Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу
Факультет інформаційних технологій
Кафедра інформаційно-телекомунікаційних технологій та систем
Данилюк Максим Дмитрович
(прізвище, ім'я, по батькові)

УДК 004.9:791.43
(індекс)

БАКАЛАВРСЬКА РОБОТА

Розроблення інформаційної системи управління кінотеатром з модулем прогнозування завантаженості залів

(назва роботи)

Інформаційні системи та технології

(назва освітньої програми)

126 − Інформаційні системи та технології

(шифр і назва спеціальності)

Робота містить результати власних досліджень, використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело:

Здобувач освітнього ступеня Данилюк М. Д.
(підпис, ініціали та прізвище здобувача)

Науковий керівник к.т.н., доцент Штаєр Л. О.
(підпис, прізвище, ім'я, по батькові, науковий ступінь, вчене звання керівника)

Допущено до захисту
В. о. завідувача кафедри

Заміховська О. Л.
(посада) (підпис) (дата) (ініціали та прізвище)

Івано-Франківськ – 2026

Івано-Франківський національний технічний університет нафти і газу
(повне найменування закладу вищої освіти)

Факультет інформаційних технологій
Кафедра інформаційно-телекомунікаційних технологій та систем
Освітній рівень бакалавр
Спеціальність 126 − Інформаційні системи та технології
(шифр і назва)

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри ІТТС
Заміховська О. Л.
«____» _____2026 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Данилюку Максиму Дмитровичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення інформаційної системи управління кінотеатром з модулем прогнозування завантаженості залів
керівник роботи, доц. каф. ІТТС Штаєр Л. О.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від " 07 " *квітня* 2026 року № 163/7
2. Строк подання студентом роботи
3. Вихідні дані до роботи Монолітний вебзастосунок на базі архітектури MVC та фреймворку Ruby on Rails 8; клієнтська частина з використанням стеку Hotwire та Tailwind CSS; локальна реляційна база даних SQLite; модуль машинного навчання для предиктивної аналітики на основі гребеневої регресії; фонова обробка завдань для обчислення прогнозів через Solid Queue; генерація фінансової звітності та фіскальних квитків у форматі PDF за допомогою бібліотеки Prawn.
4. Зміст пояснювальної записки (перелік питань, що їх належить розробити)
аналіз предметної області та існуючих систем управління завданнями; проєктування інформаційної системи; програмна реалізація системи та експлуатація програмного комплексу
5. Перелік презентаційного матеріалу (з точним зазначенням обов'язкових презентацій)
варіанти використання інформаційної системи управління завданнями; діаграма бази даних інформаційної системи управління кінотеатром з модулем прогнозування завантаженості залів; діаграма використання інформаційної системи управління кінотеатром з модулем прогнозування завантаженості залів; програмні вікна інформаційної системи управління кінотеатром з модулем прогнозування завантаженості залів (2 шт.).
6. Дата видачі завдання: 07.04.2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Термін виконання етапів	Примітка
1.	Аналіз предметної області та огляд існуючих систем управління кінотеатром	07.04.2026 - 09.04.2026 р.	Виконано
2.	Розробка варіантів використання, діаграми прецедентів та постановка задачі на розробку.	02.04.2026 - 09.04.2026 р.	Виконано
3.	Проєктування бази даних (SQLite) та побудова логічної ER-діаграми.	09.04.2026 - 16.04.2026 р.	Виконано
4.	Програмна реалізація монолітної архітектури на Ruby on Rails та налаштування бізнес-логіки продажу квитків.	16.04.2026 - 20.04.2026 р.	Виконано
5.	Розробка модуля предиктивної аналітики (гребенева регресія) та фонового конвеєра прогнозування попиту.	20.04.2026 - 29.04.2026 р.	Виконано
6.	Розробка інтерактивного клієнтського інтерфейсу касира (Hotwire, Tailwind CSS) та модуля фінансової звітності.	29.04.2026 - 07.05.2026 р.	Виконано
7.	Оцінка точності моделі (MAPE), генерація PDF-звітів та комплексне відлагодження системи.	07.05.2026 - 14.05.2026 р.	Виконано
8.	Оформлення пояснювальної записки та графічного матеріалу	22.05.2026 - 29.05.2026 р.	Виконано
9.	Подання роботи на рецензування та підготовка до захисту	29.05.2026 - 05.06.2026 р.	

Студент _____ *Данилюк М. Д.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Штаєр Л. О.*
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Розрахунково-пояснювальна записка: 77 сторінок, 30 малюнків, 3 таблиці, 21 посилань, 2 додатки.

Об'єкт дослідження – процес аналізу та прогнозування глядацького попиту в кінотеатрі.

Мета роботи – розроблення інформаційної веб-системи кінотеатру, ключовим елементом якої є інтегрований модуль предиктивної аналітики, що використовує алгоритми статистичного моделювання для математичного прогнозування завантаженості залів на майбутні сеанси.

У першій частині роботи виконано аналіз предметної області, проведено порівняння існуючих програмних рішень автоматизації кінотеатрів та обґрунтовано доцільність розроблення власної системи з вбудованим аналітичним модулем.

У другій частині здійснено проєктування архітектури системи. Обґрунтовано вибір монолітної архітектури з використанням шаблону MVC. Розроблено ER-діаграму бази даних SQLite та діаграму прецедентів. Спроєктовано алгоритмічний конвеєр для фонової обробки даних модуля прогнозування.

У третій частині описано програмну реалізацію системи. Серверну частину реалізовано на фреймворку Ruby on Rails 8. Розроблено реактивний користувацький інтерфейс на базі технологій Hotwire (Turbo, Stimulus) та Tailwind CSS. Успішно імплементовано та інтегровано математичну модель гребеневої регресії для розрахунку очікуваної кількості проданих квитків з урахуванням історії сеансів, дня тижня та часу показу.

КЛЮЧОВІ СЛОВА: УПРАВЛІННЯ КІНОТЕАТРОМ, ПРЕДИКТИВНА АНАЛІТИКА, СТАТИСТИЧНИЙ АНАЛІЗ, ПРОГНОЗУВАННЯ ПОПИТУ, RUBY ON RAILS, HOTWIRE, ГРЕБЕНЕВА РЕГРЕСІЯ.

ABSTRACT

Calculation and explanatory note: 77 pages, 30 figures, 3 tables, 21 references, 2 appendices.

Object of research – the process of analyzing and predicting audience demand in a cinema.

Purpose of the work – development of a cinema web information system, the key element of which is an integrated predictive analytics module that uses statistical modeling algorithms for mathematical predictionof hall occupancy for future showtimes.

In the first part of the work, an analysis of the subject area is performed, existing cinema automation software solutions are compared, and the feasibility of developing a proprietary system with a built-in analytical module is justified.

In the second part, the system architecture is designed. The choice of a monolithic architecture using the MVC pattern is substantiated. An ER diagram of the SQLite database and a use case diagram are developed. An algorithmic pipeline for background data processing of the forecasting module is designed.

In the third part, the software implementation of the system is described. The server side is implemented on the Ruby on Rails 8 framework. A reactive user interface based on Hotwire (Turbo, Stimulus) and Tailwind CSS technologies is developed. A mathematical Ridge Regression model is successfully implemented and integrated to calculate the expected number of sold tickets taking into account showtime history, day of the week, and time of screening.

KEYWORDS: CINEMA MANAGEMENT, PREDICTIVE ANALYTICS, STATISTICAL ANALYSIS, DEMAND FORECASTING, RUBY ON RAILS, HOTWIRE, RIDGE REGRESSION.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 7

ВСТУП 8

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПРОГРАМНИХ СИСТЕМ УПРАВЛІННЯ КІНОТЕАТРОМ 10

1.1 Аналіз програм-аналогів та встановлення вимог 10

1.2 Обґрунтування вибору технологічного стеку розробки 13

1.3 Розробка варіантів використання 17

1.4 Постановка задачі 24

Висновки до розділу 1 25

2 ПРОЄКТУВАННЯ ТА АРХІТЕКТУРА ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ КІНОТЕАТРОМ 28

2.1 Проєктування структури бази даних 28

2.2 Вибір архітектури додатка 33

2.3 Реалізація архітектурного шаблону MVC 36

2.4 Проєктування та обґрунтування структури модуля математичного прогнозування 38

2.5 Проєктування інтерфейсу користувача 47

Висновки до розділу 2 49

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЛУАТАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ 51

3.1 Програмна реалізація та інженерний аналіз методів бізнес-логіки 51

3.2 Програмна реалізація методів та алгоритмів математичного прогнозування попиту 57

3.3 Інтерфейс користувача, структура проєкту та клієнт-серверна взаємодія……..63
Висновки до розділу 3 72
ВИСНОВКИ 74
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 76
ДОДАТОК А – ЛІСТИНГ КОДУ СЕРВЕРНОЇ ЧАСТИНИ..... 78
ДОДАТОК Б – ЛІСТИНГ КОДУ ІНТЕРФЕЙСНОЇ ЧАСТИНИ..... 89

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

СУБД – Система управління базами даних.

ACID – Atomicity, Consistency, Isolation, Durability.

AJAX – Asynchronous JavaScript and XML.

API – Application Programming Interface.

CSS – Cascading Style Sheets (Каскадні таблиці стилів).

CSV – Comma-Separated Values (Значення, розділені комою).

DOM – Document Object Model (Об'єктна модель документа).

DRY – Don't Repeat Yourself (Принцип програмування «Не повторюйся»).

DTO – Data Transfer Object (Об'єкт передачі даних).

FK – Foreign Key (Зовнішній ключ).

HTML – HyperText Markup Language.

HTTP – HyperText Transfer Protocol.

JSON – JavaScript Object Notation.

MAE – Mean Absolute Error (Середня абсолютна помилка).

MAPE – Mean Absolute Percentage Error

MVC – Model-View-Controller (Модель-Представлення-Контролер).

PDF – Portable Document Format.

PK – Primary Key (Первинний ключ).

SPA – Single Page Application (Односторінковий вебзастосунок).

SQL – Structured Query Language (Мова структурованих запитів).

UML – Unified Modeling Language (Уніфікована мова моделювання).

UX/UI – User Experience / User Interface.

ARIMA – Autoregressive Integrated Moving Average.

ERB – Embedded Ruby (Вбудований Ruby).

ISO – International Organization for Standardization

JS – JavaScript.

REST – Representational State Transfer (Передача стану представлення).

ERURL – Uniform Resource Locator (Уніфікований локатор ресурсів).

ВСТУП

Актуальність. Управління сучасним кінотеатром вимагає точного планування ресурсів та витрат. Традиційно менеджери оцінюють майбутню відвідуваність сеансів інтуїтивно або спираючись на базовий досвід. Помилка в такій оцінці призводить до негативних наслідків: несподіваний аншлаг спричиняє величезні черги через нестачу касирів, а переоцінка попиту – до фінансових втрат через оплату роботи зайвого персоналу та псування нереалізованої продукції в барі. Впровадження систем прогнозування вирішує цю проблему. Створення системи, яка здатна автоматично аналізувати історичні дані продажу квитків, враховувати характеристики фільму, дні тижня та час сеансу для розрахунку точної кількості глядачів за допомогою математичних алгоритмів, є актуальним завданням для оптимізації операційної діяльності бізнесу.

Мета дослідження полягає у розробці інформаційної вебсистеми кінотеатру, ключовим елементом якої є інтегрований модуль прогнозування попиту. Цей модуль має використовувати математичні алгоритми для розрахунку завантаженості залів на майбутні сеанси, що дозволить керівництву кінотеатру завчасно оптимізувати графіки роботи персоналу, ефективно розподіляти ресурси та мінімізувати операційні втрати:

1 проаналізувати предметну область, дослідити проблему оцінки глядацького попиту та визначити недоліки ручного планування ресурсів кінотеатру;

2 сформувати чіткі функціональні та нефункціональні вимоги до архітектури системи та модуля прогнозування;

3 спроєктувати структуру реляційної бази даних, яка забезпечить коректне збереження історії проданих квитків – основи для роботи математичної моделі;

4 програмно реалізувати базовий функціонал системи управління (створення фільмів, залів та розкладу сеансів);

5 розробити та інтегрувати статистичний алгоритм (лінійну регресію) для автоматичного розрахунку очікуваної кількості глядачів на основі історичних даних, дня тижня та часу сеансу.

Об'єкт дослідження є процес аналізу та прогнозування глядацького попиту в кінотеатрі.

Предмет дослідження є математичні алгоритми та архітектурні рішення, що використовуються для розробки аналітичного модуля вебсистеми.

Методи дослідження. В роботі було використано методи аналізу літературних та технічних джерел для обробки теоретичної інформації, порівняння та систематизація виокремлених даних, методи системного аналізу та проєктування архітектури, програмування вебдодатків, а також статистичного аналізу та обробки даних з використанням алгоритмів регресії.

Практичне значення. Головна цінність розробленої системи полягає у модулі прогнозування, який генерує для власника бізнесу математично обґрунтований розрахунок кількості глядачів на майбутні сеанси. Це дозволяє заздалегідь формувати оптимальний графік виходу працівників на зміни, ефективно керувати запасами продукції, уникати зайвих витрат та суттєво підвищити загальну рентабельність кінотеатру.

Структура дипломної роботи. Робота складається зі вступу, трьох розділів, висновків та списку використаних джерел. У дипломній роботі наведено 3 таблиці, 30 рисунків. Список використаних джерел містить 21 позицій. До роботи долучено 2 додатка на 19 сторінках.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПРОГРАМНИХ СИСТЕМ УПРАВЛІННЯ КІНОТЕАТРОМ

1.1 Аналіз програм-аналогів та встановлення вимог

Предметна область цієї теми визначається реалізацією вебзастосунку для управління роботою мережі кінотеатрів. Система управління кінотеатром – це комплексне програмне забезпечення, яке використовується для автоматизації технічних, фінансових та логістичних бізнес-процесів [1]. Це включає такі завдання, як планування розкладу показів (сітка сеансів), управління продажем квитків, контроль заповнюваності залів, а також ведення звітності.

На даний момент ринок систем для кінотеатрів поділений на дві категорії: застарілі десктопні рішення, які написані під операційну систему Windows і не підтримують віддалений доступ, та хмарні сервіси-агрегатори, які вимагають регулярних фінансових відрахувань у вигляді комісії та зберігають дані клієнтів на сторонніх серверах.

Одним із поширених рішень на українському ринку є платформа mticket. Це хмарний сервіс, який інтегрується з вебсайтами кінотеатрів та дозволяє організувати онлайн-продаж квитків. Основні можливості цієї платформи включають:

- зручний інтерфейс для вибору місць на схемі залу;
- інтеграцію з платіжними системами для оплати;
- наявність мобільного додатка для покупців;
- можливість інтеграції по API з іншими системами для роботи в спільному білетному полі [2].

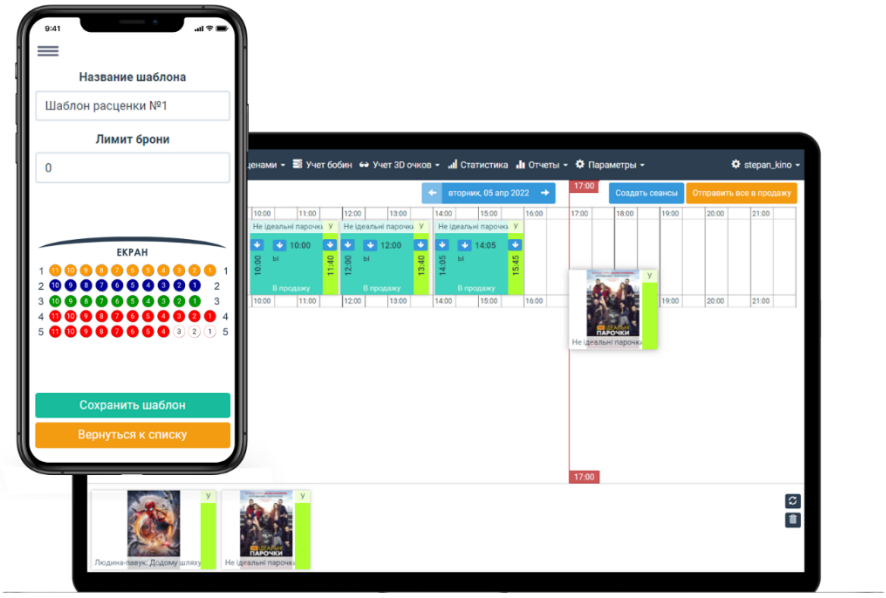

Рисунок 1.1 – Інтерфейс вибору місць у системі «mticket»

Незважаючи на поширеність, використання даного сервісу як основного інструменту управління має обмеження для власників кінотеатрів, оскільки передбачає комісію з кожного проданого квитка, що зменшує прибуток компанії. Крім того, уся база даних клієнтів та історія транзакцій знаходяться на серверах третьої сторони, що унеможливлює збір інформації для навчання власних моделей аналізу даних, а сам сервіс має обмежений функціонал для внутрішнього управлінського обліку та планування ресурсів.

Іншим відомим рішенням є застосунок SERVIO Tickets, що є системою автоматизації кінотеатрів та розважальних комплексів. Застосунок складається з комплексу програмних інструментів, що дають змогу розв'язати оперативні задачі підприємства. Серед функціональних можливостей програми виділяються:

– гнучка система налаштування сеансів та швидкий доступ касира до інформації про фільм [3];

– відображення вільних та зайнятих місць у залах з можливістю їх бронювання;

– наявність інформаційних моніторів для відвідувачів та інструментів швидкої валідації квитків біля входу [3].

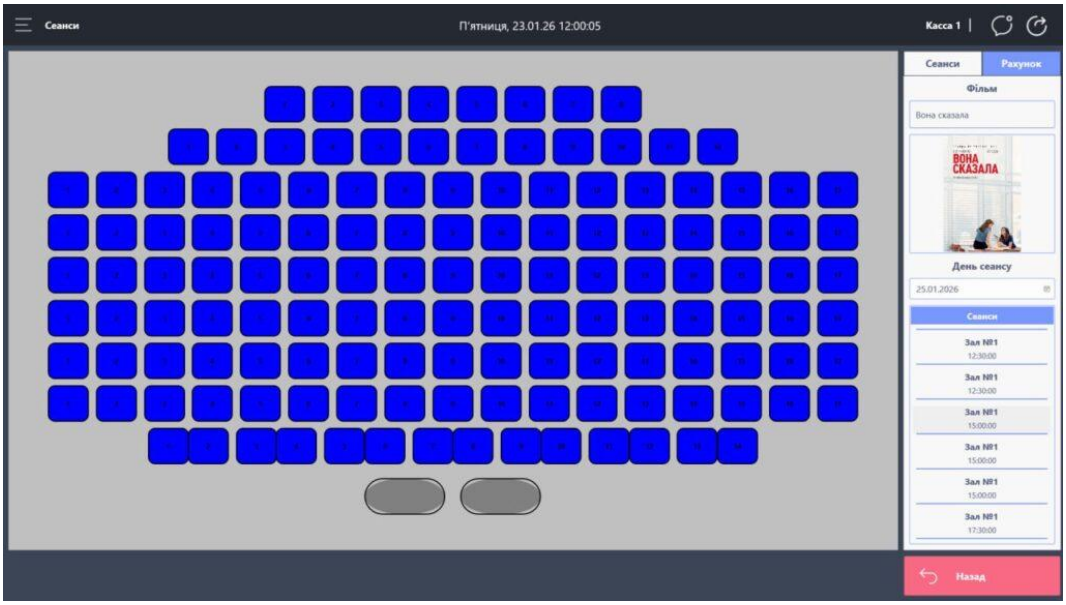

Рисунок 1.2 – Сторінка продажу квитків у «SERVIO Tickets»

Для порівняння проєкту з аналогом можна скористатись таблицею 1.1.

Таблиця 1.1 – Порівняння з аналогом

Функціонал	Розроблювана система	mticket	SERVIO Tickets	Пояснення
Продаж та бронювання квитків	+	+	+	Даний функціонал є фундаментальним для будь-якої системи обслуговування кінотеатрів.
Продаж та облік супутніх товарів	+	-	+	У розроблюваній системі після закінчення робочого дня відбувається автоматизована звірка залишків товарів.
Автоматичне формування звітів	+	-	-	При закінченні зміни система автоматично генерує фінансовий звіт. В аналогах цей процес не є автоматизованим.
Прогнозування завантаженості (Статистичний аналіз)	+	-	-	Алгоритм аналізує історію продажів і прогнозує відсоток посадки на майбутні сеанси для оптимізації персоналу.

Порівняльний аналіз показує, що розробка запропонованої інформаційної системи дозволяє поєднати переваги розглянутих аналогів,

забезпечуючи мобільність вебрішень та управлінський функціонал професійних комплексів. Це звільняє бізнес від комісійних витрат і орієнтує систему на використання вбудованого математичного модуля для автоматизації процесів аналітики та прогнозування посадки залів.

Формат вебзастосунку обрано через необхідність забезпечення доступу працівників та адміністрації з різних пристроїв (комп'ютерів чи планшетів) без потреби встановлення додаткового програмного забезпечення [4]. З метою уникнення архітектурної надмірності та складності підтримки розподілених систем, замість розділення на окремі клієнтську та серверну частини розробку інтерфейсу та бізнес-логіки інтегровано в єдину монолітну архітектуру на базі фреймворку Ruby on Rails. Для забезпечення інтерактивності, швидкодії та оновлення даних в режимі реального часу без повного перезавантаження сторінок екрана, застосовано технологічний стек Hotwire (компоненти Turbo та Stimulus) у поєднанні з утилітарним інструментом верстки Tailwind CSS.

1.2 Обґрунтування вибору технологічного стеку розробки

1.2.1 Теоретичні аспекти архітектурного проєктування вебсистем. Під час створення сучасних вебсистем важливим етапом є вибір архітектурного підходу, який забезпечує баланс між швидкістю розробки та стабільністю роботи програми. У практиці розробки існують два основні напрямки: монолітна архітектура та мікросервісна інфраструктура. Для систем середнього масштабу, таких як система управління кінотеатром, доцільним є використання монолітного підходу, коли весь програмний код додатка зберігається, тестується та виконується в одному загальному середовищі [5].

Таке рішення усуває складні мережеві затримки на взаємодію між багатьма окремими сервісами, значно спрощує процес розгортання програми на цільовому сервері та забезпечує високу цілісність бази даних. Це дозволяє зосередитися на реалізації бізнес-логіки та аналітичних функцій без інфраструктурного ускладнення кодової бази[6].

1.2.2 Математичні підходи до побудови підсистем предиктивної аналітики. Впровадження підсистеми предиктивної аналітики в систему управління кінотеатром спрямоване на розв'язання задачі прогнозування попиту, а саме – визначення очікуваної кількості проданих квитків на майбутні сеанси. Це необхідно для автоматизації управлінських рішень, таких як завчасне планування графіків роботи персоналу, уникнення черг на касах та оптимізація запасів супутньої продукції кінобару. Оскільки цільова величина (кількість глядачів у залі) є неперервним числовим показником, математична задача зводиться до побудови моделі регресійного аналізу на основі накопичених історичних даних про попередні покази фільмів.

Під час проєктування аналітичного модуля було розглянуто кілька альтернативних математичних підходів. Першою альтернативою є класична лінійна регресія методом найменших квадратів. Проте цей метод є вразливим до явища лінійної залежності між вхідними параметрами, коли вхідні фактори (наприклад, вечірній час сеансу та статус вихідного дня) сильно пов'язані між собою, що призводить до математичної нестабільності розрахунків матриць та різкого зниження точності прогнозів. Другою альтернативою є використання моделей аналізу часових рядів, таких як ARIMA. Вони ефективно працюють із безперервними послідовними процесами, але погано підходять для кінотеатрального розкладу, де сеанси є дискретними (переривчастими) подіями з різними фільмами, тривалістю показу та характеристиками залів. Застосування складних нелінійних підходів, таких як штучні нейронні мережі або алгоритми градієнтного бустингу, було відхилено через їхню високу обчислювальну складність, вимогливість до обсягу навчальної вибірки та необхідність інтеграції важких зовнішніх бібліотек, що суперечить концепції легкого монолітного застосунку.

З огляду на обмеження альтернативних варіантів, для реалізації предиктивного контуру було обрано метод гребеневої регресії [7] . Головною перевагою цього підходу є впровадження механізму L2-регуляризації. Цей математичний апарат додає спеціальний штрафний коефіцієнт до функції

помилки моделі, що штучно обмежує надмірне зростання параметрів. Це дозволяє успішно вирішити проблему лінійної залежності між вхідними параметрами та уникнути математичної нестабільності моделі при розрахунках для нових сеансів.

Додатковим аргументом на користь гребеневої регресії є етап підготовки статистичних параметрів, який дозволяє врахувати часові коливання попиту через формування набору факторів, що включає розподіл часу сеансів на періоди, календарні чинники та використання часових лагів (показників продажів за минулі тижні) [8]. Перед початком розрахунків обов'язково виконується стандартизація вхідних даних, яка зводить усі показники до єдиного масштабу [8]. З інженерної точки зору гребенева регресія є доцільним вибором для цього проєкту, оскільки її математичний розв'язок базується на лінійно-алгебраїчних операціях з матрицями. Це дозволяє реалізувати алгоритм на чистій мові Ruby за допомогою стандартної бібліотеки matrix, забезпечуючи високу швидкість обчислень у фоновому режимі без встановлення сторонніх обчислювальних пакетів..

1.2.3 Концепції управління даними та архітектури сховищ. Для забезпечення надійності операційного обліку та касових операцій архітектура центрального сховища системи базується на реляційній моделі даних із підтримкою вимог транзакційності ACID [9]. У розробленій системі управління кінотеатром реалізація цих вимог є критичною для уникнення логічних колізій. Властивість атомарності впроваджено через механізм транзакцій фреймворку. Наприклад, процес закриття робочої зміни касира виконується як єдина неподільна операція: у разі виникнення помилки на будь-якому етапі розрахунків відбувається повне автоматичне скасування усіх пов'язаних змін у базі даних.

Захист від стану гонки при одночасних спробах різних касирів продати одне й те саме місце забезпечується властивістю узгодженості. У системі цей механізм реалізовано на рівні схеми бази даних через жорсткі унікальні обмеження. Зокрема, для моделі квитка встановлено комплексне обмеження

унікальності місця в межах конкретного сеансу. Завдяки цьому, навіть якщо два касири одночасно ініціюють процес продажу одного крісла, система управління базами даних на фізичному рівні зафіксує лише першу транзакцію, а дублюючий запис буде безумовно відхилено, що гарантує цілісність інформації про завантаженість залу [10].

У межах монолітної архітектури для збереження інформації використовується вбудована (embedded) реляційна база даних SQLite. Вона функціонує безпосередньо в процесі вебзастосунку та зберігає весь масив даних в одному локальному файлі на диску. Такий підхід ліквідує мережеві накладні витрати на взаємодію між програмою та окремим сервером бази даних. Відсутність мережевих затримок забезпечує високу швидкість читання історичної інформації, що є необхідною умовою для миттєвого формування обчислювальних матриць під час роботи модуля предиктивної аналітики.

1.2.4 Технології розробки динамічних користувацьких інтерфейсів. Сучасні тенденції веброзробки диктують жорсткі вимоги до динамічності та інтерактивності клієнтської частини інформаційних систем. Замість важких клієнтських SPA-фреймворків, які потребують окремих бандлерів та API-компонентів, у промисловій розробці набув поширення підхід Hypermedia-Driven Applications, яскравим представником якого є технологічний стек Hotwire. Концептуальна ідея компонента Turbo полягає в асинхронній доставці компактних HTML-фрагментів безпосередньо з сервера, що забезпечує швидкість відгуку інтерфейсу на рівні односторінкових додатків без штучного ускладнення клієнтської кодової бази та без повного перезавантаження сторінок [11].

Використання супутньої бібліотеки Stimulus інкапсулює необхідну клієнтську логіку, зберігаючи архітектурну чистоту проєкту. Стилізація інтерфейсу в межах загальної монолітної архітектури виконується за допомогою утилітарного фреймворку Tailwind CSS[12]. Такий підхід гарантує високу швидкість рендерингу, мінімізує фінальний розмір статичних файлів

оформлення та забезпечує високу адаптивність дизайну на різних типах пристроїв.

1.3 Розробка варіантів використання

Для чіткого визначення функціональних вимог до системи було розроблено сценарії використання (прецеденти). Цей етап дозволив проаналізувати, як саме різні типи користувачів взаємодіятимуть із програмою у реальних робочих ситуаціях. У результаті вдалося сформувати точний набір функцій, правильно розподілити права доступу між ролями та підготувати надійну базу для подальшої реалізації бізнес-логіки системи.

Проєктування варіантів використання за допомогою уніфікованої мови моделювання UML дозволяє чітко специфікувати межі інформаційної системи та зафіксувати вимоги до неї ще до початку активної фази написання програмного коду. Суть цього підходу полягає в тому, що розроблювана система подається у вигляді взаємодії акторів (користувачів) із конкретними функціями додатка. Кожен варіант використання визначає певну послідовність дій, яку виконує програма під час діалогу з працівником, описуючи сервіси, які система надає користувачу, без ускладнення опису низькорівневими деталями коду. Загальна структура взаємодії акторів із системою та межі програмного комплексу відображені на розробленій діаграмі прецедентів на рисунку 1.3.

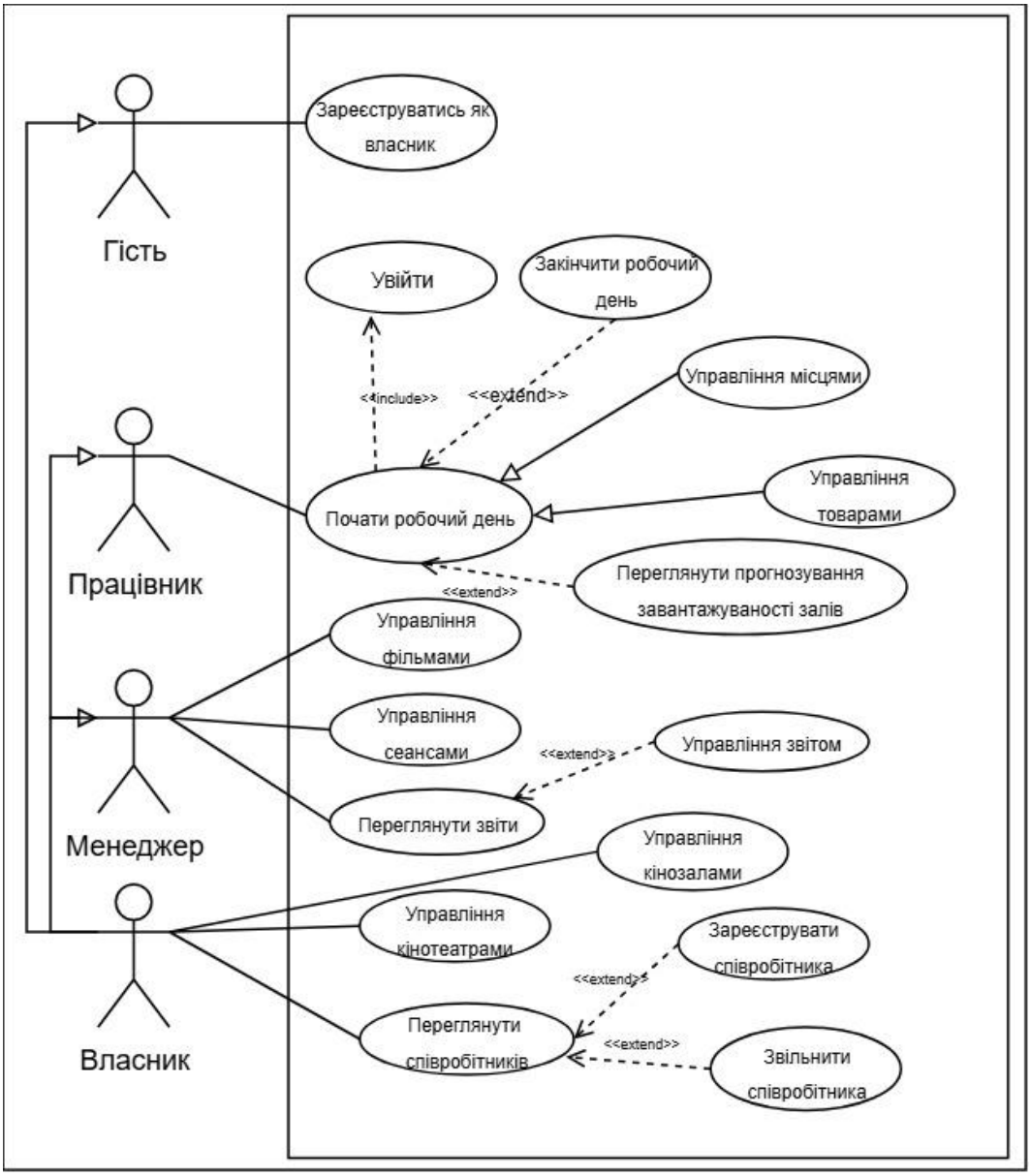

Рисунок 1.3 – Діаграма прецедентів

На основі розробленої графічної моделі виділено три головні сценарії, які детально описують операційну та адміністративну основу програми.

1.3.1. Прецедент “Управління структурою кінозалів”. Власник налаштовує структуру залів у програмі. Для моделювання залу користувач вносить його базові параметри, а саме загальну кількість рядів та кількість місць у кожному ряду. На основі цих цифр система автоматично будує та фіксує схему залу.

Поверхнева форма. Власник входить у систему і додає новий зал. Він вказує назву залу, кількість рядів та кількість місць у ряду. Програма

автоматично розраховує місткість та формує структуру приміщення. Після підтвердження дані зберігаються на сервері. Якщо параметри введено з помилкою, програма показує підказку.

Повна форма.

Назва прецеденту: «Створити та налаштувати конфігурацію кінозалу».

Сфера застосування: веб-інтерфейс адміністратора та серверна частина.

Рівень: ціль користувача.

Головні актори: власник.

Другорядні актори: сервер на базі Ruby on Rails, база даних.

Зацікавлені сторони та їхні інтереси:

– власник: прагне швидко та точно внести параметри залу в програму для подальшого продажу квитків;

– розробник: зацікавлений у надійному збереженні структури залу.

Передумови:

– користувач увійшов у систему з правами власника;

– у системі створено хоча б один кінотеатр.

Гарантія успіху:

– новий зал збережено у списку об'єктів;

– конфігурацію залу успішно зафіксовано в базі даних.

Основний сценарій успіху:

1 власник відкриває розділ управління кінозалами;

2 ініціює створення нового залу;

3 вводить назву залу, кількість рядів та місць у кожному ряду;

4 система автоматично будує конфігурацію залу за вказаними параметрами;

5 власник підтверджує збереження змін;

6 система оновлює дані та показує повідомлення про успіх.

Альтернативні сценарії:

а) Спроба збереження з помилками:

1 система знаходить помилку у введених числових даних;

2 користувач отримує сповіщення з описом проблеми під відповідним полем;

3 користувач виправляє значення та повторює дію.

Спеціальні вимоги:

– інтерфейс введення параметрів має бути простим і зрозумілим.

Список технологій:

– серверна частина: фреймворк Ruby on Rails (з використанням модулів Action Controller для обробки запитів та Active Record для об'єктно-реляційного відображення);

– клієнтська частина: HTML5, CSS3, JavaScript (DOM API для забезпечення інтерактивної взаємодії зі схемою залу в браузері);

– база даних: SQLite;

– формат обміну даними: JSON (використовується для асинхронного збереження статусів місць через AJAX-запити без перезавантаження сторінки).

Частота використання: низька (виконується при відкритті нових залів).

1.3.2. Прецедент "Обслуговування клієнтів та продаж квитків". Працівник продає квитки відвідувачам кінотеатру. Головною особливістю є те, що у вікні вибраного сеансу вже відображається готовий аналітичний прогноз завантаженості залу. Це дозволяє працівнику бачити очікуваний попит відразу на робочому екрані без додаткових переходів чи дій у системі.

Поверхнева форма. Працівник починає зміну та відкриває розклад. У вікні сеансу вже виведено індикатор прогнозу завантаженості залу, що допомагає краще орієнтувати покупців. Працівник бачить карту залу, відмічає обрані вільні місця і проводить продаж. Програма відразу оновлює стан залу.

Повна форма.

Назва прецеденту: «Продати квиток з відображенням прогнозу завантаженості».

Сфера застосування: інтерфейс працівника та серверна частина.

Рівень: ціль користувача.

Головні актори: працівник.

Другорядні актори: сервер на базі Ruby on Rails, аналітичний модуль.

Зацікавлені сторони та їхні інтереси:

– працівник: хоче швидко обслужити відвідувача, бачачи актуальний прогноз для сеансу;

– клієнт: хоче обрати та придбати зручні місця у залі.

Передумови:

– працівник увійшов у систему та розпочав робочий день;

– сеанс додано до загального розкладу.

Гарантія успіху:

– статус обраного місця в базі даних SQLite змінено на «продано».

– операція успішно зафіксована у системі.

Основний сценарій успіху:

1 працівник відкриває розклад сеансів на поточну зміну;

2 вибирає потрібний сеанс, у вікні якого відразу відображається індикатор прогнозу завантаженості;

3 відкриває карту залу та відмічає вільні місця подвійним кліком;

4 зберігає зміни та завершує транзакцію.

Альтернативні сценарії:

а) Спроба продажу недоступного або зайнятого місця:

1 система знаходить конфлікт або блокування місця;

2 працівник отримує повідомлення про помилку та пропонує клієнту інші варіанти.

Спеціальні вимоги:

– індикатор прогнозу у вікні сеансу має працювати стабільно та не уповільнювати завантаження розкладу.

Список технологій:

Серверна частина: фреймворк Ruby on Rails.

– статистичний аналіз: алгоритм математичного моделювання Ridge Regression (гребенева регресія) для динамічного розрахунку прогнозу завантаженості;

– клієнтська частина: HTML5, CSS3, JavaScript (для рендерингу карти вільних/зайнятих місць та візуалізації кольорових індикаторів попиту);

– база даних: SQLite (із застосуванням транзакцій для блокування місць під час продажу).

Частота використання: дуже висока.

1.3.3. Прецедент “Моніторинг та управління звітністю”. Менеджер або власник системи здійснює контроль за фінансовими показниками роботи через модуль звітності. Програма автоматично формує звітні дані після завершення робочих змін. Керівництво може переглядати цю інформацію, аналізувати показники та змінювати статуси звітів для внутрішнього обліку.

Поверхнева форма. Коли керівник відкриває список звітів, він бачить усі збережені документи. Після вибору запису він переглядає детальну статистику. Далі він може завантажити документ на комп'ютер або змінити його статус для підтвердження перевірки. Система зберігає ці зміни.

Повна форма.

Назва прецеденту: «Перегляд та управління фінансовими звітами».

Сфера застосування: веб-інтерфейс адміністратора та серверна частина.

Рівень: ціль користувача.

Головні актори: менеджер, власник.

Другорядні актори: сервер на базі Ruby on Rails.

Зацікавлені сторони та їхні інтереси:

– керівництво: прагне отримувати актуальну інформацію про прибутки кінотеатрів;

– бухгалтерія: потребує точних валідованих даних для обліку.

Передумови:

– користувач авторизований у системі з відповідним рівнем доступу;

– у базі даних наявні історичні дані про закриті зміни та продажі.

Гарантія успіху:

– звіт коректно виведено на екран;

– статус звіту успішно оновлено в системі за запитом користувача.

Основний сценарій успіху:

1 користувач відкриває сторінку перегляду звітів;

2 система показує список доступних документів за обраний період;

3 користувач вибирає конкретний звіт для детального ознайомлення;

4 натискає кнопку для завантаження або зміни статусу документа;

5 система обробляє запит і оновлює атрибути запису в базі даних;

6 користувач отримує підтвердження успішного виконання операції.

Альтернативні сценарії:

а) Відсутність даних за обраний період:

1 користувач запитує звіт за час, коли філії не працювали;

2 система сповіщає про те, що дані для формування документа відсутні;

3 спеціальні вимоги;

4 фінансові дані у звітах мають бути захищені від випадкових змін звичайними працівниками.

Список технологій:

Backend: Ruby on Rails.

– серверна частина: фреймворк Ruby on Rails;

– клієнтська частина: HTML5, CSS3, JavaScript (для відображення аналітичних зведених таблиць);

– база даних: SQLite (використання агрегатних SQL-запитів для швидкого підрахунку фінансових показників);

– генерація документів: спеціалізовані бібліотеки (gem-пакети) середовища Ruby для експорту та формування електронних файлів (наприклад, генерація PDF або CSV-документів).

Частота використання: середня.

Шляхом визначення ключових акторів системи (власника, менеджера та працівника каси) та опису їхніх основних сценаріїв взаємодії із програмою, було сформовано повне та чітке уявлення про функціональні вимоги до розроблюваного монолітного вебзастосунку, що дозволило перейти до етапу безпосереднього проєктування бази даних та алгоритмів предиктивного модуля.

1.4 Постановка задачі

Успішна робота будь-якого сучасного кінотеатру залежить від правильного планування ресурсів. Головна проблема цієї сфери – складність передбачити, скільки саме глядачів прийде на конкретний сеанс. Зараз більшість розважальних закладів роблять це приблизно, спираючись лише на досвід менеджерів.

Такий підхід часто призводить до фінансових втрат. Якщо на сеанс приходить несподівано багато людей, а на зміні працює мало працівників, виникають великі черги, якість обслуговування падає, і кінотеатр втрачає прибуток. Якщо ж навпаки – очікуваного ажіотажу немає, власник зазнає збитків через оплату роботи зайвого персоналу та псування продуктів у барі, які заздалегідь приготували для великої кількості людей.

Тому основною задачею цієї роботи є розробка інформаційної веб-системи, яка автоматизує збір даних про роботу кінотеатру та містить аналітичний модуль для прогнозування відвідуваності сеансів.

Для досягнення цієї мети програмний додаток має вирішувати такі конкретні завдання:

1 управління базовими процесами: створення зручного інтерфейсу для внесення інформації про кінотеатри, характеристики залів (місткість, кількість рядів та місць) та розклад фільмів;

2 облік та збереження даних: автоматична фіксація кожного проданого квитка. Це необхідно для накопичення точної історії відвідуваності, яка є основою для подальшого аналізу;

3 автоматичне прогнозування: розробка математичного модуля на основі алгоритмів статистичного аналізу. Алгоритм повинен самостійно аналізувати накопичену історію та розраховувати очікуваний відсоток завантаженості залу для майбутніх сеансів;

4 врахування ключових чинників: налаштування системи так, щоб при розрахунку прогнозу враховувалися три головні фактори: середня популярність конкретного фільму, день тижня (будній чи вихідний) та час початку сеансу (денний чи вечірній час);

5 допомога у прийнятті рішень: виведення готових результатів прогнозу у вигляді чітких числових показників (кількість очікуваних квитків та відсоток заповненості залу), щоб адміністрація могла заздалегідь бачити пікові дні та оптимізувати графік роботи працівників.

У результаті розробки має бути створено надійний інструмент, який замінить ручне планування на точний розрахунок. Це дозволить керівництву кінотеатру зменшити операційні витрати, уникнути черг та підвищити загальну ефективність бізнесу.

Висновки до розділу 1

У першому розділі виконано аналіз предметної області та обґрунтовано доцільність розробки інформаційної системи для управління мережею кінотеатрів із вбудованим модулем прогнозування.

Проведено огляд існуючих програмних рішень на ринку автоматизації квиткових процесів, зокрема платформ «mticket» та «SERVIO Tickets».

Виділено їхні ключові переваги та недоліки. Аналіз показав, що попри широкі функціональні можливості, ці системи мають спільні обмеження: жорстку прив’язку до стаціонарного робочого місця у десктопних версіях, високі комісійні збори за кожну транзакцію та зберігання клієнтської бази на сторонніх серверах. Це підтвердило необхідність розробки власного незалежного веб-рішення, яке усуне виявлені недоліки, поєднавши мобільність сучасних технологій із глибоким управлінським обліком.

Здійснено огляд підходів і технологій, що застосовуються при розробці подібних застосунків, та обґрунтовано вибір інструментів. Архітектуру системи побудовано за шаблоном MVC на базі фреймворку Ruby on Rails із використанням легкої реляційної бази даних SQLite. Це забезпечує високу швидкість розробки, надійність транзакцій та спрощує розгортання програми. Клієнтську частину реалізовано за допомогою кросплатформного стеку HTML, CSS та JavaScript для зручної роботи на будь-яких пристроях (від комп'ютерів до планшетів). Для математичного аналізу інтегровано модуль статистичного прогнозування (алгоритм гребеневої регресії), який дозволяє розраховувати очікувану завантаженість залів безпосередньо всередині системи..

На основі результатів аналізу сформульовано постановку задачі на розробку. Визначено головну мету та ключові завдання системи, які охоплюють автоматизацію продажу квитків, управління розкладом сеансів, ведення консолідованої фінансової звітності та контроль за інфраструктурою залів.

Розроблені варіанти використання (прецеденти) чітко визначили функціональність системи та межі програмного комплексу. Побудовано діаграму прецедентів та виділено чотири ієрархічні ролі: гість, працівник, менеджер та власник. Детально описано три основні сценарії взаємодії: створення конфігурації залів власником, продаж квитків працівником із використанням візуального прогнозу попиту, а також моніторинг фінансової звітності керівництвом. Це дозволило зафіксувати всі бізнес-вимоги, чітко

розподілити права доступу та підготувати надійний фундамент для подальшого проєктування бази даних і написання програмного коду.

2 ПРОЄКТУВАННЯ ТА АРХІТЕКТУРА ІНФОРМАЦІЙНОЇ СИСТЕМИ УПРАВЛІННЯ КІНОТЕАТРОМ

2.1 Проєктування структури бази даних

Основою роботи інформаційної системи є надійна структура зберігання даних. Для розробки програми було обрано реляційну модель даних на базі системи управління базами даних SQLite [9]. Цей вибір зумовлений легкістю налаштування, відсутністю потреби в окремому сервері баз даних та повною сумісністю із фреймворком Ruby on Rails через вбудований механізм Active Record.

Усі таблиці в базі даних мають первинні ключі (PK) для унікальної ідентифікації кожного запису. Зв'язки між таблицями реалізовано за допомогою зовнішніх ключів (FK), що забезпечує цілісність інформації [10]. Загальну логічну структуру бази даних із відображенням усіх сутностей, полів та зв'язків у вигляді ER-діаграми зображено на рисунку 2.1.

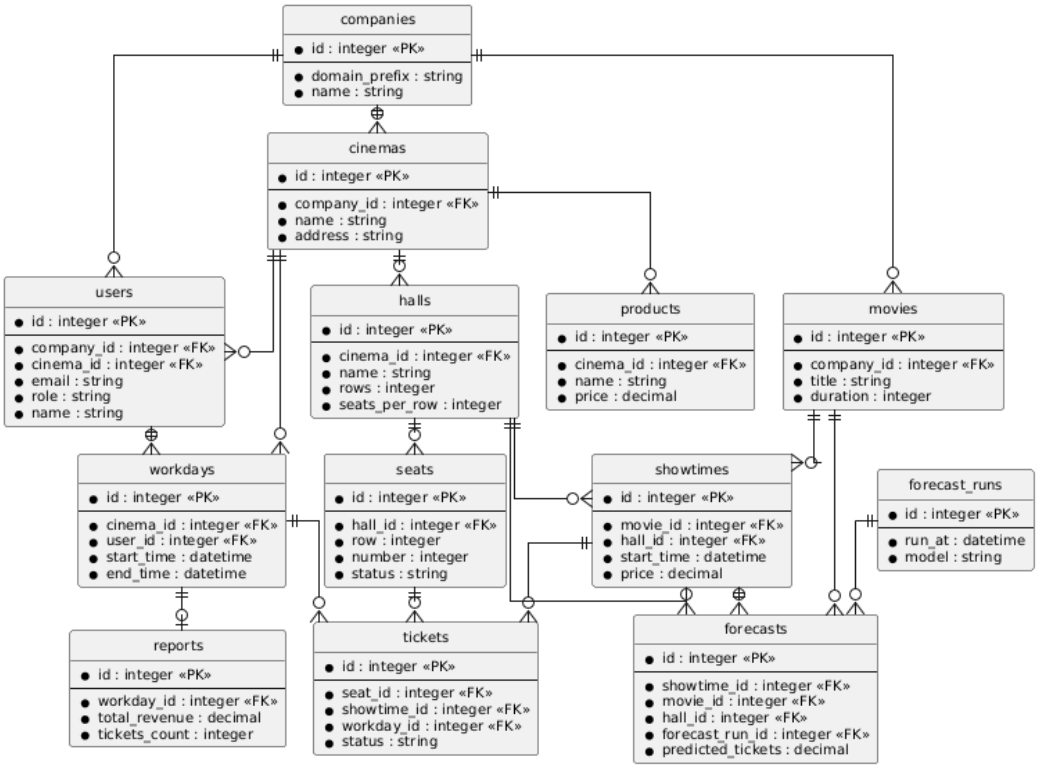

Рисунок 2.1 – Логічна структура бази даних інформаційної системи кінотеатру

Основними компонентами серверної бази даних є наступні таблиці.

Таблиця companies зберігає загальну інформацію про мережу:

– id – унікальний ідентифікатор компанії;

– name – назва мережі кінотеатрів;

– domain_prefix – унікальний текстовий префікс компанії для верифікації доменів електронної пошти;

– created_at, updated_at – часові мітки створення та оновлення запису.

Таблиця cinemas містить дані про конкретні філії (кінотеатри):

– id – унікальний ідентифікатор філії;

– company_id – зовнішній ключ для зв'язку з компанією;

– name – назва кінотеатру;

– address – фізична адреса розташування;

– created_at, updated_at – часові мітки створення та оновлення запису.

Таблиця users забезпечує ідентифікацію, авторизацію та розмежування прав персоналу:

– id – унікальний ідентифікатор користувача (первинний ключ);

– company_id – зовнішній ключ компанії;

– cinema_id – зовнішній ключ кінотеатру (філії);

– email – електронна пошта (логін);

– role – рівень доступу в системі;

– name – повне ім'я співробітника.

Таблиця halls описує конфігурацію кінозалів:

– id – унікальний ідентифікатор залу;

– cinema_id – зовнішній ключ кінотеатру, якому належить зал;

– name – назва або номер залу;

– rows – загальна кількість рядів у залі;

– seats_per_row – кількість місць у кожному ряду;

Таблиця seats зберігає інформацію про кожне конкретне місце:

– id – унікальний ідентифікатор місця;

– hall_id – зовнішній ключ залу;

– row – номер ряду;

– number – порядковий номер місця в ряду;

– status – поточний технічний стан місця (наприклад, доступне або заблоковане);

Таблиця products відповідає за облік товарів кінобару:

– id – унікальний ідентифікатор продукту (первинний ключ);

– cinema_id – зовнішній ключ кінотеатру;

– name – назва товару;

– price – вартість одиниці.

Таблиця movies представляє каталог фільмів:

– id – унікальний ідентифікатор фільму;

– company_id – зовнішній ключ компанії;

– title – назва кінокартини;

– duration – тривалість фільму у хвилинах;

Таблиця showtimes відповідає за розклад сеансів:

– id – унікальний ідентифікатор сеансу;

– movie_id – зовнішній ключ фільму;

– hall_id – зовнішній ключ залу;

– start_time – дата та час початку показу;

– price – вартість квитка на даний сеанс;

Таблиця workdays фіксує робочі зміни персоналу:

– id – унікальний ідентифікатор зміни;

– cinema_id – зовнішній ключ кінотеатру;

– user_id – зовнішній ключ працівника;

– start_time – час початку робочої зміни;

– end_time – час закриття зміни.

Таблиця tickets фіксує факт продажу квитків:

– id – унікальний ідентифікатор квитка;

– seat_id – зовнішній ключ обраного місця;

– showtime_id – зовнішній ключ сеансу;

– workday_id – зовнішній ключ робочої зміни, під час якої здійснено продаж;

– status – статус квитка (наприклад, продано);

Таблиця reports зберігає підсумкові фінансові дані:

– id – унікальний ідентифікатор звіту;

– workday_id – зовнішній ключ робочої зміни;

– total_revenue – загальна сума доходу за зміну;

– tickets_count – загальна кількість проданих квитків;

Таблиця forecast_runs виступає журналом запусків аналітичного модуля:

– id – унікальний ідентифікатор процесу розрахунку (первинний ключ);

– run_at – дата та час запуску прогнозування;

– model – назва застосованого математичного алгоритму.

Таблиця forecasts зберігає результати роботи модуля математичного прогнозування:

– id – унікальний ідентифікатор прогнозу (первинний ключ);

– showtime_id – зовнішній ключ сеансу;

– movie_id – зовнішній ключ фільму;

– hall_id – зовнішній ключ кінозалу;

– forecast_run_id – зовнішній ключ пакетного запуску;

– predicted_tickets – очікувана кількість проданих квитків.

Організація зв'язків між проектованими таблицями обумовлена реальними бізнес-процесами та ієрархічною структурою мережі кінотеатрів. Зв'язки типу «один до багатьох» між таблицями companies, cinemas, halls та seats відображають фізичний устрій підприємства, де одна компанія володіє кількома філіями, кожна філія містить декілька залів, а кожен зал складається

з фіксованої сукупності крісел. Таблиця showtimes виступає центральним сполучним вузлом, пов'язуючи сутності фільмів та залів, що дозволяє зафіксувати часові координати кожного показу.

Зв'язок «один до багатьох» між таблицями seats та tickets є критично важливим, оскільки одне й те саме статичне місце в залі протягом тривалого часу буде багаторазово продаватися на різні сеанси, генеруючи велику кількість записів у таблиці квитків.. У свою чергу, таблиця робочих змін workdays виступає контекстом для фінансової звітності: вона зв'язує користувача з філією та слугує власником для багатьох квитків, проданих касиром протягом дня, закінчуючись генерацією рівно одного підсумкового документа в таблиці reports.

Логічна структура аналітичного контуру побудована на зв'язку таблиці forecast_runs із сутністю forecasts. Оскільки процес навчання регресії відбувається періодично у фоновому режимі, один запуск обчислень створює один запис у журналі forecast_runs, який володіє багатьма предиктивними записами в forecasts для кожного майбутнього сеансу окремо. Прямі зовнішні ключі на фільми та зали всередині прогнозів оптимізують швидкість виконання SQL-запитів при формуванні зведених графіків на панелі менеджера.

Фізична реалізація спроєктованої реляційної структури бази даних у середовищі Ruby on Rails здійснюється за допомогою механізму міграцій. Поточний стан архітектури даних автоматично фіксується у системному конфігураційному файлі schema.rb з використанням предметно-орієнтованої мови фреймворку. З огляду на значний обсяг програмного коду, повний лістинг файлу schema.rb, що містить декларативний опис усіх системних сутностей, їхніх полів, типів даних та індексів зовнішніх ключів, винесено у Додаток А.

2.2 Вибір архітектури додатка

Під час проєктування інформаційної системи одним із ключових завдань є вибір глобального архітектурного підходу до організації кодової бази. Для розроблюваного програмного комплексу управління кінотеатрами було обрано монолітну архітектуру. На відміну від мікросервісного підходу, де кожна функція виноситься в окремий незалежний застосунок, класичний моноліт передбачає об'єднання всіх функціональних модулів (управління залами, продаж квитків, генерація звітів та модуль прогнозування) у межах однієї кодової бази, що запускається на єдиній платформі [13].

Вибір монолітної архітектури для цього проєкту є найбільш доцільним з інженерної та практичної точок зору. У науково-технічних працях з архітектурного моделювання зазначається, що для систем середнього масштабу та індивідуальної розробки монолітна структура є оптимальною, оскільки вона усуває складні витрати на міжсервісну мережеву взаємодію, забезпечує просту транзакційність та значно спрощує розгортання [13]. Крім того, фреймворк Ruby on Rails історично оптимізований саме під концепцію модульного моноліту, що дозволяє забезпечити високу швидкість розробки та легкість масштабування без штучного ускладнення серверної інфраструктури. Усі компоненти системи тісно пов'язані через єдину реляційну базу даних SQLite, тому використання єдиного серверного простору виключає затримки під час обміну даними між модулями продажів та статистичного аналізу.

Фізична архітектура проєкту відображається у стандартизованій структурі каталогів, яку автоматично генерує фреймворк під час ініціалізації системи [8]. Така сувора організація дозволяє розробнику підтримувати високу модульність, чітко розділяти системні та користувацькі компоненти, а також забезпечувати ефективне тестування й супровід проєкту [14]. Повний склад та призначення ключових директив і файлів розробленого вебзастосунку зведено у таблиці 2.

Таблиця 2.1 – Файлова структура Ruby on Rails додатку

Назва директорії/файлу	Призначення
app/	Містить основний код програми.
bin/	Скрипти для запуску, оновлення і налаштування програми.
config/	Файли конфігурації (наприклад, routes.rb, database.yml).
db/	Міграції, схема бази даних (schema.rb), початкові дані (seeds.rb).
lib/	Кастомні бібліотеки, модулі, утиліти.
log/	Логи роботи додатку.
public/	Статичні файли, які віддаються напряму веб-сервером (favicon, error pages).
coverage/	Файли для відстеження покритого тестами коду
spec/	Тестові файли (unit, controller, integration).
tmp/	Тимчасові файли, що використовуються під час виконання.
vendor/	Сторонні залежності: бібліотеки, плагіни, CSS/JS-фреймворки.
Gemfile	Список Ruby-гемів (залежностей) для додатку.
Gemfile.lock	Зафіксовані версії гемів, щоб забезпечити однакові залежності.
Rakefile	Опис задач, які можна виконати через rake.
README.md	Документація до проекту.

Завдяки наведеній стандартизованій структурі вдалося створити організований, передбачуваний код, який зручно підтримувати й розширювати. Наявність виділених каталогів для автоматизованого тестування (spec/ та coverage/) дозволяє контролювати стабільність роботи бізнес-логіки продажу квитків та модуля математичного прогнозування. У свою чергу, чітке ведення конфігурацій та зовнішніх залежностей через інструменти логування (log/) та менеджер пакетів (Gemfile) гарантує безпечне функціонування всієї екосистеми вебзастосунку на базі бази даних SQLite.

Для забезпечення стабільної роботи інформаційної системи, відтворюваності середовища розробки та уникнення конфліктів сумісності, у проєкті чітко зафіксовано версії базового технологічного стеку за допомогою менеджера залежностей Bundler.

Основою програмного комплексу є мова програмування Ruby (версія 3.3.5) та вебфреймворк Ruby on Rails (версія 8.1.3). Їх використання забезпечує високу швидкість обробки даних та дозволяє працювати з фоновими завданнями без налаштування додаткових зовнішніх серверів. Для підключення та керування сторонніми бібліотеками у проєкті використовується файл конфігурації Gemfile. У ньому чітко зафіксовані всі програмні залежності (геми), які згруповані за своїм функціональним призначенням. Основний перелік цих інструментів наведено у таблиці 2.2.

Таблиця 2.2 – Основні програмні залежності (Gems) інформаційної системи

Назва пакета (Gem)	Версія / Умова	Функціональне призначення у системі
sqlite3	>= 2.1	Адаптер для взаємодії з реляційною системою управління базами даних SQLite.
puma	>= 5.0	Високопродуктивний багатопотоковий вебсервер для обробки HTTP-запитів.
devise	~> 5.0	Комплексний модуль автентифікації, криптографії та управління сесіями користувачів.
turbo-rails, stimulus-rails	стандартна	Забезпечення асинхронної взаємодії та динамічного оновлення інтерфейсу (стек Hotwire).
tailwindcss-rails	~> 4.4	Інтеграція утилітарного CSS-фреймворку Tailwind для кросплатформної стилізації сторінок.
solid_queue, solid_cache, solid_cable	стандартна	Інфраструктурні компоненти для виконання фонових завдань, кешування та WebSockets безпосередньо засобами БД.
prawn	~> 2.5	Бібліотека для динамічної генерації електронних квитків та фінансових звітів у форматі PDF.
propshaft, importmap-rails	стандартна	Сучасний конвеєр збірки для управління статичними активами та JavaScript-модулями без використання Node.js.

Як видно з таблиці, підібраний набір інструментів покриває всі технічні потреби проєкту: від роботи з базою даних до побудови інтерфейсу та генерації документів. Система не перевантажена зайвими бібліотеками, а складні обчислення виконуються стандартними засобами мови Ruby. Отже, монолітна архітектура та обраний стек технологій формують необхідну технічну базу програми. Це дозволяє перейти до етапу проєктування взаємодії між внутрішніми компонентами системи за допомогою архітектурного шаблону MVC.

2.3 Реалізація архітектурного шаблону MVC

Для побудови розроблюваного програмного комплексу було обрано архітектурний шаблон MVC (Model-View-Controller). Відповідно до сучасних стандартів проєктування вебзастосунків, цей патерн забезпечує чітке розділення системи на три взаємопов'язані, але логічно незалежні компоненти: модель (дані та бізнес-логіка), представлення (інтерфейс користувача) та контролер (обробка запитів) [15]. Як зазначається у профільних матеріалах з архітектури Rails, таке розділення обов'язків (Separation of Concerns) значно полегшує процеси тестування, підтримки та масштабування системи, оскільки розробники можуть змінювати візуальну частину без ризику порушити логіку обробки баз даних [15].

Рівень моделі у спроєктованій системі відповідає за взаємодію з реляційною базою даних SQLite та виконання ключових алгоритмів бізнес-логіки. У середовищі Ruby on Rails цей компонент реалізується через патерн Active Record. Згідно з офіційною документацією фреймворку, Active Record виконує функцію об'єктно-реляційного відображення, дозволяючи об'єктам системи одночасно нести в собі збережені дані та логіку роботи з ними [16]. Вперше цей патерн був описаний Мартіном Фаулером в інженерній літературі як «об'єкт, що обгортає рядок таблиці бази даних, інкапсулює доступ до неї та додає доменну логіку до цих даних». Такий підхід дає змогу працювати з

таблицями (наприклад, кінозалами чи квитками) як із класами об'єктно-орієнтованої мови програмування, уникаючи написання прямих та громіздких SQL-запитів[16]. Саме в моделях інкапсульовано всі правила валідації вхідних даних перед їхнім збереженням. Крім того, на цьому рівні реалізовано програмну інтеграцію з модулем статистичного прогнозування для автоматичного розрахунку очікуваної завантаженості сеансів.

За формування візуальної частини та виведення інформації кінцевому користувачеві відповідає рівень представлення (View). Його головним завданням є відображення інтерфейсу користувача, рендеринг даних, отриманих від моделей, та збір користувацького вводу через екранні форми. У розробленій системі представлення створюються за допомогою вбудованого шаблонізатора ERB, який ефективно поєднує стандартну HTML-розмітку з динамічними даними. Інтерфейс динамічно підлаштовується під поточну роль авторизованого користувача: для працівника відображається інтерактивна карта залу з індикатором прогнозу попиту, а для власника бізнесу генеруються закриті модулі фінансової звітності.

Сполучною ланкою, яка координує функціонування всієї системи, виступає рівень контролера. Його основна функція полягає у прийнятті вхідних HTTP-запитів від клієнтського браузера, взаємодії з моделями для отримання або оновлення записів та виборі відповідного представлення для формування остаточної відповіді. Наприклад, під час продажу квитка контролер перевіряє рівень доступу працівника, запитує з бази даних SQLite інформацію про доступні місця та поточний прогноз завантаженості. Після підтвердження вибору контролер ініціює транзакцію через відповідну модель, фіксує зміни в базі даних і повертає оновлене вікно з повідомленням про успішне виконання операції.

2.4 Проєктування та обґрунтування структури модуля математичного прогнозування

Інтеграція підсистеми статистичного прогнозування завантаженості сеансів у загальну структуру системи вимагає створення ізольованого архітектурного контуру. Головною інженерною проблемою при побудові таких модулів усередині вебзастосунків є висока обчислювальна складність матричних операцій лінійної алгебри. Виконання цих розрахунків у головному потоці обробки запитів не відповідає вимогам точного обліку та швидкої відповіді сервера, оскільки може призвести до блокування вебсервера та уповільнення відгуку інтерфейсу користувача. Для вирішення цієї проблеми архітектуру модуля спроєктовано на основі принципу розділення відповідальності (Separation of Concerns). Усі обчислювальні алгоритми винесено в автономні сервісні об'єкти простору імен Forecasting, які виконуються як асинхронні фонові завдання (Background Jobs).

Модуль прогнозування реалізовано як цілісний фоновий конвеєр, що активується асинхронним завданням ForecastRunnerJob із визначеним часовим горизонтом планування, який за замовчуванням становить сім днів. Конвеєр послідовно проходить чотири етапи:

1 ініціалізація запису запуску та очищення застарілих даних;

2 завантаження історичної вибірки з реляційного сховища;

3 математичне моделювання за часовими сегментами;

4 генерація та збереження прогнозів для майбутнього розкладу.

Кожен із цих етапів та їхнє архітектурне обґрунтування детально розглянуто в наступних підпунктах.

2.4.1 Архітектурна організація та етапи фонового конвеєра прогнозування. Перед початком математичного моделювання конвеєр виконує підготовчі операції. Спочатку він фіксує факт запуску шляхом створення системного запису ForecastRun із точною міткою часу. Після цього виконується превентивне очищення застарілих прогнозів у визначеному

часовому вікні – від початку поточної доби до кінця граничного дня горизонту планування. Такий підхід гарантує уникнення дублікатів та конфліктів даних при повторних запусках програми.

Аналіз предметної області показує, що поведінка глядачів суттєво відрізняється залежно від часу сеансу. Тому алгоритм прогнозування використовує стратегію жорсткої сегментації, автоматично поділяючи історичну вибірку на три незалежні простори: денні сеанси будніх днів, вечірні покази буднів та сеанси вихідних днів. Розподіл виконується за чітким правилом: якщо день показу припадає на суботу або неділю, система безумовно відносить його до простору вихідного дня. Для будніх днів критерієм виступає граничний час: сеанси, що розпочинаються до 18:00, класифікуються як денні, а покази о 18:00 та пізніше – як вечірні. Цей поділ зумовлений різними життєвими циклами аудиторії: вдень у будні попит зазвичай низький і формується зі студентів чи людей із вільним графіком, увечері активність зростає після завершення робочого дня, а на вихідних досягає максимуму за рахунок сімейних відвідувань.

Для кожного з цих трьох сегментів у межах конкретного кінозалу будується індивідуальна математична модель, що дозволяє максимально точно врахувати специфіку попиту. Проте виникає інженерний ризик: у нових кінозалах або для специфічних ранкових слотів історичних даних може бути недостатньо для коректного математичного розрахунку. Для захисту від помилок обчислення (сингулярності матриць) до архітектури закладено стратегію стійкого деградування.

Перед ініціалізацією навчання система перевіряє розмір зібраної вибірки: якщо для певного часового сегмента накопичено менше п'яти прецедентів, індивідуальна модель для цього слоту відхиляється. Замість цього конвеєр автоматично перемикає обчислення на резервну глобальну модель, яка навчається на загальному масиві історичних даних усього залу без розподілу за часом. Такий архітектурний запобіжник гарантує стабільну та

безперебійну генерацію прогнозів незалежно від обсягу накопиченої статистики.

2.4.2 Алгоритм сегментації даних та стратегія стійкого деградування. Аналіз предметної області показує, що поведінка глядачів суттєво відрізняється залежно від часу сеансу. Тому алгоритм прогнозування використовує стратегію жорсткої сегментації. На архітектурному рівні простори сеансів зафіксовано у вигляді констант (рис. 2.2), які поділяють вибірку на денні сеанси буднів, вечірні покази будніх днів та сеанси вихідних днів.

```
SEGMENTS = %i[weekday_day weekday_night weekend].freeze
```

Рисунок 2.2 – Програмна декларація часових сегментів моделі

Розподіл історичних та майбутніх сеансів виконується за єдиним детермінованим алгоритмом. Якщо день проведення показу припадає на суботу або неділю, система безумовно відносить сеанс до простору вихідного дня. Для будніх днів критерієм виступає граничний час: сеанси, що розпочинаються до 18:00, класифікуються як денні, а покази о 18:00 та пізніше – як вечірні. Цей поділ зумовлений різними життєвими циклами аудиторії: вдень у будні попит зазвичай низький і складається зі студентів чи людей із вільним графіком, увечері попит зростає після завершення робочого дня, а на вихідних досягає максимуму через сімейні відвідування.

Для кожного з цих трьох сегментів у межах конкретного кінозалу будується індивідуальна математична модель, що дозволяє врахувати специфіку коливання попиту. Проте виникає інженерний ризик: у нових кінозалах або для рідкісних ранкових сеансів історичних даних може бути недостатньо для коректного розрахунку. Для захисту від сингулярності (необоротності) матриць спроєктовано стратегію стійкого деградування. Перед ініціалізацією математичного тренера система перевіряє розмір сформованої вибірки, і якщо для певного сегмента накопичено менше п'яти прецедентів, індивідуальна модель для цього слоту не будується. Замість

цього конвеєр перемикає обчислення на резервну глобальну модель, яка навчається на загальному масиві історичних даних усього залу без розподілу за часом, що гарантує стабільність генерації прогнозів.

2.4.3 Обґрунтування та структура набору вхідних статистичних факторів. Для побудови розрахункової вибірки алгоритм суворо відфільтровує незавершені покази, відсікаючи всі сеанси, що відбулися менше ніж 24 години тому, що унеможливлює явище викривлення даних. Спроєктований набір вхідних параметрів декларативно визначено у базовому сервісному класі Forecasting::RegressionTrainer, що продемонстровано на рисунку 2.3.

```
def self.feature_names
  %i[movie_popularity days_since_release lag_feature is_weekend is_morning
   is_afternoon is_evening day_of_week week_of_month is_holiday
   lag_2_weeks rolling_avg_7d]
end
```

Рисунок 2.3 – Структура набору статистичних параметрів

Усі параметри розділені на три логічні групи. Перша група описує базовий попит і включає показник популярності фільму (movie_popularity). Він розраховується як статистична медіана заповнюваності всіх минулих сеансів конкретної стрічки. Вибір медіани замість середнього арифметичного є важливим, оскільки він нівелює вплив випадкових одиничних аншлагів на загальну оцінку базового попиту. До цієї ж групи належить показник життєвого циклу релізу (days_since_release), який моделює поступовий спад відвідуваності в міру старіння фільму в прокаті.

У випадку прем'єрних показів, коли фільм є новим і ще не має історичної вибірки (проблема «холодного старту»), алгоритм використовує базові константи: підставляється значення 40% очікуваної заповнюваності для перших днів прокату та 25% для подальших періодів, що базується на усереднених показниках кіномережі.

Друга група параметрів реалізує принцип авторегресії та фіксує загальні тренди. Сюди входять часові лаги (lag_feature та lag_2_weeks), які вираховують відсоток заповнюваності аналогічного сеансу рівно один та два тижні тому у цьому ж залі. Локальний загальний тренд споживчої активності фіксується через показник ковзного середнього (rolling_avg_7d), що відображає усереднену завантаженість залу за останні сім діб незалежно від того, який саме фільм транслювався.

Третя група містить календарні та часові чинники. Маркери часу доби (is_morning, is_afternoon, is_evening) розділяють добу на три фіксовані інтервали за допомогою бінарних змінних (значення нуль або одиниця). Аналогічно реалізовано прапорці вихідного дня (is_weekend) та державних свят (is_holiday), причому логіка системи підтримує перевірку як фіксованих дат, так і рухомих свят. На завершення набір даних доповнюється індексами часу: порядковим номером дня тижня (day_of_week) та дискретним номером тижня місяця (week_of_month), що допомагає математичній моделі враховувати циклічні коливання попиту протягом місяця.

2.4.4 Математична модель нормалізації та Ridge-регресії. Математичним ядром модуля є алгоритм багатомірної гребеневої регресії (Ridge Regression), розрахунки якого інкапсульовано в методі train класу Forecasting::RegressionTrainer. Оскільки вхідні параметри мають різний масштаб (наприклад, індекс дня тижня варіюється від 0 до 6, а кількість днів з релізу може перевищувати 30), перед виконанням матричних множень система здійснює обов'язковий етап стандартизації. Обчислення виконуються для кожного стовпця матриці дизайну, е початкове значення параметра нормалізується за формулою:

$$z = \frac{x - \mu}{\sigma}, \tag{2.1}$$

де x – початкове числове значення ознаки;

μ – вибіркове середнє арифметичне цього параметра;

σ – стандартне відхилення, розраховане через корінь квадратний з вибіркової дисперсії [8].

Після стандартизації стовпці транспонуються назад у структуру рядків, і до кожного вектора на першу позицію примусово додається константа 1.0, яка виконує роль вільного члена регресії (intercept), що забезпечує зміщення прогнозної гіперплощини відносно початку координат.

Знаходження вектора вагових коефіцієнтів β здійснюється шляхом аналітичного розв'язку системи нормальних рівнянь із застосуванням штрафу L2-регуляризації [7] Програмний код, що безпосередньо реалізує цей математичний апарат у системі, наведено у Додатку А. Обчислення виконуються за формулою:

$$\beta = (X^T X + \lambda I)^{-1} X^T y, \quad (2.2)$$

де β – вектор шуканих вагових коефіцієнтів лінійної моделі;

X – матриця параметрів розрахункової вибірки;

X^T – транспонована матриця параметрів;

λ – коефіцієнт регуляризації (штрафний коефіцієнт), який контролює ступінь згладжування параметрів та запобігає надмірному зростанню значень коефіцієнтів;

I – одинична матриця ідентичності;

y – вектор цільових значень (фактичний показник завантаженості залу в історичних даних);

Для розробленої системи оптимальний коефіцієнт регуляризації підібрано на рівні λ = 0.01. Це значення було визначено шляхом емпіричного тестування алгоритму на ретроспективних (історичних) даних кінотеатру. Більші значення λ призводили до надмірного згладжування ознак та втрати чутливості моделі до днів тижня, тоді як менші значення не забезпечували достатньої математичної стабільності матриці. Обчислювальний алгоритм трансформує інформаційну матрицю $X^T X$ у двомірний масив і додає значення λ виключно до її діагональних елементів, повністю ігноруючи нульову

позицію вільного члена. Це рішення розв'язує проблему лінійної залежності між параметрами (наприклад, вечірнім часом та вихідним днем) і гарантує успішне знаходження оберненої матриці навіть на обмежених вибірках даних.

2.4.5 Архітектурна корекція та згладжування результатів. Після аналітичного знаходження матриці ваг, обчислені коефіцієнти проходять автоматичну процедуру дестандартизації шляхом їх ділення на раніше збережені стандартні відхилення. Це дозволяє виконувати миттєві прогнози для нових сеансів без повторного перерахунку всієї матриці дизайну. Для підвищення стабільності та нівелювання похибок лінійної моделі, на фінальній стадії конвеєра впроваджено механізм згладжування. Отриманий «сирий» результат регресії змішується з базовим рівнем попиту (медіанною популярністю фільму або нижнім порогом часового слоту) у пропорції 55% від моделі на 45% від базового рівня. Ця пропорція була визначена емпіричним шляхом під час тестування: вона надає невелику перевагу лінійній моделі (55%), яка детально враховує контекст конкретного сеансу (час, день тижня), але залишає вагомий вплив базового попиту (45%) як «якоря», що захищає прогноз від аномальних математичних викидів.

На згладжений показник завантаженості додатково накладається динамічний мультиплікатор життєвого циклу релізу. Логіка цього мультиплікатора спадає дискретними кроками: у перші 3 дні прокату діє коефіцієнт 1.30 (прем'єрний ажіотаж), з 4 по 7 день – 1.10, на другий тиждень прокату він падає до 0.80, і поступово знижується до 0.35 для стрічок, що перебувають у розкладі понад 20 днів. Програмна реалізація цієї логіки наведена на рисунку 2.4.

```
def release_multiplier_for(days_since_release)
  case days_since_release
  when 0..3   then 1.30
  when 4..7   then 1.10
  when 8..11  then 0.80
  when 12..13 then 0.65
  when 14..20 then 0.50
  else             0.35
  end
end
```

Рисунок 2.4 – Програмна реалізація мультиплікатора життєвого циклу релізу

Для денного сегмента буднів цей прем'єрний буст додатково послаблюється вдвічі, після чого фінальний результат примусово обмежується верхньою межею 95%, залишаючи обов'язковий п'ятивідсотковий резерв залу для технічних потреб кінотеатру.

Окрім цього, у структурі конвеєра реалізовано консервативний запобіжник: якщо для денного сеансу у будній день математична модель прогнозує низькі значення (менше 5 квитків), система автоматично коригує цей показник до безпечного мінімуму у розмірі 10% від загальної місткості залу. Таке коригування дозволяє уникнути помилок планування і гарантує, що адміністрація заздалегідь забезпечить наявність мінімально необхідного штату касового персоналу на зміні навіть у періоди найбільшого спаду відвідуваності. Сформований числовий відсоток множиться на місткість залу і округлюється, формуючи кінцеву цілочисельну кількість очікуваних квитків для запису в базу даних.

2.4.6 Оцінка точності моделювання та метрики якості. Для визначення точності розробленого математичного підходу та контролю якості прогнозів використовується статистична метрика середньої абсолютної відсоткової помилки (MAPE – Mean Absolute Percentage Error). Процес цієї оцінки автоматизовано та винесено в автономне фонове завдання ForecastAccuracyJob. Після фактичного завершення запланованих сеансів система ініціює перевірку: завантажує з бази даних згенеровані прогнози та порівнює їх із реальними даними про кількість проданих квитків на ці сеанси.

Процес оцінки точності повністю автоматизовано та винесено в окреме фонове завдання ForecastAccuracyJob. Після фактичного завершення запланованих сеансів система ініціює перевірку: вона завантажує з бази даних попередньо згенеровані прогнози та порівнює їх із реальними даними про кількість проданих квитків на ці сеанси. Щоб уникнути перевантаження бази даних під час формування перевірки, запит використовує метод includes для одночасного завантаження пов'язаних даних про кінозали та сеанси, що усуває проблему зайвих звернень до пам'яті сервера.

Вибір саме метрики MAPE, а не абсолютних показників помилки (таких як MAE – середня абсолютна помилка), має чітке бізнес-обґрунтування. Абсолютна похибка у 10 квитків для малого залу на 50 місць є суттєвою (похибка становить 20%), тоді як для просторого залу на 200 місць така ж абсолютна похибка у 10 квитків є практично непомітною (лише 5%). Метрика MAPE розраховує помилку саме у відсотковому відношенні до місткості, що робить її універсальною та незалежною від розміру конкретного кінозалу.

З архітектурної точки зору, результати цієї оцінки є службовою інформацією. Оскільки лінійному персоналу (касирам) ці дані не потрібні для щоденного обслуговування клієнтів, показники MAPE не виводяться в основний графічний інтерфейс застосунку. Натомість система зберігає ці розрахунки на рівні бази даних та системних журналів (логів). Це дозволяє системному адміністратору або розробнику періодично проводити технічний аудит моделі: якщо відсоток похибки починає системно зростати, це слугує індикатором для ручного коригування математичних параметрів (наприклад, зміни пропорцій згладжування або налаштування коефіцієнта регуляризації). Такий підхід залишає операційний інтерфейс чистим і швидким, зберігаючи при цьому повний інженерний контроль над якістю алгоритмів.

2.5 Проєктування інтерфейсу користувача

Інтерфейс розроблюваного програмного комплексу реалізовано як серверно-орієнтований вебклієнт на базі фреймворку Ruby on Rails із шаблонізацією ERB. Технологічний контур фронтенду побудований на сучасній зв’язці Propshaft для управління активами, importmap для модульного JavaScript без окремого бандлера, інструментів Hotwire (Turbo та Stimulus) для асинхронної взаємодії, а також tailwindcss-rails для декларативної стилізації. Адаптивність та кросплатформеність інтерфейсу забезпечується на рівні розмітки за допомогою гнучких контейнерів, CSS Grid, Flexbox та системних медіа-умов, завдяки чому система зберігає функціональну цілісність при переході від десктопних моніторів до екранів мобільних пристроїв [17]. Розмежування доступу до сторінок контролюється екосистемою Devise на основі рольової моделі авторизації (власник, менеджер, касир) [18].

Ключовий операційний контур касира організовано як зв’язаний ланцюг розкладу сеансів і вікна продажу квитків, доступ до якого заблоковано за відсутності активної робочої зміни в базі даних SQLite. Під час ініціалізації форми контролер передає в представлення детермінований набір даних про сеанс, параметри залу та матрицю куплених квитків. На рівні шаблону схема залу будується як параметризована сітка, де кожне місце динамічно рендериться у вигляді інтерактивного чекбокса (для вільних місць) або кнопкового індикатора (для вже проданих крісел). Графічну реалізацію розробленого інтерфейсу робочого місця касира наведено на рисунку 2.5.

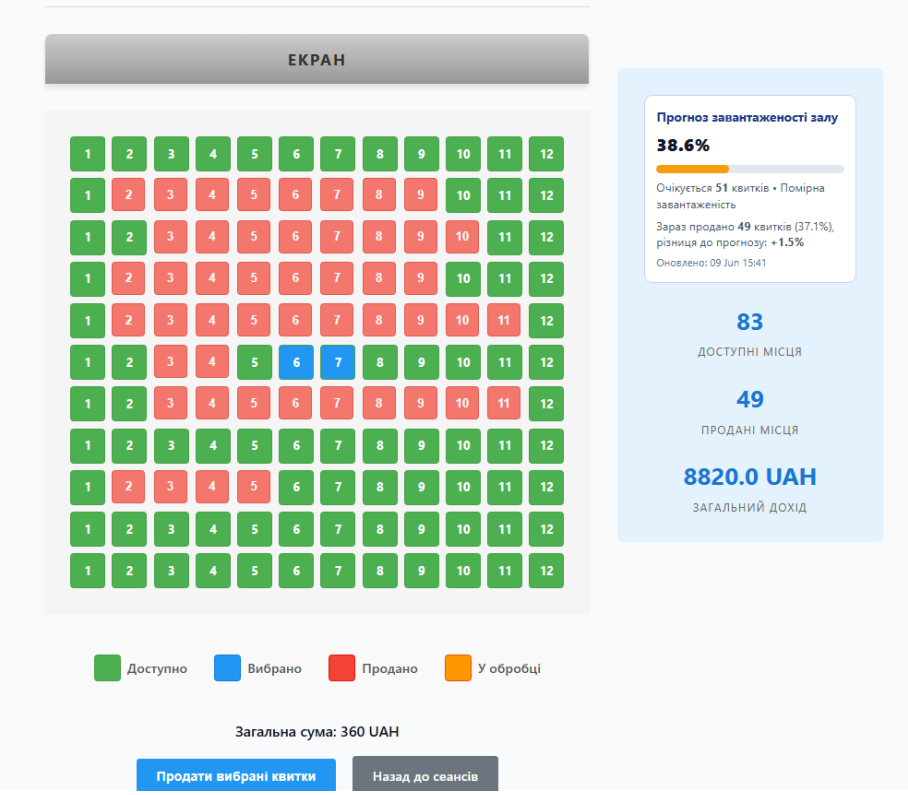

Рисунок 2.5 – Інтерфейс касового модуля системи (вікно продажу квитків)

Клієнтська логіка касового вікна реалізована за допомогою JavaScript-сценаріїв, які синхронізують стан інтерфейсу з формою, виконують інкрементальний перерахунок вартості в кошику, керують класами виділення («Доступно», «Вибрано», «Продано», «У обробці») та викликають модальне вікно для верифікації оплати перед фінальною відправкою транзакції на сервер. Серверна частина валідовує вхідні параметри, блокує повторні продажі та фіксує квитки в базі даних із прив'язкою до поточної зміни працівника.

Аналітичний компонент прогнозування інтегровано безпосередньо у праву інформаційну панель екрана продажу (рис. 2.5). Контролер автоматично підтягує результати останнього запуску моделі гребеневої регресії та виводить для касира числовий індикатор відсотка завантаженості залу, очікувану кількість квитків, різницю відхилення попиту від поточного факту продажів та рівень ризику. За відсутності розрахованих даних система інформує оператора про потребу запуску прогнозування. Адміністративна частина для

менеджерів та власників реалізована у вигляді панелей моніторингу зі структурованими таблицями для управління інфраструктурою мережі (кінотеатри, зали, фільми, розклад) та модулями фінансової звітності.

Захист від помилок введення базується на дворівневій системі ергономічних (обов'язковість полів, обмеження числових меж на клієнті) та серверних (строгі strong parameters, контроль доменів кінотеатрів та перевірка лімітів часу сеансів) запобіжників[19]. У випадку збоїв інтерфейс системно виводить деталізовані блоки помилок або флеш-повідомлення, що забезпечує високу відмовостійкість та передбачувану поведінку вебзастосунку в будь-яких експлуатаційних сценаріях.

Висновки до розділу 2

У другому розділі кваліфікаційної роботи виконано комплексне проєктування інформаційної системи управління мережею кінотеатрів. На основі аналізу бізнес-процесів обґрунтовано використання модульної монолітної архітектури та паттерна MVC у середовищі сучасного вебфреймворку Ruby on Rails, що дозволило чітко розмежувати системну логіку, конфігурації та програмний код.

Побудовано концептуальну та логічну моделі даних, які стали основою для реляційної структури бази даних під управлінням СУБД SQLite. Спроєктована схема охоплює всі ключові сутності предметної області та гарантує цілісність інформації завдяки системі каскадних обмежень і жорстко визначеним зовнішнім ключам.

Важливим інженерним рішенням стала повна структурна ізоляція обчислювальних алгоритмів модуля прогнозування попиту. Розрахунки на базі методу гребеневої регресії винесено у виділений шар сервісних об'єктів та фонових завдань. Такий підхід зберіг чистоту основного MVC-циклу, гарантуючи швидку обробку стандартних клієнтських запитів без ризику блокування вебсервера важкими математичними операціями.

Для забезпечення ефективної роботи персоналу розроблено ергономічні принципи та макети користувацького інтерфейсу з фокусом на робоче місце касира. Інтеграція сучасного фронтенд-стеку (Hotwire, Turbo, Stimulus та Tailwind CSS) забезпечила високу кросплатформеність та інтерактивність системи, а впровадження дворівневої клієнт-серверної валідації сформувало надійний захист від операційних помилок введення даних.

Спроєктована архітектурна та інфраструктурна база повністю задовольняє функціональні вимоги технічного завдання і є міцним фундаментом для безпосередньої програмної реалізації комплексу в наступному розділі роботи.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЛУАТАЦІЯ ПРОГРАМНОГО КОМПЛЕКСУ

3.1 Програмна реалізація та інженерний аналіз методів бізнес-логіки

3.1.1 Програмна реалізація моделей бази даних та забезпечення цілісності інформації. Під час розробки моделі квитка головним завданням є створення зв'язків із сеансами, робочими змінами та конкретними місцями, а також захист системи від помилок під час продажу. Щоб уникнути випадкового продажу одного місця двом покупцям, система проводить перевірку унікальності місця в межах обраного сеансу.

Програмний код, що визначає ці зв’язки та правила перевірки для об’єкта квитка, наведено на рисунку 3.1.

```
class Ticket < ApplicationRecord
  belongs_to :showtime
  belongs_to :seat
  belongs_to :workday, optional: true

  validates :status, presence: true, inclusion: { in: %w[sold pending cancelled] }
  validates :showtime_id, :seat_id, presence: true
  validates :seat_id, uniqueness: { scope: :showtime_id }
end
```

Рисунок 3.1 – Програмна реалізація моделі квитка із правилами перевірки даних

Аналіз наведеного коду показує, що модель квитка обмежує допустимі статуси продажу. Завдяки правилу перевірки унікальності з параметром області видимості, який обмежує пошук дублікатів не всією базою, а виключно контекстом конкретного кіносеансу, база даних автоматично відхиляє спроби забронювати те саме крісло на одну й ту саму подію двічі. Це забезпечує цілісність касових операцій безпосередньо на рівні об’єктно-реляційного відображення, запобігаючи збереженню конфліктних транзакцій [16].

Наступним етапом розробки є створення моделі розкладу сеансів, яка керує розподілом фільмів за часом та залами. Крім базових перевірок наявності фільму або залу, система повинна контролювати, щоб сеанси не накладалися один на одного в межах одного приміщення. Початковий варіант коду моделі сеансу з описом основних атрибутів та правилами перевірки показано на рисунку 3.2.

```ruby
class Showtime < ApplicationRecord
  belongs_to :movie
  belongs_to :hall
  has_many :tickets, dependent: :destroy
  has_many :forecasts, dependent: :destroy

  validates :movie_id, :hall_id, :start_time, presence: true
  validates :price, presence: true, numericality: { greater_than: 0 }
  validates :start_time, uniqueness: { scope: :hall_id }
  validate :does_not_overlap_with_existing_showtime
```

Рисунок 3.2 – Специфікація атрибутів та базових перевірок моделі сеансів

Модель розкладу містить метод перевірки даних, який ініціюється перед кожним збереженням нового запису. Для визначення того, чи не перетинаються сеанси за часом, безпосередньо у класі сеансу розроблено допоміжний метод порівняння overlaps_with?. Його логіка ґрунтується на математичному принципі перетину двох відрізків. Метод спочатку перевіряє наявність обов'язкових даних: якщо один із сеансів не має часу початку або тривалості фільму, він повертає негативний результат для уникнення системної помилки. Далі розраховуються точки початку та завершення для обох подій, де час завершення є сумою початку сеансу та тривалості фільму. Умова перетину виконується лише тоді, коли початок першого сеансу менший за кінець другого, а початок другого менший за кінець першого. Програмний код цього розрахунку наведено на рисунку 3.3.

```ruby
def overlaps_with?(other)
  return false if other.nil? || start_time.blank? || other.start_time.blank?
  return false if movie.blank? || movie.duration.blank? || other.movie.blank? || other.movie.duration.blank?

  start_a = start_time
  end_a = start_a + movie.duration.minutes
  start_b = other.start_time
  end_b = start_b + other.movie.duration.minutes

  start_a < end_b && start_b < end_a
```

Рисунок 3.3 – Програмна реалізація методу порівняння часових інтервалів сеансів

Описаний алгоритм порівняння використовується закритим методом валідації з модифікатором доступу private, функціонування якого обмежено виключно внутрішньою логікою поточного об'єкта. Метод перевірки конфліктів виступає як системний бар'єр бізнес-логіки, а його програмну реалізацію проілюстровано на рисунку 3.4.

```ruby
private

def does_not_overlap_with_existing_showtime
  return if hall_id.blank? || start_time.blank? || movie.blank? || movie.duration.blank?

  conflict = Showtime.includes(:movie)
                     .where(hall_id: hall_id)
                     .where.not(id: id)
                     .detect { |showtime| overlaps_with?(showtime) }

  return if conflict.blank?
end
```

Рисунок 3.4 – Програмна реалізація перевірки конфліктів у розкладі кінозалу

Алгоритм спочатку ізолює дані, вибираючи лише сеанси обраного залу, а потім вилучає запис, який зараз редагується, щоб уникнути порівняння сеансу із самим собою. Далі система послідовно застосовує метод порівняння інтервалів до кожного знайденого сеансу. Використання технології випереджального завантаження пов'язаних даних за допомогою методу includes, що дозволяє заздалегідь витягнути всю необхідну інформацію з бази в оперативну пам'ять одним загальним запитом, оптимізує цей процес та дозволяє уникнути проблеми багаторазових послідовних звернень до сервера. У разі виявлення накладки система відхиляє операцію та формує діагностичне повідомлення для менеджера, вказуючи назву фільму та час конфліктного

показу. Таким чином, запроваджені інструменти забезпечують стабільність розкладу, блокуючи створення сеансів із помилками в часі.

3.1.2 Програмна реалізація обробки транзакцій у контролері касових операцій. Координація взаємодії між клієнтським інтерфейсом та базою даних реалізована у контролері касових операцій (TicketsController). Для забезпечення чистоти коду та дотримання принципу єдиної відповідальності, логіку підготовки масивів даних винесено у спеціалізований сервісний об'єкт TicketShowService. Під час звернення працівника до форми продажу контролер ініціює виклик цього сервісу, який вивантажує структуру обраного кінозалу, агрегує масив зайнятих місць та інтегрує результати модуля предиктивної аналітики. Використання сервісного підходу суттєво зменшує обсяг контролера та ізолює складні базисні запити від логіки відображення інтерфейсу. Програмну реалізацію цього процесу підготовки даних наведено на рисунку 3.5.

```
def new
  service_data = TicketShowService.new(params[:showtime_id]).load_data

  @showtime = service_data[:showtime]
  @hall = service_data[:hall]
  @seats = service_data[:seats]
  @sold_seat_ids = service_data[:sold_seat_ids]
  @movie = service_data[:movie]
  @tickets_by_seat = service_data[:tickets_by_seat]
  @latest_forecast_for_showtime = service_data[:latest_forecast]

  forecast_data = service_data[:forecast_data]
  @predicted_occupancy_pct = forecast_data[:predicted_occupancy_pct]
  @predicted_tickets_count = forecast_data[:predicted_tickets_count]
  @forecast_generated_at = forecast_data[:forecast_generated_at]
end
```

Рисунок 3.5 – Ініціалізація підготовки даних касового екрана за допомогою TicketShowService

Процедура реєстрації факту фінансового продажу ініціюється відправкою асинхронного запиту до методу створення записів (create). На початковому етапі алгоритм аналізує вхідні параметри, перевіряючи наявність обов'язкового ідентифікатора сеансу та списку обраних місць. Після первинної

валідації процес збереження квитків делегується спеціальному інкапсульованому об'єкту TicketCreatorService. Цей підхід гарантує безпечне виконання бізнес-правил: сервіс приймає ідентифікатори обраних місць, перевіряє активність робочої зміни (current_workday) та ініціює ітераційне збереження. У разі порушення прав доступу або логічних помилок, сервіс генерує власне виключення PermissionDenied. Фрагмент виклику сервісу та перехоплення помилок відображено на рисунку 3.6.

```
  showtime = Showtime.find(showtime_id)

  result = TicketCreatorService.new(
    showtime: showtime,
    seat_ids: seat_ids,
    workday: current_workday,
    current_user: current_user
  ).call

  handle_ticket_creation_result(result, showtime, generate_pdf)
rescue TicketCreatorService::PermissionDenied => e
  flash[:alert] = e.message
  redirect_to new_ticket_path(showtime_id: showtime_id)
```

Рисунок 3.6 – Логіка делегування створення квитків у TicketCreatorService з обробкою виключень

Після завершення обробки масиву місць, результати повертаються до контролера у вигляді хеш-структури result, яка містить кількість успішно створених квитків та масив помилок. Контролер передає цей результат у закритий метод handle_ticket_creation_result, який відповідає за зворотний зв'язок із користувачем. Замість аварійної зупинки, система формує комбіноване інформаційне повідомлення, вказуючи як успішно зарезервовані місця, так і перелік виявлених конфліктів. Це убезпечує касовий процес від зупинки у випадку одночасних запитів від різних працівників. Цей етап логіки відображено на рисунку 3.7.

```ruby
def handle_ticket_creation_result(result, showtime, generate_pdf)
  created_count = result[:created_count]
  errors = result[:errors]
  sold_tickets = result[:sold_tickets]

  if created_count > 0
    notice = "#{created_count} ticket(s) sold successfully"
    notice += " (Errors: #{errors.join(', ')})" if errors.any?
    flash[:notice] = notice
  elsif errors.any?
    flash[:alert] = "No tickets sold. #{errors.join(', ')}"
  else
    flash[:alert] = "No seats selected."
  end

  if generate_pdf && sold_tickets.any?
    handle_pdf_generation(showtime, sold_tickets)
  else
    redirect_to new_ticket_path(showtime_id: showtime.id)
  end
end
```

Рисунок 3.7 – Алгоритм формування комбінованих повідомлень про результати продажу

На завершальній стадії обробки система перевіряє статус прапорця generate_pdf. Якщо працівник ініціював друк квитків, контролер викликає допоміжний об'єкт TicketPdfGenerator. Конструкція перехоплення системних помилок rescue навколо логіки створення PDF ізолює транзакцію від можливих збоїв зовнішніх бібліотек генерації документів: якщо на сервері виникне помилка залежностей під час рендерингу файлу (наприклад, MissingDependencyError), дані про продаж квитків гарантовано залишаться збереженими в базі, а система попередить працівника про тимчасову недоступність функції друку. Механізм цієї ізоляції наведено на рисунку 3.8.

```
def handle_pdf_generation(showtime, sold_tickets)
  showtime = Showtime.includes(:movie, :hall).find(showtime.id)
  begin
    pdf_data = TicketPdfGenerator.new(showtime: showtime, tickets: sold_tickets).render
    send_data(
      pdf_data,
      filename: pdf_filename_for(showtime),
      type: "application/pdf",
      disposition: "attachment"
    )
  rescue TicketPdfGenerator::MissingDependencyError => e
    redirect_to new_ticket_path(showtime_id: showtime.id)
  end
end
```

Рисунок 3.8 – Програмний механізм ізольованої генерації PDF-документів

Таким чином, використання сервісних об'єктів замість написання всього програмного коду у файлі контролера забезпечує структурне розділення обов'язків. Контролер відповідає виключно за прийом запитів від користувача та повернення відповідей, тоді як перевірка даних, обробка помилок та збереження квитків (бізнес-логіка) виконуються окремими ізольованими класами. Такий підхід зменшує обсяг основного коду та структурує процес додавання нових функцій.

Водночас робота касового модуля забезпечує системну фіксацію кожного проданого квитка в базі даних. З часом ці транзакції формують впорядкований масив історичних даних. Ця накопичена інформація про минулі продажі є базовим компонентом для роботи алгоритмів математичного моделювання. Процес використання цих даних для предиктивного прогнозування (математичного розрахунку очікуваної завантаженості залів на майбутніх сеансах) розглядається у наступному підрозділі.

3.2 Програмна реалізація методів та алгоритмів математичного прогнозування попиту

Програмна реалізація предиктивного контуру повністю інтегрована в загальну архітектурну модель розробленого вебдодатка. Усі обчислювальні процеси виконуються як асинхронні фонові завдання за допомогою

інфраструктури ActiveJob, що дозволяє ізолювати ресурсомісткі матричні розрахунки від основного потоку обробки HTTP-запитів вебсервера. Повний лістинг вихідного коду аналітичної підсистеми наведено у Додатку А.

3.2.1 Фонова ініціалізація та програмне кешування. Центральним компонентом підсистеми є фоновий клас ForecastRunnerJob. Головним методом ініціалізації виступає perform, який приймає параметр горизонту планування. На початковому етапі алгоритм виконує превентивне каскадне очищення застарілих записів прогнозів для цільового проміжку часу за допомогою методу delete_all, що гарантує ідемпотентність операцій та відсутність дублікатів у базі даних.

Для оптимізації обчислювальних ресурсів під час циклічної обробки розкладу впроваджено механізм локального кешування в оперативній пам'яті. Цей різновид кешування базується на програмному патерні мемоїзації та функціонує виключно протягом життєвого циклу поточного фонового завдання. Оскільки різні сеанси часто транслюють один і той самий фільм, послідовні звернення до бази даних для розрахунку базової популярності створюють надмірне навантаження на систему.

Для вирішення проблеми циклічних запитів ініціалізуються тимчасові структури даних у форматі асоціативних масивів – словник популярності фільмів movie_occupancy_cache та словник дат релізів release_dates_cache. Алгоритм доступу до даних побудований на логіці умовного присвоєння. Під час ітерації система перевіряє наявність ідентифікатора фільму як ключа у словнику. Якщо запис відсутній, програма виконує агрегатний запит до бази даних, розраховує математичний результат і зберігає його в пам'ять. При всіх наступних зверненнях до характеристик цього ж фільму на інших сеансах, система миттєво повертає готове числове значення з кешу, оминаючи звернення до постійного сховища. Такий підхід мінімізує кількість дискових операцій читання та прискорює загальний час обчислень конвеєра. Механізм ініціалізації прогнозного вікна та роботи зі словниками проілюстровано на рисунку 3.9.

```
def perform(horizon_days: 7)
  run = ForecastRun.create!(
      run_at: Time.current,
      horizon_days: horizon_days,
      model: 'segmented_regression'
      )
  horizon_days = horizon_days.to_i
  window_start = Time.current.beginning_of_day
  window_end = (window_start + (horizon_days - 1).days).end_of_day

  Forecast.where('showtime_at >= ? AND showtime_at <= ?', window_start, window_end).delete_all
  movie_occupancy_cache = {}
  release_dates_cache = {}
```

Рисунок 3.9 – Алгоритм ініціалізації прогнозного вікна та очищення дублікатів

3.2.2 Алгоритм пакетної агрегації навчальної вибірки. Збір історичних даних делеговано приватному методу gather_training_rows_for. Для унеможливлення витоку даних алгоритм вилучає з бази лише ті сеанси, час початку яких менший за поточний момент щонайменше на 24 години.

Замість ітеративного завантаження об'єктів квитків у пам'ять сервера, підрахунок проданих місць виконується єдиним пакетним SQL-запитом із використанням методу групування group(:showtime_id).count. Отриманий агрегований масив матеріалізується в оперативній пам'яті, після чого програма здійснює обчислення цільової змінної occupancy_pct для кожного історичного прецеденту. Оптимізовану процедуру пакетного завантаження статистики відображено на рисунку 3.10.

```
def gather_training_rows_for(hall, capacity, movie_cache = {}, release_dates_cache = {})
  past_showtimes = hall.showtimes.where('start_time < ?', Time.current - 24.hours).includes(:movie).to_a

  showtime_ids = past_showtimes.map(&:id)
  return [] if showtime_ids.empty?

  ticket_counts = Ticket.where(showtime_id: showtime_ids, status: %w[sold pending]).group(:showtime_id).count
```

Рисунок 3.10 – Оптимізований метод пакетної агрегації історичних продажів

3.2.3 Програмна сегментація та логіка стійкого деградування. Після формування загального масиву параметрів метод split_into_segments розділяє його на три незалежні часові простори: денні будні, вечірні будні та вихідні сеанси. Процес класифікації реалізовано за допомогою методу select, який

фільтрує об'єкти залежно від булевого маркера is_weekend та години початку сеансу.

Для захисту математичного ядра від помилок обчислення оберненої матриці на малих вибірках реалізовано захисну логіку (fallback). У методі train_segmented_models система перевіряє розмір відфільтрованого масиву: якщо кількість спостережень становить менше п'яти, алгоритм присвоює покажчикам значення nil. У такому випадку під час прогнозування конвеєр автоматично делегує розрахунки резервній глобальній моделі, яка навчається на всьому масиві даних кінозалу. Процес сегментації та ініціалізації моделей наведено на рисунку 3.11.

```ruby
def train_segmented_models(hall_id, segmented_rows)
  SEGMENTS.each_with_object({}) do |segment, result|
    seg_rows = segmented_rows[segment]

    if seg_rows.size < 5
      result[segment] = { trainer: nil, model: nil }
      next
    end

    trainer = Forecasting::RegressionTrainer.new(seg_rows)
    model   = trainer.train(ridge: 0.01)
    result[segment] = { trainer: trainer, model: model }
  end
```

Рисунок 3.11 – Алгоритм сегментації сеансів та захисної ініціалізації моделей

3.2.4 Реалізація математичного тренера та матричних обчислень. Обробка даних та побудова моделі лінійної регресії виконується у класі Forecasting::RegressionTrainer. Метод train відповідає за нормалізацію набору параметрів за допомогою математичних функцій модуля Math для розрахунку дисперсії та стандартного відхилення. Після стандартизації нормалізовані масиви перетворюються на об'єкти вбудованої бібліотеки Matrix мови Ruby.

Аналітичне розв'язання системи нормальних рівнянь відбувається шляхом безпосередньої модифікації матриці коваріацій. Програмний алгоритм трансформує матрицю у двомірний масив за допомогою методу to_a та ітеративно додає штрафний коефіцієнт ridge_val до її діагональних

елементів, ігноруючи позицію нульового індексу. Після зворотної конвертації система обчислює обернену матрицю методом inverse. Для запобігання аварійної зупинці фонового завдання у випадку сингулярності матриці застосовано блок перехоплення помилок rescue StandardError, який безпечно зупиняє розрахунок. Цей програмний механізм наведено на рисунку 3.12.

```
if ridge && ridge.to_f != 0.0
  ridge_val = ridge.to_f
  mat = xtx.to_a
  (0...mat.size).each do |i|
    mat[i][i] += (i == 0 ? 0.0 : ridge_val)
  end
  xtx = Matrix.rows(mat)
end

begin
  beta_std = xtx.inverse * xt * y
rescue StandardError
  return nil
end
```

Рисунок 3.12 – Програмна модифікація матриці коваріацій та обчислення ваг

Після отримання стандартизованого вектора коефіцієнтів програма виконує їхню дестандартизацію, коригуючи значення вільного члена регресії. Це дозволяє методу predict миттєво обчислювати прогнози через скалярний добуток масивів без повторної нормалізації вхідних параметрів.

3.2.5 Програмне ансамблювання та згладжування результатів. На фінальному етапі методу perform отриманий результат матричного множення піддається обов'язковому блендингу (ансамблюванню). Сирий прогноз змішується з базовим рівнем попиту у фіксованій пропорції (55% моделі та 45% базової медіани). На згладжений показник накладається мультиплікатор життєвого циклу релізу, який розраховується методом release_multiplier_for.

Отримане значення примусово обмежується верхньою та нижньою межами за допомогою методу clamp, враховуючи технічний ліміт заповнюваності у 95% та захисний мінімум для відповідного часового слоту. Фінальне відсоткове значення множиться на місткість кінозалу, округлюється

та зберігається в базу даних як очікувана кількість квитків. Механізм згладжування та збереження результатів представлено на рисунку 3.13.

```
baseline_occupancy = [movie_pop, slot_floor].max
raw_occupancy      = active_trainer.predict(active_model, features)

occupancy = raw_occupancy.nil? ? baseline_occupancy : ((raw_occupancy * 0.55) + (baseline_occupancy * 0.45))
occupancy = slot_floor if occupancy < slot_floor
occupancy = 0.95 if occupancy > 0.95

days_since_release    = calculate_days_since_release(showtime.movie, date, release_dates_cache)
new_release_multiplier = release_multiplier_for(days_since_release)

occupancy = (occupancy * new_release_multiplier).clamp(slot_floor, 0.95)
predicted tickets = (occupancy * capacity).round
```

Рисунок 3.13 – Програмний механізм ансамбльованого згладжування прогнозів

Таким чином, розроблена програмна підсистема предиктивного аналізу попиту забезпечує автономний розрахунок очікуваної завантаженості кінозалів. Використання інфраструктури фонових завдань, механізмів локального кешування та методів пакетної агрегації історичних даних дозволяє системі обробляти великі масиви інформації без створення критичного навантаження на основний потік вебсервера. Програмне розділення відповідальності між збором даних та безпосереднім навчанням моделей спрощує подальшу підтримку коду.

Інтеграція математичного апарату матричних обчислень безпосередньо у програмний код, доповнена логікою ансамблювання та згладжування результатів, формує кінцевий прогноз у вигляді розрахункової кількості проданих квитків. Отримані аналітичні дані записуються в реляційну базу даних для їх подальшого використання у плануванні розкладу. Візуалізація цих розрахунків, а також загальна взаємодія працівників кінотеатру з розробленими функціональними модулями здійснюється через графічний інтерфейс користувача, принципи програмної реалізації якого розглядаються у наступному підрозділі.

3.3 Інтерфейс користувача, структура проєкту та клієнт-серверна взаємодія

3.3.1 Елементи інтерфейсу та сценарії взаємодії (UX/UI). Користувацький інтерфейс системи побудований на базі компонентного підходу із застосуванням фреймворку Tailwind CSS. Головним фокусом під час проєктування візуальної частини було створення ергономічного робочого середовища для лінійного персоналу (касирів та менеджерів), яке дозволяє швидко обслуговувати клієнтів та контролювати операційні процеси.

Будь-яка взаємодія працівника з внутрішніми ресурсами кінотеатру починається із захищеного екрана входу, реалізованого на базі індустріального фреймворку Devise. Він забезпечує криптографічне шифрування паролів алгоритмом bcrypt та захист від перехоплення сесій. На практиці співробітник вводить свою корпоративну адресу електронної пошти та пароль у стандартну безпечну форму. Загальний вигляд сторінки авторизації персоналу представлено на рисунку 3.14.

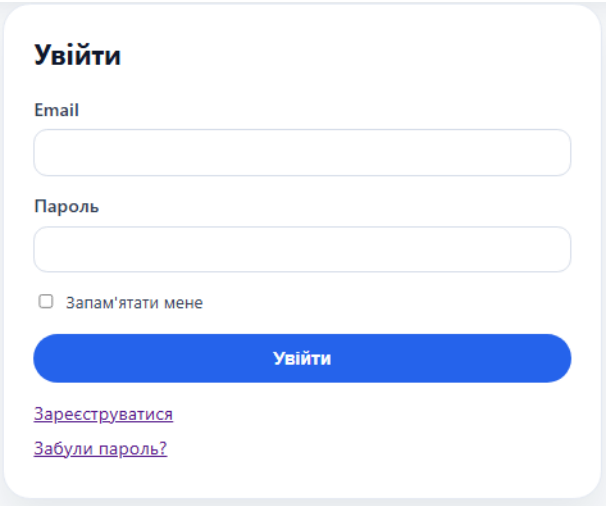

Рисунок 3.14 – Екран автентифікації користувачів (Сторінка входу)

Характерною особливістю архітектури є наявність обов'язкового етапу ініціалізації робочого дня. Одразу після успішного входу в систему, для працівника автоматично відкривається модальне вікно вибору активної зміни та кінотеатру. Цей крок є критично важливим для коректного функціонування

касового модуля: він динамічно прив'язує поточну сесію касира до конкретної фізичної філії. Усі подальші продажі квитків та товарів автоматично записуватимуться в базу даних саме на цю зміну для точного підрахунку виручки в кінці дня. Інтерфейс вибору робочої зміни ілюструє рисунок 3.15.

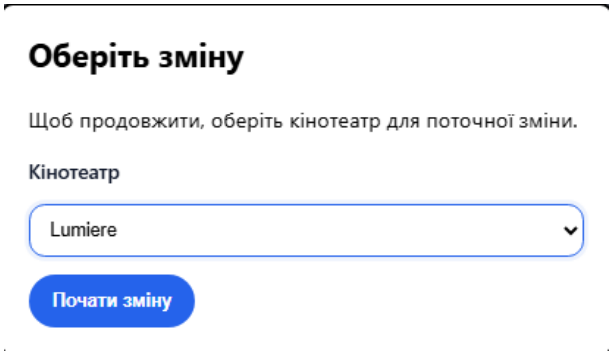

Рисунок 3.15 – Модальне вікно вибору кінотеатру для поточної зміни

Для нових власників кіномережі передбачено окрему екранну форму створення облікового запису та первинної конфігурації параметрів організації. Система підтримує реєстрацію бізнес-профілю з обов'язковою перевіркою та валідацією корпоративних доменних імен для автоматичного формування логінів майбутніх співробітників за шаблоном електронної пошти. Роботу цієї форми відображено на рисунку 3.16.

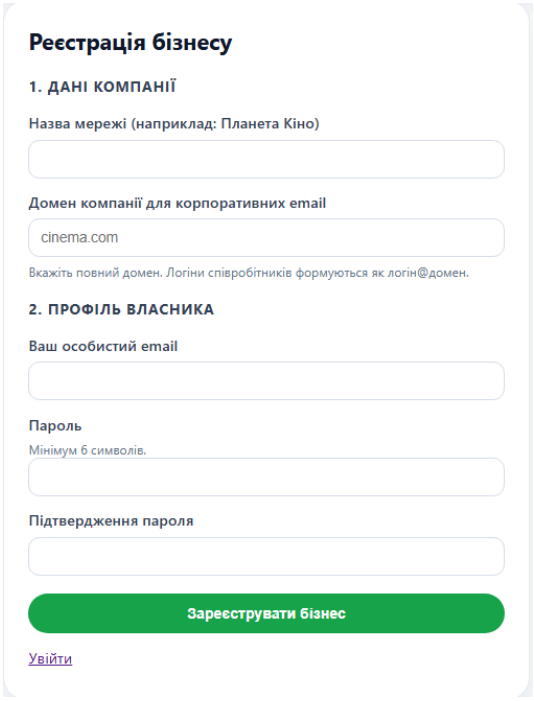

Рисунок 3.16 – Екран реєстрації корпоративного профілю кіномережі

Після підтвердження початку зміни працівник автоматично перенаправляється у свій головний робочий простір – операційну панель співробітника. Цей екран агрегує ключові показники поточної зміни: ім'я авторизованого користувача, статус активного кінотеатру, час останнього запуску предиктивного контуру та середній рівень завантаженості залів. Це дозволяє менеджеру або касиру миттєво оцінити поточну комерційну ситуацію в закладі. Головну робочу панель співробітника наведено на рисунку 3.17.

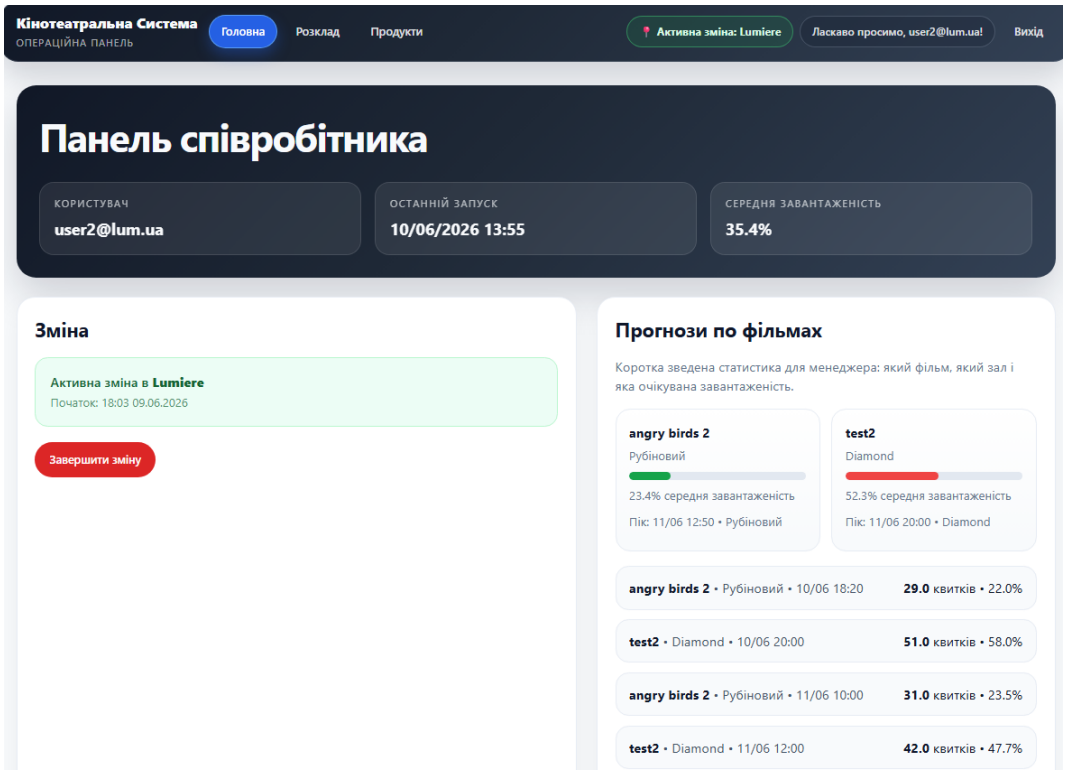

Рисунок 3.17 – Панель співробітника з аналітичними показниками зміни

Для планування подальшої роботи персоналу передбачено спеціалізований екран моніторингу прогнозів. У цій таблиці працівник бачить розклад на 7 днів наперед. Кожен сеанс містить точну розраховану кількість очікуваних глядачів та планову виручку. Для швидкого візуального сприйняття розрахунки дублюються кольоровим прогрес-баром: зелений колір маркує низький попит, помаранчевий – помірний, а червоний попереджає персонал про критичний аншлаг, вимагаючи посиленої уваги касирів. Описану панель моніторингу представлено на рисунку 3.18.

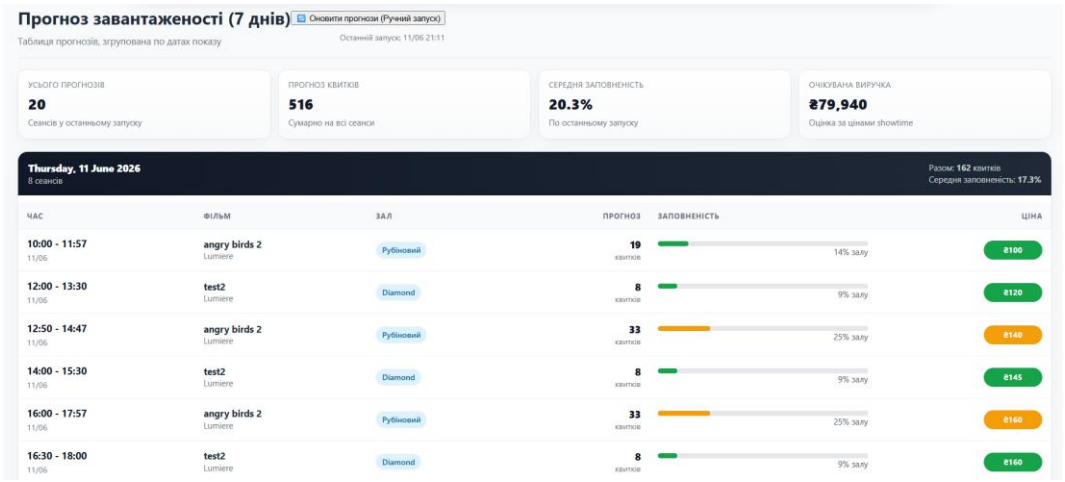

Рисунок 3.18 – Панель предиктивного моніторингу та прогнозу завантаженості

Безпосереднє обслуговування клієнтів здійснюється через інтерфейс календаря сеансів. Він представляє розклад у вигляді ергономічних карток. З цього екрана касир може візуально відстежувати розклад, бачити поточну кількість проданих квитків та одним натисканням кнопки переходити безпосередньо до модуля продажу місць на обраний фільм (рис. 3.19).

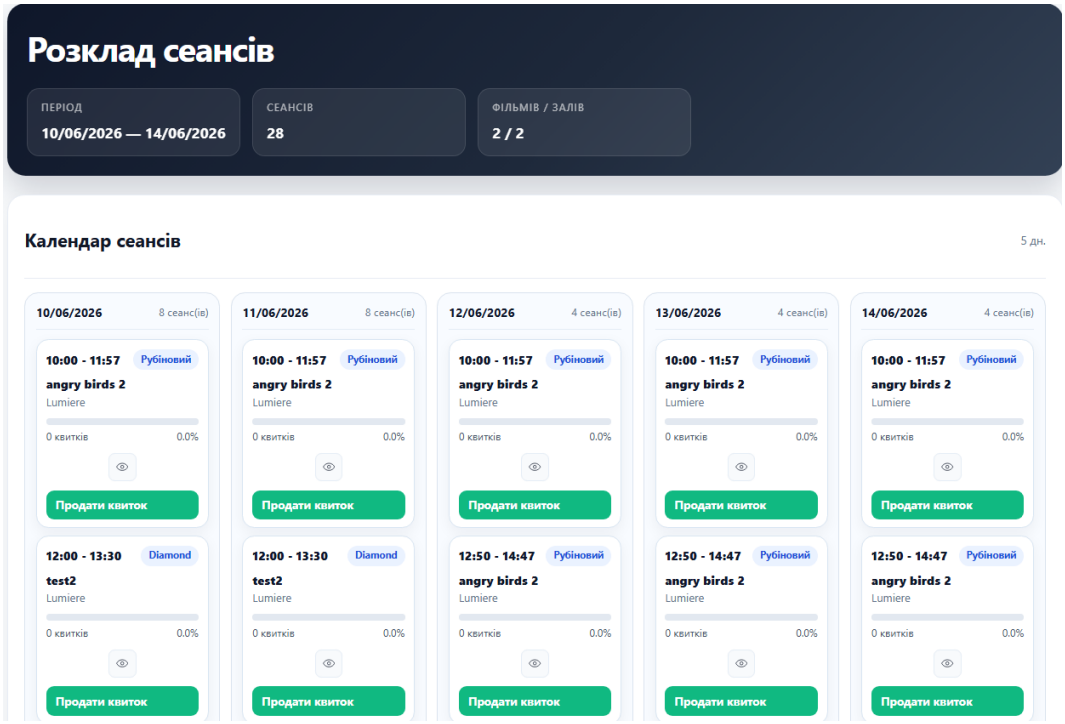

Рисунок 3.19 – Інтерфейс календаря сеансів та управління розкладом

Окрім продажу квитків, до щоденних обов'язків працівника входить управління асортиментом кінобару. Для цього розроблено окремий модуль обліку товарів. В рамках відкритої зміни касир може фіксувати поточні продажі попкорну чи напоїв, реєструвати нові надходження продукції на склад та контролювати актуальні ціни. Система автоматично перераховує загальну барну виручку працівника в режимі реального часу. Інтерфейс робочого місця працівника кінобару наведено на рисунку 3.20.

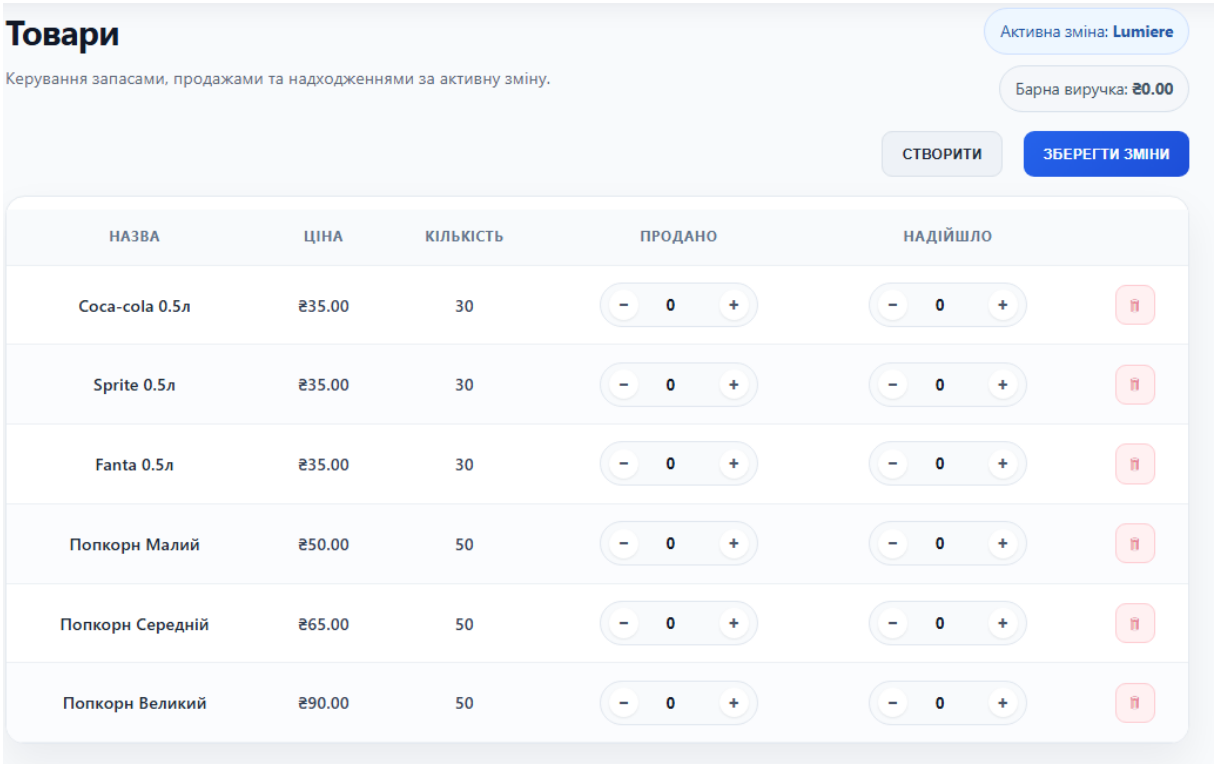

Рисунок 3.20 – Модуль операційного управління запасами та виручкою кінобару

3.3.2 Модульна організація та файлова структура програмного комплексу. Архітектурна організація файлової системи розробленого вебзастосунку базується на стандартизованих конвенціях фреймворку Ruby on Rails, які суворо регламентують розташування кожного типу компонентів у відповідних директоріях. Такий детермінований підхід до структурування кодової бази відповідає принципу «угода важливіша за конфігурацію» і гарантує передбачувану навігацію по проєкту незалежно від масштабу системи. Центральним вузлом кодової бази є директорія app/, яка містить усі

функціональні компоненти програмного комплексу, розподілені між піддиректоріями відповідно до їхньої архітектурної ролі в патерні MVC.

Шар моделей, зосереджений у app/models/, представлений 14 окремими файлами для кожної сутності предметної області, включаючи cinema.rb, hall.rb, showtime.rb, ticket.rb, workday.rb, report.rb, forecast.rb, movie.rb, product.rb, user.rb, seat.rb, company.rb, forecast_run.rb та базовий клас application_record.rb. Кожна модель інкапсулює визначення асоціацій (has_many, belongs_to), правила внутрішньої валідації чинників та методи бізнес-логіки, безпосередньо пов'язані з відповідною таблицею бази даних SQLite через механізм Active Record, забезпечуючи дотримання принципу єдиної відповідальності (Single Responsibility Principle).

Шар контролерів, розміщений у app/controllers/, реалізує обробку вхідних HTTP-запитів відповідно до логіки предметної області. Базовий application_controller.rb містить загальні інструменти автентифікації, фільтрації та спільні методи захисту, тоді як основні контролери (cinemas_controller.rb, halls_controller.rb, showtimes_controller.rb, tickets_controller.rb, workdays_controller.rb, reports_controller.rb) обробляють стандартні RESTful-операції для кожної сутності предметної області. Вкладений простір імен app/controllers/users/ містить registrations_controller.rb для розширення функціональності екосистеми Devise, що дозволяє користувачам реєструвати нові бізнес-профілі (компанії) з автоматичним транзакційним зв'язуванням з обліковим записом власника. Спільна логіка винесена у модулі піддиректорії app/controllers/concerns/, забезпечуючи дотримання принципу DRY (Don't Repeat Yourself).

Паралельно з цим шар представлень у директорії app/views/ структурований відповідно до логіки 11 контролерів за допомогою 53 ERB-шаблонів, де базовий каркас сторінок винесено у layouts/application.html.erb, глобальна навігаційна панель централізована у layouts/_navbar.html.erb, а модальні вікна та повторювані компоненти розташовані в shared/ для максимального перевикористання коду (reusability).

Важливим інженерним рішенням стало виділення модуля предиктивної аналітики та допоміжної складної бізнес-логіки в окремий сервісний шар додатка. Фонові завдання зосереджені в директорії app/jobs/, де класи ForecastRunnerJob та ForecastAccuracyJob обробляють ресурсомісткі довготривалі операції без блокування основного HTTP-циклу вебсервера [20]. Математичне ядро Forecasting::RegressionTrainer розташоване у папці app/services/forecasting/, утворюючи ізольований простір імен для побудови сегментованих моделей регресії. Додаткові сервісні об'єкти, такі як BatchShowtimeCreator для пакетної генерації сеансів, а також генератори звітів та квитків ReportPdfGenerator і TicketPdfGenerator, винесені в app/services/ для інкапсуляції бізнес-правил. Такий розподіл відокремлює матричні обчислення від основного циклу обробки запитів, завдяки чому контролери не містять аналітичної логіки, а сервіси не мають прямого доступу до HTTP-контексту.

Конфігурація маршрутизації у файлі config/routes.rb декларативно відображає URL-простір системи, де RESTful-маршрути організовані безпосередньо на кореневому рівні для забезпечення прямого доступу до ендпоінтів управління квитками, фільмами та залами [21]. Безпека сесій та авторизація діють через глобальні фільтри автентифікації на базі Devise та специфічні обмеження доступу безпосередньо всередині контролерів (наприклад, метод authorize_manager_or_admin! для операцій з продуктами кінобару).

Керування зовнішніми залежностями здійснюється через маніфест Gemfile та фіксуючий файл Gemfile.lock, де бібліотеки розподілені на групи безпеки (devise), фронтенду та асинхронної комунікації (turbo-rails, stimulus-rails, tailwindcss-rails), статичних ресурсів (propshaft), генерації PDF (prawn) та сучасного асинхронного оброблення задач і кешування (solid_queue, solid_cache, solid_cable). Математичний контур регресії реалізований виключно на базі стандартної бібліотеки matrix та базових операцій мови Ruby, без сторонніх науково-обчислювальних пакетів.

Інфраструктура бази даних підтримується каталогом db/, який містить 17 файлів міграцій (із поточною версією схеми 20260606120000), актуальну структуру schema.rb та початкові дані seeds.rb. Автоматизовані тести зосереджені в директорії test/ відповідно до конвенцій фреймворку Minitest і охоплюють тести бізнес-логіки моделей, контролерів та інтеграційні специфікації, а компоненти асинхронної обробки подій конфігуруються в каталозі config/initializers/ для управління background job-ами, що разом забезпечує високу надійність та детермінованість програмного комплексу перед його промисловим розгортанням.

3.3.3 Типізація даних та механізми клієнт-серверної взаємодії. Для забезпечення високої швидкодії, мінімізації мережевого трафіку та слабкого зв'язку (Loose Coupling) між бекенд-платформою та клієнтським інтерфейсом, обмін інформацією в системі суворо типізований та стандартизований у форматі JSON. Серверний шар за допомогою спеціалізованих серіалізаторів трансформує вихідні об'єкти бази даних у полегшений текстовий пакет Data Transfer Object (DTO). Структура цього пакета передбачає жорстку типізацію: часові мітки start_time уніфіковані за стандартом ISO 8601 із збереженням часової зони для коректної роботи аналітичних моделей, числові параметри відсотків прогнозу приводяться до типу Float, а показники місткості залів та кількості квитків – до цілочисельного типу Integer, оскільки продаж дробової кількості місць фізично неможливий.

Завдяки такій типізації клієнтська частина додатка завжди отримує передбачувані структури даних, що унеможливлює помилки динамічного рендерингу на фронтенді. Асинхронна комунікація та динамічне оновлення інтерфейсу без повного перезавантаження сторінок реалізовані за допомогою технологій Turbo та Turbo Streams, що дозволяє вебсерверу трансляційно впорскувати оновлені типи даних через вебсокети безпосередньо в цільові контейнери браузера при зміні розкладу. Складна клієнтська логіка та керування станами інтерактивної схеми залів покладені на JavaScript-компоненти фреймворку Stimulus, які розбирають типізований JSON,

переключають класи Tailwind CSS для крісел та здійснюють локальний перерахунок вартості замовлення. Повний лістинг клієнтського програмного коду (Stimulus-контролерів), який забезпечує асинхронну реактивність касового екрана та управління станами посадкових місць, наведено у Додатку Б. Наскрізний життєвий цикл клієнт-серверного обміну типізованими даними, процеси автентифікації користувача, моніторинг попиту та фіксація квитків у межах транзакцій ActiveRecord наочно проілюстровані на діаграмі послідовності на рисунку 3.21.

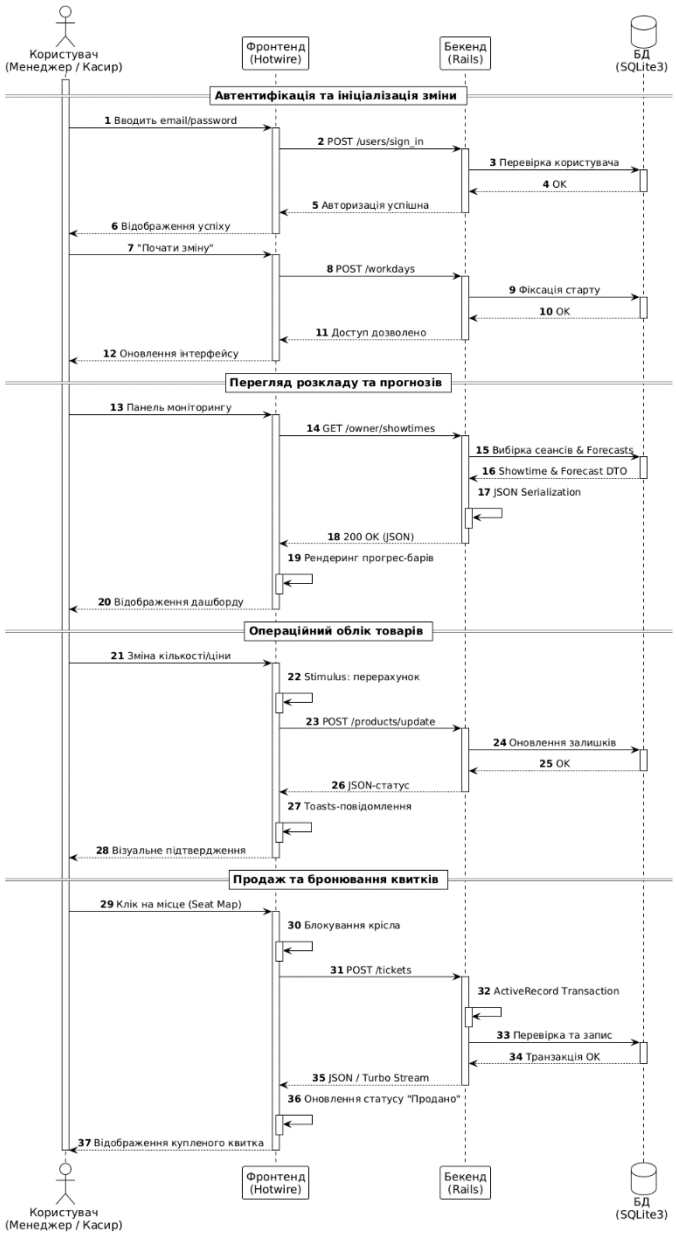

Рисунок 3.21 – Діаграма послідовності клієнт-серверної взаємодії під час обробки касових транзакцій

Під час обробки критичних транзакцій, таких як викуп квитка, згенерований текстовий JSON-пакет миттєво парситься клієнтським скриптом для динамічного оновлення статусу місць. У разі виникнення помилок валідації або колізій типів під час SQL-транзакції, Rails-сервер відхиляє операцію та повертає JSON-відповідь із кодом помилки. Вона автоматично перехоплюється Stimulus-контролером на клієнтській стороні, що ініціює виведення динамічного toasts-повідомлення у відповідному сигнальному класі Tailwind CSS без перезавантаження активної сторінки користувача.

Висновки до розділу 3

У третьому розділі кваліфікаційної роботи було здійснено комплексну програмну реалізацію спроєктованої інформаційної системи керування кінотеатром із вбудованим модулем предиктивного аналізу попиту. Практична розробка базувалася на вебфреймворку Ruby on Rails, що дозволило забезпечити надійність кінцевого продукту завдяки суворому дотриманню архітектурного патерну MVC.

На початковому етапі розробки було імплементовано реляційну структуру бази даних, яка охоплює ключові моделі предметної області: кінозали, розклад, квитки та фінансові звіти. Завдяки механізмам міграцій та транзакцій Active Record створено стійку схему збереження даних, що унеможливлює виникнення колізій (наприклад, подвійного продажу одного місця) під час паралельної роботи кількох касирів. Для захисту операційного середовища розгорнуто індустріальний стек автентифікації Devise із криптографічним хешуванням паролів, що надійно ізолює адміністративні контури системи.

Центральним інженерним етапом розділу стала розробка та програмна оптимізація підсистеми предиктивного аналізу попиту. Увесь обчислювальний конвеєр успішно винесено у фоновий асинхронний шар за допомогою інфраструктури ActiveJob. Для уникнення надмірного

навантаження на базу даних під час підготовки навчальних вибірок впроваджено механізми пакетної агрегації SQL-запитів та алгоритми тимчасового локального кешування в оперативній пам'яті (мемоїзації).

Математичне ядро лінійної Ridge-регресії реалізовано засобами мови Ruby з використанням стандартної бібліотеки матричних обчислень. Для забезпечення безперебійної роботи конвеєра розроблено багаторівневу систему захисту: впроваджено перехоплення системних помилок під час інверсії матриць, а також логіку стійкого деградування (fallback), яка автоматично перемикає розрахунки на резервну глобальну модель у випадку нестачі історичних даних для конкретного часового сегмента. Точність результатів додатково підвищено шляхом програмного ансамблювання сирого прогнозу з медіанними показниками та застосуванням мультиплікатора життєвого циклу прокату.

Для зручної взаємодії персоналу з розробленими аналітичними інструментами спроєктовано ергономічний адаптивний веб-інтерфейс із використанням CSS-фреймворку Tailwind. Проблему постійного перезавантаження сторінок під час зміни розкладу чи продажу квитків вирішено інтеграцією реактивного стека Hotwire (Turbo та Stimulus). Асинхронна клієнт-серверна комунікація була суворо стандартизована через типізовані об'єкти передачі даних (DTO) у форматі JSON, що гарантує цілісність відображення касового екрана в режимі реального часу.

Таким чином, інтеграція наведених інженерних, алгоритмічних та архітектурних рішень забезпечила створення повноцінного та відмовостійкого програмного комплексу, що повністю задовольняє функціональні вимоги. Розроблена інформаційна система не лише успішно автоматизує операційні процеси продажу квитків та касового обліку, але й формує для керівництва незалежний аналітичний контур. Використання вбудованої математичної моделі прогнозування попиту дозволяє завчасно оптимізувати розподіл персоналу та ресурсів кінотеатру, що підтверджує успішне виконання поставлених завдань та досягнення головної мети кваліфікаційної роботи.

ВИСНОВКИ

У дипломній роботі розроблено інформаційну вебсистему кінотеатру з інтегрованим модулем прогнозування попиту, яка охоплює клієнтську частину, серверну бізнес-логіку на базі вебфреймворку та окремий сервісний шар для обробки алгоритмів математичного моделювання.

У першому розділі проаналізовано предметну область та розглянуто існуючі програмні рішення. Аналіз показав, що більшість аналогів або не мають вбудованих аналітичних функцій для внутрішнього управлінського обліку, або вимагають сплати фінансових комісій та не забезпечують повного контролю над базою даних. На основі цього обґрунтовано необхідність розробки власної системи, яка б поєднувала мобільність сучасних вебрішень, управлінський функціонал та вбудований математичний модуль для автоматизації процесів прогнозування посадки залів без залежності від сторонніх платформ.

У другому розділі виконано проєктування архітектури системи. Розроблено варіанти використання, що описують ключові сценарії роботи персоналу. Спроєктовано реляційну структуру бази даних під управлінням СУБД SQLite. Обґрунтовано вибір монолітного архітектурного підходу та зафіксовано стек технологій (Ruby on Rails, Hotwire, Tailwind CSS). Визначено компонентну файлову структуру додатка з окремим сервісним шаром, що дозволило ізолювати обчислювальні алгоритми прогнозування від основного потоку обробки запитів та усунуло необхідність використання важких клієнтських фреймворків чи сторонніх серверів пам'яті.

У третьому розділі описано практичну програмну реалізацію всіх підсистем. Автентифікацію персоналу побудовано на основі фреймворку Devise із застосуванням криптографічного алгоритму bcrypt. Управління касовими операціями реалізовано всередині ізольованих транзакцій Active Record, що гарантує захист від колізій подвійного продажу місць. Головним інженерним досягненням стала розробка аналітичного контуру на базі

алгоритму Ridge-регресії. Для оптимізації продуктивності системи впроваджено пакетну агрегацію SQL-запитів та механізми локального кешування в оперативній пам'яті. Весь обчислювальний конвеєр винесено у фоновий асинхронний шар за допомогою інфраструктури ActiveJob та пакету Solid Queue. Клієнтський інтерфейс розроблено на базі реактивного стека Turbo та Stimulus, що забезпечило динамічне оновлення екранів у реальному часі без перезавантаження сторінок завдяки обміну строго типізованими даними у форматі JSON.

Практичне значення розробленої системи полягає в тому, що вона автоматизує рутинні процеси (продаж квитків, облік товарів кінобару, ведення змін) та одночасно генерує для бізнесу математично обґрунтовані прогнози відвідуваності. Система аналізує розклад у фоновому режимі та виводить показники завантаженості безпосередньо на робочих екранах касирів та менеджерів. Такий підхід замінює інтуїтивне управління точним розрахунком, дозволяючи заздалегідь формувати оптимальний графік працівників, керувати запасами та мінімізувати операційні втрати.

Система є архітектурно цілісною і придатною до подальшого розширення – зокрема, додавання клієнтського онлайн-порталу для відвідувачів, інтеграції із зовнішніми платіжними шлюзами або застосування більш складних багаторівневих статистичних моделей для довгострокового фінансового планування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. kinotehnik.net – CinemaTech Community UA. kinotehnik.net – CinemaTech Community UA. URL: https://kinotehnik.net/ (дата звернення: 10.05.2026).

2. mticket | Автоматизація квиткових процесів. mticket. URL: https://mticket.com.ua/#products (дата звернення: 10.05.2026).

3. SERVIO Tickets для автоматизації кінотеатрів – Програмне забезпечення для управління продажем квитків | SERVIO. Expert Solution. URL: https://expertsolution.com.ua/produkty/pz/servio-tickets-dlya-avtomatyzatsiyi-zahodiv/ (дата звернення: 10.05.2026).

4. WebCase – Розробка веб-додатків. WebCase – Розробка веб-додатків. URL: https://webcase.com.ua/uk/razrabotka-web-prilozhenij/ (дата звернення: 13.05.2026).

5. Newman S. Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith. O'Reilly Media, 2019. 272 p.

6. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media, 2020. 416 p.

7. Géron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. 3rd ed. O'Reilly Media, 2022. 856 p.

8. Джеймс Г., Віттон Д., Гасті Т., Тібширані Р. Вступ до статистичного навчання з прикладами на R та Python. Київ: К.І.С., 2021. 464 с.

9. Silberschatz A., Korth H. F., Sudarshan S. Database System Concepts. 7th ed. McGraw-Hill Education, 2019. 1376 p.

10. Coronel C., Morris S. Database Systems: Design, Implementation, & Management. 13th ed. Cengage Learning, 2018. 816 p.

11. HTML Over The Wire | Hotwire. HTML Over The Wire | Hotwire. URL: https://hotwired.dev/ (дата звернення: 01.06.2026).

12. Installing with Vite - Installation. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. URL: https://tailwindcss.com/docs/installation/using-vite (дата звернення: 01.06.2026).

13. Software Architecture Guide. martinfowler.com. URL: https://martinfowler.com/architecture/ (дата звернення: 13.06.2026).

14. The Rails Initialization Process â€ ” Ruby on Rails Guides. Ruby on Rails Guides. URL: https://guides.rubyonrails.org/initialization.html (date of access: 13.06.2026).

15. Medium – MVC Architecture in Rails. Medium – MVC Architecture in Rails. URL: https://medium.com/@er.sumitsah/mvc-architecture-in-rails-94881c860ccd (дата звернення: 13.06.2026).

16. Active Record Basics a Ruby on Rails Guides. Ruby on Rails Guides. URL: https://guides.rubyonrails.org/active_record_basics.html (дата звернення: 13.06.2026).

17. Responsive design - Core concepts. Tailwind CSS - Rapidly build modern websites without ever leaving your HTML. URL: https://tailwindcss.com/docs/responsive-design (дата звернення: 13.06.2026).

18. GitHub - heartcombo/devise: Flexible authentication solution for Rails with Warden. GitHub. URL: https://github.com/heartcombo/devise (дата звернення: 13.06.2026).

19. Securing Rails Applications a Ruby on Rails Guides. Ruby on Rails Guides. URL: https://guides.rubyonrails.org/security.html (дата звернення: 13.06.2026).

20. GitHub - rails/solid_queue: Database-backed Active Job backend. GitHub. URL: https://github.com/rails/solid_queue (дата звернення: 13.06.2026).

21. Rails Routing from the Outside In a Ruby on Rails Guides. Ruby on Rails Guides. URL: https://guides.rubyonrails.org/routing.html (дата звернення: 13.06.2026).

ДОДАТОК А

ЛІСТИНГ КОДУ СЕРВЕРНОЇ ЧАСТИНИ

СХЕМА БАЗИ ДАНИХ

```
ActiveRecord::Schema[8.1].define(version:
2026_06_11_180340) do
  create_table "cinemas", force: :cascade do
|t|
    t.string "address"
    t.integer "company_id", null: false
    t.datetime "created_at", null: false
    t.string "name"
    t.datetime "updated_at", null: false
    t.index ["company_id"], name:
"index_cinemas_on_company_id"
  end

  create_table "companies", force: :cascade
do |t|
    t.datetime "created_at", null: false
    t.string "domain_prefix"
    t.string "name"
    t.datetime "updated_at", null: false
  end

  create_table "forecast_runs", force:
:cascade do |t|
    t.datetime "created_at", null: false
    t.decimal "global_mape", precision: 5,
scale: 2
    t.integer "horizon_days", default: 1,
null: false
    t.string "model"
    t.text "model_weights"
    t.text "params"
    t.datetime "run_at", null: false
    t.datetime "updated_at", null: false
  end

  create_table "forecasts", force: :cascade
do |t|
    t.datetime "created_at", null: false
    t.integer "forecast_run_id", null: false
    t.integer "hall_id", null: false
    t.integer "movie_id"
    t.decimal "predicted_fill_pct",
precision: 5, scale: 2
    t.decimal "predicted_tickets", precision:
10, scale: 2
    t.datetime "showtime_at", null: false
    t.integer "showtime_id"
    t.datetime "updated_at", null: false
    t.index ["forecast_run_id"], name:
"index_forecasts_on_forecast_run_id"
    t.index ["hall_id"], name:
"index_forecasts_on_hall_id"
    t.index ["movie_id"], name:
"index_forecasts_on_movie_id"
    t.index ["showtime_id"], name:
"index_forecasts_on_showtime_id"
  end

  create_table "halls", force: :cascade do
|t|
    t.integer "cinema_id", null: false
    t.datetime "created_at", null: false
    t.string "name"
    t.integer "rows"
    t.integer "seats_per_row"
    t.datetime "updated_at", null: false
    t.index ["cinema_id"], name:
"index_halls_on_cinema_id"
  end

  create_table "movies", force: :cascade do
|t|
    t.integer "company_id"
    t.datetime "created_at", null: false
    t.datetime "deleted_at"
    t.text "description"
    t.integer "duration"
    t.string "title"
    t.datetime "updated_at", null: false
    t.index ["company_id"], name:
"index_movies_on_company_id"
    t.index ["deleted_at"], name:
"index_movies_on_deleted_at"
  end

  create_table "products", force: :cascade do
|t|
    t.integer "amount"
    t.integer "cinema_id", null: false
    t.datetime "created_at", null: false
    t.string "name"
    t.decimal "price"
    t.integer "sold_amount"
    t.datetime "updated_at", null: false
```

```
    t.index ["cinema_id"], name:
"index_products_on_cinema_id"
  end

  create_table "reports", force: :cascade do
|t|
    t.datetime "created_at", null: false
    t.text "product_details"
    t.integer "status", default: 0
    t.integer "tickets_count"
    t.decimal "total_revenue"
    t.decimal "total_sales", precision: 10,
scale: 2, default: "0.0", null: false
    t.datetime "updated_at", null: false
    t.integer "workday_id", null: false
    t.index ["workday_id"], name:
"index_reports_on_workday_id"
  end

  create_table "seats", force: :cascade do
|t|
    t.datetime "created_at", null: false
    t.integer "hall_id", null: false
    t.integer "number"
    t.integer "row"
    t.string "status"
    t.datetime "updated_at", null: false
    t.index ["hall_id"], name:
"index_seats_on_hall_id"
  end

  create_table "showtimes", force: :cascade
do |t|
    t.datetime "created_at", null: false
    t.integer "hall_id", null: false
    t.integer "movie_id", null: false
    t.decimal "price"
    t.datetime "start_time"
    t.datetime "updated_at", null: false
    t.index ["hall_id"], name:
"index_showtimes_on_hall_id"
    t.index ["movie_id"], name:
"index_showtimes_on_movie_id"
  end

  create_table
"solid_queue_blocked_executions", force:
:cascade do |t|
    t.string "concurrency_key", null: false
    t.datetime "created_at", null: false
    t.datetime "expires_at", null: false
    t.bigint "job_id", null: false
    t.integer "priority", default: 0, null:
false
    t.string "queue_name", null: false
    t.index ["concurrency_key", "priority",
"job_id"], name:
"index_solid_queue_blocked_executions_for_rel
ease"
    t.index ["expires_at",
"concurrency_key"], name:
"index_solid_queue_blocked_executions_for_mai
ntenance"
    t.index ["job_id"], name:
"index_solid_queue_blocked_executions_on_job_
id", unique: true
  end

  create_table
"solid_queue_claimed_executions", force:
:cascade do |t|
    t.datetime "created_at", null: false
    t.bigint "job_id", null: false
    t.bigint "process_id"
    t.index ["job_id"], name:
"index_solid_queue_claimed_executions_on_job_
id", unique: true
    t.index ["process_id", "job_id"], name:
"index_solid_queue_claimed_executions_on_proc
ess_id_and_job_id"
  end

  create_table
"solid_queue_failed_executions", force:
:cascade do |t|
    t.datetime "created_at", null: false
    t.text "error"
    t.bigint "job_id", null: false
    t.index ["job_id"], name:
"index_solid_queue_failed_executions_on_job_i
d", unique: true
  end

  create_table "solid_queue_jobs", force:
:cascade do |t|
    t.string "active_job_id"
    t.text "arguments"
    t.string "class_name", null: false
    t.string "concurrency_key"
    t.datetime "created_at", null: false
    t.datetime "finished_at"
    t.integer "priority", default: 0, null:
false
    t.string "queue_name", null: false
    t.datetime "scheduled_at"
```

```
    t.datetime "updated_at", null: false
    t.index ["active_job_id"], name:
"index_solid_queue_jobs_on_active_job_id"
    t.index ["class_name"], name:
"index_solid_queue_jobs_on_class_name"
    t.index ["finished_at"], name:
"index_solid_queue_jobs_on_finished_at"
    t.index ["queue_name", "finished_at"],
name: "index_solid_queue_jobs_for_filtering"
    t.index ["scheduled_at", "finished_at"],
name: "index_solid_queue_jobs_for_alerting"
  end

  create_table "solid_queue_pauses", force:
:cascade do |t|
    t.datetime "created_at", null: false
    t.string "queue_name", null: false
    t.index ["queue_name"], name:
"index_solid_queue_pauses_on_queue_name",
unique: true
  end

  create_table "solid_queue_processes",
force: :cascade do |t|
    t.datetime "created_at", null: false
    t.string "hostname"
    t.string "kind", null: false
    t.datetime "last_heartbeat_at", null:
false
    t.text "metadata"
    t.string "name", null: false
    t.integer "pid", null: false
    t.bigint "supervisor_id"
    t.index ["last_heartbeat_at"], name:
"index_solid_queue_processes_on_last_heartbea
t_at"
    t.index ["name", "supervisor_id"], name:
"index_solid_queue_processes_on_name_and_supe
rvisor_id", unique: true
    t.index ["supervisor_id"], name:
"index_solid_queue_processes_on_supervisor_id
"
  end

  create_table
"solid_queue_ready_executions", force:
:cascade do |t|
    t.datetime "created_at", null: false
    t.bigint "job_id", null: false
    t.integer "priority", default: 0, null:
false
    t.string "queue_name", null: false
    t.index ["job_id"], name:
"index_solid_queue_ready_executions_on_job_id
", unique: true
    t.index ["priority", "job_id"], name:
"index_solid_queue_poll_all"
    t.index ["queue_name", "priority",
"job_id"], name:
"index_solid_queue_poll_by_queue"
  end

  create_table
"solid_queue_recurring_executions", force:
:cascade do |t|
    t.datetime "created_at", null: false
    t.bigint "job_id", null: false
    t.datetime "run_at", null: false
    t.string "task_key", null: false
    t.index ["job_id"], name:
"index_solid_queue_recurring_executions_on_jo
b_id", unique: true
    t.index ["task_key", "run_at"], name:
"index_solid_queue_recurring_executions_on_ta
sk_key_and_run_at", unique: true
  end

  create_table "solid_queue_recurring_tasks",
force: :cascade do |t|
    t.text "arguments"
    t.string "class_name"
    t.string "command", limit: 2048
    t.datetime "created_at", null: false
    t.text "description"
    t.string "key", null: false
    t.integer "priority", default: 0, null:
false
    t.string "queue_name"
    t.string "schedule", null: false
    t.boolean "static", default: true, null:
false
    t.datetime "updated_at", null: false
    t.index ["key"], name:
"index_solid_queue_recurring_tasks_on_key",
unique: true
    t.index ["static"], name:
"index_solid_queue_recurring_tasks_on_static"
  end

  create_table
"solid_queue_scheduled_executions", force:
:cascade do |t|
    t.datetime "created_at", null: false
    t.bigint "job_id", null: false
```

```
    t.integer "priority", default: 0, null:
false
    t.string "queue_name", null: false
    t.datetime "scheduled_at", null: false
    t.index ["job_id"], name:
"index_solid_queue_scheduled_executions_on_jo
b_id", unique: true
    t.index ["scheduled_at", "priority",
"job_id"], name:
"index_solid_queue_dispatch_all"
  end

  create_table "solid_queue_semaphores",
force: :cascade do |t|
    t.datetime "created_at", null: false
    t.datetime "expires_at", null: false
    t.string "key", null: false
    t.datetime "updated_at", null: false
    t.integer "value", default: 1, null:
false
    t.index ["expires_at"], name:
"index_solid_queue_semaphores_on_expires_at"
    t.index ["key", "value"], name:
"index_solid_queue_semaphores_on_key_and_valu
e"
    t.index ["key"], name:
"index_solid_queue_semaphores_on_key",
unique: true
  end

  create_table "tickets", force: :cascade do
|t|
    t.datetime "created_at", null: false
    t.integer "seat_id", null: false
    t.integer "showtime_id", null: false
    t.string "status"
    t.datetime "updated_at", null: false
    t.integer "workday_id", null: false
    t.index ["seat_id"], name:
"index_tickets_on_seat_id"
    t.index ["showtime_id"], name:
"index_tickets_on_showtime_id"
    t.index ["workday_id"], name:
"index_tickets_on_workday_id"
  end

  create_table "users", force: :cascade do
|t|
    t.integer "cinema_id"
    t.integer "company_id"
    t.datetime "created_at", null: false
    t.string "email", default: "", null:
false
    t.string "encrypted_password", default:
"", null: false
    t.string "name"
    t.datetime "remember_created_at"
    t.datetime "reset_password_sent_at"
    t.string "reset_password_token"
    t.string "role"
    t.datetime "updated_at", null: false
    t.index ["cinema_id"], name:
"index_users_on_cinema_id"
    t.index ["company_id"], name:
"index_users_on_company_id"
    t.index ["email"], name:
"index_users_on_email", unique: true
    t.index ["reset_password_token"], name:
"index_users_on_reset_password_token",
unique: true
  end

  create_table "workdays", force: :cascade do
|t|
    t.decimal "bar_sales_total", precision:
10, scale: 2, default: "0.0", null: false
    t.integer "cinema_id", null: false
    t.datetime "created_at", null: false
    t.datetime "end_time"
    t.text "product_snapshot"
    t.datetime "start_time"
    t.datetime "updated_at", null: false
    t.integer "user_id", null: false
    t.index ["cinema_id"], name:
"index_workdays_on_cinema_id"
    t.index ["user_id"], name:
"index_workdays_on_user_id"
  end

  add_foreign_key "cinemas", "companies"
  add_foreign_key "forecasts",
"forecast_runs"
  add_foreign_key "forecasts", "halls"
  add_foreign_key "forecasts", "movies"
  add_foreign_key "forecasts", "showtimes"
  add_foreign_key "halls", "cinemas"
  add_foreign_key "movies", "companies"
  add_foreign_key "products", "cinemas"
  add_foreign_key "reports", "workdays"
  add_foreign_key "seats", "halls"
  add_foreign_key "showtimes", "halls"
  add_foreign_key "showtimes", "movies"
  add_foreign_key
"solid_queue_blocked_executions",
"solid_queue_jobs", column: "job_id",
on_delete: :cascade
```

```
  add_foreign_key
"solid_queue_claimed_executions",
"solid_queue_jobs", column: "job_id",
on_delete: :cascade
  add_foreign_key
"solid_queue_failed_executions",
"solid_queue_jobs", column: "job_id",
on_delete: :cascade
  add_foreign_key
"solid_queue_ready_executions",
"solid_queue_jobs", column: "job_id",
on_delete: :cascade
  add_foreign_key
"solid_queue_recurring_executions",
"solid_queue_jobs", column: "job_id",
on_delete: :cascade
  add_foreign_key
"solid_queue_scheduled_executions",
"solid_queue_jobs", column: "job_id",
on_delete: :cascade
  add_foreign_key "tickets", "seats"
  add_foreign_key "tickets", "showtimes"
  add_foreign_key "tickets", "workdays"
  add_foreign_key "users", "cinemas",
on_delete: :nullify
  add_foreign_key "users", "companies"
  add_foreign_key "workdays", "cinemas"
  add_foreign_key "workdays", "users"
end
```

ФОНОВИЙ КОНВЕЄР ПРОГНОЗУВАННЯ

```
class ForecastRunnerJob < ApplicationJob
  queue_as :default

  SEGMENTS = %i[weekday_day weekday_night
weekend].freeze

  def perform(horizon_days: 7)
    run = ForecastRun.create!(
        run_at: Time.current,
        horizon_days: horizon_days,
        model: 'segmented_regression'
        )
    horizon_days = horizon_days.to_i
    window_start =
Time.current.beginning_of_day
    window_end = (window_start +
(horizon_days - 1).days).end_of_day

    Forecast.where('showtime_at >= ? AND
showtime_at <= ?', window_start,
window_end).delete_all
    movie_occupancy_cache = {}
    release_dates_cache = {}

    Hall.find_each do |hall|
      capacity = hall.capacity.to_i
      next if capacity <= 0


      all_rows =
gather_training_rows_for(hall, capacity,
movie_occupancy_cache, release_dates_cache)
      next if all_rows.empty?

      segmented_rows =
split_into_segments(all_rows)

      models =
train_segmented_models(hall.id,
segmented_rows)

      global_trainer =
Forecasting::RegressionTrainer.new(all_rows)
      global_model   =
global_trainer.train(ridge: 0.01)

      upcoming_showtimes =
hall.showtimes.where('start_time >= ? AND
start_time <= ?', window_start, window_end)
      upcoming_showtimes.each do |showtime|
        hour    = showtime.start_time.hour
        is_wknd =
showtime.start_time.saturday? ||
showtime.start_time.sunday?
        date    = showtime.start_time.to_date

        segment = classify_segment(hour,
is_wknd)
        active_trainer =
models[segment][:trainer]
        active_model   =
models[segment][:model]

        if active_model.nil?
```

```
          active_trainer = global_trainer
          active_model   = global_model
        end
        next if active_model.nil?

        movie_pop =
cached_historical_occupancy(showtime.movie_id
, movie_occupancy_cache) ||
          default_occupancy_for_movie(
            showtime.movie,
            date,
            release_dates_cache,
            hour: hour,
            is_weekend: is_wknd
          )

        is_morning   = (hour >= 8 && hour <
12)
        is_afternoon = (hour >= 12 && hour <
17)
        is_evening   = (hour >= 17)
        slot_floor   =
minimum_floor_for_slot_with_day_factor(hour,
is_wknd)

        features = {
          movie_popularity: movie_pop,
          lag_feature:      lag_feature_for_s
howtime(showtime, days_ago: 7),
          lag_2_weeks:      lag_feature_for_s
howtime(showtime, days_ago: 14),
          rolling_avg_7d:   rolling_avg_7d(ha
ll, showtime, capacity),
          start_time:       showtime.start_ti
me,
          movie_id:         showtime.movie_id
,
          hall_id:          showtime.hall_id,
          release_date:     get_movie_release
_date(showtime.movie_id,
release_dates_cache),
          is_weekend:       is_wknd,
          is_morning:       is_morning,
          is_afternoon:     is_afternoon,
          is_evening:       is_evening,
          day_of_week:      day_of_week(date)
,
          week_of_month:    week_of_month(dat
e),
          is_holiday:       is_holiday?(date)
        }

        baseline_occupancy = [movie_pop,
slot_floor].max
        raw_occupancy      =
active_trainer.predict(active_model,
features)

        occupancy = raw_occupancy.nil? ?
baseline_occupancy : ((raw_occupancy * 0.55)
+ (baseline_occupancy * 0.45))
        occupancy = slot_floor if occupancy <
slot_floor
        occupancy = 0.95 if occupancy > 0.95

        days_since_release    =
calculate_days_since_release(showtime.movie,
date, release_dates_cache)
        new_release_multiplier =
release_multiplier_for(days_since_release)

        if segment == :weekday_day
          new_release_multiplier = 1.0 +
(new_release_multiplier - 1.0) * 0.5
        end

        occupancy = (occupancy *
new_release_multiplier).clamp(slot_floor,
0.95)
        predicted_tickets = (occupancy *
capacity).round

        if segment == :weekday_day &&
predicted_tickets < 5
          conservative_tickets = (0.10 *
capacity).round
          predicted_tickets =
conservative_tickets
        end

        Forecast.create!(
          forecast_run:       run,
          hall:               hall,
          showtime:           showtime,
          movie:              showtime.movie,
          showtime_at:        showtime.start_
time,
          predicted_tickets:  predicted_ticke
ts,
          predicted_fill_pct: (occupancy *
100).round(2)
        )
      end
    end
```

```
    ForecastAccuracyJob.perform_now(forecast_
run_id: run.id, days_back: 7)
  end

  private

  def is_holiday?(date)
    return false if date.nil?
    month_day = "#{date.month}-#{date.day}"
    fixed_holidays = %w[
      1-1    # New Year's Day
      3-8    # International Women's Day
      5-1    # Labour Day
      6-28   # Constitution Day of Ukraine
      7-15   # Statehood Day
      8-24   # Independence Day of Ukraine
      10-1   # Day of Defenders and
Defendresses of Ukraine
      12-25  # Christmas Day
    ]

    return true if
fixed_holidays.include?(month_day)

    easter_dates = [
      Date.new(2024, 5, 5),   # Easter 2024
      Date.new(2025, 4, 20),  # Easter 2025
      Date.new(2026, 4, 12),  # Easter 2026
      Date.new(2027, 5, 2),   # Easter 2027
      Date.new(2028, 4, 16),  # Easter 2028
    ]

    easter_dates.include?(date)
  end

  def week_of_month(date)
    return 1 if date.day <= 7
    return 2 if date.day <= 14
    return 3 if date.day <= 21
    4
  end

  def day_of_week(date)
    date.wday
  end

  def lag_feature_for_showtime(showtime,
days_ago: 7)
    return 0.0 if showtime.nil?
    target_time = showtime.start_time -
days_ago.days
    historical = Showtime.where(movie_id:
showtime.movie_id, hall_id: showtime.hall_id)
                         .where('start_time
<= ?', target_time)
                         .order(start_time:
:desc)
                         .first

    return 0.0 if historical.nil?
    cap = historical.hall.capacity.to_i
    return 0.0 if cap <= 0
    historical.tickets.where(status: %w[sold
pending]).count.to_f / cap
  end

  def rolling_avg_7d(hall, showtime,
capacity)
    return 0.25 if showtime.nil? || capacity
<= 0
    cutoff_time = showtime.start_time -
7.days
    recent_showtimes = hall.showtimes
                           .where('start_time
>= ?', cutoff_time)
                           .where('start_time
< ?', showtime.start_time)
                           .to_a
    return 0.25 if recent_showtimes.empty?

    showtime_ids = recent_showtimes.map(&:id)
    ticket_counts = Ticket.where(showtime_id:
showtime_ids, status: %w[sold pending])
                          .group(:showtime_id
)
                          .count

    occupancies = recent_showtimes.map { |st|
ticket_counts[st.id].to_i.to_f / capacity }
    occupancies.empty? ? 0.25 :
occupancies.sum / occupancies.length
  end

  def classify_segment(hour, is_weekend)
    return :weekend      if is_weekend
    return :weekday_day  if hour < 18
    :weekday_night
  end

  def split_into_segments(rows)
    {
      weekday_day:   rows.select { |r|
!r[:is_weekend] && r[:start_time].hour < 18
},
```

```
      weekday_night: rows.select { |r|
!r[:is_weekend] && r[:start_time].hour >= 18
},
      weekend:       rows.select { |r|
r[:is_weekend] }
    }
  end

  def train_segmented_models(hall_id,
segmented_rows)
    SEGMENTS.each_with_object({}) do
|segment, result|
      seg_rows = segmented_rows[segment]

      if seg_rows.size < 5
        result[segment] = { trainer: nil,
model: nil }
        next
      end

      trainer =
Forecasting::RegressionTrainer.new(seg_rows)
      model   = trainer.train(ridge: 0.01)
      result[segment] = { trainer: trainer,
model: model }
    end
  end

  def
release_multiplier_for(days_since_release)
    case days_since_release
    when 0..3   then 1.30
    when 4..7   then 1.10
    when 8..11  then 0.80
    when 12..13 then 0.65
    when 14..20 then 0.50
    else             0.35
    end
  end

  def gather_training_rows_for(hall,
capacity, movie_cache = {},
release_dates_cache = {})
    past_showtimes =
hall.showtimes.where('start_time < ?',
Time.current -
24.hours).includes(:movie).to_a

    showtime_ids = past_showtimes.map(&:id)
    return [] if showtime_ids.empty?

    ticket_counts = Ticket.where(showtime_id:
showtime_ids, status: %w[sold
pending]).group(:showtime_id).count

    past_showtimes.map do |showtime|
      sold_tickets =
ticket_counts[showtime.id].to_i
      occupancy    = sold_tickets.to_f /
capacity

      hour    = showtime.start_time.hour
      is_wknd = showtime.start_time.saturday?
|| showtime.start_time.sunday?
      date    = showtime.start_time.to_date

      movie_pop =
cached_historical_occupancy(showtime.movie_id
, movie_cache) ||
        default_occupancy_for_movie(
          showtime.movie,
          date,
          release_dates_cache,
          hour: hour,
          is_weekend: is_wknd
        )

      {
        occupancy_pct:    occupancy,
        movie_popularity: movie_pop,
        lag_feature:      lag_feature_for_sho
wtime(showtime, days_ago: 7),
        lag_2_weeks:      lag_feature_for_sho
wtime(showtime, days_ago: 14),
        rolling_avg_7d:   rolling_avg_7d(hall
, showtime, capacity),
        start_time:       showtime.start_time
,
        movie_id:         showtime.movie_id,
        hall_id:          showtime.hall_id,
        release_date:     get_movie_release_d
ate(showtime.movie_id, release_dates_cache),
        is_weekend:       is_wknd,
        is_morning:       (hour >= 8 && hour
< 12),
        is_afternoon:     (hour >= 12 && hour
< 17),
        is_evening:       (hour >= 17),
        day_of_week:      day_of_week(date),
        week_of_month:    week_of_month(date)
,
        is_holiday:       is_holiday?(date),
```

```
        days_since_release:
calculate_days_since_release(showtime.movie,
date, release_dates_cache)
      }
    end
  end

  def
calculate_historical_occupancy(movie_id)
    past_showtimes = Showtime.where(movie_id:
movie_id)
                             .where('start_ti
me < ?', Time.current - 24.hours)
                             .includes(:hall)

    return nil if past_showtimes.empty?
    showtime_ids = past_showtimes.map(&:id)
    return nil if showtime_ids.empty?
    ticket_counts = Ticket.where(showtime_id:
showtime_ids, status: %w[sold
pending]).group(:showtime_id).count
    occupancies = past_showtimes.filter_map
do |s|
      cap = s.hall.capacity.to_i
      next if cap <= 0
      ticket_counts[s.id].to_i.to_f / cap
    end
    return nil if occupancies.empty?

    sorted = occupancies.sort
    median = sorted.length.odd? ?
sorted[sorted.length / 2] :
(sorted[sorted.length / 2 - 1] +
sorted[sorted.length / 2]) / 2.0
    median
  end

  def cached_historical_occupancy(movie_id,
cache)
    cache[movie_id] ||=
calculate_historical_occupancy(movie_id)
  end

  def get_movie_release_date(movie_id, cache)
    cache[movie_id] ||=
Showtime.where(movie_id:
movie_id).order(start_time:
:asc).first&.start_time&.to_date
  end

  def
minimum_floor_for_slot_with_day_factor(hour,
is_weekend)
    if is_weekend
      return 0.10 if hour >= 8 && hour < 12
      return 0.12 if hour >= 12 && hour < 17
      return 0.15
    end

    return 0.005 if hour < 18

    0.15
  end

  def calculate_days_since_release(movie,
target_date, release_dates_cache)
    return 999 if movie.nil? ||
target_date.nil?

    release_date =
get_movie_release_date(movie.id,
release_dates_cache)
    return 999 if release_date.nil?

    [(target_date - release_date).to_i,
0].max
  end

  def default_occupancy_for_movie(movie,
target_date, release_dates_cache, hour: nil,
is_weekend: nil)
    return 0.40 if movie.nil? ||
target_date.nil?

    release_date =
get_movie_release_date(movie.id,
release_dates_cache)
    return 0.40 if release_date.nil?

    days_since_release = (target_date -
release_date).to_i

    if days_since_release >= 0 &&
days_since_release <= 3
      return 0.20 if !is_weekend && hour &&
hour < 18
      return 0.40
    end
    0.25
  end
end
```

МАТЕМАТИЧНА МОДЕЛЬ

```
require 'matrix'

module Forecasting
  class RegressionTrainer
    def initialize(rows)
      @rows = rows
    end

    def train(ridge: 1.0)
      return nil if @rows.size < 6
      feature_names =
self.class.feature_names

      x_raw = @rows.map do |row|
        feature_vector(row).values_at(*featur
e_names)
      end
      y = Vector.elements(@rows.map { |r|
r[:occupancy_pct].to_f })

      means = []
      stds = []
      x_std = x_raw.transpose.map do |col|
        col = col.map(&:to_f)
        mean = col.sum / col.size
        variance = col.map { |v| (v - mean)
** 2 }.sum / col.size
        std = Math.sqrt(variance)
        std = 1.0 if std == 0.0
        means << mean
        stds << std
        col.map { |v| (v - mean) / std }
      end

      x_rows = x_std.transpose.map { |r|
[1.0] + r }
      x = Matrix.rows(x_rows)
      xt = x.transpose
      xtx = xt * x

      if ridge && ridge.to_f != 0.0
        ridge_val = ridge.to_f
        mat = xtx.to_a
        (0...mat.size).each do |i|
          mat[i][i] += (i == 0 ? 0.0 :
ridge_val)
        end
        xtx = Matrix.rows(mat)
      end

      begin
        beta_std = xtx.inverse * xt * y
      rescue StandardError
        return nil
      end

      intercept = beta_std[0]
      coefs = [intercept]
      beta_std.to_a[1..-1].each_with_index do
|b, idx|
        coefs << (b.to_f / stds[idx])
      end

      intercept_adj = coefs[0].to_f
      coefs[1..-1].each_with_index do |coef,
idx|
        intercept_adj -= coef.to_f *
means[idx]
      end
      coefs[0] = intercept_adj

      { feature_names:
feature_names.map(&:to_s), coefs: coefs,
means: means, stds: stds }
    end

    def predict(model, features)
      return nil if model.nil? ||
model[:coefs].nil?
      coefs = model[:coefs]
      feature_vector =
feature_vector(features)
      vec = [1.0] +
self.class.feature_names.map { |name|
feature_vector[name].to_f }
      vec.zip(coefs).map { |a, b| a * b.to_f
}.sum
    end

    def self.feature_names
      %i[movie_popularity days_since_release
lag_feature is_weekend is_morning
       is_afternoon is_evening day_of_week
week_of_month is_holiday
       lag_2_weeks rolling_avg_7d]
    end

    private
```

```
    def feature_vector(row)
      row = row || {}
      {
        movie_popularity:
row[:movie_popularity].to_f,
        days_since_release:
days_since_release_for(row),
        lag_feature: lag_feature_for(row),
        is_weekend:
flag_value(row[:is_weekend]),
        is_morning:
flag_value(row[:is_morning]),
        is_afternoon:
flag_value(row[:is_afternoon]),
        is_evening:
flag_value(row[:is_evening]),
        day_of_week: row[:day_of_week].to_f,
        week_of_month:
row[:week_of_month].to_f,
        is_holiday:
flag_value(row[:is_holiday]),
        lag_2_weeks: (row[:lag_2_weeks] ||
0).to_f,
        rolling_avg_7d: (row[:rolling_avg_7d]
|| 0.25).to_f
      }
    end

    def flag_value(value)
      return 0.0 if value.nil?
      return 1.0 if value == true
      return 0.0 if value == false

      return value.to_f > 0.0 ? 1.0 : 0.0 if
value.is_a?(Numeric)

      normalized = value.to_s.strip.downcase
      return 1.0 if %w[1 true t yes
y].include?(normalized)

      0.0
    end

    def days_since_release_for(row)
      start_time = row[:start_time]
      release_date = row[:release_date] ||
movie_release_date_for(row[:movie_id])

      return (row[:days_since_release] ||
0).to_f if start_time.blank? ||
release_date.blank?

      [(start_time.to_date -
release_date.to_date).to_i, 0].max.to_f
    end

    def movie_release_date_for(movie_id)
      return nil if movie_id.blank?

      movie = Movie.find_by(id: movie_id)
      return nil if movie.nil?

      movie.try(:release_date) ||
movie.created_at
    end

    def lag_feature_for(row)
      return (row[:lag_feature] || 0).to_f if
row[:lag_feature]

      start_time = row[:start_time]
      movie_id = row[:movie_id]
      hall_id = row[:hall_id]

      return 0.0 if start_time.blank? ||
movie_id.blank? || hall_id.blank?

      target_time = start_time - 7.days
      historical_showtime =
Showtime.where(movie_id: movie_id, hall_id:
hall_id)
                                      .where('s
tart_time <= ?', target_time)
                                      .order(st
art_time: :desc)
                                      .first

      return 0.0 unless historical_showtime

      hall_capacity =
historical_showtime.hall&.capacity.to_i
      return 0.0 if hall_capacity <= 0

      sold_tickets =
historical_showtime.tickets.where(status:
%w[sold pending]).count
      sold_tickets.to_f / hall_capacity
    end
  end
end
```

ДОДАТОК Б

ЛІСТИНГ КОДУ ІНТЕРФЕЙСНОЇ ЧАСТИНИ

КАСОВИЙ ЕКРАН ПРОДАЖУ КВИТКІВ

```
<div class="tickets-container">
  <div style="display: flex; gap: 2rem;">
    <div style="flex: 1;">
      <div class="booking-header">
        <h1>Продаж квитків</h1>
        <div class="movie-info">
          <h2><%= @movie.title %></h2>
          <p class="ticket-
price"><strong>Ціна:</strong> <%=
@showtime.price %> UAH</p>
          <p><strong>Дата та час:</strong>
<%= @showtime.start_time.strftime("%B %d, %Y
at %H:%M") %></p>
          <p><strong>Кінотеатр:</strong> <%=
@hall.cinema.name %></p>
          <p><strong>Зал:</strong> <%=
@hall.name %></p>
        </div>
      </div>

      <div class="screen-container">
        <div class="screen">ЕКРАН</div>
      </div>

      <%= form_with url: tickets_path,
method: :post, local: true, class: "ticket-
form", data: { turbo: false } do |form| %>
    <%= hidden_field_tag
"ticket[showtime_id]", @showtime.id %>
    <%= hidden_field_tag
"ticket[generate_pdf]", "0", id: "generate-
pdf-flag" %>

    <div class="seat-map">
      <div class="seat-grid" style="grid-
template-columns: repeat(<%=
@hall.seats_per_row %>, 1fr); gap: 8px;">
        <% @seats.each do |seat| %>
          <% ticket =
@tickets_by_seat[seat.id] %>
          <% if ticket && ticket.status !=
'cancelled' %>
            <button type="button"
class="seat-label seat-status-<%=
ticket.status %>" data-seat-id="<%= seat.id
%>" data-ticket-id="<%= ticket.id %>"
title="Правий клік — опції: Місце <%=
seat.row %>-<%= seat.number %> (<%=
ticket.status %>)">
              <div class="seat-text"><%=
seat.number %></div>
            </button>
          <% else %>
            <label class="seat-label"
title="Місце <%= seat.row %>-<%= seat.number
%>">
              <%= check_box_tag
"ticket[seat_ids][]", seat.id %>
              <div class="seat-text"><%=
seat.number %></div>
            </label>
          <% end %>
        <% end %>
      </div>
    </div>

    <div class="legend">
      <div class="legend-item">
        <div class="legend-box
available"></div>
        <span>Доступно</span>
      </div>
      <div class="legend-item">
        <div class="legend-box
selected"></div>
        <span>Вибрано</span>
      </div>
      <div class="legend-item">
        <div class="legend-box sold"></div>
        <span>Продано</span>
      </div>
      <div class="legend-item">
        <div class="legend-box"
style="background-color: #ff9800; border-
color: #e65100;"></div>
        <span>У обробці</span>
      </div>
    </div>

    <div class="actions">
      <p id="total-price-display"
style="margin-bottom: 1rem; font-size:
1.1rem; font-weight: 600;">Загальна сума: 0
UAH</p>
```

```
      <%= form.submit "Продати вибрані квитки", class: "btn btn-primary", id: "sell-tickets-submit" %>
      <%= link_to "Назад до сеансів", showtimes_path, class: "btn btn-secondary" %>
    </div>
      <% end %>
    </div>

    <div style="flex: 0 0 320px; display: flex; flex-direction: column; justify-content: center;">
      <% sold_tickets_count = @showtime.tickets.where(status: ['Sold', 'sold']).count %>
      <% unavailable_tickets_count = @showtime.tickets.where.not(status: ['Available', 'available', 'cancelled', nil]).count %>
      <% available_seats_count = @seats.count - unavailable_tickets_count %>
      <% total_earned = sold_tickets_count * (@showtime.price || 0) %>
      <% current_occupancy_pct = @seats.any? ? (sold_tickets_count.to_f / @seats.count * 100.0) : 0.0 %>

      <div class="stats-panel">
        <% if @latest_forecast_for_showtime.present? %>
          <% predicted_pct = @predicted_occupancy_pct.to_f %>
          <% predicted_tickets = @predicted_tickets_count.to_i %>
          <% occupancy_gap = (predicted_pct - current_occupancy_pct).round(1) %>
          <div class="forecast-box">
            <p class="forecast-title">Прогноз завантаженості залу</p>
            <div class="forecast-main"><%= predicted_pct.round(1) %>%</div>
            <div class="forecast-track">
              <div class="forecast-fill" style="width: <%= [predicted_pct, 100].min %>%; background: <%= forecast_occupancy_color(predicted_pct) %>;"></div>
            </div>
            <div class="forecast-copy">
              Очікується <strong><%= predicted_tickets %></strong> квитків • <%= forecast_occupancy_label(predicted_pct) %>
            </div>
            <div class="forecast-copy">
              Зараз продано <strong><%= sold_tickets_count %></strong> квитків (<%= current_occupancy_pct.round(1) %>%), різниця до прогнозу: <strong><%= occupancy_gap >= 0 ? "+#{occupancy_gap}" : occupancy_gap %>%</strong>
            </div>
            <% if @forecast_generated_at.present? %>
              <div class="forecast-muted">Оновлено: <%= l(@forecast_generated_at, format: :short) %></div>
            <% end %>
          </div>
        <% else %>
          <div class="forecast-box">
            <p class="forecast-title">Прогноз завантаженості залу</p>
            <div class="forecast-copy">Для цього сеансу поки немає прогнозу. Запустіть побудову прогнозів, щоб бачити очікуване навантаження під час продажу квитків.</div>
          </div>
        <% end %>

        <div class="stat-item">
          <div class="stat-value"><%= available_seats_count %></div>
          <div class="stat-label">Доступні місця</div>
        </div>
        <div class="stat-item">
          <div class="stat-value"><%= sold_tickets_count %></div>
          <div class="stat-label">Продані місця</div>
        </div>
        <div class="stat-item">
          <div class="stat-value"><%= total_earned %> UAH</div>
          <div class="stat-label">Загальний дохід</div>
        </div>
      </div>
    </div>
  </div>
</div>

<div id="payment-modal" class="payment-modal hidden" aria-hidden="true">
```

```
  <div class="payment-modal__backdrop" data-close-payment-modal="true"></div>
  <div class="payment-modal__card" role="dialog" aria-modal="true" aria-labelledby="payment-modal-title">
    <h3 id="payment-modal-title">Очікується оплата</h3>
    <p class="payment-modal__copy">Перевірте обрані місця перед підтвердженням. Після оплати буде згенеровано PDF з квитками.</p>
    <div class="payment-modal__summary">
      <div>
        <span>Кількість місць</span>
        <strong id="payment-selected-count">0</strong>
      </div>
      <div>
        <span>Сума до оплати</span>
        <strong id="payment-total-amount">0 UAH</strong>
      </div>
    </div>
    <p id="payment-modal-status" class="payment-modal__status">Ініціалізація платежу...</p>
    <div class="payment-modal__actions">
      <button type="button" class="btn btn-secondary" id="payment-cancel-btn">Скасувати</button>
      <button type="button" class="btn btn-primary" id="payment-confirm-btn" disabled>Підтвердити оплату</button>
    </div>
  </div>
</div>

<div id="seat-context-menu" class="context-menu" style="display: none; position: fixed; z-index: 1000;">
  <div class="context-menu-item" data-action="cancelled">Скасувати місце</div>
  <div class="context-menu-item" data-action="sold">Позначити як продане</div>
  <div class="context-menu-item" data-action="pending">Позначити як в обробці</div>
</div>

<script>
  document.addEventListener('turbo:load', function() {
    const ticketForm = document.querySelector('.ticket-form');
    const generatePdfFlag = document.getElementById('generate-pdf-flag');
    const paymentModal = document.getElementById('payment-modal');
    const paymentStatus = document.getElementById('payment-modal-status');
    const paymentSelectedCount = document.getElementById('payment-selected-count');
    const paymentTotalAmount = document.getElementById('payment-total-amount');
    const paymentCancelBtn = document.getElementById('payment-cancel-btn');
    const paymentConfirmBtn = document.getElementById('payment-confirm-btn');
    const seatCheckboxes = document.querySelectorAll('input[name="ticket[seat_ids][]"]');
    const totalPriceDisplay = document.getElementById('total-price-display');
    const ticketPrice = <%= @showtime.price.to_f %>;
    const contextMenu = document.getElementById('seat-context-menu');
    let currentSeatId = null;
    let currentTicketId = null;
    let paymentConfirmed = false;
    let paymentReadyTimer = null;

    function selectedSeatsCount() {
      return document.querySelectorAll('input[name="ticket[seat_ids][]"]:checked').length;
    }

    function closePaymentModal() {
      if (!paymentModal) {
        return;
      }
      paymentModal.classList.add('hidden');
      paymentModal.setAttribute('aria-hidden', 'true');
      if (paymentReadyTimer) {
        clearTimeout(paymentReadyTimer);
      }
    }

    function openPaymentModal() {
```

```
      const count = selectedSeatsCount();
      if (!paymentModal || count === 0) {
        return;
      }

      paymentSelectedCount.textContent =
count;
      paymentTotalAmount.textContent =
`${count * ticketPrice} UAH`;
      paymentStatus.textContent = 'Очікуйте
підтвердження платежу...';
      paymentConfirmBtn.disabled = true;

      paymentModal.classList.remove('hidden')
;
      paymentModal.setAttribute('aria-
hidden', 'false');

      paymentReadyTimer =
setTimeout(function() {
        paymentStatus.textContent = 'Оплату
можна підтвердити';
        paymentConfirmBtn.disabled = false;
      }, 900);
    }

    function updateTotalPrice() {
      const selectedCount =
selectedSeatsCount();
      if (totalPriceDisplay) {
        totalPriceDisplay.innerText =
`Загальна сума: ${selectedCount *
ticketPrice} UAH`;
      }
    }

    if (ticketForm) {
      ticketForm.addEventListener('submit',
function(event) {
        if (paymentConfirmed) {
          return;
        }

        const count = selectedSeatsCount();
        if (count === 0) {
          return;
        }

        event.preventDefault();
        openPaymentModal();
      });
    }

    if (paymentCancelBtn) {
      paymentCancelBtn.addEventListener('clic
k', function() {
        closePaymentModal();
      });
    }

    if (paymentConfirmBtn) {
      paymentConfirmBtn.addEventListener('cli
ck', function() {
        if (!ticketForm) {
          return;
        }
        paymentConfirmBtn.disabled = true;
        paymentConfirmed = true;
        if (generatePdfFlag) {
          generatePdfFlag.value = '1';
        }
        closePaymentModal();
        ticketForm.submit();
      });
    }

    if (paymentModal) {
      paymentModal.addEventListener('click',
function(event) {
        if (event.target.closest('[data-
close-payment-modal="true"]')) {
          closePaymentModal();
        }
      });
    }

    seatCheckboxes.forEach(function(checkbox)
{
      const seatLabel =
checkbox.closest('.seat-label');

      if (seatLabel && checkbox.checked) {
        seatLabel.classList.add('seat-
selected');
      }

      checkbox.addEventListener('change',
function() {
        if (seatLabel) {
          seatLabel.classList.toggle('seat-
selected', checkbox.checked);
        }
        updateTotalPrice();
      });
    });
```

```
    document.querySelectorAll('.seat-
label[data-seat-
id]').forEach(function(seatBtn) {
      seatBtn.addEventListener('contextmenu',
function(e) {
        e.preventDefault();
        currentSeatId =
seatBtn.getAttribute('data-seat-id');
        currentTicketId =
seatBtn.getAttribute('data-ticket-id');
        contextMenu.style.display = 'block';
        contextMenu.style.left = e.pageX +
'px';
        contextMenu.style.top = e.pageY +
'px';
      });
    });

    document.addEventListener('click',
function(e) {
      if (e.target.closest('.context-menu')
|| e.target.closest('[data-seat-id]')) {
        return;
      }
      contextMenu.style.display = 'none';
    });

    document.addEventListener('keydown',
function(e) {
      if (e.key === 'Escape') {
        contextMenu.style.display = 'none';
        closePaymentModal();
      }
    });

    document.querySelectorAll('.context-menu-
item').forEach(function(item) {
      item.addEventListener('click',
function() {
        const action =
item.getAttribute('data-action');
        if (currentTicketId) {
          fetch(`/tickets/${currentTicketId}`
, {
            method: 'PATCH',
            headers: {
              'Content-Type':
'application/json',
              'X-CSRF-Token':
document.querySelector('meta[name="csrf-
token"]').getAttribute('content')
            },
            body: JSON.stringify({ ticket: {
status: action } })
          })
          .then(response => {
            if (!response.ok) {
              return
response.json().then(data => {
                throw new Error(data.errors ?
data.errors.join(', ') : `HTTP
${response.status}`);
              });
            }
            return response.json();
          })
          .then(data => {
            if (data.success) {
              location.reload();
            } else {
              alert('Помилка: ' +
(data.errors ? data.errors.join(', ') :
'Невідома помилка'));
            }
          })
          .catch(error => {
            console.error('Error:', error);
            alert('Помилка оновлення квитка:
' + error.message);
          });
        }
        contextMenu.style.display = 'none';
      });
    });

    updateTotalPrice();
  });
</script>
```

ПАНЕЛЬ УПРАВЛІННЯ КІНОБАРОМ

```
<div class="products-page" data-controller="products">
  <div class="products-header">
    <div>
      <h1>Товари</h1>
```

```
      <p class="products-lead">Керування запасами, продажами та надходженнями за активну
зміну.</p>
    </div>

    <div class="products-summary">
      <% if @active_workday.present? %>
        <div class="summary-pill">
          Активна зміна: <strong><%= @active_workday.cinema.name %></strong>
        </div>
        <div class="summary-pill muted">
          Барна виручка: <strong><%= number_to_currency(@active_workday.bar_sales_total, unit:
"₴", precision: 2) %></strong>
        </div>
      <% else %>
        <div class="summary-pill warning">
          Спочатку потрібно відкрити активну зміну
        </div>
      <% end %>
    </div>
  </div>

  <div class="products-toolbar">
    <button type="button" class="btn btn-secondary" data-action="products#openCreateDialog" <%=
"disabled" unless @active_workday.present? %>>
      СТВОРИТИ
    </button>

    <%= button_tag "ЗБЕРЕГТИ ЗМІНИ",
      type: "submit",
      form: "products-bulk-form",
      class: "btn btn-primary",
      disabled: !@active_workday.present? %>
  </div>

  <% if @products.any? %>
    <%= form_with url: bulk_update_products_path, method: :patch, local: true, id: "products-
bulk-form", class: "products-form" do %>
      <div class="table-frame">
        <table class="products-table">
          <thead>
            <tr>
              <th>Назва</th>
              <th>Ціна</th>
              <th>Кількість</th>
              <th>Продано</th>
              <th>Надійшло</th>
              <th></th>
            </tr>
          </thead>
          <tbody>
            <% @products.each do |product| %>
              <tr class="product-row" data-product-row data-initial-amount="<%= product.amount
%>">
```

```
                <td class="product-name"><%= product.name %></td>
                <td class="product-price"><%= number_to_currency(product.price, unit: "₴",
precision: 2) %></td>
                <td class="product-stock"><%= product.amount %></td>

                <td>
                  <div class="adjuster">
                    <button type="button" class="adjust-button" data-
action="products#changeQuantity" data-adjust-target="sold" data-direction="minus">−</button>
                    <input type="number" class="qty-input sold-qty" name="products[<%= product.id
%>][sold_qty]" value="0" readonly>
                    <button type="button" class="adjust-button" data-
action="products#changeQuantity" data-adjust-target="sold" data-direction="plus">+</button>
                  </div>
                </td>

                <td>
                  <div class="adjuster">
                    <button type="button" class="adjust-button" data-
action="products#changeQuantity" data-adjust-target="arrived" data-direction="minus">−</button>
                    <input type="number" class="qty-input arrived-qty" name="products[<%=
product.id %>][arrived_qty]" value="0" readonly>
                    <button type="button" class="adjust-button" data-
action="products#changeQuantity" data-adjust-target="arrived" data-direction="plus">+</button>
                  </div>
                </td>

                <td class="product-actions">
                  <% if current_user.admin_or_manager? %>
                    <%= link_to "🖉", edit_product_path(product), class: "edit-button", title:
"Редагувати" %>
                  <% end %>
                  <%= link_to "🗑", product_path(product), class: "delete-button", data: {
turbo_method: :delete, turbo_confirm: "Ви впевнені, що хочете видалити цей товар?" } %>
                  <input type="hidden" name="products[<%= product.id %>][id]" value="<%=
product.id %>">
                </td>
              </tr>
            <% end %>
          </tbody>
        </table>
      </div>
    <% end %>
  <% else %>
    <div class="empty-state">
      <h2>Товари ще не додані</h2>
      <p>Створи першу позицію через модалку праворуч угорі.</p>
    </div>
  <% end %>

  <dialog class="product-dialog" data-products-target="dialog">
    <div class="dialog-backdrop" data-action="click->products#closeCreateDialog">
      <div class="dialog-panel" data-action="click->products#stopPropagation">
```

```
        <div class="dialog-header">
          <h2>Створити товар</h2>
          <button type="button" class="dialog-close" data-
action="products#closeCreateDialog">×</button>
        </div>

        <%= form_with model: @new_product, local: true, class: "new-product-form" do |f| %>
          <% if @new_product.errors.any? %>
            <div id="error_explanation">
              <h2><%= pluralize(@new_product.errors.count, "error") %> prohibited this product
from being saved:</h2>
              <ul>
                <% @new_product.errors.full_messages.each do |message| %>
                  <li><%= message %></li>
                <% end %>
              </ul>
            </div>
          <% end %>

          <div class="form-grid">
            <div class="form-group">
              <%= f.label :name, "Назва" %>
              <%= f.text_field :name, placeholder: "Назва товару" %>
            </div>

            <div class="form-group">
              <%= f.label :price, "Ціна" %>
              <%= f.number_field :price, placeholder: "0", step: 0.01 %>
            </div>

            <div class="form-group">
              <%= f.label :amount, "Початкова кількість" %>
              <%= f.number_field :amount, placeholder: "0", value: 0 %>
            </div>

            <div class="form-group">
              <%= f.label :sold_amount, "Початково продано" %>
              <%= f.number_field :sold_amount, placeholder: "0", value: 0 %>
            </div>
          </div>

          <div class="dialog-actions">
            <button type="button" class="btn btn-secondary" data-
action="products#closeCreateDialog">Скасувати</button>
            <%= f.submit "Створити", class: "btn btn-primary" %>
          </div>
        <% end %>
      </div>
    </div>
  </dialog>
</div>
```

Бібліографічна довідка

Тема роботи: Розроблення інформаційної системи управління кінотеатром з модулем прогнозування завантаженості залів.

Обсяг бакалаврської роботи 77 сторінок.

Перелік графічного матеріалу:

1. КРБ.ІСТ - 08.00.00.000 С1. Варіанти використання інформаційної системи управління завданнями.
2. КРБ.ІСТ - 08.00.00.000 С1. Діаграма бази даних інформаційної системи управління кінотеатром з модулем прогнозування завантаженості залів.
3. КРБ.ІСТ - 08.00.00.000 С1. Діаграма використання інформаційної системи управління кінотеатром з модулем прогнозування завантаженості залів.
4. КРБ.ІСТ - 08.00.00.001. Програмні вікна інформаційної системи управління кінотеатром з модулем прогнозування завантаженості залів (2 шт.).

Дата закінчення БР

Студент Максим ДАНИЛЮК