроектувати архітектуру програмного комплексу.
- Розробити логічну та фізичну структуру реляційної бази даних для надійного збереження інформації.
- Спроектувати алгоритми управління репертуаром, контролю колізій та синхронізації стану місць у реальному часі.
- Реалізувати серверний програмний інтерфейс та інтерактивний користувацький інтерфейс адміністративної і клієнтської підсистем.
- Провести комплексне тестування розробленого програмного забезпечення.

Об'єктом дослідження є процеси автоматизації централізованого управління ресурсами, формування розкладу подій та менеджменту продажів у мережах кінотеатрів. У межах цієї роботи досліджується інфраструктура підприємства кінопрокату, яка об'єднує адміністративне керування репертуарним планом філій, синхронізацію розкладу, рольове розмежування доступу, а також контур автоматизації продажів і збору бізнес-аналітики.

Предметом дослідження є методи, алгоритми, архітектурні патерни, вебтехнології та засоби безпеки, що застосовуються для створення програмного забезпечення централізованого керування мережею кінотеатрів.

Методи дослідження базуються на інтеграції загальних принципів системного аналізу предметної області, концепцій об'єктно-орієнтованого проєктування, архітектурних підходів розділення обов'язків, а також методів реляційного моделювання баз даних.

Практичне значення отриманих результатів полягає у створенні працездатного, масштабованого та готового до розгортання програмного комплексу. Матеріали та програмний код роботи можуть бути використані реальними мережами кінотеатрів для оптимізації репертуарного планування, автоматизації контролю роботи філій та підвищення ефективності комерційної діяльності.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ УПРАВЛІННЯ МЕРЕЖЕЮ КІНОТЕАТРІВ

1.1. Постановка задачі

Основною метою розробки випускної кваліфікаційної роботи є проєктування, алгоритмічна оптимізація та подальша програмна реалізація архітектурно стійкої, масштабованої та безпечної вебсистеми управління мережею кінотеатрів. Проєкт спрямований на створення єдиного інформаційного простору для розгалуженої структури підприємств кінопрокату, що дозволить ліквідувати десинхронізацію даних між окремими філіями, автоматизувати рутинні операції адміністративного персоналу та оптимізувати процеси цифрового обслуговування клієнтів.

З урахуванням специфіки роботи мережі кінотеатрів, проєктована вебсистема має вирішувати такі основні завдання:

По-перше, реалізація модуля управління організаційною структурою компанії. Сюди входить конфігурування даних про кінотеатри мережі та технічні параметри кінозалів. При створенні залу система повинна автоматично генерувати координатну сітку за вказаною кількістю рядів і місць у ряду з динамічним призначенням категорій комфорту, що впливають на базове ціноутворення квитків. Крім того, підсистема забезпечує рольове розмежування доступу для персоналу: суперадміністратор керує всіма об'єктами глобально, тоді як права менеджера конкретної філії обмежені переглядом та редагуванням залів і сеансів лише у своєму підрозділі.

По-друге, автоматизація обліку та формування єдиного репертуару. Для усунення людського фактора та прискорення наповнення бази даних, система має містити інструментарій асинхронної інтеграції з міжнародним медіаресурсом The Movie Database через серверний програмний інтерфейс. Менеджери повинні мати змогу виконувати пошук фільмів і здійснювати автоматичний імпорт метаданих українською мовою, включаючи тривалість,

опис сюжету, вікові обмеження, посилання на відеотрейлери та графічні ресурси постерів.

По-третє, автоматизація оперативного планування розкладу та контролю часових колізій сеансів. Вебсистема повинна самостійно перевіряти доступність обраного залу на рівні серверної валідації. При додаванні нового сеансу час його закінчення має розраховуватися динамічно на основі тривалості самого фільму із обов'язковим додаванням технологічної паузи тривалістю 20 хвилин. Ця пауза необхідна персоналу для очищення та підготовки залу до наступного показу. Спроби менеджерів поставити сеанси впритул або з накладанням за часом у межах одного залу мають блокуватися системою із виведенням відповідного інформаційного повідомлення. Крім того, у випадку скасування сеансу адміністратором, алгоритми повинні передбачати автоматичне анулювання всіх пов'язаних квитків та запуск процедур фінансового повернення коштів клієнтам.

По-четверте, впровадження підсистеми управління продажами та транзакційного контролю в реальному часі. Для запобігання конфліктам паралельного конкурентного вибору одних і тих самих місць кількома покупцями одночасно, необхідно впровадити асинхронний механізм короткострокового розподіленого блокування. У момент вибору місця користувачем, система повинна фіксувати це блокування в базі даних на 10 хвилин і миттєво транслювати оновлений статус місця всім іншим активним відвідувачам сторінки сеансу без необхідності перезавантаження вікна браузера. Обмін цими подіями має відбуватися через подієво-орієнтовані хаби вебсокетів. Паралельно з цим, спеціальний фоновий сервіс на сервері повинен щохвилини перевіряти таблицю утримань, автоматично звільняючи не оплачені протягом встановленого ліміту місця для оптимізації комерційного завантаження залу.

По-п'яте, інтеграція модуля безпечної обробки фінансових транзакцій, автоматизації випуску та верифікації квитків. Вебсистема має інтегруватися з міжнародним платіжним шлюзом для обробки банківських карток на

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						9

фронтенді, тоді як бекенд-частина повинна асинхронно приймати результати платежів. Після успішного списання коштів платіжною системою та підтвердження замовлення вебсистема автоматично сформує векторний PDF-квиток із вбудованим унікальним QR-кодом для швидкої валідації контролером на вході, а також надішле його на електронну пошту клієнта. Насамкінець, для керівництва мережі реалізується модуль консолідації бізнес-аналітики, який агрегує фінансові показники, розраховує чистий дохід, обсяги повернень за звітний період та візуалізує динаміку продажів і рейтинг найпопулярніших фільмів у вигляді інтерактивних графіків для прийняття стратегічних управлінських рішень.

1.2. Порівняльний аналіз наявних аналогів

Для визначення оптимальних архітектурних та функціональних рішень під час проєктування вебсистеми необхідно провести дослідження та аналіз наявних програмних продуктів, які використовують провідні оператори кінопрокату на ринку України. Основним комерційним аналогом у даній предметній області є інформаційна система всеукраїнської мережі кінотеатрів «Multiplex», особливості функціонування якої можна детально дослідити на прикладі її публічного вебсайту. Ця платформа забезпечує автоматизацію взаємодії з відвідувачами поєднуючи функції інформаційного табло, каталогу медіаданих та транзакційного сервісу продажу.

Аналіз головної сторінки інтерфейсу дозволяє оцінити підхід до організації користувацького досвіду, де основний акцент зроблено на візуальному сприйнятті афіші. Система використовує повноекранну плиткову структуру презентації фільмів, яка виконує роль інтерактивного банера з можливістю горизонтальної навігації. Важливою деталлю є маркування окремих кінострічок спеціальними графічними індикаторами статусу подій, наприклад, позначка «Presale» для фільмів, на які відкрито попередній продаж квитків. Верхня навігаційна панель містить інструменти для швидкої зміни географічної локації користувача, що свідчить про використання єдиної централізованої бази даних для управління багатьма регіональними

кінотеатрами з однієї панелі адміністратора. Також на панелі розміщено функціональні елементи для авторизації клієнтів та спеціалізовану форму швидкого оформлення повернення куплених квитків, що автоматизує фінансові операції на рівні самообслуговування (рис. 1.1).

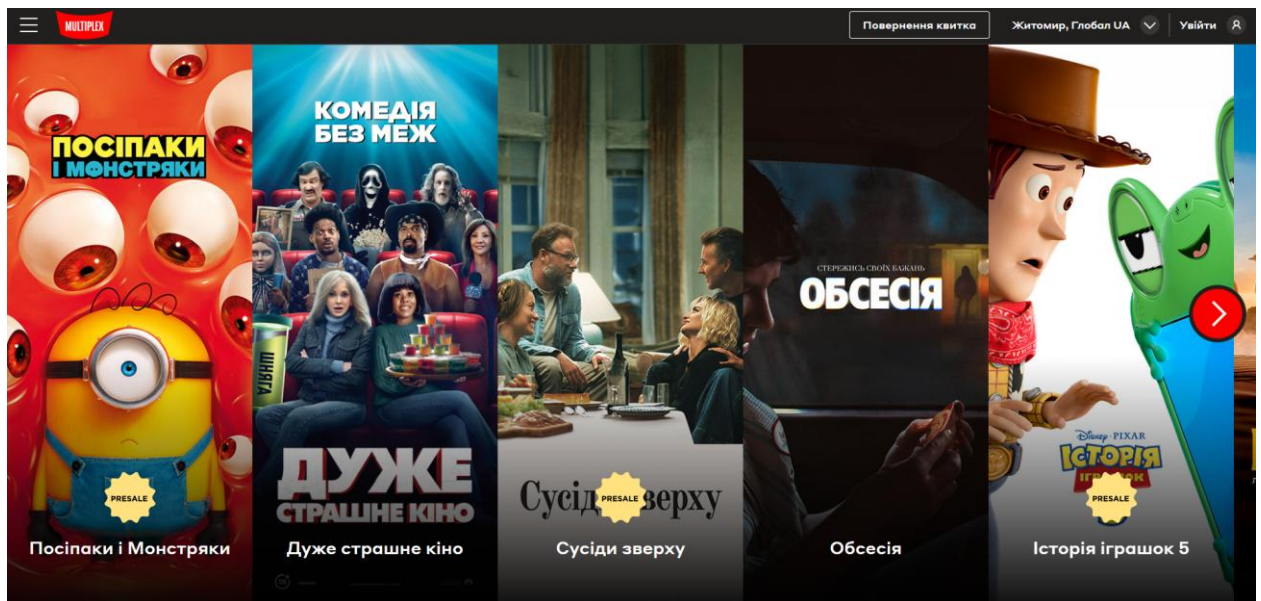


Рис. 1.1. Вигляд головної сторінки сайту «Multiplex»

Перехід до картки конкретного фільму демонструє глибоку деталізацію інформаційного наповнення системи. На сторінці відображається вичерпний перелік метаданих кінострічки, що включає вікові обмеження, рік випуску, оригінальну назву, склад творчої групи, мову дубляжу, жанрову приналежність та текстовий опис сюжету. Наявність інтегрованого відеоплеєра для перегляду трейлерів підтверджує взаємодію системи із зовнішніми медіаплатформами. З правого боку сторінки реалізовано динамічний модуль вибору дати та часу сеансів для конкретного кінотеатру. При цьому інтерфейс підтримує фільтрацію за технологічними форматами показу та спеціальними опціями інклюзивної адаптації. Графічний аналіз розкладу підтверджує наявність автоматизованого контролю часових слотів, адже система чітко обчислює тривалість кожного сеансу та унеможливорює накладання показів у межах одного залу (рис. 1.2).



Рис. 1.2. Сторінка картки фільму на сайті «Multiplex»

Процес безпосереднього вибору місць на інтерактивній схемі залу відображає архітектурну логіку обробки транзакцій у реальному часі. У верхній правій частині інтерфейсу відображається таймер зворотного відліку, налаштований на сім хвилин. Це візуально підтверджує наявність у системі алгоритму короткострокового блокування ресурсів у базі даних, який захищає обрані місця від паралельного викупу іншими клієнтами на період проведення платежу. Схема залу розділена на кодовані за кольором зони комфорту, кожна з яких має власний фінансовий тариф (рис. 1.3).

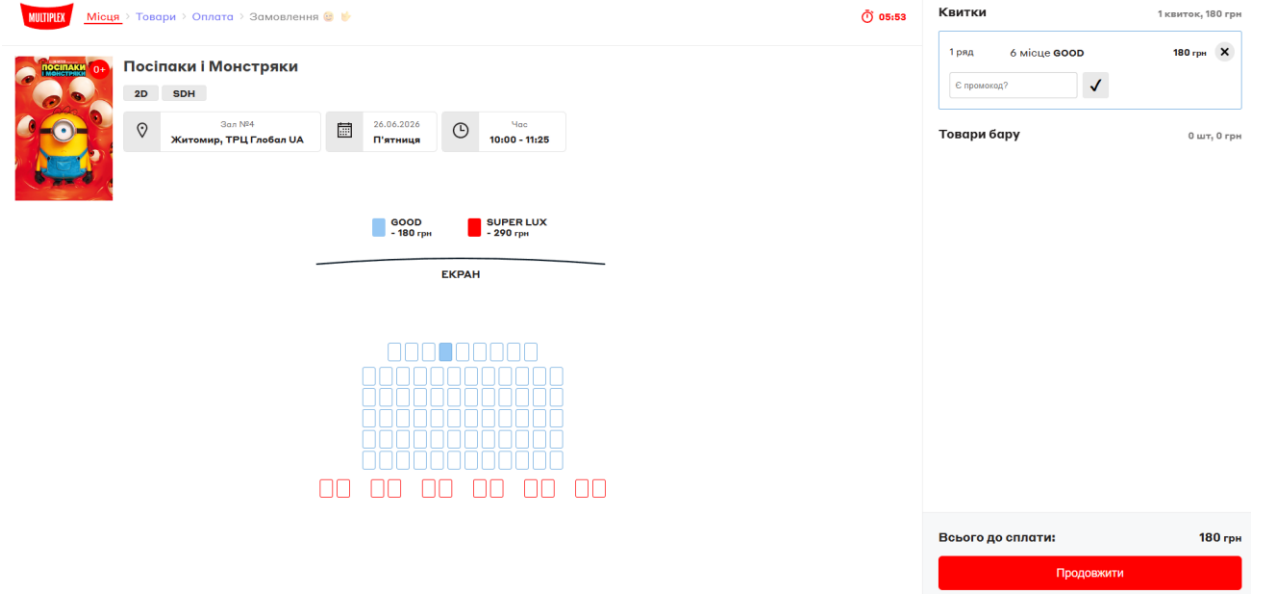


Рис. 1.3. Сторінка інтерактивної схеми залу на сайті «Multiplex»

У сфері автоматизації кінопрокату варто також виділити вебсистему мережі кінотеатрів «Планета Кіно». Вона втілює іншу концепцію взаємодії з користувачем, фокусуючись на інтерактивності, детальному тегуванні контенту та розгалуженій системі лояльності.

Аналіз структури головної сторінки дозволяє виявити використання комбінованого підходу до презентації афіші. На відміну від попереднього продукту, дана платформа поєднує горизонтальну карусель медіакартки із розгорнутим текстовим та графічним блоком актуальної допрем'єрної події. Верхнє меню навігації містить розширений набір інструментів фільтрації за часовими проміжками та найближчими днями показу. Прямо на головній сторінці під описом фоновому фільму система виводить доступні найближчі сеанси із зазначенням фірмової технології відтворення звуку та зображення «CINETECH+», що вказує на складну внутрішню структуру зв'язків між розкладом та технічними характеристиками залів (рис. 1.4).

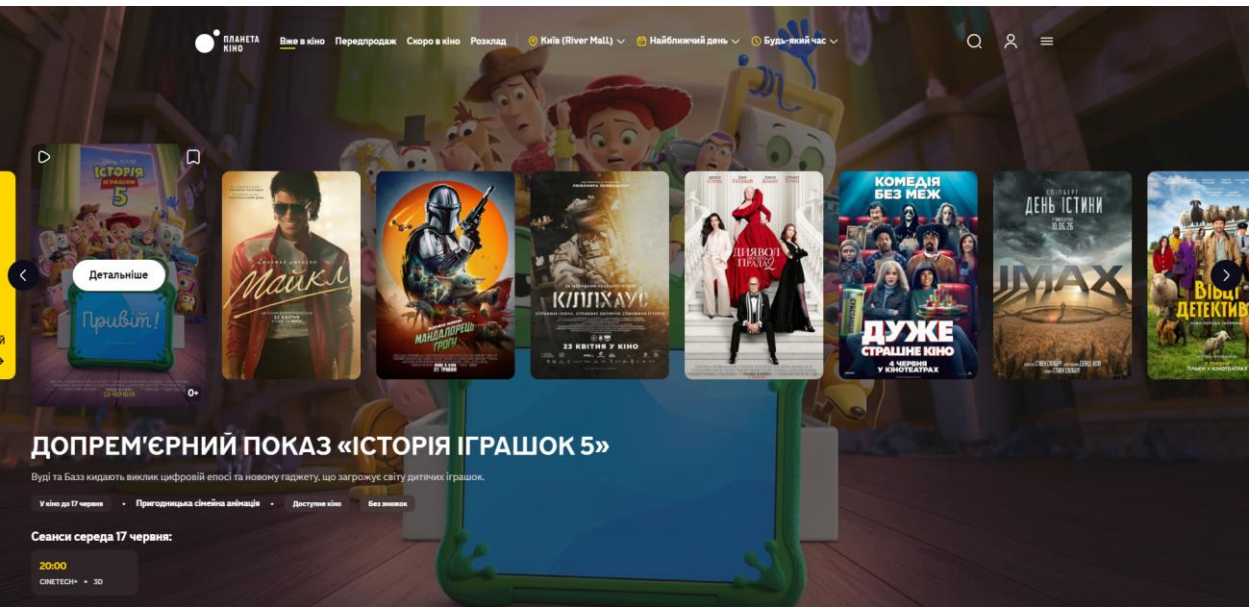


Рис. 1.4. Вигляд головної сторінки сайту «Планета Кіно»

Сторінка детальної інформації про фільм ілюструє унікальний підхід до аналізу контенту. Окремо виділено інтеграційні оцінки популярності з міжнародного ресурсу IMDb та рейтингу кінокритиків, що свідчить про використання асинхронних конвеєрів імпорту сторонніх даних. Модуль розкладу сеансів у нижній частині картки адаптований під преміальні формати

кінопоказу, такі як «RE'LUX» або «4DX», що вимагає динамічної зміни логіки ціноутворення залежно від технологічного обладнання залу (рис. 1.5).

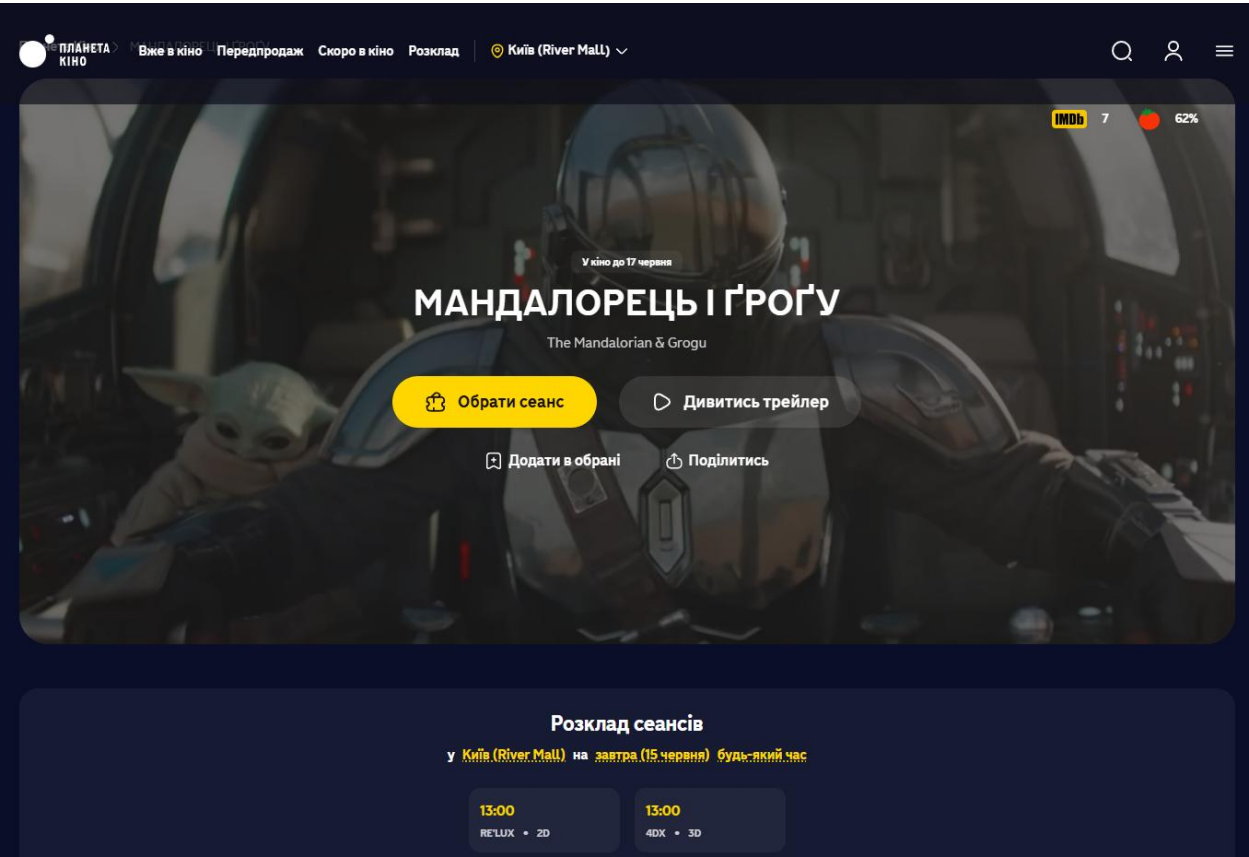


Рис. 1.5. Сторінка детальної інформації про фільм на сайті «Планета Кіно»

Екран безпосереднього оформлення квитків, представлений на рисунку 1.6, відображає особливості геометрії залів підвищеного комфорту «RE'LUX». Схема залу має унікальну структуру: місця згруповані парами у вигляді відокремлених лож, де кожне місце позиціонується за буквеним індексом ряду та індивідуальним номером. Справа розміщено пусту інформаційну панель замовлення з повідомленням про динамічну появу квитків у момент кліку. У нижній частині екрана наведено кольорове кодування статусів місць, де окремо виділено не лише зайняті та вільні місця, а й опцію оплати накопиченими бонусами системи лояльності із фіксованою внутрішньою вартістю у 15000 балів. Це підтверджує наявність алгоритмів перерахунку віртуальної валюти на серверній стороні аналога. При цьому відсутність явного таймера зворотного відліку може свідчити про приховане блокування місць у базі даних, що знижує інформативність інтерфейсу для клієнта (рис. 1.6).

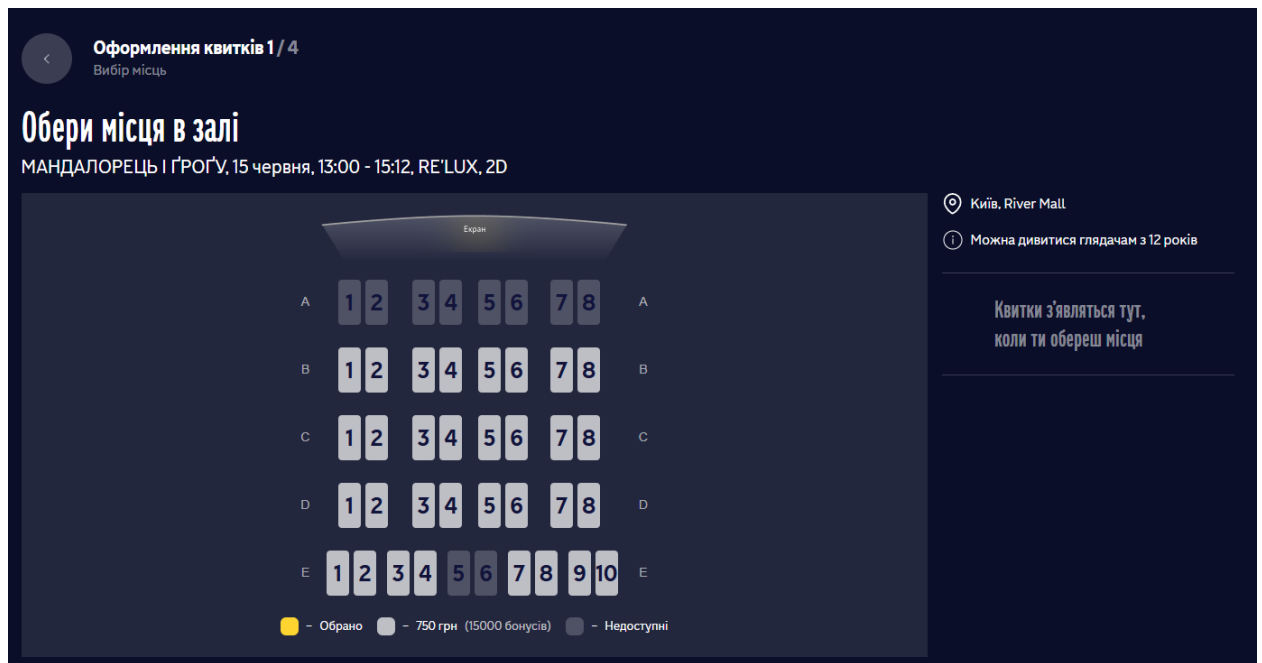


Рис. 1.6. Сторінка оформлення квитків на сайті «Планета Кіно»

Порівняльну характеристику наявних аналогів та проєктованої вебсистеми наведено в таблиці 1.1.

Таблиця 1.1

Порівняльний аналіз характеристик проєктованої системи та її аналогів

Критерії порівняння інформаційних систем	Вебплатформа «Multiplex»	Вебплатформа «Планета Кіно»	Проектowana вебсистема
Синхронізація місць у реальному часі	Реалізовано через внутрішні сервери	Реалізовано асинхронно	Інтегровано на базі WebSockets
Автоматичний імпорт репертуару	Кастомні API дистриб'юторів	Інтеграція з медіа-базами (IMDb)	Пряма інтеграція з TMDB API
Контроль часових колізій сеансів	Фіксовані інтервали розкладу	Фіксовані інтервали розкладу	Автоматичний розрахунок
Модульність платіжних систем	Інтеграція з локальним шлюзом LiqPay	Закрита внутрішня білінг-система	Інтеграція Stripe через вебхуки
Ізоляція доступу для філій	Невідомо	Невідомо	Розділення за ідентифікатором CinemaId

Таким чином, результати порівняльного аналізу підтверджують практичну доцільність розробки централізованої вебсистеми управління мережею кінотеатрів.

1.3. Вибір архітектури вебсистеми

Ефективність функціонування розгалуженої вебсистеми управління мережею кінотеатрів, що характеризується постійним потоком конкурентних

запитів від користувачів та великим обсягом транзакційних операцій, напряду залежить від обраного архітектурного стилю. У сучасній інженерії програмного забезпечення основними підходами до побудови подібних комплексів є монолітна, мікросервісна, а також багат шарова архітектури. Для проєктованої вебсистеми було обрано саме багат шарову архітектуру з чітким розмежуванням обов'язків на основі патерну Clean Architecture [2].

Монолітна архітектура передбачає об'єднання компонентів інтерфейсу користувача, бізнес-логіки та інфраструктури доступу до даних у межах єдиного виконаного модуля або процесу. Головними перевагами монолітного підходу є простота початкової розробки та легкість тестування взаємодії між компонентами [5]. Проте для системи управління мережею кінотеатрів монолітна модель створює суттєві технологічні ризики. Зі зростанням кількості підключених філій та обсягу квиткових операцій монолітна система стає складною для масштабування, оскільки її можна розгортати лише цілком, що призводить до неефективного перевикористання апаратних ресурсів. Крім того, висока зв'язність коду призводить до того, що помилка в одному модулі, наприклад, у підсистемі генерації аналітичних звітів, може спричинити повне падіння всієї вебсистеми, включаючи важливий сервіс онлайн-оплати та бронювання місць.

Мікросервісна архітектура базується на розділенні системи на набір слабкозв'язаних, незалежно розгорнутих сервісів, кожен з яких відповідає за окрему бізнес-функцію. Кожен мікросервіс функціонує автономно, має власну базу даних і взаємодіє з іншими компонентами через мережеві протоколи. Цей стиль забезпечує високу масштабованість та ізоляцію помилок, проте його впровадження для централізованих систем управління середнього масштабу супроводжується надмірними інфраструктурними витратами. Недоліками мікросервісного підходу в рамках даної предметної області є складність підтримки консистентності розподілених даних у реальному часі, суттєве збільшення мережевих затримок при інтенсивному обміні між сервісами, необхідність налаштування складних сервісних сіток та значне підвищення

вартості утримання хмарної інфраструктури, що робить його економічно та технічно нераціональним на початкових етапах розгортання системи [11, 12].

Для реалізації вебсистеми було обрано архітектурний стиль Clean Architecture у поєднанні з патерном CQRS та контейнеризованою моделлю розгортання. Цей підхід дозволяє досягти ключових переваг мікросервісів, таких як ізоляція логіки та слабка зв'язність, без накладання надмірної інфраструктурної складності. Основний принцип обраної архітектури полягає в тому, що ядро системи, яке включає шари сутностей предметної області та варіантів використання, є повністю незалежним від зовнішніх деталей, до яких відносяться бази даних, вебфреймворки, платіжні сервіси чи інтерфейси користувача. Напрямок залежностей у коді завжди йде від зовнішніх інфраструктурних шарів до внутрішнього ядра, що реалізується через механізми інверсії залежностей [3].

Розподіл обов'язків за обраною архітектурною моделлю Clean Architecture для вебсистеми включає чотири основні рівні, кожен з яких виконує суворо визначені інженерні функції. Найглибшим рівнем є шар сутностей, який інкапсулює фундаментальні бізнес-об'єкти предметної області, такі як кінотеатри, сеанси, зали, місця та квитки. Ці класи описують загальну логіку поведінки даних, містять базові правила валідації сутностей і спроектовані як чисті C# класи без будь-яких прив'язок до конкретних технологій персистентного збереження, обробників запитів чи зовнішніх веб-протоколів [2, 16].

Над шаром сутностей розташовується шар бізнес-логіки застосунку, який безпосередньо координує виконання варіантів використання вебсистеми. У цьому шарі зосереджені інтерфейси взаємодії із зовнішнім світом, абстракції контексту бази даних, а також конвеєри поведінкової валідації вхідних параметрів. Важливою інженерною особливістю цього рівня є імплементація архітектурного патерну CQRS за допомогою бібліотеки MediatR [15]. Логіка застосунку чітко розділяється на команди, які змінюють стан системи, та запити, спрямовані на читання інформації. Кожна команда або запит

обробляється ізольованим класом, що дозволяє уникнути розростання коду в контролерах, спрощує написання модульних тестів і підвищує загальну читабельність програмного комплексу [8].

Шар інфраструктури є зовнішнім щодо ядра і забезпечує безпосередню реалізацію всіх абстракцій та інтерфейсів, задекларованих на рівні бізнес-логіки застосунку. Саме тут конфігуруються фізичні параметри контексту бази даних Entity Framework Core для взаємодії з СКБД PostgreSQL, імплементуються сервіси інтеграції з міжнародним платіжним шлюзом Stripe, розгортаються алгоритми автоматичного формування векторних PDF-квитків, а також налаштовується логіка вебсокетових хабів SignalR для миттєвої синхронізації карти залу. Завдяки принципу інверсії залежностей, інфраструктурні компоненти можна легко модифікувати або повністю замінювати без жодного впливу на стабільність внутрішньої бізнес-логіки.

Найвищим рівнем архітектури виступає шар представлення, представлений у проєкті у вигляді серверного веб-інтерфейсу ASP.NET Core Web API та реалізованих SignalR-ендпоінтів. Головна задача цього шару полягає в організації точки входу для зовнішніх клієнтських запитів від React-фронтенду. Шар представлення приймає HTTP-запити, виконує первинну автентифікацію користувачів на основі JWT-токенів, перевіряє рольові обмеження доступу адміністративного персоналу та транслює отримані дані у відповідні об'єкти команд або запитів CQRS, передаючи їх медіатору для подальшої обробки ядром системи. Такий підхід гарантує повну незалежність бізнес-правил від деталей реалізації транспортного протоколу або особливостей побудови користувацького інтерфейсу.

Водночас обране поєднання Clean Architecture та патерну CQRS створює надійний фундамент для потенційного переходу до мікросервісної архітектури у майбутньому [9]. Бізнес-логіка вебсистеми вже на етапі проєктування розділена на ізольовані шари, а потоки читання та модифікації даних чітко розмежовані за допомогою команд і запитів. У випадку стрімкого глобального масштабування мережі кінотеатрів та критичного зростання навантаження

окремі модулі системи, такі як сервіс утримання місць або модуль обробки платіжних вебхуків, можна буде винести в автономні мікросервіси. Це дозволить масштабувати інфраструктуру без необхідності комплексного рефакторингу всього коду вебсистеми.

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для практичної реалізації спроектованої шаруватої архітектури вебсистеми управління мережею кінотеатрів та забезпечення транзакційної логіки у реальному часі було здійснено вибір інструментальних засобів для серверної, інфраструктурної та клієнтської частин програмного комплексу.

Основою серверної частини обрано кросплатформну екосистему .NET 8.0 та вебфреймворк ASP.NET Core Web API [26]. Цей вибір зумовлений високими показниками продуктивності платформи, нативною підтримкою асинхронного програмування за допомогою конструкцій `async/await`, наявністю вбудованого контейнера впровадження залежностей (Dependency Injection) та строгими механізмами типізації мови C# [1, 14]. ASP.NET Core дозволяє створювати полегшені RESTful-сервіси, які мінімізують затримки при обробці запитів користувачів. Екосистема .NET 8.0 забезпечує оптимізацію використання пам'яті завдяки впровадженню нових типів даних та покращеної роботи збирача сміття, що важливо при одночасному обслуговуванні великої кількості сесій покупців. Для реалізації асинхронної взаємодії в реальному часі під час інтерактивного вибору місць на карті залу обрано технологію ASP.NET Core SignalR. Вона повністю абстрагує роботу з низькорівневим протоколом WebSockets і забезпечує двосторонній безперервний обмін повідомленнями між браузером клієнта та сервером, що дозволяє миттєво синхронізувати статуси місць без перевантаження сторінок. Крім того, SignalR підтримує автоматичне відновлення підключення у разі нестабільного інтернет-з'єднання на боці користувача. Технологія дозволяє групувати клієнтів у віртуальні кімнати за ідентифікаторами конкретних кіносеансів для економії мережевого трафіку сервера [18].

Як систему керування базами даних обрано реляційну PostgreSQL 16. Вона повністю відповідає вимогам ACID, що є необхідним для забезпечення ізоляції паралельних транзакцій та запобігання колізіям під час резервування місць. Взаємодія між серверним кодом та СКБД реалізується за допомогою ORM-фреймворку Entity Framework Core. Він дозволяє налаштовувати зв'язки таблиць через інструменти Fluent API та оптимізує SQL-запити. Використання Entity Framework Core у поєднанні з архітектурним патерном CQRS дозволяє відмовитися від надмірних шарів абстракції на кшталт класичних репозиторіїв. Запити до бази даних будуються за допомогою технології LINQ безпосередньо всередині обробників команд. Це забезпечує гнучкість під час компіляції складних вибірок, дозволяє використовувати механізм відстеження змін лише там, де це необхідно для модифікації стану сутностей. Швидкість читання розкладу сеансів при цьому підвищується завдяки викликам без відстеження стану компонентів [4, 13].

Для забезпечення безпеки, автентифікації та авторизації користувачів і персоналу застосовано фреймворк Microsoft ASP.NET Core Identity у поєднанні з протоколом JWT. Механізм автентифікації на основі безстанційних токенів дозволяє повністю захистити кінцеві точки Web API. Права доступу розділяються на основі ролей користувачів, де для локальних менеджерів філій впроваджено фільтр доступу за унікальним ідентифікатором кінотеатру. Важливим елементом інфраструктури обслуговування клієнтів є модуль онлайн-оплати, для реалізації якого обрано міжнародну платіжну платформу Stripe. Інтеграція зі Stripe API дозволяє безпечно обробляти банківські картки безпосередньо в інтерфейсі застосунку без необхідності збереження конфіденційних фінансових даних клієнтів на серверах кінотеатру. Обробка успішних платежів та автоматичний запуск конвеєра генерації квитків реалізується за допомогою механізму асинхронних вебхуків (Webhooks), коли сервер Stripe надсилає підписане криптографічним ключем повідомлення на спеціальний ендпоінт вебсистеми, ініціюючи фіксацію куплених місць у базі даних [28].

У процесі організації інфраструктури розробника та тестування вебсистеми було застосовано гібридну модель розгортання сервісів. Зовнішні залежності системи, які є важливими для її роботи, але не потребують постійної модифікації коду, повністю ізольовані в контейнерах Docker за допомогою утиліти Docker Compose. До цієї інфраструктури входить контейнер бази даних PostgreSQL 16, забезпечений механізмом Docker Volumes для персистентного збереження даних, а також допоміжний контейнер поштової заглушки Mailpit. Використання Mailpit дозволяє локально перехоплювати та тестувати автоматичне надсилання згенерованих PDF-квитків на SMTP-рівні через зручний вебінтерфейс без ризику розсилки реального спаму. Водночас ядро серверного застосунку на етапі розробки запускається безпосередньо нативно на машині розробника. Такий крок є інженерно виправданим, оскільки він дозволяє використовувати інструменти швидкого налагодження коду, гарячого перезапуску застосунку та суттєво економить час, усуваючи потребу у тривалому перезбиранні важких Docker-образів після кожної виправленої помилки бізнес-логіки [21].

Клієнтську частину вебсистеми спроектовано на базі бібліотеки React 19 та інструменту збирання Vite з використанням мови TypeScript. Застосування React дозволяє реалізувати компонентний підхід до побудови інтерфейсу та мінімізувати прямі маніпуляції з DOM-деревом за рахунок механізму Virtual DOM, тоді як TypeScript забезпечує строгу типізацію фронтенд-коду, що знижує ймовірність виникнення помилок на етапі розробки [7, 19]. Використання мови TypeScript дозволяє створити єдину систему контрактів даних між сервером та клієнтом, коли інтерфейси відповідей API контролюються на етапі збирання проєкту. Це мінімізує ризики обробки некоректних полів або пустих значень об'єктів фільмів чи залів. Керування глобальним станом застосунку – зокрема збереження даних авторизованого користувача, кошика квитків та поточного сеансу SignalR – реалізовано за допомогою компактної бібліотеки керування станом. Такий підхід дозволяє уникнути комплексності, притаманної Redux, і гарантує високу

продуктивність рендерингу інтерактивної карти місць. Для стилізації інтерфейсу застосовано Tailwind CSS v4, який дозволяє гнучко налаштовувати адаптивну верстку під мобільні та десктопні пристрої користувачів, а для побудови графіків бізнес-аналітики в адмін-панелі використано спеціалізовану бібліотеку Recharts.

Для стабільного функціонування та розгортання вебсистеми сформульовано технічні вимоги до апаратного забезпечення. На етапі розробки та локального тестування гібридної інфраструктури комп'ютер розробника повинен бути оснащений багатоядерним процесором із тактовою частотою не менше 2,5 ГГц, обсягом оперативної пам'яті від 16 ГБ для одночасного запуску Docker-контейнерів бази даних та інструменту Mailpit, середовища розробки й браузера, а також швидким твердотільним накопичувачем із щонайменше 50 ГБ вільного простору. Для продуктивного сервера, де надалі буде розгорнуто систему за допомогою повного стеку Docker Compose, мінімальні вимоги включають два віртуальних ядра процесора, 4 ГБ оперативної пам'яті та 20 ГБ дискового простору, чого цілком достатньо для обслуговування середньої мережі кінотеатрів. Кінцевий користувач системи може взаємодіяти з системою через будь-який сучасний веббраузер на мобільному телефоні або персональному комп'ютері з мінімальними вимогами до швидкості інтернет-з'єднання.

Висновки до першого розділу

У першому розділі випускної кваліфікаційної роботи було проведено комплексний системний аналіз предметної області, пов'язаної з процесами автоматизації та централізованого управління розгалуженими мережами кінотеатрів. На основі детального дослідження внутрішньої структури та бізнес-процесів підприємств кінопрокату було сформульовано та формалізовано детальну постановку інженерної задачі для проєктування і подальшої програмної реалізації вебсистеми. Визначено та обґрунтовано ключові функціональні вимоги до архітектури системи, серед яких пріоритетне інженерне місце посідають модулі автоматичного координатного

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						22

налаштування геометрії кінозалів, асинхронний імпорт репертуару медіаданих, контроль часових колізій сеансів із врахуванням обов’язкових технологічних пауз на прибирання, а також утримання місць і обробка онлайн-транзакцій у режимі реального часу.

Проведений порівняльний аналіз наявних комерційних аналогів на ринку України, зокрема інформаційних вебплатформ кіномереж «Multiplex» та «Планета Кіно», дозволив виявити їхні архітектурні переваги та інфраструктурні обмеження на основі дослідження їхніх публічних інтерфейсів. Встановлено, що великі комерційні системи є пропрієтарними рішеннями з жорсткою прив’язкою до внутрішніх квиткових серверів та локальних платіжних шлюзів, таких як LiqPay, що унеможливорює їх використання чи гнучку адаптацію іншими суб’єктами господарювання без значних фінансових витрат. Результати проведеного дослідження повністю підтвердили технічну та економічну доцільність розробки відкритого масштабованого програмного комплексу, спроможного забезпечити високу швидкість обробки конкурентних запитів клієнтів.

На основі детального аналізу сучасних архітектурних стилів та патернів було обґрунтовано вибір шаруватої структури Clean Architecture у поєднанні з концепцією розділення відповідальності CQRS. Такий підхід забезпечив повну незалежність бізнес-правил ядра системи від деталей зовнішньої інфраструктури, баз даних і транспортних протоколів представлення даних, а також сформував надійну основу для потенційного винесення навантажених модулів в автономні мікросервіси у разі подальшого глобального розширення мережі компанії. За результатами дослідження ринку інструментальних засобів було сформовано стабільний технологічний стек, який базується на використанні екосистеми .NET 8.0, фреймворку ASP.NET Core Web API, технології SignalR, реляційної СКБД PostgreSQL 16, клієнтської бібліотеки React 19 та мови TypeScript.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБСИСТЕМИ

2.1. Специфікація вимог та моделювання варіантів використання

Проектування архітектури централізованої вебсистеми управління мережею кінотеатрів базується на принципах об'єктно-орієнтованого аналізу та вимагає чіткого формального визначення меж системи, її взаємодії із зовнішніми акторами та деталізації функціональних вимог. Для високорівневого моделювання поведінки системи та визначення обов'язків кожної групи користувачів застосовується уніфікована мова моделювання UML, зокрема діаграми прецедентів, які дозволяють зафіксувати вимоги до програмного комплексу у вигляді послідовних сценаріїв взаємодії. Декомпозиція загальної моделі використання на окремі діаграми для кожного актора дозволяє детально візуалізувати повний спектр функціональних можливостей вебсистеми, мінімізувати архітектурну зв'язність та забезпечити високу наочність проектування [25].

У рамках проєктованої вебсистеми виділено чотирьох внутрішніх акторів, кожен з яких має унікальний набір прав доступу, розмежованих на рівнях шару представлення та бізнес-логіки застосунку, а також одного зовнішнього системного актора.

Першим актором системи є Клієнт, який може взаємодіяти з системою як анонімно, так і в статусі автентифікованого користувача. Його функціональний простір охоплює процеси пошуку медіаконтенту та безпосереднього придбання квитків. Клієнт має можливість переглядати актуальну афішу та вивчати детальну картку кінострічки з переглядом трейлерів. На етапі замовлення прецедент вибору місць на схемі залу асинхронно взаємодіє з SignalR-хабом для відображення зайнятих місць. Процес безпосередньої онлайн-оплати інтегрується з платіжним шлюзом Stripe через захищене вікно введення реквізитів. Окремим важливим прецедентом самообслуговування для клієнта є можливість ініціації повернення купленого квиткового пакета у власному профілі з автоматичним

Для детальної візуалізації функціональних можливостей відвідувача вебсистеми розроблено діаграму прецедентів клієнтського рівня (рис. 2.1).



ИПЗ.КР.Б-121-26-ПЗ

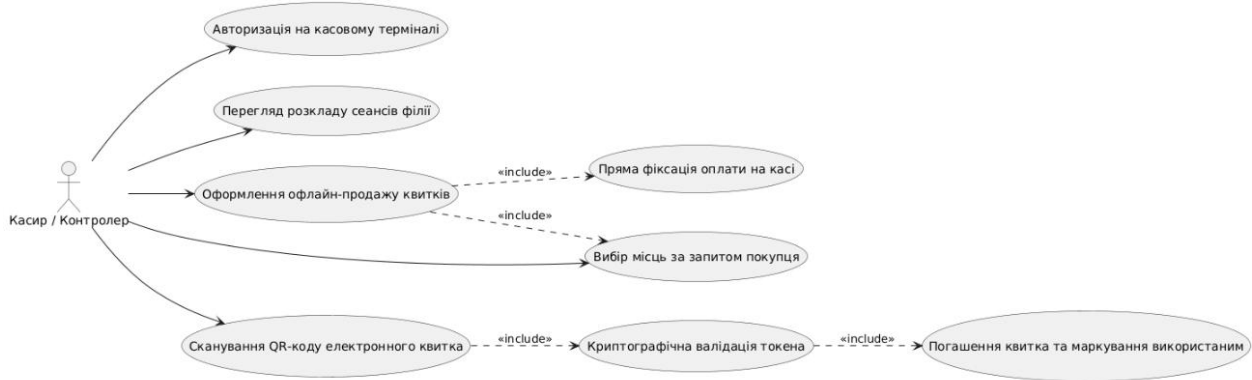


Рис. 2.2. Діаграма прецедентів для актора Касир / Контролер

Третім актором виступає Менеджер кінотеатру. Його прецеденти пов'язані з адмініструванням локальних ресурсів філії та формуванням сітки подій. Повноваження менеджера ізольовані рамками його кінотеатру за допомогою фільтрації сутностей за зовнішнім ключем CinemaId на рівні бізнес-логіки застосунку. Менеджер здійснює конфігурування залів, де запуск процесу автоматичної генерації місць відбувається на основі вказаної кількості рядів та сидінь із можливістю гнучкого розподілу місць за категоріями комфорту. Головним операційним прецедентом менеджера є планування розкладу сеансів. При спробі додати новий сеанс система в обов'язковому порядку запускає вбудований конвеєр валідації часових колізій, який обчислює тривалість фільму та автоматично додає 20-хвилинну технічну перерву на прибирання, блокуючи накладання подій у межах одного залу. Також менеджер має доступ до перегляду локальних звітів про заповнюваність залів у своїй філії (рис. 2.3).



Рис. 2.3. Діаграма прецедентів для актора Менеджер кінотеатру

Четвертим актором є Суперадміністратор мережі, який володіє глобальними та необмеженими правами доступу до керування всією

організаційно-технічною структурою компанії. Його прецеденти включають створення та конфігурування нових об’єктів кінотеатрів у базі даних із прив’язкою до міст і фізичних адрес. Для мінімізації ручної праці персоналу суперадміністратор ініціює прецедент автоматизованого наповнення глобального каталогу фільмів, який асинхронно взаємодіє із зовнішнім актором-сервісом The Movie Database через REST API, імпортуючи описи, вікові рейтинги та графічні постери українською мовою. Крім того, суперадміністратор здійснює повне керування обліковими записами всього персоналу мережі, призначає ролі Identity та має ексклюзивний доступ до інтерактивного модуля глобальної бізнес-аналітики Recharts, де агрегуються фінансові показники чистого доходу та обсягів повернень по всій компанії (рис. 2.4).



Рис. 2.4. Діаграма прецедентів для актора Суперадміністратор

Для деталізації бізнес-логіки розробленої моделі та узагальнення наскрізних процесів взаємодії користувачів із системою сформовано детальну специфікацію сценаріїв використання. У таблиці 2.1 приведено розгорнутий опис основних сценаріїв використання вебсистеми управління мережею кінотеатрів (табл. 2.1).

Таблиця 2.1

Основні сценарії використання

Основний актор	Найменування	Формулювання
Касир / Контролер	Касове оформлення офлайн-продажів	Оформлення квитків на касі кінотеатру за готівковий розрахунок

Основний актор	Найменування	Формулювання
Касир / Контролер	QR-валідація та погашення квитків	Сканування QR-коду квитка при вході до зали, перевірка валідності та зміна статусу на використаний
	Автентифікація	Вхід до системи з перевіркою відповідності ролі Cashier та автоматичним зчитуванням ідентифікатора поточної філії
Менеджер кінотеатру	Рольова авторизація менеджера	Доступ до локальної панелі управління із застосуванням глобального фільтра CinemaId на рівні бази даних для повної ізоляції даних інших філій
	Автоматична координатна генерація	Запуск алгоритму, який на основі введеної матриці рядів автоматично створює записи місць у базі даних
	Планування розкладу сеансів	Створення нових карток сеансів із прив'язкою фільму до конкретного залу, часу початку та встановленням базової вартості квитків
	Валідація часових колізій сеансів	Автоматичний перехресний аналіз розкладу, який вираховує тривалість стрічки з бази даних і блокує створення сеансу, якщо час перетинається з іншим фільмом
	Моніторинг заповнюваності залів	Перегляд локальних звітів щодо динаміки викупу місць на різні сеанси для оперативного коригування цінової політики філії
Суперадміністратор	Управління каталогом фільмів	Наповнення загальної бази даних фільмів, які стають доступними для вибору менеджерами всіх локальних філій при формуванні розкладу
	Асинхронний REST-імпорт TMDB	Автоматичне завантаження метаданих фільму з бази TMDB

Основний актор	Найменування	Формулювання
Суперадміністратор	Модерація облікових записів персоналу	Створення профілів для нових менеджерів та касирів мережі, призначення відповідних Identity-ролей та прив'язка працівників до конкретних кінотеатрів
	Моніторинг фінансової аналітики	Побудова інтерактивних діаграм і трендів продажів у реальному часі на основі агрегованих даних PostgreSQL для прийняття управлінських рішень
Клієнт	JWT-автентифікація профілю	Реєстрація або вхід користувача до системи
	Динамічний пошук та фільтрація	Відображення поточної афіші фільмів філії з можливістю сортування за жанрами
	Інтерактивне SignalR-бронювання	Клік на координатну сітку залу, запуск 10-хвилинного таймера блокування місця у PostgreSQL та трансляція статусу іншим вікнам через відкритий вебсокет-канал
	Онлайн-оплата квитків через Stripe	Введення реквізитів у форму Stripe Elements, асинхронна обробка результату через платіжні вебхуки та автоматичний випуск PDF-документа
	Автоматизований рефанд замовлень	Оформлення скасування квитка в особистому кабінеті клієнта з миттєвим анулюванням місць у базі та надсиланням запиту на повернення коштів у Stripe

2.2. Розробка структури бази даних

Важливим етапом інженерного проєктування централізованої вебсистеми управління мережею кінотеатрів є розробка реляційної моделі бази даних у СКБД PostgreSQL. Структура збереження даних повинна забезпечувати цілісність інформації, виключати критичні аномалії надлишковості завдяки раціональній нормалізації та підтримувати високу швидкість виконання транзакційних операцій під час конкурентного онлайн-

бронювання місць [10, 22]. На основі виділених варіантів використання та аналізу бізнес-сутностей було сформовано логічну структуру бази даних, яка повністю покриває транзакційну логіку, систему безпеки на основі JWT-токенів, аудит дій користувачів та механізми тимчасового утримання ресурсів у реальному часі. Усього розроблена архітектура сховища містить 14 основних операційних та допоміжних таблиць бізнес-логіки (Users, Cinemas, Halls, Seats, Sessions, Movies, Bookings, BookingSeats, Tickets, Payments, SelectedSeats, RefreshToken, AuditLogs, Discounts), а також інтегрує в себе 7 інфраструктурних сутностей стандартного контуру Microsoft ASP.NET Core Identity для керування правами доступу й автентифікації користувачів, повну версію якої у вигляді ER-діаграми винесено у Додаток А.

Для полів із фіксованим набором значень на рівні прикладного коду застосовано перелічувані типи. Їх валідація в СКБД PostgreSQL забезпечується збереженням у цілочисельному форматі, що оптимізує дисковий простір та прискорює індексацію.

База даних вебсистеми містить інформацію про користувачів та персонал мережі кінотеатрів. Структуру та типи даних для збереження облікових записів наведено в таблиці 2.2.

Таблиця 2.2

Опис полів таблиці Users

Назва поля	Тип даних у PostgreSQL	Функціональний опис призначення атрибута
Id	uuid (PK)	Унікальний ідентифікатор облікового запису користувача
CinemaId	uuid (FK)	Зовнішній ключ прив'язки працівника до конкретного кінотеатру мережі.
Email	varchar(256)	Електронна пошта, що виступає логіном в системі
FirstName	varchar(100)	Ім'я користувача
LastName	varchar(100)	Прізвище користувача
DateOfBirth	Date	Дата народження відвідувача для контролю вікових рейтингів
CreatedAt	timestamp with time zone	Дата і точний час реєстрації профілю в системі

Оскільки адміністративні користувачі керують конкретними філіями, наступним кроком було розроблено структуру для збереження даних про самі кінотеатри. Опис полів для цієї сутності міститься в таблиці 2.3.

Таблиця 2.3

Опис полів таблиці Cinemas

Назва поля	Тип даних у PostgreSQL	Функціональний опис призначення атрибута
Id	uuid (PK)	Глобальний унікальний ідентифікатор філії кінотеатру
Name	varchar(200)	Комерційна назва конкретного кінотеатру мережі
Address	varchar(500)	Фізична адреса розташування будівлі
City	varchar(100)	Місто, в якому функціонує цей об'єкт

У кожному кінотеатрі функціонує декілька залів із різними технічними характеристиками. Опис структури збереження інформації про кінозали наведено в таблиці 2.4.

Таблиця 2.4

Опис полів таблиці Halls

Назва поля	Тип даних у PostgreSQL	Функціональний опис призначення атрибута
Id	uuid (PK)	Унікальний ідентифікатор кінозалу
CinemaId	uuid (FK)	Зовнішній ключ для зв'язку з конкретним кінотеатром
Name	varchar(100)	Технічна назва або номер залу
TotalRows	integer	Загальна кількість рядів у геометрії залу
SeatsPerRow	integer	Кількість місць в одному ряду
IsActive	boolean	Прапорець доступності залу для призначення сеансів

Для кожного залу автоматично генерується координатна сітка відповідно до кількості рядів і місць. Специфікацію полів для опису місць у залі представлено в таблиці 2.5.

Таблиця 2.5

Опис полів таблиці Seats

Назва поля	Тип даних у PostgreSQL	Функціональний опис призначення атрибута
Id	uuid (PK)	Ідентифікатор конкретного місця в залі
HallId	uuid (FK)	Зовнішній ключ прив'язки до геометрії залу
Row	varchar(10)	Позначення ряду
Number	integer	Порядковий номер місця в межах ряду
Type	integer	Категорія комфорту місця

На основі створених залів та каталогу фільмів менеджери формують розклад кіносеансів. Структуру таблиці для збереження даних про сеанси наведено в таблиці 2.6.

Таблиця 2.6

Опис полів таблиці Sessions

Назва поля	Тип даних у PostgreSQL	Функціональний опис призначення атрибута
Id	uuid (PK)	Ідентифікатор запланованого сеансу
MovieId	uuid (FK)	Зовнішній ключ зв'язку з каталогом фільмів
HallId	uuid (FK)	Зовнішній ключ фізичного залу проведення показу
StartTime	timestamp with time zone	Час початку демонстрації кінострічки
MovieDurationInMinutes	integer	Тривалість фільму з урахуванням блоку трейлерів
BasePrice	numeric(18,2)	Базова вартість квитка на стандартне місце
Status	varchar(50)	Стан сеансу

Коли користувачі бронюють або купують квитки на конкретні сеанси, система створює запис про замовлення. Параметри збереження даних про замовлення наведено в таблиці 2.7.

Таблиця 2.7

Опис полів таблиці Bookings

Назва поля	Тип даних у PostgreSQL	Функціональний опис призначення атрибута
Id	uuid (PK)	Унікальний ідентифікатор замовлення

продовж. табл. 2.7

Назва поля	Тип даних у PostgreSQL	Функціональний опис призначення атрибута
UserId	uuid (FK)	Посилання на обліковий запис покупця
SessionId	uuid (FK)	Посилання на конкретний сеанс для оптимізації вибірок
TotalPrice	numeric(18,2)	Підсумкова сума замовлення до сплати
BookingTime	timestamp with time zone	Час ініціації та формування чека замовлення
Status	varchar(50)	Статус оплати

Для онлайн-замовлень, які оплачуються через платіжний шлюз Stripe, система фіксує результати фінансових транзакцій. Опис відповідних полів наведено в таблиці 2.8.

Таблиця 2.8

Опис полів таблиці Payments

Назва поля	Тип даних у PostgreSQL	Функціональний опис призначення атрибута
Id	uuid (PK)	Ідентифікатор фінансової транзакції
BookingId	uuid (FK)	Посилання на оплачуване замовлення
StripePaymentIntentId	varchar(255)	Технічний унікальний індекс операції у системі Stripe
Amount	numeric(18,2)	Фізична сума списаних грошових коштів
Status	varchar(50)	Фінансовий статус транзакції від платіжного шлюзу
PaidAt	timestamp with time zone	Точний час успішного проведення платежу

Оскільки в одному замовленні може бути декілька квитків, для зв'язку між обраними місцями та чеком оплати використовується проміжна таблиця, структура якої наведена в таблиці 2.9.

Таблиця 2.9

Опис полів таблиці BookingSeats

Назва поля	Тип даних у PostgreSQL	Функціональний опис призначення атрибута
Id	uuid (PK)	Ідентифікатор зв'язку між місцем та чеком

Назва поля	Тип даних у PostgreSQL	Функціональний опис призначення атрибута
BookingId	uuid (FK)	Зовнішній ключ приналежності до замовлення
SeatId	uuid (FK)	Зовнішній ключ обраного місця з сітки залу

Після успішного підтвердження транзакції для кожного викупленого місця генерується унікальний електронний квиток із QR-кодом. Специфікацію полів цієї фінальної таблиці наведено в таблиці 2.10.

Таблиця 2.10

Опис полів таблиці Tickets

Назва поля	Тип даних у PostgreSQL	Функціональний опис призначення атрибута
Id	uuid (PK)	Глобальний ідентифікатор випущеного квитка
BookingSeatId	uuid (FK)	Посилання на транзакційну сутність зарезервованого місця
TicketCode	varchar(100)	Унікальний ідентифікатор квитка, зашитий у QR-код для сканування касиром
IsUsed	boolean	Прапорець погашення квитка касиром на вході в зал

Таблиці допоміжного контуру, що не винесені окремо у вигляді специфікацій (включаючи SelectedSeats, RefreshToken, Movies, Discounts, AuditLogs), мають аналогічну реляційну структуру, побудовану на базі системних типів uuid та timestamp with time zone. Таблиця SelectedSeats виступає оперативним транзакційним буфером для збереження тимчасового блокування місць у розрізі сеансів і користувачів, обмежуючи тривалість доступу атрибутом LockedUntil. Цілісність даних на рівні СКБД PostgreSQL забезпечується налаштуванням правил поведінки зовнішніх ключів. Для довідкових і батьківських сутностей (Cinemas, Movies або Seats) встановлено обмеження RESTRICT, що блокує видалення об'єктів за наявності пов'язаних квитків чи сеансів, тоді як для допоміжних логів та транзакційних зв'язків застосовано механізм CASCADE для автоматичного каскадного очищення сховища [29].

2.3. Проєктування та реалізація алгоритмів роботи системи

Оскільки логіка функціонування вебсистеми управління мережею кінотеатрів безпосередньо залежить від приналежності користувача до певної рольової моделі, аналіз виключно статичної структури реляційних даних є недостатнім для повного розуміння архітектурних процесів. Динаміка виконання ключових операцій та поведінка системи під час обробки запитів потребують опису на рівні поведінкових алгоритмів. Для формалізації та візуалізації цих процесів у межах інженерного проєктування застосовуються діаграми активності UML, які відображають послідовність виконання дій, умовні переходи, паралельні гілки та циклічні процеси. Це дозволяє детально описати механізми функціонування бізнес-логіки системи та взаємодію її компонентів [25].

Одним із ключових алгоритмів вебсистеми є формування розкладу сеансів адміністративним персоналом, що ґрунтується на автоматичному виявленні часових конфліктів. Обробник команди `CreateSessionCommand`, реалізований у шарі `Application`, приймає параметри нового сеансу та отримує тривалість обраного фільму з каталогу медіаданих. Після цього система розраховує час завершення показу з урахуванням обов'язкової двадцятихвилинної технологічної перерви, необхідної для підготовки та прибирання кінозалу. На основі сформованого часового інтервалу виконується перевірка наявності перетинів із вже запланованими сеансами засобами PostgreSQL та LINQ. У разі виявлення конфлікту операція завершується винятком валідації, що запобігає формуванню некоректного розкладу ще до моменту збереження транзакції. Послідовність кроків та логіку умовних розгалужень цього процесу детально візуалізовано за допомогою діаграми активностей UML (рис. 2.5).

PostgreSQL. Подія видалення автоматично тригерить SignalR-ендпоінт, який повертає місцям статус вільних для вибору у вікнах усіх підключених користувачів, забезпечуючи високу динаміку та актуальність інтерфейсу.

Одним із найбільш складних інтерактивних алгоритмів вебсистеми є процес онлайн-бронювання місць та проведення електронної оплати квитків. Особливістю цього процесу є необхідність одночасної синхронізації стану місць між усіма активними користувачами, забезпечення транзакційної цілісності даних та інтеграції із зовнішнім платіжним сервісом Stripe. Послідовність виконання цього сценарію відображено на діаграмі активності, наведеній на рисунку 2.6.

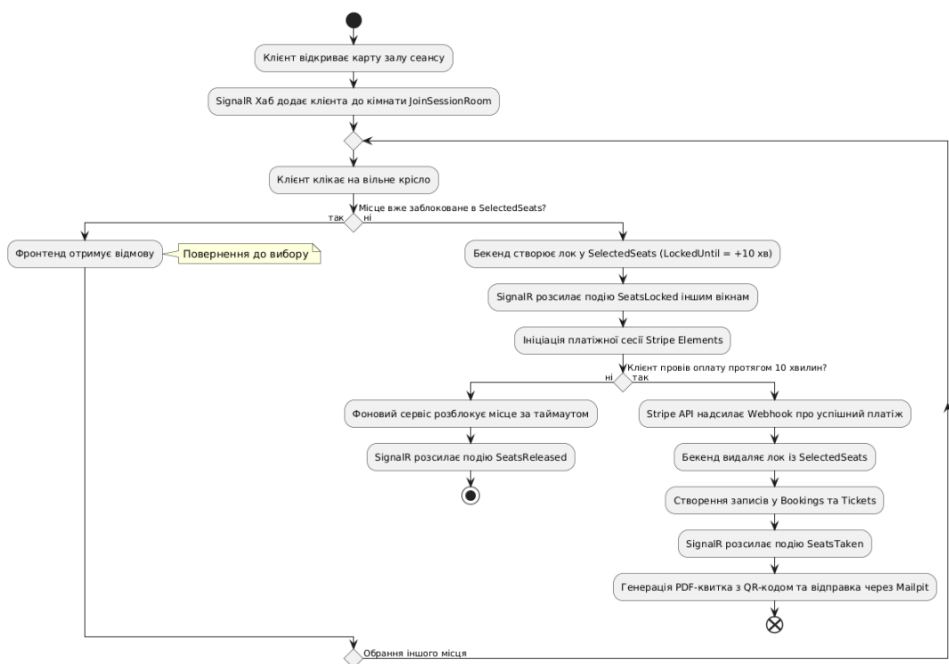


Рис. 2.6. Діаграма активності процесу онлайн-бронювання та оплати квитків

Взаємодія розпочинається після успішної автентифікації користувача, коли вебклієнт надсилає запит на отримання схеми місць обраного кіносеансу. Одночасно із завантаженням схеми сервер додає користувача до окремої SignalR-групи, сформованої для відповідного сеансу, що забезпечує синхронізацію змін між усіма підключеними клієнтами в режимі реального часу. Під час вибору місця система перевіряє його поточний стан у таблиці SelectedSeats. Якщо місце вже заблоковане або придбане іншим користувачем, операція бронювання відхиляється. В іншому випадку створюється запис про тимчасове резервування строком на 10 хвилин, після чого інформація про

зміну статусу місця миттєво передається всім учасникам відповідної групи. Після успішного завершення оплати платіжний сервіс Stripe надсилає вебхук, обробка якого завершує процедуру бронювання: тимчасовий запис вилучається, у базі даних створюються постійні записи про бронювання та придбані квитки, а користувачу автоматично формується електронний PDF-квиток.

Отже, спроектовані поведінкові алгоритми охоплюють як адміністративні процеси управління мережею кінотеатрів, так і клієнтські сценарії взаємодії із системою. Реалізовані механізми автоматичного контролю часових колізій, рольового розмежування доступу, синхронізації стану місць у режимі реального часу та очищення прострочених бронювань забезпечують цілісність даних, стабільність функціонування системи й коректність виконання бізнес-процесів.

2.4. Програмна реалізація та архітектура взаємодії компонентів вебсистеми

В основі програмної реалізації розгалуженої вебсистеми управління мережею кінотеатрів лежить сучасна клієнт-серверна архітектура, що базується на принципі розподілу зон відповідальності між автономними обчислювальними вузлами системи. Серверний компонент виступає ізольованим ядром бізнес-логіки, яке повністю контролює процеси персистентного збереження даних, верифікацію прав доступу та асинхронну обробку транзакцій, тоді як клієнтський Single Page Application застосунок відповідає за рендеринг інтерфейсу, утримання стану користувача та обробку подій. Інформаційний обмін між зазначеними частинами системи організовано за допомогою архітектурного стилю REST поверх захищеного протоколу HTTP з використанням стандартизованого формату обміну даними JSON. Крім того, для забезпечення інтерактивності високонавантажених модулів інтегровано дуплексний протокол WebSockets під керуванням технології SignalR, що дозволяє мінімізувати мережеві затримки при трансляції станів кінозалів.

Серверна частина системи функціонує в середовищі кросплатформної екосистеми .NET 8.0 під керуванням фреймворку ASP.NET Core Web API. Замість класичного монолітного підходу, структурування серверного коду виконано відповідно до шаблону Clean Architecture, де ключовим інструментом розв'язання внутрішніх залежностей є вбудований контейнер Dependency Injection. Обробка вхідних HTTP-запитів координується тонким шаром API-контролерів, які не містять безпосередньої бізнес-логіки, а лише виконують роль маршрутизаторів. Для реалізації динамічних процесів застосовано архітектурний патерн Command Query Responsibility Segregation, впроваджений за допомогою спеціалізованої бібліотеки MediatR. Кожен запит на модифікацію даних інкапсулюється в окремий об'єкт команди та передається до відповідного ізольованого обробника. Така декомпозиція дозволяє модифікувати бізнес-правила системи без ризику порушення цілісності суміжних модулів.

Взаємодія з реляційним сховищем PostgreSQL організована на рівні інфраструктурного шару за допомогою об'єктно-реляційного мапера Entity Framework Core. Для оптимізації продуктивності в умовах значного зростання кількості запитів під час популярних кінопрем'єр, EF Core налаштовано на роботу в режимі автоматичного об'єднання контекстів бази даних у пул. Це дозволяє повторно використовувати раніше відкриті фізичні підключення до PostgreSQL, суттєво заощаджуючи ресурси процесора сервера на етапах ініціалізації сесій. Конфігурування SQL-запитів у рамках CQRS-обробників виконується виключно із застосуванням типізованого інструментарію LINQ. Всі динамічні параметри, що надходять від клієнтів, проходять обов'язкову параметризацію на рівні ORM-транслятора, що повністю нівелює можливість виконання несанкціонованого коду та захищає вебсистему від ін'єкційного зламу.

Безпека інфраструктури та розмежування рівнів доступу до REST-ендпоінтів реалізована через технологію JWT (JSON Web Token) у зв'язці з Microsoft ASP.NET Core Identity. Процедура реєстрації користувачів

супроводжується криптографічним хешуванням паролів за допомогою алгоритму PBKDF2 з ітераційним захистом, виключаючи збереження відкритих текстових значень. Авторизована сесія супроводжується генерацією двох типів токенів: короткоживучого `access_token` та довгоживучого `refresh_token`. Роль кожного актора перевіряється на етапі проходження запиту через спеціалізовані фільтри авторизації. Для менеджерів кінотеатрів діє каскадна глобальна фільтрація даних, яка на рівні побудови SQL-запитів автоматично додає обмеження за ідентифікатором `CinemaId`, запобігаючи витоку комерційної інформації між різними філіями кіномережі.

Клієнтська частина вебсистеми управління мережею кінотеатрів спроектована як Single Page Application на базі сучасної бібліотеки React 19 та високопродуктивного інструменту збирання Vite з використанням строго типізованої мови TypeScript. Архітектурно фронтенд побудовано на базі атомарних, перевикористовуваних компонентів, що суттєво спрощує супровід та розширення візуальних елементів програми. Для організації мережевої взаємодії застосовано HTTP-клієнт Axios, конфігурація якого містить глобальні перехоплювачі. Перехоплювач запитів автоматично зчитує актуальний JWT із захищеного сховища клієнта та підставляє його в заголовок `Authorization: Bearer` для кожного звернення до API. У свою чергу, перехоплювач відповідей реалізує безшовний алгоритм обробки помилок із кодом 401: у разі закінчення терміну дії токена доступу, клієнт асинхронно надсилає запит на ендпоінт оновлення, отримує нову пару токенів і повторює початковий запит користувача абсолютно непомітно для інтерфейсу [6].

Управління глобальним станом автентифікації та збереження інформації про поточну роль користувача реалізовано за допомогою оптимізованих кастомних сховищ, інтегрованих із механізмами реактивності React. На основі цих даних функціонує декларативна система клієнтської маршрутизації, яка на етапі аналізу дерева компонентів блокує спроби несанкціонованого переходу на захищені сторінки. Якщо неавторизований відвідувач намагається вручну ввести адресу панелі бізнес-аналітики чи інтерфейсу касового продажу

квитків, система миттєво перенаправляє його на сторінку JWT-авторизації, забезпечуючи повну синхронізацію клієнтської безпеки із правилами захисту серверного Web API.

У підсумку, застосування шаблону Clean Architecture спільно з патерном CQRS забезпечує чітке розділення обов'язків між інфраструктурним, прикладним та презентаційним шарами вебсистеми. Таке структурування спрощує подальшу підтримку коду, оптимізує ізольоване тестування окремих сервісів ядра і дозволяє гнучко нарощувати функціонал вебсистеми за умов зміни бізнес-вимог.

Висновки до другого розділу

У другому розділі випускної кваліфікаційної роботи було проведено повний комплекс інженерно-проектних робіт, спрямованих на побудову детальної архітектурної та поведінкової моделі централізованої вебсистеми управління мережею кінотеатрів. На основі розробленої класифікації користувачів було виконано декомпозицію функціональних вимог, що дозволило побудувати чотири окремі компактні діаграми варіантів використання UML для Суперадміністратора, Касира-контролера, Менеджера філії та Клієнта мережі. Такий підхід забезпечив високу наочність проектування меж системи. Розроблена наскрізна специфікація сценаріїв та створена діаграма активностей дозволили покроково формалізувати життєвий цикл управління сеансами, автоматизувати виявлення часових колізій при формуванні розкладу, а також структурувати операційну діяльність персоналу філій.

На основі аналізу інформаційних потоків та транзакційних вимог було спроектовано та нормалізовано логічну структуру реляційної бази даних у СКБД PostgreSQL, яка включає розгалужену систему таблиць бізнес-логіки та вбудований інфраструктурний контур автентифікації. Розроблена ER-діаграма, винесена до матеріалів додатків, відображає схему збереження даних, де враховано реальні вимоги комерційного застосунку, включаючи ведення аудит-логів дій суперадміністратора, зберігання довготривалих

токенів оновлення сесій, а також виділення ізольованої оперативної таблиці для фіксації часових лімітів утримання місць. Було обґрунтовано правила перенесення концептуальної моделі у фізичне сховище за допомогою конфігурацій Fluent API фреймворку Entity Framework Core, що дозволило зафіксувати точність фінансових типів даних та впровадити складені унікальні індекси для захисту конвеєра від аномалій дублювання конкурентних транзакцій.

Також було спроектовано алгоритм автоматичного виявлення часових конфліктів при плануванні розкладу менеджерами філій, який на рівні шару бізнес-логіки автоматично вираховує тривалість показу фільмів і закладає обов'язкові 20-хвилинні технологічні паузи на прибирання, що виключає помилки планування розкладу через людський фактор. На додаток до адміністративних функцій, формалізовано алгоритм онлайн-бронювання та оплати місць, де за допомогою SignalR-хабів реалізовано двосторонню трансляцію геометрії залу в реальному часі, розподілене блокування обмежено 10-хвилинним таймером транзакційного утримання місць, а фіксація успішних операцій спирається на обробку захищених вебхуків платіжного шлюзу Stripe.

Наприкінці розділу було детально описано програмну реалізацію та архітектуру взаємодії компонентів вебсистеми, побудовану на базі .NET 8.0 та React 19 із використанням мови TypeScript. Обґрунтовано застосування шаблону Clean Architecture та патерну CQRS за допомогою бібліотеки MediatR для серверної частини, що дозволило відокремити ендпоінти Web API від безпосередніх бізнес-правил ядра і сформувати фундамент для масштабування проєкту. Описано клієнтські інструменти безпеки, що базуються на перехоплювачах Axios Interceptors для оновлення короткоживучих токенів JWT та декларативних захищених маршрутах Protected Routes.

РОЗДІЛ 3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ КОМПОНЕНТІВ ВЕБСИСТЕМИ

3.1. Порядок встановлення та налаштування параметрів системи

Забезпечення стабільного функціонування та високої відмовостійкості централізованої вебсистеми «StarPlex» вимагає чіткої регламентації процесів інфраструктурного розгортання, конфігурування системних параметрів та ініціалізації середовища виконання. Для детального моделювання фізичної архітектури програмного комплексу та наочного відображення розподілу виконуваних модулів між апаратними ресурсами було розроблено діаграму розгортання UML (Рис. 3.1). Зазначена схема дозволяє зафіксувати топологію мережевої взаємодії компонентів на рівні фізичних або віртуальних обчислювальних вузлів системи [25].

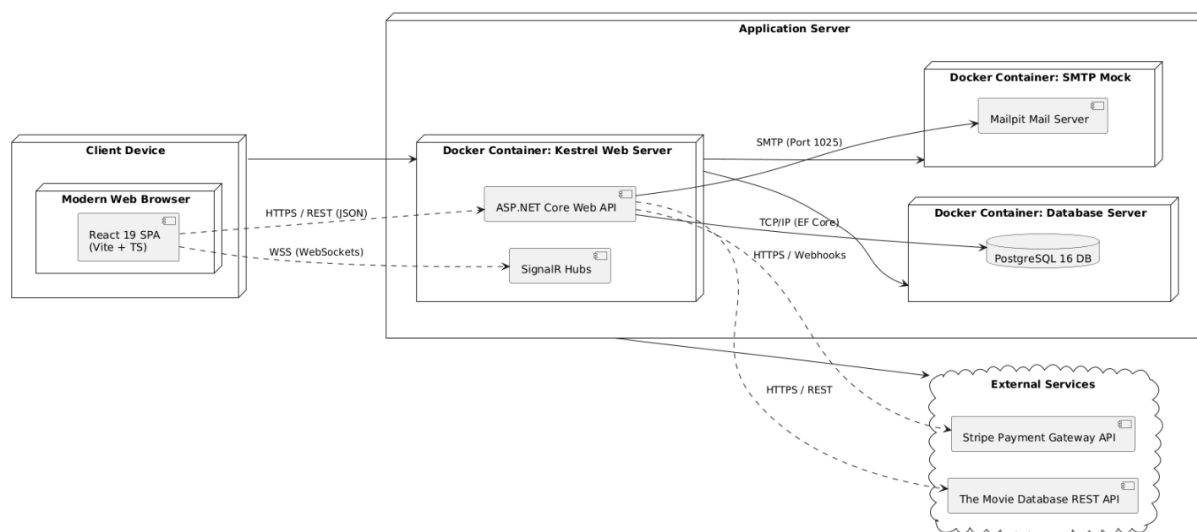


Рис. 3.1. Діаграма розгортання інфраструктурних компонентів вебсистеми

Представлена топологія фізичного розподілу ілюструє взаємодію чотирьох ізольованих середовищ виконання. На боці кінцевого пристрою користувача функціонує сучасний вебпереглядач, який утримує в оперативній пам'яті та виконує скомпільований клієнтський застосунок React 19. Продуктивний серверний вузол об'єднує в собі три автономні Docker-контейнери, що взаємодіють всередині внутрішньої віртуальної мережі через фіксовані порти. Вебсервер Kestrel забезпечує працездатність ядра на базі ASP.NET Core Web API та хабів SignalR, опрацьовуючи вхідні транзакційні запити та підтримуючи відкриті вебсокет-канали. Контейнер СКБД

PostgreSQL 16 відповідає за надійне збереження реляційних даних, а допоміжний інфраструктурний контейнер Mailpit виступає локальним поштовим сервером для перехоплення та тестування конвеєра генерації PDF-квитків. Зовнішній контур взаємодії охоплює REST-інтерфейси платформ Stripe та TMDb.

Для первинного розгортання та успішного налаштування програмного комплексу в умовах локальної робочої станції розробника або цільового Linux-сервера необхідно забезпечити наявність попередньо встановленого інструментарію контейнеризації Docker Daemon та утиліти Docker Compose. Оскільки ядро серверної частини на етапі розробки запускається нативно, на цільовій машині розгортається пакет SDK .NET 8.0, який надає доступ до компілятора мови C# та інструментів командного рядка EF Core CLI. Отримання актуальної кодової бази та подальший супровід версій виконується за допомогою клієнта системи контролю версій Git шляхом клонування репозиторію з віддаленого сервера.

Процес початкової ініціалізації сховища даних передбачає автоматизоване розгортання контейнера PostgreSQL за допомогою конфігураційного файлу docker-compose.yml. Для забезпечення персистентності інформації та запобігання втраті транзакційних записів при перезапуску контейнерів налаштовується ізольоване монтування дискових томів (Docker Volumes). Після підняття сервісу бази даних, розробник виконує команду `dotnet ef database update` у консолі серверного проєкту `StarPlex.Infrastructure`. Ця дія активує механізм міграцій Entity Framework Core, який автоматично трансліює C#-класи сутностей предметної області у фізичні таблиці PostgreSQL, створює необхідні зовнішні ключі, обмеження цілісності та B-Tree індекси [17].

Наступним кроком є точне налаштування параметрів конфігурації у файлі `appsettings.json` або через змінні оточення (Environment Variables) операційної системи. Секція `ConnectionStrings` наповнюється параметрами доступу до PostgreSQL, які включають адресу хоста, назву бази, унікальне ім'я

користувача та пароль. Крім параметрів підключення до сховища, у конфігураційному дереві визначаються параметри безпеки `JwtSettings`, що містять довгий криптографічний секретний ключ для підпису `claims`, та параметри інтеграції із платіжною системою, включаючи приватний ключ `Stripe:SecretKey` та унікальний ідентифікатор вебхуку `Stripe:WebhookSecret`. Винесення зазначених маркерів та ключів за межі вихідного коду програми є фундаментальною інженерною вимогою безпеки, що виключає випадковий витік конфіденційних даних до публічних репозиторіїв.

Клієнтська частина вебсистеми конфігурується незалежно через файли налаштувань `.env.development`. Головним параметром тут виступає змінна `VITE_API_BASE_URL`, яка вказує базову адресу серверних ендпоінтів для налаштування екземпляра `Axios`. Встановлення зовнішніх залежностей та пакетів для фронтенду виконується за допомогою менеджера пакетів `npm` шляхом виконання команди інсталяції в кореневому каталозі `React`-проєкту.

Фінальний запуск системи в робочий стан передбачає одночасну активацію серверного та клієнтського процесів. Вебсервер `Kestrel` починає прослуховувати визначений у конфігурації порт, реєструє `SignalR`-маршрути та ініціює роботу фонового `Background Worker` для очищення застарілих блокувань у таблиці `SelectedSeats`. Для автоматизації початкового адміністрування системи та розгортання базової конфігурації безпеки в інфраструктурному шарі реалізовано програмний модуль `RoleSeeder`. Під час першого старту сервера цей модуль автоматично перевіряє наявність системних ролей у базі даних `PostgreSQL`. За потреби створюється повний набір ролей: `SuperAdmin`, `CinemaManager`, `Cashier` та `Customer`. Після цього система самостійно реєструє перші облікові дані адміністратора із заздалегідь визначеними параметрами. Програмну реалізацію та повний вихідний код розробленого інфраструктурного модуля автоматичного розгортання конфігурацій безпеки приведено у матеріалах додатків (Додаток Б). Такий підхід повністю виключає потребу в ручному втручанні у структуру СКБД через сторонні утиліти адміністрування, надаючи розробнику миттєвий доступ

до адміністративного інтерфейсу для подальшого наповнення бази об'єктами кінотеатрів, конфігурування залів та асинхронного імпорту фільмів.

Під час переведення програмного комплексу в режим промислової експлуатації клієнтський застосунок оптимізують за допомогою команди `prun build`. Вона мінімізує обсяг кінцевого коду й генерує набір статичних файлів, готових до розгортання. Для захисту серверного API та забезпечення шифрування трафіку в обов'язковому порядку налаштовується зворотний проксі-сервер із конфігурацією SSL-сертифікатів, а також підключаються автономні сервіси щоденного резервного копіювання та логування критичних станів системи.

3.2. Архітектура програмного комплексу та його розгортання

Взаємодія користувачів та адміністративного персоналу з функціональними модулями вебсистеми «StarPlex» реалізована через адаптивний односторінковий застосунок, розроблений на основі компонентного підходу бібліотеки React 19. Візуальна структура екранних форм спроектована з метою забезпечення максимальної швидкості відгуку на дії акторів, мінімізації кількості повних перезавантажень сторінок завдяки засобам клієнтської маршрутизації та надання інформативного доступу до керування медіаконтентом кіомережі [7, 20]. Наскрізним елементом усіх сторінок є верхній навігаційний заголовок, який містить інтерактивне меню вибору поточного міста та кінотеатру, посилання на загальнодоступну афішу, а для авторизованих користувачів динамічно відображає привітання, посилання на кабінет з аватаром профілю та кнопку деавторизації.

Адміністративна частина вебсистеми адаптована під ієрархічну модель доступу і розгортається починаючи з модулів глобального конфігурування організаційної структури підприємства кінопрокату, доступних суперадміністратору мережі. Первинним етапом розширення та логістичного налаштування компанії є інтерфейс управління філіями, призначений для реєстрації нових кінотеатрів (рис. 3.2).

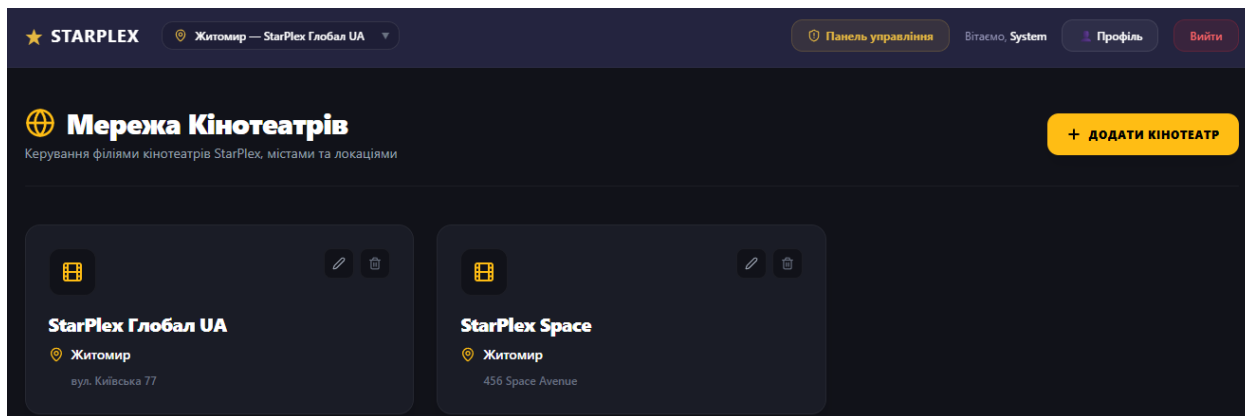


Рис. 3.2. Сторінка додавання та редагування філій кіномережі

На наведеному графічному представленні (рис. 3.2) відображено інтерфейс суперадміністратора, призначений для моніторингу наявних об'єктів кіномережі. Компонент забезпечує відображення списку філій із фіксацією їхніх назв, міст розташування та адрес. Кожен елемент інтерфейсу містить функціональні блоки для переходу до налаштування залів, редагування параметрів запису або каскадного видалення сутності з бази даних.

Для внесення до реєстру нової філії передбачено активацію елемента «Додати кінотеатр», що викликає відповідне вікно (рис. 3.3).

Рис. 3.3. Модальне вікно додавання нового кінотеатру

Модальне вікно (рис. 3.3) містить поля введення текстових метаданих: комерційної назви, міста та повної адреси локації. При підтвердженні операції клієнтська частина формує асинхронний POST-запит до серверної частини застосунку, обробка якого забезпечує додавання нового запису до таблиці Cinemas СКБД PostgreSQL. Після ініціалізації об'єктів мережі виконується розподіл прав доступу персоналу для операційного обслуговування філій за допомогою модуля адміністрування користувачів (рис. 3.4).

КОРИСТУВАЧ	EMAIL	ПРИВ'ЯЗКА ДО ЛОКАЦІЇ	ПОТОЧНА РОЛЬ	ДІЇ
Закусило Володимир	vova.zakusylo124@gmail.com	Житомир — StarPlex Глобал UA	МЕНЕДЖЕР ФІЛІЇ	НАЛАШТУВАТИ
Zakusylo Volodymyr	testusen@starplex.com	Житомир — StarPlex Space	МЕНЕДЖЕР ФІЛІЇ	НАЛАШТУВАТИ
Zakusylo Volodymyr	vova.zakusylo@gmail.com	Житомир — StarPlex Space	МЕНЕДЖЕР ФІЛІЇ	НАЛАШТУВАТИ
Закусило Володимир	1987@gmail.com	Глобальний доступ	КЛІЄНТ	НАЛАШТУВАТИ
Surname User	exmpl@gmail.com	Глобальний доступ	КЛІЄНТ	НАЛАШТУВАТИ
Zakusylo Vova	vova@gmail.com	Глобальний доступ	КЛІЄНТ	НАЛАШТУВАТИ
Administrator System	admin1@starplex.com	Глобальний доступ	ГЛОБАЛЬНИЙ АДМІН	НАЛАШТУВАТИ

Рис. 3.4. Сторінка управління персоналом у системі «StarPlex»

Інтерфейсний компонент (рис. 3.4) реалізує взаємодію з підсистемою безпеки Microsoft ASP.NET Core Identity. У табличній формі відображаються ідентифікаційні дані співробітників, їхні електронні адреси, поточні системні ролі, а також логічна прив'язка до конкретного кінотеатру за допомогою зовнішнього ключа CinemaId.

Зміна повноважень та переприв'язка працівника до іншої локації здійснюється через виклик окремого модального вікна (рис. 3.5), дані з якого проходять обов'язкову серверу валідацію на рівні бізнес-правил перед оновленням транзакцій у СКБД.

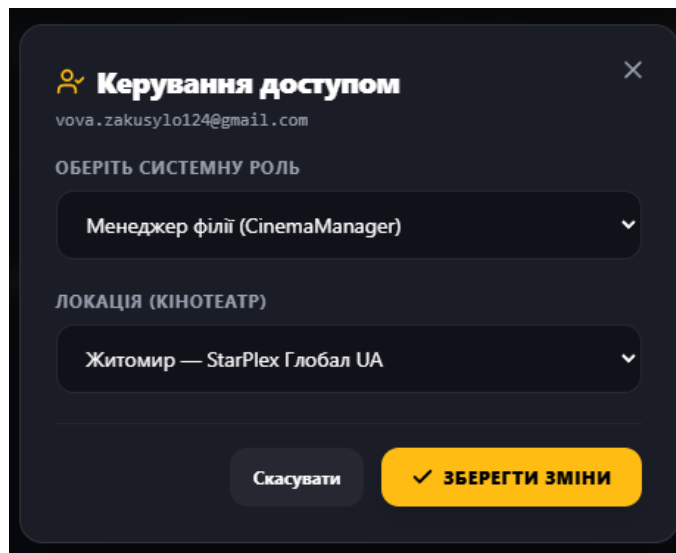


Рис. 3.5. Вікно налаштування доступу у системі «StarPlex»

Вікно налаштування доступу містить випадні списки для вибору системної ролі в ієрархії Identity та призначення цільового кінотеатру мережі. Після збереження змін серверний обробник оновлює атрибути (claims) користувача в базі даних підсистеми автентифікації, що забезпечує формування оновленого набору прав при наступній генерації JWT-токена.

Наступним інфраструктурним рівнем системи є формування єдиної медіатеки фільмів, доступної для планування розкладу в усіх підключених філіях мережі (рис. 3.6).

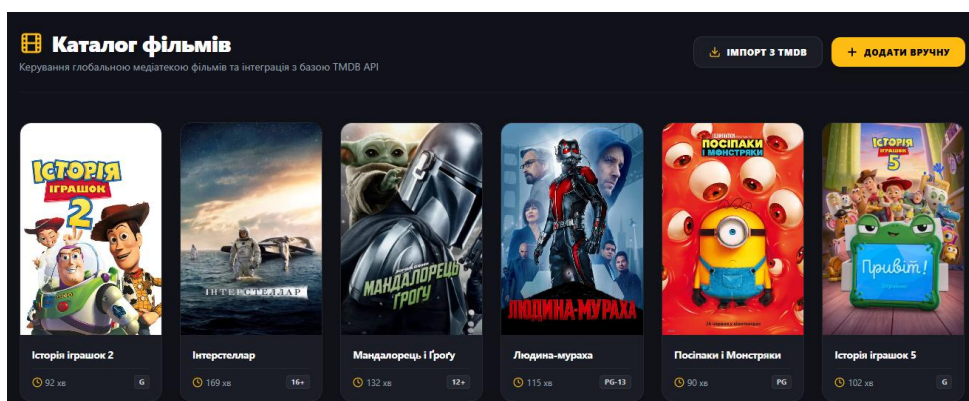


Рис. 3.6. Глобальний каталог фільмів у модулі суперадміністратора

Цей модуль керування дозволяє переглядати список наявних у базі даних фільмів у вигляді карток із відображенням постерів, оригінальних назв та вікових рейтингів. Для оптимізації процесу наповнення бази даних у системі реалізовано інтеграцію із зовнішнім сервісом TMDB API, доступ до якої здійснюється через активацію елемента «Імпорт з TMDB» (рис. 3.7).

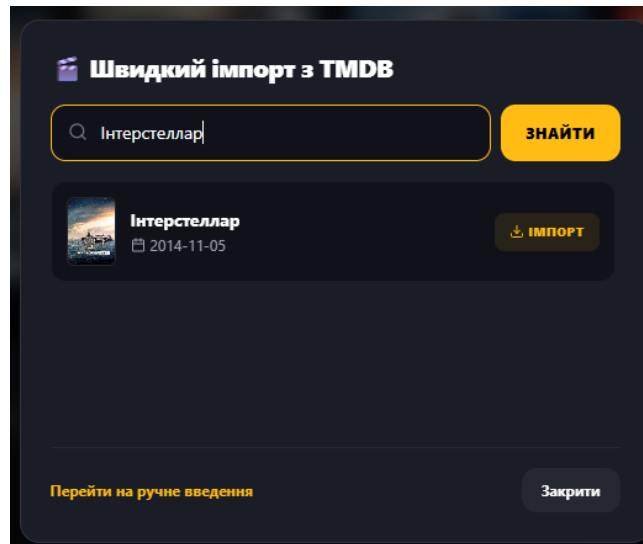


Рис. 3.7. Інтерфейс імпорту метаданих фільмів через TMDb API

Компонент імпорту забезпечує інтерактивний пошук медіаконтенту за текстовим запитом у сторонньому репозиторії. Після вибору цільової стрічки система надсилає асинхронний запит до API, автоматично отримує метадані та заповнює ними відповідні поля без необхідності ручного введення.

Для випадків, коли медіаконтент відсутній у глобальних базах даних або потребує специфічного ручного коригування, передбачено окреме вікно додавання (рис. 3.8).

Рис. 3.8. Вікно ручного додавання та налаштування параметрів фільму

Вікно додавання кінострічки містить структурований набір полів для точного налаштування сутності фільму перед збереженням у реляційному сховищі. Функціонал інтерфейсу передбачає введення назви, вибір жанрових тегів, визначення тривалості, вікового обмеження, дати початку прокату, посилання на офіційний YouTube-трейлер, а також завантаження фонового зображення та постера. Після підтвердження операції валідований об'єкт передається на серверний ендпоінт для додавання запису до таблиці Movies.

Формування єдиного каталогу фільмів дозволяє перейти до безпосереднього конфігурування фізичних ресурсів конкретних філій – кінозалів та їхньої геометрії (рис. 3.9).

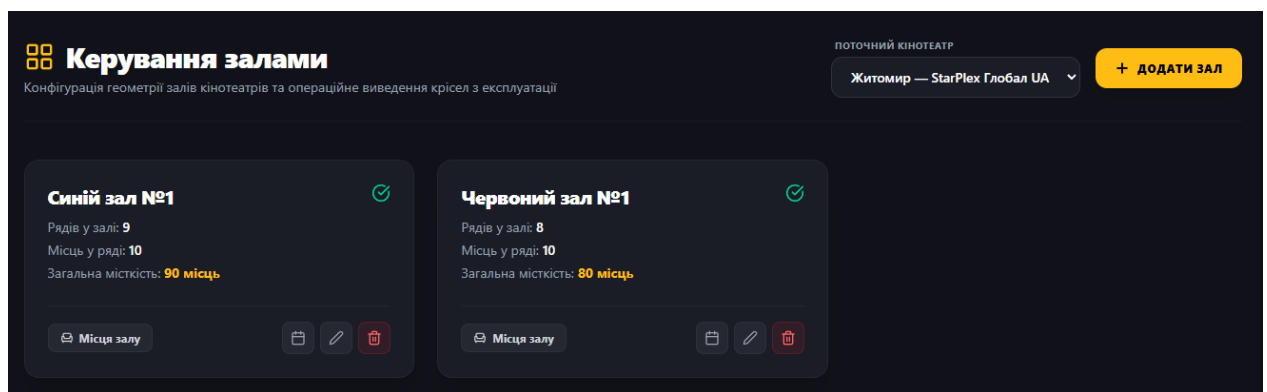


Рис. 3.9. Сторінка управління залами кінотеатру

На наведеній сторінці (рис. 3.9) відображено інтерфейс керування фізичними ресурсами кінотеатру. Локальний менеджер філії взаємодіє виключно з об'єктами свого підрозділу, тоді як суперадміністратор володіє глобальними повноваженнями. Для суперадміністратора інтерфейс розширюється випадним меню, що дозволяє динамічно перемикатися між будь-якими кінотеатрами мережі для моніторингу їхнього стану. Компонент візуалізує картки залів із зазначенням кількості рядів, місць, загальної місткості та індикатора технічної доступності локації для призначення сеансів.

Для детального налаштування схеми залу та оперативного керування категоріями місць передбачено перехід до інтерактивного графічного модуля (рис. 3.10).

На графічному таймлайні представлено добову сітку розкладу сеансів у розрізі окремих залів кінотеатру. Цей інтерфейс демонструє роботу алгоритмів автоматичного виявлення часових колізій. При додаванні або переміщенні сеансу система зчитує тривалість фільму з бази даних і обов’язково закладає 20-хвилинний інтервал для технічної підготовки залу персоналом. Спроби накладання часових слотів блокуються на рівні серверної валідації. Локальний менеджер керує розкладом лише для своєї філії, тоді як суперадміністратор має доступ до редагування таймлайнів будь-якої філії мережі.

Для мінімізації рутинної роботи персоналу при довгостроковому плануванні репертуару в системі реалізовано інструментарій автоматизації розкладу (рис. 3.12).

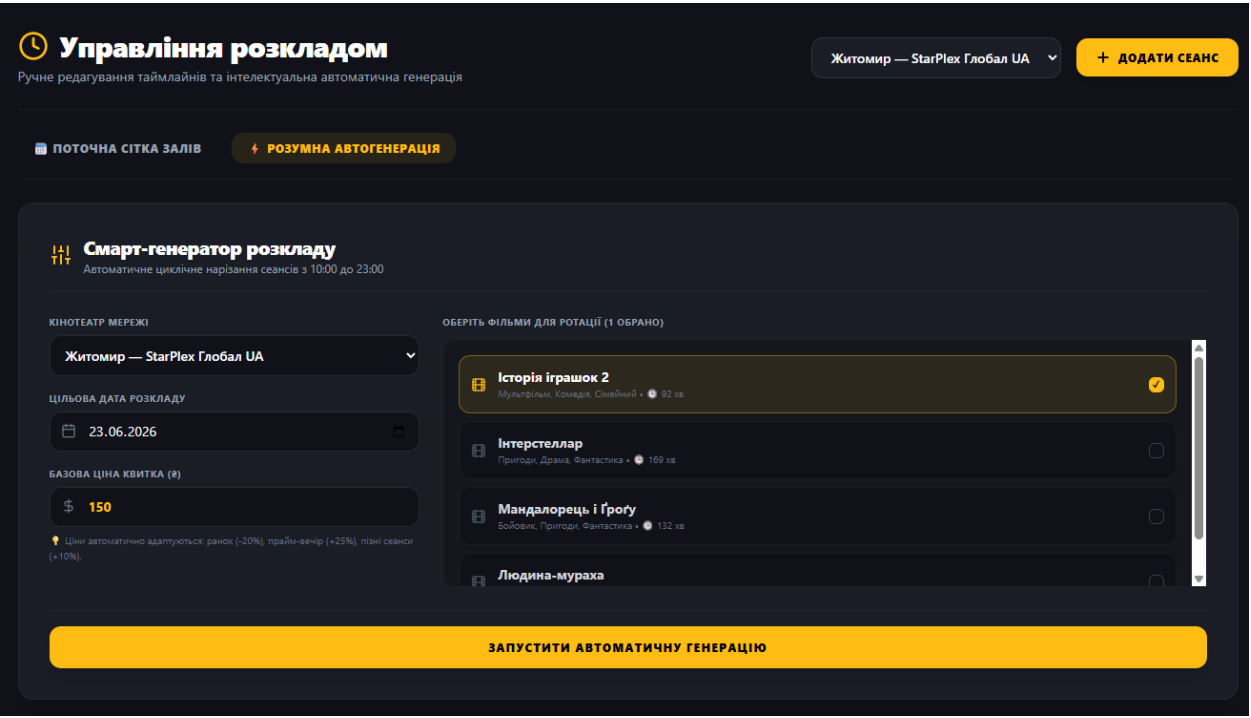


Рис. 3.12. Сторінка автогенерації розкладу

Інтерфейс генератора призначений для автоматичного циклічного формування розкладу сеансів на цільову дату за визначеними критеріями. Користувач обирає фільми для ротації, вказує базову вартість квитка та запускає алгоритм автогенерації. Серверна частина самостійно заповнює добовий таймлайн залів з 10:00 до 23:00, автоматично розраховує час початку подій, додає технологічні паузи на прибирання, а також адаптує ціноутворення. Відповідно до розподілу прав доступу, менеджер філії

обмежений своєю локацією, тоді як суперадміністратор може запускати автогенерацію для будь-якого кінотеатру мережі, використовуючи випадний список вибору філій.

Після формування розкладу сеансів та конфігурування геометрії залів система забезпечує повний спектр інструментів для безпосередньої реалізації квитків, що виконується через спеціалізований інтерфейс касира кінотеатру (рис. 3.13).

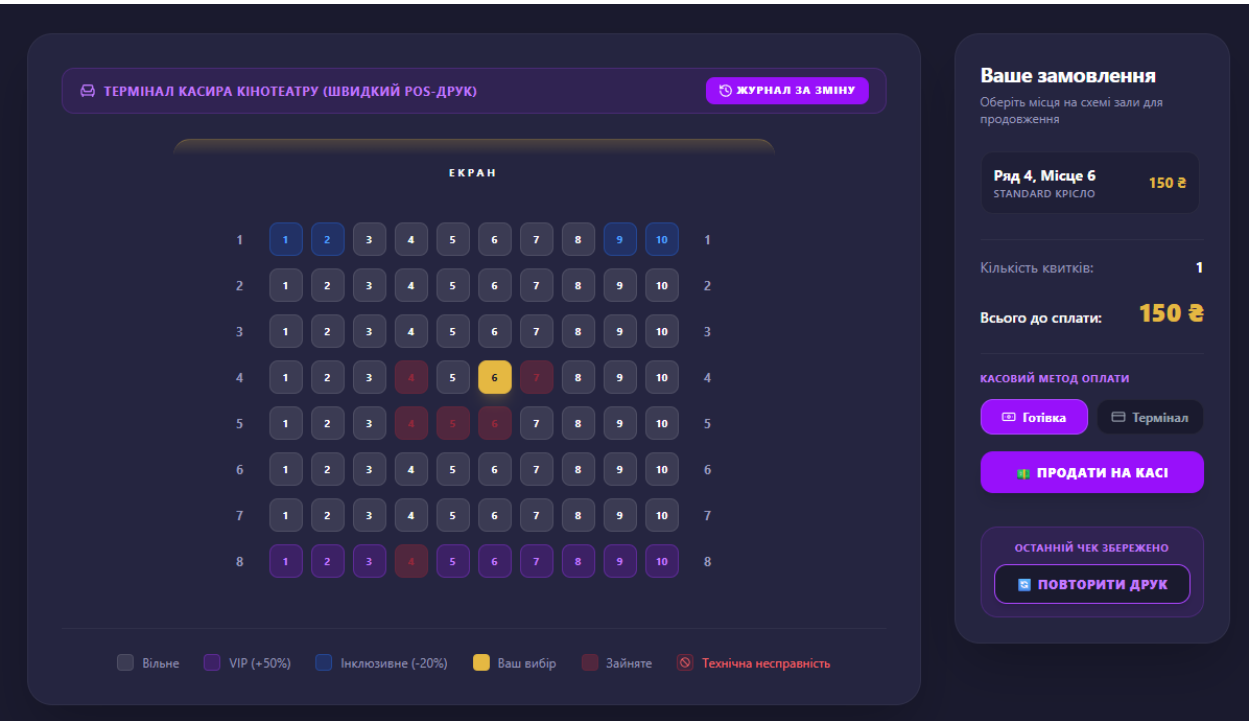


Рис. 3.13. Інтерфейс термінала касира філії для офлайн-продажів

Робоче місце касира-контролера оптимізоване для швидкого обслуговування черги на касі кінотеатру. Компонент підтримує безперервне з’єднання через SignalR-хаб, завдяки чому на схемі залу синхронізуються та блокуються для вибору місця, що вже зарезервовані або куплені іншими користувачами онлайн. Бічна панель «Ваше замовлення» автоматично розраховує підсумкову вартість квитків залежно від обраної категорії місця, надає можливість вибору локального методу оплати та містить функціональний елемент для прямої фіксації транзакції в обхід зовнішнього інтернет-еквайрингу Stripe з можливістю повторного друку останнього збереженого чека.

Для суворого обліку розрахункових операцій та можливості оперативного скасування транзакцій у системі реалізовано модуль локального аудиту продажів (рис. 3.14).

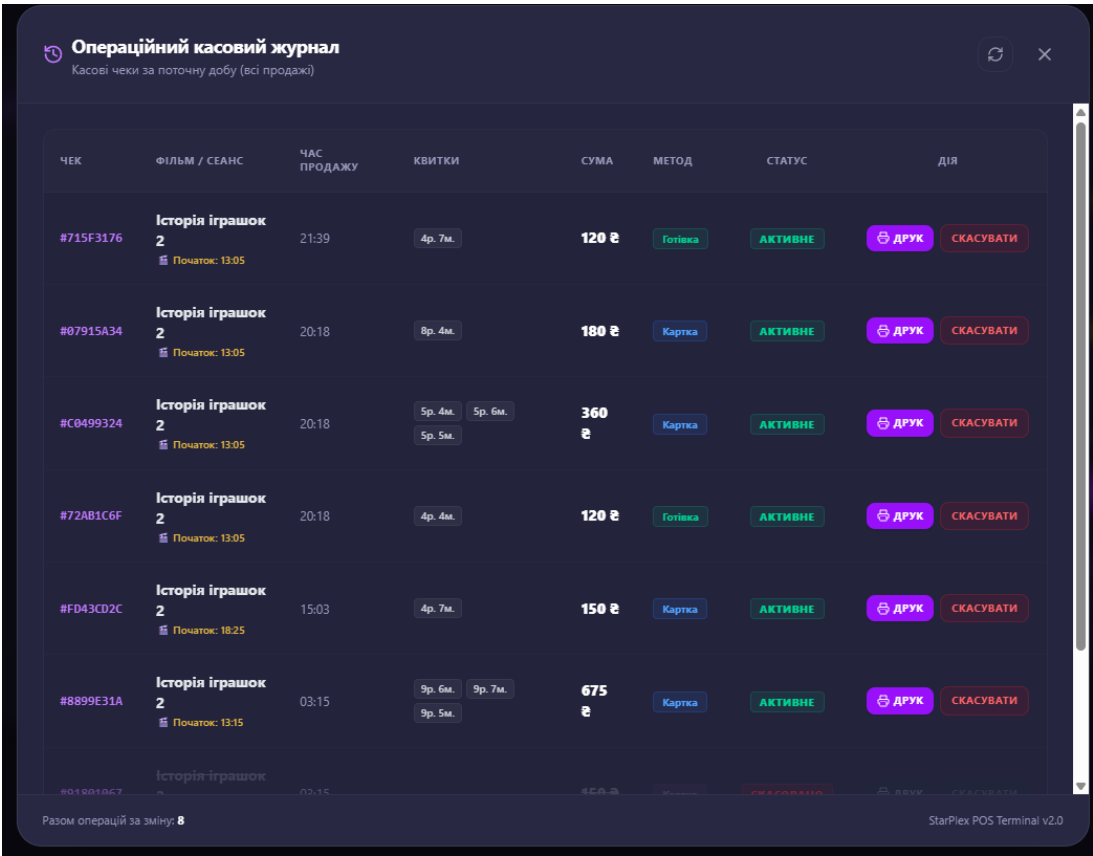


Рис. 3.14. Касовий журнал реєстрації та скасування розрахункових операцій

Інтерфейс касового журналу відображає деталізовану таблицю всіх фінансових операцій, проведених на території поточної філії за зміну. Для кожної транзакції фіксується унікальний буквено-цифровий ідентифікатор чека, назва фільму, час продажу, точні координати місць у залі, підсумкова сума та фінансовий статус. Функціональний блок «Дія» дозволяє касиру миттєво відправити квиток на повторний друк або ініціювати процедуру скасування операції за запитом клієнта, що автоматично анулює запис в таблиці Bookings СКБД PostgreSQL і повертає місцям статус вільних для вибору.

Організація внутрішніх операційних процесів філій та наповнення репертуарної сітки дозволяє сформувати зовнішній інформаційний шар системи, орієнтований на взаємодію з кінцевим споживачем послуг (рис. 3.15).

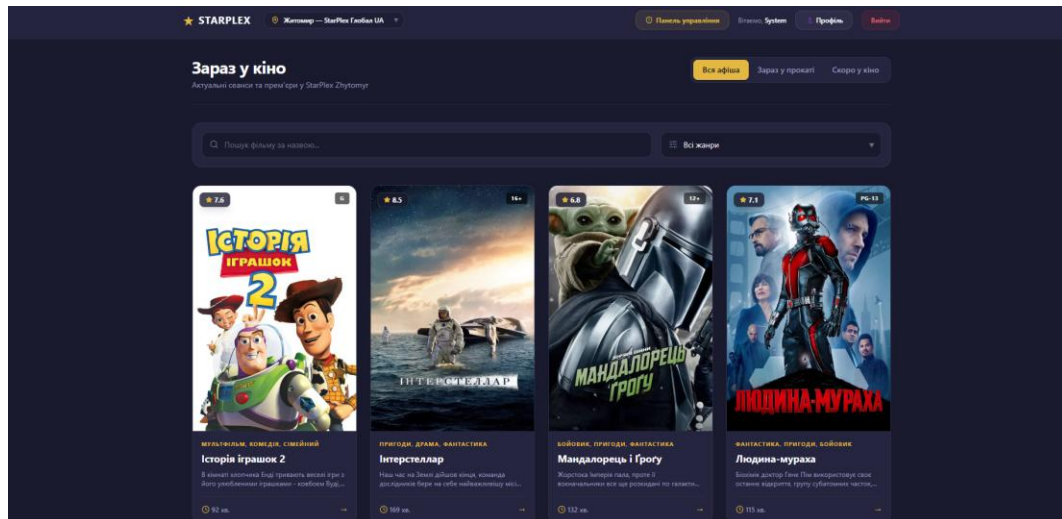


Рис. 3.15. Головна сторінка та інтерфейс поточної афіші

Публічний інтерфейс застосунку виконує функцію цифрової вітрини мережі кінотеатрів. Компонент «Зараз у кіно» динамічно наповнюється даними через надсилання асинхронного GET-запиту до серверного Web API. Отриманий масив об'єктів фільмів візуалізується у вигляді карток медіаконтенту, що містять локалізовані постери, назви, текстові описи сюжетів, тривалість стрічок, мітки вікових рейтингів та числові оцінки. У верхній частині форми розташовані інструменти швидкого пошуку за назвою та елемент фільтрації за жанрами. Фільтрація реалізована на основі стану React, що забезпечує оновлення інтерфейсу без повного перезавантаження DOM-дерева [27].

Для отримання детальної інформації щодо конкретної кінострічки та вибору сеансу користувач переходить до інформаційного модуля (рис. 3.16).

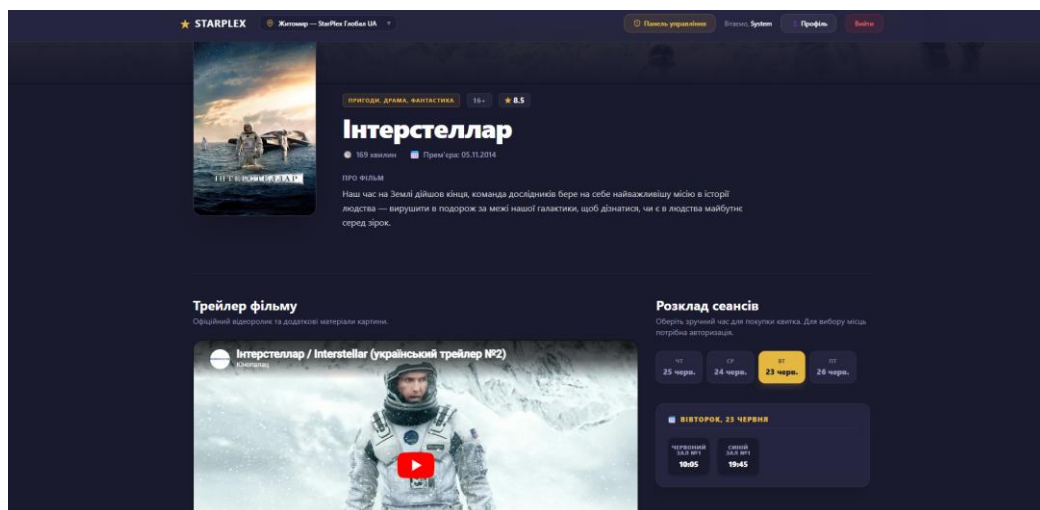


Рис. 3.16. Інформаційна картка фільму з модулем вибору сеансів

На цій сторінці відображаються розширені метадані фільму, інтегрований плеєр для перегляду офіційного трейлера з платформи YouTube та календарний модуль розкладу сеансів у розрізі кінозалів обраної філії. Календарна стрічка забезпечує можливість перемикання між днями тижня, ініціюючи динамічну фільтрацію списку подій на стороні клієнта. Сеанси відображають точний час початку показу, назви залу та вартості квитка. Кожна часова мітка є активним маршрутом, який перенаправляє авторизованого користувача до транзакційного контуру координатної сітки залу для онлайн-бронювання та оплати через платіжний шлюз Stripe.

Процес безготівкового розрахунку за зарезервовані місця реалізовано шляхом інтеграції з міжнародним платіжним шлюзом (рис. 3.17).

Рис. 3.17. Інтерфейс онлайн-оплати замовлення через платіжний шлюз Stripe

Інтерфейс оплати інтегрований у застосунок за допомогою компонента Stripe Elements. Вікно відображає суму до сплати, інформацію про замовлення та поля для введення реквізитів банківської картки. Після ініціалізації транзакції Stripe обробляє платіж та асинхронно надсилає вебхук на серверний Web API. Отриманий запит фіксує успішний статус транзакції в таблиці Payments і остаточно закріплює обрані місця за користувачем.

Дані про всі транзакції, здійснені через платіжний шлюз Stripe або локальні касові термінали філій, автоматично агрегуються системою. Ці відомості використовуються для розрахунку аналітичних метрик та формування звітності у головному модулі статистики (рис. 3.18).

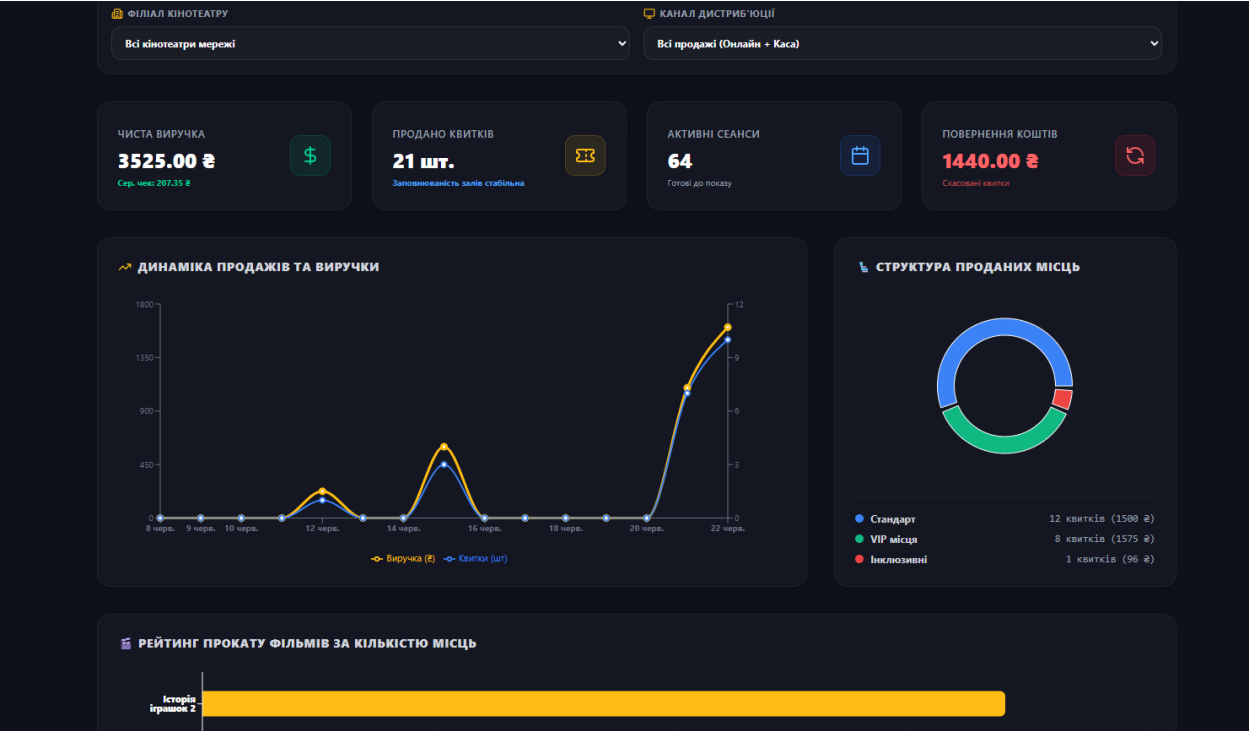


Рис. 3.18. Панель моніторингу зведеної бізнес-аналітики мережі кінотеатрів

Модуль аналітики суперадміністратора призначений для консолідації та візуалізації накопичених статистичних даних. На основі інформації з бази даних та за допомогою компонентів бібліотеки Recharts інтерфейс відображає обсяг загального виторгу, кількість проданих квитків і середній чек. Лінійний графік візуалізує помісячну динаміку доходів, а кругова діаграма демонструє розподіл куплених місць за категоріями комфорту. Розроблений інструментарій забезпечує можливість аналізу фінансових показників, оцінки завантаженості філій та гнучкого налаштування тарифної політики мережі.

3.3. Тестування роботи програмного комплексу

Тестування розробленого програмного комплексу є фундаментальною стадією життєвого циклу інженерної розробки, спрямованою на виявлення дефектів, перевірку стійкості бізнес-правил ядра та підтвердження повної відповідності вебсистеми висунутим функціональним вимогам. Для комплексного аналізу працездатності централізованої вебсистеми «StarPlex»

було застосовано ручний метод верифікації шляхом послідовного проєктування та виконання структурованих тест-кейсів. Кожен тест-кейс є формалізованою матрицею, яка фіксує передумови, чіткий вектор дій розробника, очікувані системні реакції на рівні клієнт-серверного обміну та фактично отриманий результат. Сформований пул перевірок охоплює всі критичні інформаційні конвеєри та рівні безпеки розробленої вебсистеми [23].

Перший вектор верифікації спрямований на контроль цілісності шару безпеки, аналіз захисту закритих клієнтських маршрутів (Protected Routes) та перевірку механізмів авторизації на базі токенів JWT, що відображено у таблиці 3.1.

Таблиця 3.1

Тест-кейс 1 - Верифікація механізмів захисту маршрутів та JWT-авторизації

№	Послідовність інженерних кроків	Очікувана реакція системи	Отриманий результат	Статус
1	Пряме введення захищеного шляху особистого кабінету без активного токена	Перехоплення запиту Protected Route компонентом і перенаправлення на авторизацію	Як і очікувалось	Виконано
2	Спроба неавторизованого переходу до панелі управління менеджера філії	Автоматичне блокування доступу клієнтським роутером	Як і очікувалось	Виконано
3	Реєстрація нового користувача з передачею незаповненої форми	Спрацювання валідаторів форми, блокування відправки POST-запиту	Як і очікувалось	Виконано
4	Реєстрація профілю на електронну адресу, яка вже зафіксована у реляційній базі даних	Сервер повертає помилку Identity з повідомленням про дублювання логіна	Як і очікувалось	Виконано
5	Авторизація за допомогою валідних облікових даних адміністратора	Успішна обробка запиту, запис claims у контекст, розблокування доступу до панелі суперадміністратора	Як і очікувалось	Виконано

Другий блок перевірок покриває інструменти суперадміністратора щодо автоматизованого імпорту контенту із зовнішніх репозиторіїв та обмеження завантаження статичних файлів для формування єдиної медіатеки мережі, що наведено у таблиці 3.2.

Таблиця 3.2

Тест-кейс 2 – Тестування модулів імпорту медіаданих з TMDb та файлових фільтрів

№	Послідовність інженерних кроків	Очікувана реакція системи	Отриманий результат	Статус
1	Надсилання команди асинхронного імпорту фільму за унікальним кодом ресурсу TMDb	Сервер робить REST-запит до API TMDb, завантажує постери й описи українською, створює картку фільму	Як і очікувалось	Виконано
2	Спроба завантаження файлу обкладинки постера у невідповідному системному форматі	Проміжний обробник блокує завантаження на етапі перевірки розширення, повертаючи помилку	Як і очікувалось	Виконано
3	Спроба завантаження графічного файлу постера, фізичний розмір якого перевищує ліміт у 5 Мбайт	Сервер перериває обробку форми, скасовує запис і сповіщає про перевищення максимального ліміту	Як і очікувалось	Виконано

Третій етап верифікації спрямований на контроль логіки планування розкладу, перевірку серверних алгоритмів автогенерації сеансів та тестування ізоляції даних менеджерів філій, що зафіксовано у таблиці 3.3.

Таблиця 3.3

Тест-кейс 3 – Тестування бізнес-правил планування розкладу, валідації колізій та автогенерації сеансів

№	Послідовність інженерних кроків	Очікувана реакція системи	Отриманий результат	Статус
1	Спроба створення менеджером сеансу на час, який повністю або частково перетинається з іншим фільмом у цьому залі	Серверний CQRS-обробник блокує команду, сеанс не додається до бази даних	Як і очікувалось	Виконано
2	Створення сеансу на час, який формально вільний, але не враховується обов'язковий 20-хвилинний інтервал прибирання залу	Алгоритм додає тривалість фільму та CleanUp-інтервал, виявляє колізію з часом початку наступної події та відхиляє запис	Як і очікувалось	Виконано

№	Послідовність інженерних кроків	Очікувана реакція системи	Отриманий результат	Статус
3	Ініціалізація модуля циклічної автогенерації репертуарного плану менеджером на обрану майбутню дату	Серверний алгоритм автоматично заповнює таймлайн залів, розраховує час початку сеансів, додає паузи та генерує масив записів	Як і очікувалось	Виконано
4	Автентифікація в системі під профілем локального менеджера філії та спроба перегляду розкладу іншого міста	Глобальний інфраструктурний фільтр за CinemaId обмежує SQL-запити, відображаючи зали та сеанси лише поточної філії	Як і очікувалось	Виконано

Четвертий блок перевірок охоплює тестування інструментів адміністрування мережі, зокрема верифікацію процесів додавання нових філій до структури системи, динамічного розмежування прав доступу персоналу та коректності агрегації фінансових показників на модулі статистики, що наведено в таблиці 3.4.

Таблиця 3.4

Тест-кейс 4 – Тестування інструментів глобального адміністрування, управління персоналом та бізнес-аналітики

№	Послідовність інженерних кроків	Очікувана реакція системи	Отриманий результат	Статус
1	Ініціалізація створення нової філії в системі суперадміністратором через модальне вікно конфігурування	Успішна валідація полів та запис нової сутності до таблиці Cinemas СКБД PostgreSQL	Як і очікувалось	Виконано
2	Динамічна зміна системної ролі користувача на CinemaManager та його прив'язка до створеного ідентифікатора філії	Серверна частина актуалізує перелік claims у базі даних підсистеми автентифікації, ізолюючи доступ менеджера межами вказаної філії	Як і очікувалось	Виконано
3	Спроба локального менеджера філії отримати доступ до інтерфейсів глобального моніторингу або дашборду іншого підрозділу	Проміжний обробник безпеки блокує запит на етапі валідації токена, повертаючи клієнту код помилки 403 Forbidden	Як і очікувалось	Виконано

№	Послідовність інженерних кроків	Очікувана реакція системи	Отриманий результат	Статус
4	Агрегація нових фінансових транзакцій з усіх підключених POS-терміналів та онлайн-шлюзів у центральному аналітичному контурі	Компоненти бізнес-аналітики React/Recharts перераховують загальну виручку мережі, середній чек та пропорційний розподіл попиту	Як і очікувалось	Виконано

П'ятий етап присвячено аналізу процесів синхронізації розподілених ресурсів у реальному часі через SignalR-хаби під час паралельної взаємодії користувачів, що наведено у таблиці 3.5.

Таблиця 3.5

Тест-кейс 5 – Тестування SignalR-конвеєра бронювання місць на схемі залу

№	Послідовність інженерних кроків	Очікувана реакція системи	Отриманий результат	Статус
1	Вибір вільного місця на координатній сітці залу в першому браузері авторизованого клієнта	Створення запису в БД, забарвлення місця в жовтий колір, активація 10-хвилинного таймера	Як і очікувалось	Виконано
2	Перевірка стану аналогічної карти залу у другому незалежному вікні браузера іншого користувача	SignalR-хаб транслює подію, місце миттєво забарвлюється в сірий колір для всіх учасників кімнати	Як і очікувалось	Виконано
3	Спроба другого користувача натиснути на утримуване поточним клієнтом жовте місце	Локальний стан React блокує клік	Як і очікувалось	Виконано
4	Штучне розірвання мережевого з'єднання у першого користувача (Емуляція Offline) під час активного утримання місця	SignalR автоматично ініціює відновлення з'єднання у разі збою, тоді як поточний стан бронювання зберігається в базі даних до відновлення сесії користувача.	Як і очікувалось	Виконано
5	Очікування закінчення 10-хвилинного ліміту утримання броні без переходу користувача до кроку оплати	Фоновий сервіс .NET видаляє прострочене блокування з PostgreSQL, SignalR повертає місцям сірий колір	Як і очікувалось	Виконано

Шостий тест-кейс спрямований на перевірку фінансового контуру вебсистеми, коректності інтеграції захищеного API-інструментарію Stripe та обробки асинхронних платіжних вебхуків, що зафіксовано у таблиці 3.6.

Таблиця 3.6

Тест-кейс 6 – Верифікація підсистеми онлайн-еквайрингу Stripe та випуску квитків

№	Послідовність інженерних кроків	Очікувана реакція системи	Отриманий результат	Статус
1	Свідоме введення некоректних або тестових реквізитів картки з недостатнім балансом у Stripe Elements	Фрейм Stripe повертає помилку транзакції, статус замовлення в базі залишається у стані Pending	Як і очікувалось	Виконано
2	Проведення оплати за заброньовані місця за допомогою валідної тестової картки Stripe	Шлюз Stripe успішно списує кошти та надсилає підписаний Webhook на серверний ендпоінт Web API	Як і очікувалось	Виконано

У ході проведення комплексного ручного тестування було успішно перевірено всі найважливіші операційні та інфраструктурні сценарії функціонування централізованої вебсистеми «StarPlex». За результатами виконання спроектованих тест-кейсів не виявлено жодних розбіжностей між очікуваними реакціями ядра системи та фактично отриманими відповідями клієнтської і серверної частин програмного комплексу. Сформована верифікаційна матриця доводить, що розроблена система працює стабільно, повністю відповідає критеріям надійності, захищена від логічних помилок конкурентного доступу та готова до промислової експлуатації [24].

Висновки до третього розділу

У третьому розділі випускної кваліфікаційної роботи було успішно виконано етап практичної реалізації, інфраструктурного розгортання та комплексної верифікації централізованої вебсистеми «StarPlex». На основі розроблених архітектурних рішень було побудовано та впроваджено діаграму розгортання UML, яка формалізувала фізичний розподіл компонентів системи на апаратному рівні. Обґрунтовано та реалізовано сучасну контейнеризовану модель розгортання в ізольованих Docker-середовищах, де взаємодія між

вебсервером Kestrel під керуванням ASP.NET Core Web API, реляційною СКБД PostgreSQL 16 та інфраструктурним поштовим сервером Mailpit відбувається всередині замкненої віртуальної мережі. Реалізовано автоматичне налаштування параметрів безпеки та генерації ролей, що дало змогу виключити ручне втручання в структуру бази даних під час первинного розгортання інфраструктури.

На основі компонентного підходу бібліотеки React 19 та інструментарію TypeScript описано структури адміністративних, операційних та користувацьких інтерфейсів системи. У роботі наведено опис функціонування екранних форм фінансової статистики на базі бібліотеки Recharts, а також модулів керування філіями та мережею кінотеатрів. Окрему увагу приділено інтерактивним графічним конфігураторам місць кінозалів та засобам автоматичної генерації репертуарної сітки. Також представлено інтерфейси термінала касира й журналу реєстрації чеків для офлайн-продажів, які функціонують у зв'язці з платіжним шлюзом Stripe.

Фінальним етапом роботи став аналіз результатів верифікації програмного комплексу шляхом розробки та виконання шести структурованих тест-кейсів ручного тестування. Створені верифікаційні матриці дозволили всебічно перевірити працездатність механізмів захисту маршрутів та модулів автоматичного імпорту медіаданих із зовнішніх API. Також перевірено логіку автогенерації сеансів, SignalR-трансляцію утримання місць різними користувачами, точність відсікання часових колізій і технологічних пауз при плануванні розкладу філій. Результати тестування довели повну відповідність фактичних реакцій системи очікуваній поведінці її програмної логіки.

ВИСНОВКИ

У випускній кваліфікаційній роботі було повністю вирішено актуальне науково-практичне завдання, що полягало в аналізі, проєктуванні, програмній реалізації та верифікації централізованої вебсистеми «StarPlex». Розроблена система призначена для автоматизації процесів операційного керування розгалуженою мережею кінотеатрів. У ході виконання роботи було досягнуто поставленої мети та успішно реалізовано всі інфраструктурні й операційні вимоги шляхом впровадження сучасного стеку технологій. Спроектowana архітектурна модель дозволила об'єднати гнучкі інтерфейсні рішення на базі клієнтського фреймворку з типізованою серверною екосистемою, сформувавши масштабований інструмент для автоматизації бізнес-процесів сучасної індустрії кінопрокату.

Під час теоретичного та аналітичного етапів дослідження було детально формалізовано функціональні межі системи за допомогою побудови та опису специфікацій діаграм варіантів використання UML для чотирьох ключових акторів, із фокусом на рольові моделі суперадміністратора та локального менеджера філії. Це дозволило чітко розмежувати сценарії взаємодії та спроектувати концептуальну схему бази даних у СКБД PostgreSQL, яка пройшла повний цикл нормалізації для усунення критичних аномалій надлишковості даних. Впровадження конфігурацій Fluent API на рівні об'єктно-реляційного мапера Entity Framework Core забезпечило точність фінансових розрахунків і високу швидкість виконання пошукових операцій завдяки оптимізації B-Tree індексів, а також гарантувало ізоляцію паралельних транзакцій на рівні СКБД через налаштування складених унікальних обмежень.

Практична реалізація серверної частини вебсистеми була успішно виконана на базі платформи .NET 8.0 Web API з дотриманням архітектурного шаблону Clean Architecture та поведінкового патерну CQRS за допомогою бібліотеки MediatR. Такий підхід дозволив повністю ізолювати REST-ендпоінти від ядра програмної логіки, спростивши подальше масштабування

системи. Було розроблено та впроваджено алгоритми запобігання часовим колізіям при плануванні розкладу з урахуванням тривалості фільмів та автоматичних пауз на прибирання залів. Інтеграція підсистеми безпеки Microsoft ASP.NET Core Identity та безстанційних JWT-токенів у зв'язці з каскадною фільтрацією за ідентифікатором кінотеатру забезпечила рольове розмежування доступу та захист інформації від несанкціонованого доступу.

Клієнтський рівень вебсистеми реалізовано як односторінковий застосунок на базі React 19, інструменту Vite та мови TypeScript. Впровадження технології SignalR для підтримки двосторонніх вебсокет-каналів дозволило організувати синхронізацію та візуалізацію станів посадкових місць на інтерактивній схемі залу в реальному часі. Проведення транзакцій, обробка безготівкових платежів та централізоване зведення аналітики реалізовані через інтеграцію платіжного шлюзу Stripe та інтерактивних дашбордів. Процес розгортання інфраструктури було повністю контейнеризовано за допомогою інструментів Docker Compose. Успішне виконання спроектованих тест-кейсів ручного тестування довело стабільність, відсутність логічних помилок та готовність вебсистеми «StarPlex» до впровадження й експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Troelsen A., Japikse P. Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming. 11th ed. Berkeley : Apress, 2022. 1705 p.
2. Martin R. C. Clean Architecture: A Craftsman's Guide to Software Structure and Design. Boston : Prentice Hall, 2017. 432 p.
3. Seemann M., van Deursen S. Dependency Injection Principles, Practices, and Patterns. Shelter Island : Manning Publications, 2019. 552 p.
4. Skeet J. C# in Depth. 4th ed. Shelter Island : Manning Publications, 2019. 528 p.
5. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. Sebastopol : O'Reilly Media, 2020. 422 p.
6. Flanagan D. JavaScript: The Definitive Guide. 7th ed. Sebastopol : O'Reilly Media, 2020. 706 p.
7. Banks A., Porcello E. Learning React: Modern Patterns for Developing React Apps. 2nd ed. Sebastopol : O'Reilly Media, 2020. 307 p.
8. Fowler M. Patterns of Enterprise Application Architecture. Boston : Addison-Wesley, 2002. 560 p.
9. Crouse C., Smith S. Clean Architecture with .NET. Birmingham : Packt Publishing, 2026. 490 p.
10. Ciolli G., Mejías B., Angelakos J., Kumar V., Riggs S. PostgreSQL 16 Administration Cookbook. Birmingham : Packt Publishing, 2023. 636 p.
11. Richardson C. Microservices Patterns: With examples in Java. Shelter Island : Manning Publications, 2018. 520 p.
12. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. Sebastopol : O'Reilly Media, 2021. 616 p.
13. Freeman A. Pro ASP.NET Core 7. 10th ed. Shelter Island : Manning Publications, 2023. 1256 p.
14. Price M. J. C# 12 and .NET 8 – Modern Cross-Platform Development Fundamentals. 8th ed. Birmingham : Packt Publishing, 2023. 826 p.
15. Esposito D., Saltarello A. Microsoft .NET: Architecting Applications for the Enterprise. 2nd ed. Redmond : Microsoft Press, 2014. 416 p.

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						67

16. Evans E. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston : Addison-Wesley, 2003. 560 p.
17. Kleppmann M. Designing Data-Intensive Applications. Sebastopol : O'Reilly Media, 2017. 614 p.
18. Ingebrigtsen E. SignalR Blueprints. Birmingham : Packt Publishing, 2015. 244 p.
19. Chinnathambi K. Learning React. 2nd ed. Boston : Addison-Wesley Professional, 2018. 304 p.
20. Accomazzo A., Murray N., Lerner A. Fullstack React: The Complete Guide to ReactJS and Friends. San Francisco : Fullstack.io, 2017. 836 p.
21. Poulton N. Docker Deep Dive. Birmingham : Packt Publishing, 2020. 250 p.
22. Schonig H. J. Mastering PostgreSQL 15: Advanced techniques to build and manage scalable, reliable, and fault-tolerant database applications. 5th ed. Birmingham : Packt Publishing, 2023. 522 p.
23. Aniche M. Effective Software Testing: A developer's guide. Shelter Island : Manning Publications, 2022. 328 p.
24. Whittaker J. A. Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design. Boston : Addison-Wesley, 2010. 256 p.
25. UML 2.5.1 Specification / Object Management Group. URL: <https://www.omg.org/spec/UML/2.5.1/About-UML/> (дата звернення: 10.05.2026).
26. Microsoft .NET Documentation / Microsoft Learn. URL: <https://learn.microsoft.com/en-us/dotnet/> (дата звернення: 15.05.2026).
27. React Documentation / React Dev. URL: <https://react.dev/> (дата звернення: 25.05.2026).
28. Stripe API Reference / Stripe Documentation. URL: <https://stripe.com/docs/api> (дата звернення: 01.06.2026).
29. PostgreSQL 16.0 Documentation / PostgreSQL Global Development Group. URL: <https://www.postgresql.org/docs/16/> (дата звернення: 15.05.2026).
30. OWASP Top 10:2021 / The Most Critical Web Application Security Risks. URL: <https://owasp.org/www-project-top-ten/> (дата звернення: 25.05.2026).

ДОДАТКИ

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						69

ER-діаграма

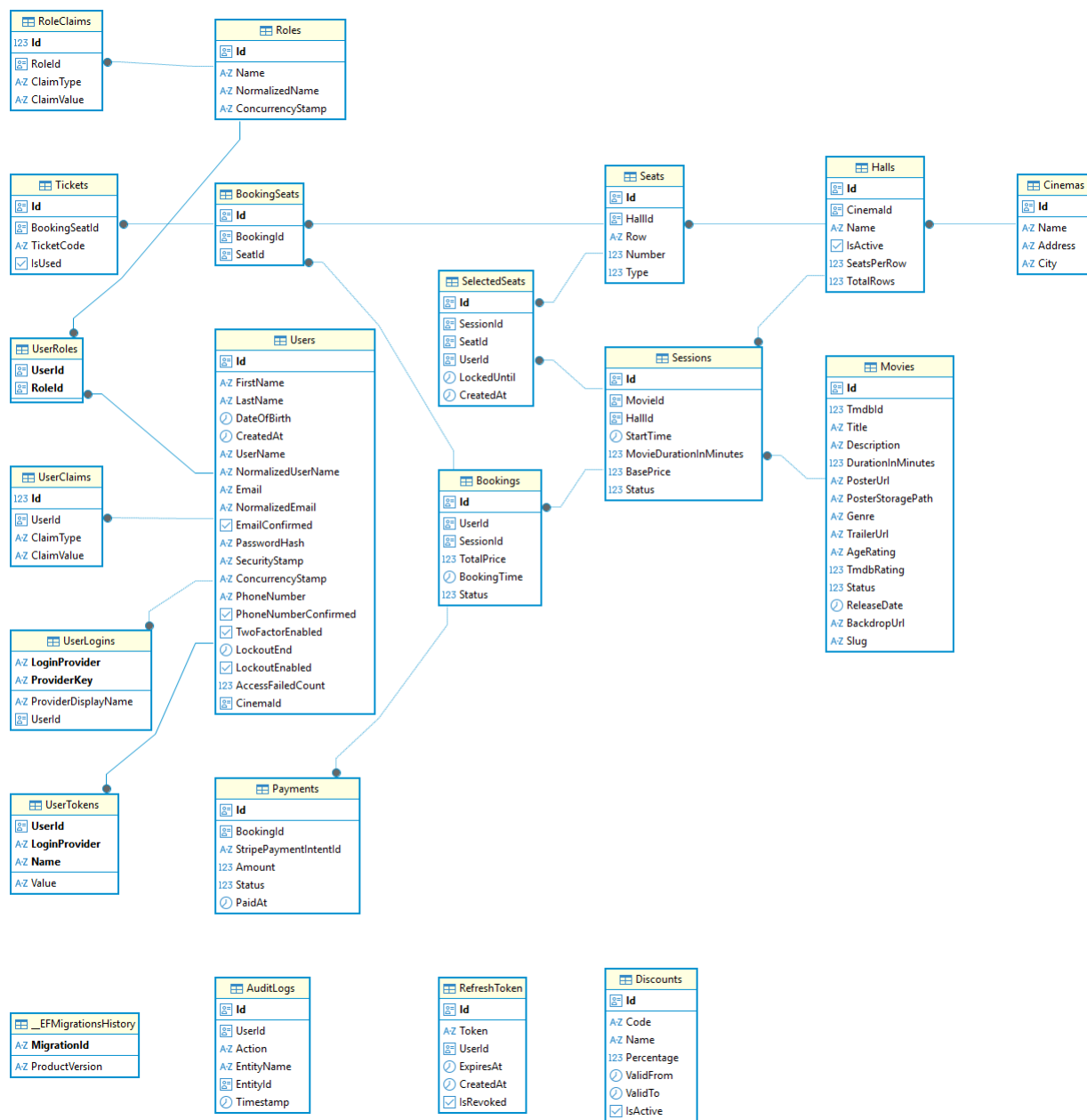


Рис. А.1. ER-діаграма

Лістинг програми RoleSeeder.cs

```

namespace StarPlex.Infrastructure.Identity;

public static class RoleSeeder
{
    public static async Task SeedDataAsync(
        RoleManager<IdentityRole<Guid>> roleManager,
        UserManager<ApplicationUser> userManager)
    {
        string[] roles = ["SuperAdmin", "CinemaManager", "Cashier", "Customer"];

        foreach (var roleName in roles)
        {
            if (!await roleManager.RoleExistsAsync(roleName))
            {
                await roleManager.CreateAsync(new IdentityRole<Guid>(roleName));
            }
        }

        var adminEmail = "admin1@starplex.com";
        var adminUser = await userManager.FindByEmailAsync(adminEmail);

        if (adminUser == null)
        {
            var defaultAdmin = new ApplicationUser(
                firstName: "System",
                lastName: "Administrator",
                dateOfBirth: new DateTime(1990, 1, 1)
            )
            {
                UserName = adminEmail,
                Email = adminEmail,
                EmailConfirmed = true
            };

            var createResult = await userManager.CreateAsync(defaultAdmin, "Admin123!");

            if (createResult.Succeeded)
            {
                await userManager.AddToRoleAsync(defaultAdmin, "SuperAdmin");
            }
        }
    }
}

```

Лістинг програми IdentityService.cs

```

namespace StarPlex.Infrastructure.Identity;

public class IdentityService : IIdentityService
{
    private readonly UserManager<ApplicationUser> _userManager;
    private readonly IApplicationDbContext _context;
    private readonly JwtSettings _jwtSettings;

    public IdentityService(
        UserManager<ApplicationUser> userManager,
        IApplicationDbContext context,
        IOptions<JwtSettings> jwtSettings)
    {
        _userManager = userManager;
        _context = context;
        _jwtSettings = jwtSettings.Value;
    }

    public async Task<bool> IsEmailUniqueAsync(string email)
    {

```

```

        var user = await _userManager.FindByEmailAsync(email);
        return user == null;
    }

    public async Task<AuthResponse?> RegisterAsync(string email, string password, string firstName, string lastName, DateTime
dateOfBirth)
    {
        if (!await IsEmailUniqueAsync(email))
            return null;

        var user = new ApplicationUser(firstName, lastName, dateOfBirth)
        {
            UserName = email,
            Email = email
        };

        var result = await _userManager.CreateAsync(user, password);
        if (!result.Succeeded)
            return null;

        await _userManager.AddToRoleAsync(user, "Customer");

        return await GenerateAuthResponseAsync(user);
    }

    public async Task<AuthResponse?> LoginAsync(string email, string password)
    {
        var user = await _userManager.FindByEmailAsync(email);
        if (user == null) return null;

        var isValid = await _userManager.CheckPasswordAsync(user, password);
        if (!isValid) return null;

        return await GenerateAuthResponseAsync(user);
    }

    public async Task<AuthResponse?> RefreshTokenAsync(string refreshTokenStr)
    {
        var storedToken = await _context.RefreshTokens
            .FirstOrDefaultAsync(x => x.Token == refreshTokenStr);

        if (storedToken == null || !storedToken.IsActive)
            return null;

        storedToken.IsRevoked = true;
        await _context.SaveChangesAsync(CancellationToken.None);

        var user = await _userManager.FindByIdAsync(storedToken.UserId.ToString());
        if (user == null) return null;

        return await GenerateAuthResponseAsync(user);
    }

    public async Task<bool> RevokeTokenAsync(string refreshTokenStr)
    {
        var storedToken = await _context.RefreshTokens
            .FirstOrDefaultAsync(x => x.Token == refreshTokenStr);

        if (storedToken == null) return false;

        storedToken.IsRevoked = true;
        await _context.SaveChangesAsync(CancellationToken.None);
        return true;
    }

    private async Task<AuthResponse> GenerateAuthResponseAsync(ApplicationUser user)
    {
        var userRoles = await _userManager.GetRolesAsync(user);

        var claims = new List<Claim>
        {

```

					ИПЗ.КР.Б-121-26-ПЗ	Арк.
						72

```

        new(JwtRegisteredClaimNames.Sub, user.Id.ToString()),
        new(JwtRegisteredClaimNames.Email, user.Email ?? string.Empty),
        new(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),
        new("firstName", user.FirstName),
        new("lastName", user.LastName)
    };

    if (user.CinemaId.HasValue)
    {
        claims.Add(new Claim("cinemaId", user.CinemaId.Value.ToString()));
    }

    foreach (var role in userRoles)
    {
        claims.Add(new Claim(ClaimTypes.Role, role));
    }

    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(_jwtSettings.Secret));
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);
    var expiry = DateTime.UtcNow.AddMinutes(_jwtSettings.ExpiryInMinutes);

    var token = new SecurityTokenDescriptor
    {
        Subject = new ClaimsIdentity(claims),
        Expires = expiry,
        Issuer = _jwtSettings.Issuer,
        Audience = _jwtSettings.Audience,
        SigningCredentials = creds
    };

    var tokenHandler = new JwtSecurityTokenHandler();
    var securityToken = tokenHandler.CreateToken(token);
    var accessTokenStr = tokenHandler.WriteToken(securityToken);

    var refreshTokenStr = GenerateSecureRefreshTokenString();

    var refreshTokenEntity = new RefreshToken(user.Id, refreshTokenStr, DateTime.UtcNow.AddDays(7));

    _context.RefreshTokens.Add(refreshTokenEntity);
    await _context.SaveChangesAsync(CancellationToken.None);

    return new AuthResponse
    {
        Token = accessTokenStr,
        RefreshToken = refreshTokenStr,
        TokenExpiry = expiry,
        UserId = user.Id,
        Email = user.Email ?? string.Empty,
        FirstName = user.FirstName,
        LastName = user.LastName,
        Roles = userRoles
    };
}

private static string GenerateSecureRefreshTokenString()
{
    var randomNumber = new byte[64];
    using var rng = RandomNumberGenerator.Create();
    rng.GetBytes(randomNumber);
    return Convert.ToBase64String(randomNumber);
}
}

```

Лістинг програми CurrentUserService.cs

```

namespace StarPlex.Infrastructure.Services;

public class CurrentUserService : ICurrentUserService
{
    private readonly IHttpContextAccessor _httpContextAccessor;

```

```

public CurrentUserService(IHttpContextAccessor httpContextAccessor)
{
    _httpContextAccessor = httpContextAccessor;
}

public Guid? UserId
{
    get
    {
        var sub = _httpContextAccessor.HttpContext?.User?.FindFirstValue(ClaimTypes.NameIdentifier)
            ?? _httpContextAccessor.HttpContext?.User?.FindFirstValue(JwtRegisteredClaimNames.Sub);
        return Guid.TryParse(sub, out var id) ? id : null;
    }
}

public string? Role => _httpContextAccessor.HttpContext?.User?.FindFirstValue(ClaimTypes.Role);

public Guid? CinemaId
{
    get
    {
        var cinemaIdStr = _httpContextAccessor.HttpContext?.User?.FindFirst("cinemaId")?.Value;
        return Guid.TryParse(cinemaIdStr, out var id) ? id : null;
    }
}

public bool IsSuperAdmin => Role == "SuperAdmin";
}

```

Лістинг програми GetSessionSeatMapQueryHandler.cs

```

namespace StarPlex.Application.Features.Sessions.Queries.GetSessionSeatMap;

public class GetSessionSeatMapQueryHandler : IRequestHandler<GetSessionSeatMapQuery, List<SeatMapDto>>
{
    private readonly IApplicationDbContext _context;

    public GetSessionSeatMapQueryHandler(IApplicationDbContext context)
    {
        _context = context;
    }

    public async Task<List<SeatMapDto>> Handle(GetSessionSeatMapQuery request, CancellationToken cancellationToken)
    {
        var utcNow = DateTime.UtcNow;

        var session = await _context.Sessions
            .AsNoTracking()
            .FirstOrDefaultAsync(s => s.Id == request.SessionId, cancellationToken);

        if (session == null) throw new NotFoundException("Session", request.SessionId);

        var allSeats = await _context.Seats
            .AsNoTracking()
            .Where(s => s.HallId == session.HallId)
            .ToListAsync(cancellationToken);

        var activeLocks = await _context.SelectedSeats
            .AsNoTracking()
            .Where(ss => ss.SessionId == request.SessionId && ss.LockedUntil > utcNow)
            .ToListAsync(cancellationToken);

        var takenSeatIds = await _context.BookingSeats
            .AsNoTracking()
            .Where(bs => bs.Booking.SessionId == request.SessionId &&
                (bs.Booking.Status == BookingStatus.Confirmed ||
                 bs.Booking.Status == BookingStatus.Pending))
            .Select(bs => bs.SeatId)
            .ToListAsync(cancellationToken);

        var seatMap = allSeats.Select(seat =>

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						74

```

{
    string status = "Available";
    Guid? lockedBy = null;

    if (takenSeatIds.Contains(seat.Id))
    {
        status = "Taken";
    }
    else
    {
        var currentLock = activeLocks.FirstOrDefault(l => l.SeatId == seat.Id);
        if (currentLock != null)
        {
            status = "Locked";
            lockedBy = currentLock.UserId;
        }
    }

    decimal priceMultiplier = seat.Type switch
    {
        SeatType.VIP => 1.5m,
        SeatType.Disabled => 0.8m,
        _ => 1.0m
    };

    int.TryParse(seat.Row, out var parsedRow);

    return new SeatMapDto
    {
        SeatId = seat.Id,
        Row = seat.Row,
        SeatNumber = seat.Number,
        SeatType = seat.Type.ToString(),
        PriceMultiplier = priceMultiplier,
        Status = status,
        LockedByUserId = lockedBy,
        RowNumberForSorting = parsedRow
    };
})
.OrderBy(s => s.RowNumberForSorting)
.ThenBy(s => s.SeatNumber)
.ToList();

return seatMap;
}
}

```

Лістинг програми CreateSessionCommandHandler.cs

```

namespace StarPlex.Application.Features.Sessions.Commands.CreateSession;

public class CreateSessionCommandHandler : IRequestHandler<CreateSessionCommand, Guid>
{
    private readonly IApplicationDbContext _context;
    private readonly ICurrentUserService _currentUserService;

    public CreateSessionCommandHandler(IApplicationDbContext context, ICurrentUserService currentUserService)
    {
        _context = context;
        _currentUserService = currentUserService;
    }

    public async Task<Guid> Handle(CreateSessionCommand request, CancellationToken cancellationToken)
    {
        if (_context is not DbContext dbContext)
            throw new InvalidOperationException("Database context is not compatible with transactions.");

        var hall = await _context.Halls
            .AsNoTracking()
            .FirstOrDefaultAsync(h => h.Id == request.HallId, cancellationToken);
    }
}

```

```

if (hall == null) throw new NotFoundException("Cinema Hall", request.HallId);

if (!_currentUserService.IsSuperAdmin && _currentUserService.CinemaId != hall.CinemaId)
{
    throw new InvalidOperationException("You do not have permission to manage this cinema.");
}

var movie = await _context.Movies
    .AsNoTracking()
    .FirstOrDefaultAsync(m => m.Id == request.MovieId, cancellationToken);

if (movie == null) throw new NotFoundException("Movie", request.MovieId);

var startTimeUtc = request.StartTime.Kind == DateTimeKind.Utc
    ? request.StartTime
    : DateTime.SpecifyKind(request.StartTime, DateTimeKind.Utc);

var cleanUpDuration = Session.CleanUpDurationInMinutes;
var endTimeUtc = startTimeUtc.AddMinutes(movie.DurationInMinutes + cleanUpDuration);

await using var transaction = await dbContext.Database.BeginTransactionAsync(cancellationToken);
try
{
    var hasCollision = await _context.Sessions
        .AsNoTracking()
        .Where(s => s.HallId == request.HallId && s.Status == SessionStatus.Active)
        .AnyAsync(s => s.StartTime < endTimeUtc &&
            s.StartTime.AddMinutes(s.MovieDurationInMinutes + Session.CleanUpDurationInMinutes) > startTimeUtc,
            cancellationToken);

    if (hasCollision)
    {
        throw new InvalidOperationException("Time slot collision detected. This hall is already occupied by another session
during the specified time.");
    }

    var session = new Session(
        request.MovieId,
        request.HallId,
        startTimeUtc,
        movie.DurationInMinutes,
        request.BasePrice,
        request.BasePrice,
        SessionStatus.Active
    );

    if (movie.Status == MovieStatus.ComingSoon)
    {
        movie.Status = MovieStatus.NowShowing;
    }

    _context.Sessions.Add(session);
    await _context.SaveChangesAsync(cancellationToken);

    await transaction.CommitAsync(cancellationToken);
    return session.Id;
}
catch
{
    await transaction.RollbackAsync(cancellationToken);
    throw;
}
}

```

Лістинг програми LockSeatsCommandHandler.cs

namespace StarPlex.Application.Features.Bookings.Commands.LockSeats;

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						76

```

public class LockSeatsCommandHandler : IRequestHandler<LockSeatsCommand, LockSeatsResponseDto>
{
    private readonly IApplicationDbContext _context;
    private readonly ISeatLockService _seatLockService;
    private readonly ISeatHubService _seatHubService;

    public LockSeatsCommandHandler(IApplicationDbContext context, ISeatLockService seatLockService, ISeatHubService seatHubService)
    {
        _context = context;
        _seatLockService = seatLockService;
        _seatHubService = seatHubService;
    }

    public async Task<LockSeatsResponseDto> Handle(LockSeatsCommand request, CancellationToken cancellationToken)
    {
        var session = await _context.Sessions
            .AsNoTracking()
            .FirstOrDefaultAsync(s => s.Id == request.SessionId, cancellationToken);

        if (session == null)
            throw new NotFoundException("Session", request.SessionId);

        if (session.Status != SessionStatus.Active)
            throw new InvalidOperationException("Cannot lock seats for an inactive or completed session.");

        var seatsInHallCount = await _context.Seats
            .AsNoTracking()
            .Where(s => s.HallId == session.HallId && request.SeatIds.Contains(s.Id))
            .CountAsync(cancellationToken);

        if (seatsInHallCount != request.SeatIds.Count)
            throw new InvalidOperationException("Some of the selected seats do not belong to this cinema hall.");

        var success = await _seatLockService.LockSeatsAsync(
            request.SessionId,
            request.SeatIds,
            request.UserId,
            cancellationToken);

        if (!success)
        {
            throw new InvalidOperationException("One or more selected seats are already locked or booked by another user.");
        }

        await _seatHubService.NotifySeatsLockedAsync(
            request.SessionId,
            request.SeatIds,
            request.UserId,
            cancellationToken);

        return new LockSeatsResponseDto
        {
            IsSuccess = true,
            Message = "Seats successfully locked for 10 minutes.",
            ExpiryTimeUtc = DateTime.UtcNow.AddMinutes(10)
        };
    }
}

```

Лістинг програми GenerateScheduleCommandHandler.cs

```

namespace StarPlex.Application.Features.Sessions.Commands.GenerateSchedule;

public class GenerateScheduleCommandHandler : IRequestHandler<GenerateScheduleCommand, int>
{
    private readonly IApplicationDbContext _context;
    private const int CleanUpDurationInMinutes = 20;

    public GenerateScheduleCommandHandler(IApplicationDbContext context)
    {
    }
}

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						77

```

    _context = context;
}

public async Task<int> Handle(GenerateScheduleCommand request, CancellationToken cancellationToken)
{
    var halls = await _context.Halls
        .Where(h => h.CinemaId == request.CinemaId && h.IsActive)
        .ToListAsync(cancellationToken);

    if (!halls.Any())
        throw new InvalidOperationException("There are no active halls in this cinema to generate a schedule.");

    var movies = await _context.Movies
        .Where(m => request.MovieIds.Contains(m.Id) && m.Status == MovieStatus.NowShowing)
        .OrderByDescending(m => m.TmdbRating)
        .ToListAsync(cancellationToken);

    if (!movies.Any())
        throw new InvalidOperationException("No active movies were found for generation.");

    int sessionsCreated = 0;
    int movieIndex = 0;

    foreach (var hall in halls)
    {
        var startOfDay = new DateTime(request.TargetDate.Year, request.TargetDate.Month, request.TargetDate.Day, 10, 0, 0,
            DateTimeKind.Utc);
        var endOfDay = new DateTime(request.TargetDate.Year, request.TargetDate.Month, request.TargetDate.Day, 23, 0, 0,
            DateTimeKind.Utc);

        var currentTrackTime = startOfDay;

        while (currentTrackTime < endOfDay)
        {
            var currentMovie = movies[movieIndex % movies.Count];

            var sessionStartTime = currentTrackTime;
            var sessionEndTime = sessionStartTime.AddMinutes(currentMovie.DurationInMinutes);

            if (sessionEndTime > endOfDay)
            {
                break;
            }

            bool hasCollision = await _context.Sessions.AnyAsync(s =>
                s.HallId == hall.Id &&
                s.Status == SessionStatus.Active &&
                ((sessionStartTime >= s.StartTime && sessionStartTime < s.StartTime.AddMinutes(s.MovieDurationInMinutes +
                    CleanUpDurationInMinutes)) ||
                (sessionEndTime > s.StartTime && sessionEndTime <= s.StartTime.AddMinutes(s.MovieDurationInMinutes +
                    CleanUpDurationInMinutes))),
                cancellationToken);

            if (!hasCollision)
            {
                decimal calculatedPrice = sessionStartTime.Hour switch
                {
                    >= 10 and < 12 => request.BasePrice * 0.80m,
                    >= 12 and < 17 => request.BasePrice,
                    >= 17 and < 21 => request.BasePrice * 1.25m,
                    _ => request.BasePrice * 1.10m
                };

                var newSession = new Session
                {
                    Id = Guid.NewGuid(),
                    MovieId = currentMovie.Id,
                    HallId = hall.Id,
                    StartTime = sessionStartTime,
                    MovieDurationInMinutes = currentMovie.DurationInMinutes,
                    OriginalPrice = request.BasePrice
                };
            }
        }
    }
}

```

					ИПЗ.КР.Б-121-26-ПЗ	Арк.
						78

```

        BasePrice = Math.Round(calculatedPrice, 0),
        Status = SessionStatus.Active
    };

    _context.Sessions.Add(newSession);
    sessionsCreated++;
}

var nextTimeWithCleanUp = sessionEndTime.AddMinutes(CleanUpDurationInMinutes);
int minutesToSubtractOrAdd = nextTimeWithCleanUp.Minute % 5;

currentTrackTime = minutesToSubtractOrAdd == 0
    ? nextTimeWithCleanUp
    : nextTimeWithCleanUp.AddMinutes(5 - minutesToSubtractOrAdd);
movieIndex++;
}
}

if (sessionsCreated > 0)
{
    await _context.SaveChangesAsync(cancellationToken);
}

return sessionsCreated;
}
}

```

Лістинг програми MoviesController.cs

```

namespace StarPlex.API.Controllers;

[ApiController]
[Route("api/[controller]")]
public class MoviesController : ControllerBase
{
    private readonly IMediator _mediator;

    public MoviesController(IMediator mediator)
    {
        _mediator = mediator;
    }

    [HttpGet]
    [AllowAnonymous]
    public async Task<ActionResult<List<MovieDto>>> GetMovies([FromQuery] MovieStatus? status, CancellationToken
cancellationToken)
    {
        var result = await _mediator.Send(new GetMoviesQuery { Status = status }, cancellationToken);
        return Ok(result);
    }

    [HttpGet("{id:guid}")]
    [AllowAnonymous]
    public async Task<ActionResult<MovieDto>> GetMovieById(Guid id, CancellationToken cancellationToken)
    {
        var result = await _mediator.Send(new GetMovieByIdQuery { Id = id }, cancellationToken);
        if (result == null) return NotFound($"Movie with ID '{id}' was not found.");
        return Ok(result);
    }

    [HttpGet("search-tmdb")]
    [Authorize(Roles = "CinemaManager,SuperAdmin")]
    public async Task<ActionResult<List<TmdbMovieResultDto>>> SearchTmdb([FromQuery] string query, CancellationToken
cancellationToken)
    {
        var result = await _mediator.Send(new SearchMoviesQuery { Query = query }, cancellationToken);
        return Ok(result);
    }

    [HttpPost("import-tmdb/{tmdbId:int}")]
    [Authorize(Roles = "CinemaManager,SuperAdmin")]

```

					ІПЗ.КР.Б-121-26-ПЗ	Арк.
						79

```

public async Task<ActionResult<Guid>> ImportMovie([FromRoute] int tmdbId, CancellationToken cancellationToken)
{
    var command = new CreateMovieCommand { TmdbId = tmdbId };
    var movieId = await _mediator.Send(command, cancellationToken);
    return CreatedAtAction(nameof(GetMovieById), new { id = movieId }, movieId);
}

[HttpPut("{id:guid}")]
[Authorize(Roles = "CinemaManager,SuperAdmin")]
public async Task<ActionResult> UpdateMovie(Guid id, [FromBody] UpdateMovieCommand command, CancellationToken
cancellationToken)
{
    if (id != command.Id) return BadRequest("ID in URL does not match ID in request body.");
    await _mediator.Send(command, cancellationToken);
    return NoContent();
}

[HttpDelete("{id:guid}")]
[Authorize(Roles = "CinemaManager,SuperAdmin")]
public async Task<ActionResult> DeleteMovie(Guid id, CancellationToken cancellationToken)
{
    await _mediator.Send(new DeleteMovieCommand { Id = id }, cancellationToken);
    return NoContent();
}
}

```

Лістинг програми AnalyticsController.cs

```

namespace StarPlex.API.Controllers;

[ApiController]
[Route("api/[controller]")]
[Authorize(Roles = "SuperAdmin,CinemaManager")]
public class AnalyticsController : ControllerBase
{
    private readonly IMediator _mediator;

    public AnalyticsController(IMediator mediator)
    {
        _mediator = mediator;
    }

    [HttpGet("admin-stats")]
    public async Task<ActionResult<AdminDashboardStatsDto>> GetAdminStats(
        [FromQuery] int days = 14,
        [FromQuery] Guid? cinemaId = null,
        [FromQuery] string isOnline = "all",
        CancellationToken cancellationToken = default)
    {
        var result = await _mediator.Send(new GetAdminDashboardStatsQuery
        {
            DaysPeriod = days,
            CinemaId = cinemaId,
            IsOnline = isOnline
        }, cancellationToken);

        return Ok(result);
    }
}

```