Міністерство освіти і науки України
Луцький національний технічний університет
Факультет комп'ютерних та інформаційних технологій
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА

ЗА СТУПЕНЕМ ВИЩОЇ ОСВІТИ «БАКАЛАВР»

РОЗРОБКА ВЕБ-САЙТУ ДЛЯ БРОНЮВАННЯ КВИТКІВ У КІНОТЕАТРІ

DEVELOPMENT OF A WEBSITE FOR BOOKING MOVIE TICKETS

спеціальність 122 Комп'ютерні науки

освітня програма «Комп'ютерні науки»

Виконав: здобувач вищої освіти
групи КН-42
Антонюк Олександр Іванович

_____
(підпис)

Керівник: асистент
Сачук Вікторія Олегівна

_____
(підпис)

Кваліфікаційну роботу
допущено до захисту
«___» _____ 2026 р.
Гарант освітньої програми:
д.пед.н., професор
Тулашвілі Юрій Йосипович

_____
(підпис)

Луцьк – 2026 року

ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ

Факультет *комп'ютерних та інформаційних технологій*
Кафедра *комп'ютерних наук*
Ступінь вищої освіти: *бакалавр*
Галузь знань: *12 Інформаційні технології*
Спеціальність: *122 Комп'ютерні науки*
Освітня програма: *«Комп'ютерні науки»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____Валерій ЛІЩИНА
«16» січня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА
ПЕРШОГО (БАКАЛАВРСЬКОГО) РІВНЯ ВИЩОЇ ОСВІТИ
Антонюк Олександр Іванович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи «Розробка веб-сайту для бронювання квитків у кінотеатрі»

Керівник асистент Сачук Вікторія Олегівна

затверджені наказом закладу вищої освіти від *«16» січня 2026* р. № 32/01-02

2. Строк подання здобувачем вищої освіти кваліфікаційної роботи *«02»червня 2026 р.*

3. Вихідні дані до роботи: Опрацьовано зарубіжні дослідження в області проєктування та реалізації веборієнтованих інформаційних систем бронювання квитків, сучасних архітектурних підходів до побудови односторінкових застосунків (SPA), а також компонентно-орієнтованої розробки клієнтських інтерфейсів.

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

Аналіз сучасного стану проблеми організації веборієнтованих систем бронювання квитків, існуючих методів та засобів її розв'язання. Виконано аналіз і вибір засобів проєктування на основі сучасного фронтенд-стеку (React, TypeScript, Vite). Описано функціональне наповнення системи та розроблено її структурну організацію. Проведено оцінку ергономічних та надійнісних характеристик вебзастосунку, сформовано функціонально-структурну схему роботи об'єкта проєктування.

5. Перелік графічного матеріалу: 5 рисунків, презентація

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис	
		завдання видав	завдання прийняв
Аналіз проблеми за темою роботи та постановка завдань дослідження	*Сачук В. О.*		
Теоретичне дослідження	*Сачук В. О.*		
Практична реалізація	*Сачук В. О.*		
Спеціальні розрахунки	*Сачук В. О.*		
Показник запозичень тексту	_____ %		
Інструментальна перевірка	*Кошелюк В. А.*		
Нормоконтроль	*Сачук В. О.*		
Гарант ОПП	*Тулашвілі Ю. Й.*		

7. Дата видачі завдання «16» січня 2026 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи бакалавра	Строк виконання етапів роботи	Примітка
1	*Провести огляд літературних джерел по темі кваліфікаційної роботи*	*до 17.02.2026 р*	
2	*Провести аналіз загальної проблеми і вибір напрямків дослідження*	*до 28.02.2026 р.*	
3	*Розробити функціональну схему роботи програмного продукту*	*до 12.03.2026 р*	
4	*Описати засоби розробки об'єкта проєктування*	*до 26.03.2026 р.*	
5	*Практична реалізація об'єкта проєктування*	*до 30.04.2026 р.*	
6	*Провести спеціальні розрахунки*	*до 18.05.2026 р.*	
7	*Здача чистового варіанту кваліфікаційної роботи бакалавра на кафедру*	*до 02.06.2026 р.*	

Здобувач вищої освіти _____ Олександр АНТОНЮК

Керівник роботи _____ Вікторія САЧУК

АНОТАЦІЯ

Антонюк О. І. Веб-орієнтована система бронювання квитків у кінотеатрі з використанням сучасного фронтенд-стека (React, TypeScript, Vite). Рукопис.

Кваліфікаційна робота бакалавра ОП «Комп'ютерні науки». Луцький національний технічний університет. Луцьк, 2026.

Кваліфікаційна робота бакалавра складається зі вступу, чотирьох розділів, висновків, списку використаних джерел та додатків, у межах якої в першому розділі виконано аналіз предметної області, розглянуто сучасні веборієнтовані сервіси бронювання квитків, проаналізовано існуючі аналоги та визначено основні функціональні вимоги до системи; у другому розділі наведено специфікацію функціональних і нефункціональних вимог, описано архітектуру системи та обґрунтовано вибір технологічного стеку, зокрема використання React 19, TypeScript та Vite для побудови клієнтської частини застосунку; Третій розділ присвячено програмній реалізації вебзастосунку бронювання квитків у кінотеатрі, здійснено розробку користувацького інтерфейсу з використанням бібліотеки React, реалізовано маршрутизацію за допомогою react-router-dom, а також організовано керування станом і асинхронними запитами із застосуванням @tanstack/react-query; у четвертому розділі розглянуто питання забезпечення якості програмного забезпечення та оптимізації вебзастосунку, зокрема використання інструментів статичного аналізу коду ESLint для дотримання стандартів розробки та підвищення підтримуваності програмного коду.

Ключові слова: вебзастосунок, система бронювання, React, TypeScript, Vite, Tailwind CSS, React Query, i18next, фронтенд-розробки

ANNOTATION

Antoniuk Oleksandr. Web-Based Movie Ticket Booking System Using a Modern Frontend Stack (React, TypeScript, Vite). Manuscript.

Bachelor's Thesis in the Educational Program Computer Science. Lutsk National Technical University. Lutsk, 2026.

The bachelor's qualification work consists of an introduction, four chapters, conclusions, a list of references, and appendices. In the first chapter, the subject area is analyzed, modern web-based ticket booking services are reviewed, existing analogues are examined, and the main functional requirements for the system are defined. The second chapter presents the specification of functional and non-functional requirements, describes the system architecture, and justifies the choice of the technological stack, in particular the use of React 19, TypeScript, and Vite for building the client-side of the application. The third chapter is devoted to the implementation of a web-based cinema ticket booking application, including the development of the user interface using the React library, the implementation of routing with react-router-dom, as well as state management and asynchronous requests using @tanstack/react-query. The fourth chapter addresses issues of software quality assurance and optimization of the web application, including the use of static code analysis tools such as ESLint to ensure compliance with development standards and improve code maintainability.

Keywords: web application, booking system, React, TypeScript, Vite, Tailwind CSS, React Query, i18next, frontend development.

ЗМІСТ

ВСТУП

Цифровізація процесів організації дозвілля та розвитку онлайн-сервісів бронювання розглядається як один із пріоритетних напрямів сучасної інформаційної інженерії, що обумовлено зростанням навантаження на веборієнтовані системи та підвищенням вимог до швидкодії інтерфейсів і точності обробки транзакційних даних. Розширення використання цифрових платформ у сфері кіноіндустрії формує необхідність у побудові архітектурних рішень, здатних забезпечувати мінімізацію часу взаємодії користувача із системою за одночасного збереження коректності бізнес-логіки та узгодженості даних.

Наявні рішення на ринку вебзастосунків для бронювання квитків демонструють неоднорідний рівень функціональної реалізації, де поряд із високонавантаженими корпоративними платформами присутні спрощені системи з обмеженою гнучкістю інтерфейсу та недостатньою масштабованістю архітектури. У таких умовах підвищується значущість застосування сучасних frontend-підходів, орієнтованих на побудову односторінкових застосунків із динамічним оновленням стану та оптимізованою взаємодією з серверною частиною.

Проєктування веборієнтованої системи бронювання квитків передбачає інтеграцію компонентно-орієнтованої архітектури, механізмів асинхронної обробки запитів та адаптивного інтерфейсу, що дозволяє забезпечити стабільну взаємодію користувача з функціональними модулями системи, включаючи перегляд афіші, вибір сеансів та резервування місць у кінозалі. У межах такого підходу формуються передумови для побудови масштабованого програмного продукту з розширюваною структурою.

Метою кваліфікаційної роботи є розробка та програмна реалізація веборієнтованої системи бронювання квитків у кінотеатрі на основі сучасного фронтенд-стеку React, TypeScript та Vite, що забезпечує підвищену

продуктивність, структурну масштабованість і стабільність роботи інтерфейсної частини застосунку.

Об’єктом дослідження визначено процеси автоматизації онлайн-бронювання квитків у кінотеатрі в умовах застосування сучасних вебтехнологій, що функціонують у клієнт-серверній архітектурі.

Предметом дослідження виступає сукупність програмно-алгоритмічних методів та інструментальних засобів проєктування веборієнтованої платформи бронювання квитків, реалізованої на основі React, TypeScript та пов’язаних технологій фронтенд-екосистеми, спрямованих на забезпечення ефективної взаємодії користувача з системою та коректної обробки даних у реальному часі.

Для досягнення поставленої мети необхідно вирішити такі завдання:

– реалізувати користувацький інтерфейс із використанням React та TypeScript;

– забезпечити маршрутизацію застосунку за допомогою react-router-dom;

– організувати керування станом і асинхронними запитами із застосуванням @tanstack/react-query;

– реалізувати адаптивний дизайн із використанням Tailwind CSS та супутніх інструментів;

– провести тестування системи на відповідність функціональним та ергономічним вимогам.

Результати проведеного дослідження та практичні аспекти розробки веб-платформи для бронювання квитків у кінотеатрі були апробовані, узагальнені та представлені на міжнародній науковій конференції «Сучасні інформаційні технології: від теорії до практики». У публікації детально висвітлено архітектурні рішення проєкту, зокрема реалізацію клієнтської частини застосунку з використанням компонентного підходу, організацію керування станом користувача, а також механізми інтерактивної взаємодії з інтерфейсом при виборі місць у кінозалі (додаток А).

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ЧАСТИНИ

1.1 Аналіз сучасного стану проблеми

Інтенсивний розвиток інформаційних технологій, супроводжуваний безперервним удосконаленням веб-орієнтованих засобів, зумовлює трансформацію підходів до організації дозвіллєвої діяльності, зокрема процесів бронювання квитків і керування розкладом кінопоказів. Активне впровадження програмних рішень, інтегрованих у мережеве середовище, забезпечує автоматизовану взаємодію між кінцевим користувачем і інформаційною системою, надаючи оперативний доступ до відомостей про кінофільми, часові параметри сеансів, доступність місць і можливість негайного оформлення транзакцій [1]. За таких умов формуються підвищені вимоги до програмного забезпечення, що має поєднувати ергономічність інтерфейсу, надійність збереження даних і ефективність механізмів резервування.

Зміна поведінкових патернів користувачів, детермінована поширенням цифрових сервісів, обумовлює зростання попиту на дистанційні інструменти взаємодії, що забезпечують вибір контенту, визначення місць і здійснення платіжних операцій без фізичної присутності.

Переважання таких практик, супроводжуване прагненням до мінімізації часових витрат, актуалізує потребу у високопродуктивних веб-додатках із низькою латентністю, підтримкою багатомовності та інтуїтивно зрозумілою структурою навігації, одночасно орієнтованих на забезпечення захисту персональних даних [2]. У зазначеному контексті розроблення інтегрованого веб-рішення для відображення афіші та реалізації квиткового сервісу набуває прикладного значення.

Поряд із зовнішніми аспектами функціонування, істотної уваги потребує оптимізація внутрішніх процесів, пов’язаних із адмініструванням кінопоказів, обробкою запитів і контролем використання ресурсів. Централізована система управління, реалізована на основі єдиного інформаційного середовища,

забезпечує узгодженість розкладу, унеможливлює конфлікти бронювання та дозволяє здійснювати моніторинг завантаженості кінозалів у режимі реального часу [3]. За умов підвищеного навантаження така автоматизація виступає необхідною передумовою стабільності функціонування, мінімізуючи ризики, притаманні ручному управлінню.

Архітектура програмного рішення сформована з урахуванням принципів модульності та масштабованості, передбачаючи розподіл функціональних обов'язків між клієнтським і серверним рівнями. Клієнтська частина, реалізована із застосуванням сучасних інструментів розробки інтерфейсів, забезпечує динамічну візуалізацію даних і взаємодію з користувачем, тоді як серверний компонент, обробляючи запити до прикладного інтерфейсу, реалізує бізнес-логіку, пов'язану з резервуванням і доступом до сховища даних [4-5].

Зберігання інформації організовано із застосуванням вбудованої реляційної бази даних, інтегрованої через відповідний драйвер, що забезпечує автономність роботи системи та стабільність виконання операцій без залежності від зовнішніх ресурсів [6-7].

Функціональна модель системи охоплює повний цикл взаємодії користувача із сервісом, включаючи перегляд розкладу, вибір місць, формування замовлення та подальше управління бронюваннями. У процесі виконання транзакцій реалізовано механізми перевірки коректності введених даних і зміни станів замовлень, що забезпечує відстеження їхнього життєвого циклу.

На клієнтському рівні передбачено інтерфейси для доступу до розкладу, оформлення замовлень і перегляду історії бронювань, де здійснюється розмежування активних і завершених операцій відповідно до встановлених статусів. Для активних бронювань передбачено можливість їх скасування.

Серверний компонент забезпечує обробку запитів різного типу, включаючи перевірку працездатності, отримання переліку кінофільмів, додавання нових записів, формування розкладу сеансів із використанням об'єднаних запитів до кількох сутностей, а також перевірку доступності місць і реєстрацію бронювань [5]. Забезпечення цілісності даних досягається шляхом

впровадження унікальних обмежень на комбінації атрибутів, що ідентифікують місця в межах конкретного сеансу, унеможливлюючи дублювання резервувань навіть за умов конкурентного доступу. У випадку порушення цих обмежень або відсутності відповідного сеансу система формує стандартизовані повідомлення про помилки.

Організація сховища даних передбачає використання взаємопов’язаних таблиць, структура яких ініціалізується автоматично під час першого запуску програмного забезпечення. Ініціалізація супроводжується створенням початкового набору тестових даних, що забезпечує можливість негайного тестування функціональних можливостей без попереднього ручного наповнення бази.

Архітектурна специфіка системи визначається поєднанням локального емулювання та серверної взаємодії, що дозволяє забезпечити гнучкість у процесі розробки та поступовий перехід до повністю централізованої моделі. Частина логіки, реалізована на клієнтському рівні із застосуванням механізмів локального збереження, використовується для моделювання поведінки системи, тоді як критичні операції, пов’язані з обробкою транзакцій, передаються на сервер для гарантованого виконання.

Ідентифікація окремих сутностей, зокрема сеансів, супроводжується використанням спеціалізованих позначень, що визначають необхідність звернення до серверного компонента при виконанні операцій резервування.

Сформований програмний комплекс, інтегруючи клієнтські інтерфейси, серверну логіку та механізми збереження даних, забезпечує автоматизацію процесів взаємодії між користувачами та системами кінопрокату, підтримуючи стабільність функціонування і створюючи передумови для подальшого розширення функціональних можливостей відповідно до змінних вимог середовища. Реалізована система характеризується модульністю та зручністю подальшого супроводу. Архітектурні рішення, використані під час розробки, забезпечують можливість інтеграції додаткових сервісів і масштабування системи без суттєвого ускладнення її структури.

1.2 Аналіз аналогів та обґрунтування технічного напряму розробки

Стан сучасного ринку вебзастосунків для онлайн-бронювання квитків і керування кіносеансами характеризується значною насиченістю програмними рішеннями, орієнтованими переважно на великі мережеві структури кінотеатрів, тоді як функціональні можливості рішень для локального або навчального використання залишаються обмеженими. У більшості випадків відсутня достатня гнучкість конфігурації, ускладнюється адаптація до специфічних вимог малих систем, а також не передбачається ефективна підтримка локального зберігання даних, що зумовлює необхідність критичного аналізу наявних підходів до проєктування подібних інформаційних систем.

Поширені комерційні платформи бронювання квитків у кінотеатрах реалізують комплекс функцій, що охоплює перегляд афіші, вибір місць у кінозалі, інтеграцію платіжних механізмів та збереження історії транзакцій, при цьому забезпечуючи централізоване управління даними та автоматизацію ключових операцій обслуговування користувачів [2]. Застосування подібних систем призводить до підвищення швидкодії обробки запитів і зменшення навантаження на офлайн-інфраструктуру, однак одночасно супроводжується ускладненням адміністрування, залежністю від хмарних сервісів і підвищенням витрат на технічну підтримку.

У паралельному контексті систем електронної комерції реалізовано підходи, спрямовані на оптимізацію обробки замовлень, контроль доступності ресурсів і запобігання дублюванню транзакцій, що набуває безпосередньої релевантності для систем кінотеатрального бронювання. При інтенсивній конкурентній взаємодії користувачів за обмежені ресурси, зокрема місця в межах одного сеансу, критичною стає коректна синхронізація станів системи та забезпечення унікальності кожної операції бронювання. Ефективність автоматизованих механізмів управління замовленнями, підтверджена результатами досліджень, визначається зниженням впливу людського фактора та підвищенням стабільності обробки транзакцій [3].

Ключовим елементом сучасних вебзастосунків виступає клієнтська частина, функціональні характеристики якої безпосередньо впливають на якість взаємодії користувача із системою. Використання бібліотеки React забезпечує побудову компонентно орієнтованої архітектури інтерфейсу, що підтримує повторне використання елементів, динамічне оновлення стану та ефективну обробку змін у режимі реального часу [4]. У межах сценаріїв перегляду сеансів, вибору місць і оформлення бронювання це дозволяє мінімізувати затримки інтерфейсу та забезпечити узгодженість відображення даних. Додаткове застосування TypeScript сприяє підвищенню структурної надійності коду, обмежуючи виникнення типових логічних помилок на етапі компіляції.

Серверний рівень реалізується із застосуванням Express.js, що забезпечує побудову REST API та обробку HTTP-запитів у відповідності до принципів модульної архітектури [5]. Через API здійснюється взаємодія між клієнтською частиною та базою даних, включаючи отримання переліку фільмів, формування розкладу сеансів, перевірку доступності місць і створення бронювань. Гнучкість маршрутизації та мінімалістична структура фреймворку дозволяють розширювати функціональність без суттєвого ускладнення програмної логіки.

Рівень зберігання даних доцільно реалізовувати на основі SQLite як компактної реляційної системи керування базами даних, що функціонує без необхідності окремого серверного розгортання та забезпечує транзакційну цілісність операцій [6]. Використання бібліотеки better-sqlite3 [7] дозволяє підвищити продуктивність взаємодії з базою даних та спростити реалізацію операцій читання і запису. Критичним елементом забезпечення цілісності даних виступає впровадження унікального обмеження для комбінації session_id, rownumber і seatnumber, що виключає можливість дублювання бронювань у межах одного сеансу.

Архітектурна модель, побудована на гібридному принципі, передбачає поєднання локального шару емулювання з використанням localStorage та SessionsContext із серверною взаємодією через API та SQLite. Така структура дозволяє розмежувати функціональні рівні відповідальності, забезпечуючи

паралельне існування швидкого прототипування інтерфейсної частини та стабільної серверної логіки обробки критичних операцій. Використання локального шару сприяє прискоренню розробки мінімально життєздатного продукту, тоді як серверна складова формує основу для переходу до повноцінної виробничої системи з гарантованою цілісністю даних.

Аналіз існуючих рішень демонструє наявність структурних обмежень, пов’язаних із надмірною складністю комерційних систем або недостатнім функціональним наповненням спрощених аналогів, що у сукупності формує потребу у розробленні збалансованого програмного продукту. Формування власного вебзастосунку, що поєднує локальну базу даних, клієнт-серверну архітектуру, механізми контролю бронювання та адаптивний інтерфейс, дозволяє забезпечити оптимізацію співвідношення між функціональністю, простотою використання та технічною надійністю системи.

1.3 Вибір та аналіз інструментарію проектування системи

Проєктування архітектури вебзастосунку для онлайн-кінотеатру та системи бронювання квитків визначається необхідністю узгодження вимог до продуктивності, масштабованості, супровідності програмного коду та забезпечення цілісності даних, що обробляються в умовах конкурентного доступу користувачів. Оскільки предметна область передбачає роботу з реальними бронюваннями, критичним стає контроль стану місць у залі, коректна синхронізація транзакцій та мінімізація затримок при оновленні інтерфейсу, що безпосередньо впливає на консистентність відображуваної інформації.

Реалізація клієнтської частини із застосуванням React ґрунтується на компонентному підході, за якого інтерфейс декомпозується на незалежні логічні блоки, що взаємодіють через механізми керування станом застосунку [4]. Така модель забезпечує інкрементальне оновлення DOM без повного перезавантаження сторінки, що є суттєвим чинником для сценаріїв, пов’язаних із динамічним вибором місць і відображенням доступності сеансів. Додаткове

використання TypeScript, запроваджуючи статичну типізацію, зменшує ймовірність виникнення помилок на етапі компіляції, особливо в умовах складної взаємодії між UI-компонентами та бізнес-логікою.

Серверна частина побудована на основі Express.js, що забезпечує реалізацію REST API з мінімальними накладними витратами та високою гнучкістю конфігурації [5]. Через визначені маршрути здійснюється обробка запитів на отримання списків фільмів і сеансів, перевірку доступності місць, а також ініціалізацію процесів бронювання. Архітектурна простота фреймворку дозволяє інтегрувати додаткові модулі без суттєвого ускладнення структури застосунку, що є критичним при поступовому розширенні функціональності.

Рівень зберігання даних реалізується із застосуванням SQLite як вбудованої реляційної системи керування базами даних, що функціонує без необхідності окремого серверного розгортання та забезпечує підтримку транзакційної обробки [6]. Інтеграція через бібліотеку better-sqlite3 дозволяє досягти підвищеної продуктивності завдяки синхронній моделі доступу до даних і зменшенню накладних витрат, характерних для асинхронних рішень [7]. Ключовим елементом забезпечення цілісності даних виступає запровадження унікального обмеження на комбінацію session_id, rownumber і seatnumber, що виключає можливість дублювання бронювань у межах одного сеансу навіть за умов паралельного доступу.

Архітектурна модель системи реалізує клієнт-серверну взаємодію через REST API, що відповідає загальноприйнятим принципам побудови сучасних вебзастосунків і забезпечує чіткий поділ відповідальності між рівнями системи [8]. На клієнтській стороні додатково використовується локальний стан і механізми тимчасового збереження даних, що зменшує кількість мережевих запитів і підвищує швидкодію інтерфейсу під час взаємодії користувача із системою [9].

Комбінований підхід до організації зберігання та обробки даних передбачає одночасне використання локального шару localStorage та контекстів стану і серверної частини, яка відповідає за критичні операції бронювання. Така

схема дозволяє поєднати швидкість прототипування інтерфейсних рішень із надійністю централізованого збереження транзакцій, що є характерним для сучасних гібридних систем класу ticketing та e-commerce [2].

Додатковим механізмом забезпечення цілісності виступає обробка конфліктних ситуацій на рівні API, де у випадку спроби повторного бронювання одного й того самого місця повертається код помилки 409, що узгоджується з принципами REST та HTTP-стандартами обробки конфліктів стану ресурсу. Такий підхід дозволяє формалізувати реакцію системи на порушення обмежень цілісності даних і забезпечує передбачувану поведінку клієнтської частини.

Розширення функціональних можливостей, зокрема через підтримку багатомовності інтерфейсу на основі підходу i18n, підвищує адаптивність системи до різних груп користувачів і відповідає сучасним вимогам до UX/UI-рішень, у межах яких пріоритетними визначаються локалізація, доступність та узгодженість взаємодії з інтерфейсом [1]. У сукупності запропонована архітектура, поєднуючи легкий серверний шар, компонентний фронтенд і вбудоване сховище даних, формує структурно цілісну основу для побудови продуктивного та масштабованого вебзастосунку з можливістю подальшого розширення функціональності.

1.4 Завдання на кваліфікаційну роботу бакалавра

Мета кваліфікаційної роботи полягає у розробленні вебзастосунку для онлайн-кінотеатру та системи бронювання квитків, що забезпечує організовану взаємодію користувача з розкладом кінопоказів, механізмом вибору місць у кінозалі та процедурою оформлення бронювання в режимі реального часу. У процесі реалізації передбачається створення програмного рішення, структурно поділеного на клієнтську і серверну складові, доповнені локальним сховищем даних, що використовується для забезпечення стабільності функціонування в умовах обмежених ресурсів та під час реалізації MVP-версії.

Основним завданням визначається аналіз предметної області систем онлайн-бронювання з подальшою формалізацією функціональних вимог до програмного забезпечення. У сучасних інформаційних системах електронної комерції, де обробляються транзакційні дані великої кількості користувачів, критичними виступають швидкість виконання запитів, надійність збереження інформації та забезпечення цілісності даних при конкурентному доступі до спільних ресурсів. У зв’язку з цим окрему увагу приділено реалізації механізмів контролю унікальності бронювань, а також обробці конфліктних ситуацій, що виникають у разі повторного резервування вже зайнятих місць.

Проєктування архітектури системи здійснюється на основі клієнт-серверної моделі, що передбачає чіткий розподіл відповідальності між рівнями обробки даних і взаємодії з користувачем. Клієнтська частина реалізується із застосуванням сучасних вебтехнологій, орієнтованих на побудову динамічних інтерфейсів із можливістю оперативного оновлення стану без перезавантаження сторінки. Використання React забезпечує компонентну організацію інтерфейсу, за якої окремі функціональні блоки ізольовано реалізують відповідні частини UI, що підвищує масштабованість і спрощує супровід програмного продукту [4]. Додаткове застосування TypeScript, запроваджуючи статичну типізацію, зменшує ймовірність виникнення логічних помилок у процесі розробки та підвищує структурну узгодженість коду [10].

Серверна частина системи реалізується на основі Express.js, що забезпечує побудову REST API з ефективною маршрутизацією та обробкою HTTP-запитів [5]. У межах бекенд-логіки виконується обробка запитів, пов’язаних із фільмами, сеансами та бронюваннями, а також реалізуються механізми перевірки доступності місць і забезпечення цілісності даних при одночасному доступі декількох користувачів. Архітектурна простота фреймворку дозволяє розширювати функціональність системи без суттєвого ускладнення її структурної організації.

Проєктування структури бази даних передбачає використання SQLite як вбудованої реляційної системи керування даними, що функціонує без

необхідності окремого серверного розгортання та забезпечує підтримку транзакційної обробки. Застосування бібліотеки better-sqlite3 забезпечує підвищену продуктивність за рахунок синхронного доступу до даних і оптимізації операцій читання та запису [11].

У структурі сховища передбачено збереження інформації про фільми, кінозали, сеанси та бронювання, що формує цілісну модель предметної області. Додатково реалізується механізм унікальності комбінації session_id, rownumber та seatnumber, що виключає можливість дублювання бронювань одного й того самого місця.

Реалізація бізнес-логіки бронювання передбачає виконання послідовності операцій, пов’язаних із вибором місць, перевіркою їх доступності, створенням запису бронювання та зміною його статусу після завершення транзакції. Поділ бронювань на стани RESERVED та PAID забезпечує структуроване відображення життєвого циклу замовлення та дозволяє організувати історію користувацьких операцій. Така модель відповідає типовим архітектурним рішенням сучасних систем електронної комерції та сервісів онлайн-бронювання [1].

Окремо реалізується гібридна архітектура, що поєднує локальний стан застосунку з серверною взаємодією через API. Частина даних зберігається у локальному середовищі для забезпечення швидкого доступу та спрощення тестування, тоді як операції, що мають критичне значення для цілісності системи, виконуються через серверний рівень. Такий підхід дозволяє поєднати високу швидкість взаємодії користувача з надійністю централізованого зберігання даних [3].

У межах кваліфікаційної роботи необхідно розробити повноцінний вебзастосунок, який забезпечує перегляд списку фільмів і сеансів, вибір місць у кінозалі, оформлення та збереження бронювання, контроль доступності місць, ведення історії бронювань користувача, а також стабільну взаємодію клієнта з сервером через REST API. Реалізація такого програмного продукту передбачає створення багаторівневої клієнт-серверної архітектури, у межах якої фронтенд

частина відповідає за взаємодію користувача із системою, а серверна частина забезпечує обробку запитів, перевірку коректності даних та взаємодію з базою даних. Особливу увагу необхідно приділити механізмам синхронізації стану місць у режимі реального часу, оскільки одночасна робота декількох користувачів із одним кіносеансом створює ризик конфліктів при бронюванні.

Функціональні можливості системи повинні охоплювати повний цикл взаємодії користувача із сервісом онлайн-бронювання. Користувач має отримати можливість переглядати афішу доступних фільмів, інформацію про жанри, тривалість та час проведення сеансів, а також здійснювати навігацію між окремими сторінками застосунку без перезавантаження інтерфейсу. Після вибору конкретного сеансу система повинна відображати структуру кінозалу із зазначенням доступних та зайнятих місць, що забезпечує наочність процесу бронювання та спрощує взаємодію користувача з інтерфейсом.

Оформлення бронювання передбачає реалізацію механізмів перевірки доступності місць перед збереженням даних у базі. Для цього серверна частина повинна виконувати перевірку унікальності кожного бронювання та блокувати можливість повторного резервування одного й того самого місця. Важливим аспектом є також реалізація історії бронювань користувача, яка дозволяє відображати інформацію про попередні замовлення, їх поточний статус та деталі оформлених квитків. Подібний функціонал відповідає сучасним вимогам до сервісів електронної комерції та систем онлайн-обслуговування користувачів.

Для забезпечення стабільної взаємодії між клієнтською та серверною частинами застосунку необхідно реалізувати REST API, яке забезпечуватиме обмін даними у стандартизованому форматі. Через API повинні виконуватись операції отримання списків фільмів і сеансів, надсилання запитів на створення бронювання, перевірка доступності місць та отримання інформації про історію операцій користувача.

Такий підхід дозволяє чітко розмежувати функції окремих рівнів системи та забезпечити можливість подальшого масштабування програмного продукту.Результатом виконання роботи має стати функціональний

вебзастосунок, який демонструє практичне застосування сучасного frontend та backend-стеку, а також ефективну інтеграцію з базою даних. Розроблений програмний продукт повинен забезпечувати стабільну роботу основних функціональних модулів, високу швидкодію інтерфейсу та коректну обробку запитів користувачів у режимі реального часу.

У межах реалізації застосунку передбачається використання сучасних технологій веброзробки, що дозволяють створити масштабовану та зручну систему із можливістю подальшого розширення функціоналу.Практичний результат роботи має підтвердити ефективність використання компонентного підходу до побудови інтерфейсу, принципів клієнт-серверної взаємодії та реляційної моделі збереження даних. Реалізована система повинна демонструвати можливість інтеграції сучасних засобів управління станом застосунку, механізмів асинхронної обробки HTTP-запитів та технологій локального збереження даних.

Додатково створений програмний комплекс має формувати основу для подальшої модернізації, включаючи інтеграцію платіжних сервісів, впровадження розширених механізмів авторизації користувачів, аналітичних модулів і засобів адміністрування системи.

РОЗДІЛ 2
СПЕЦИФІКАЦІЯ ВИМОГ ДО РОЗРОБЛЮВАНОЇ СИСТЕМИ

2.1 Обґрунтування вибору шляхів, технологій (алгоритмів) і засобів вирішення поставленого завдання

Під час проєктування вебзастосунку для онлайн-кінотеатру та системи бронювання квитків визначальним чинником виступає обґрунтований вибір технологічного стеку, узгодженого з вимогами до продуктивності, масштабованості, супровідності програмного коду та стабільності функціонування системи. З урахуванням динамічного характеру предметної області, пов’язаної з постійною зміною станів сеансів, місць і бронювань, критичною є організація ефективної взаємодії між клієнтською та серверною частинами, а також забезпечення консистентності збережених даних.

Реалізація клієнтського рівня здійснюється із застосуванням React, що забезпечує компонентний підхід до побудови інтерфейсу, у межах якого логічно ізольовані модулі взаємодіють через контрольований механізм стану застосунку [4]. Використання віртуального DOM дозволяє мінімізувати кількість безпосередніх операцій з реальним DOM-деревом, що суттєво підвищує продуктивність рендерингу у сценаріях інтенсивної взаємодії користувача із системою, зокрема під час вибору місць або оновлення переліку бронювань. Додаткове застосування TypeScript, базоване на статичній типізації, зменшує кількість помилок компіляційного та логічного характеру, підвищуючи передбачуваність поведінки компонентів у складних інтерфейсних структурах.

Серверна частина реалізується на основі Express.js, який функціонує як легковаговий фреймворк для побудови REST API та забезпечує ефективну маршрутизацію HTTP-запитів [5].

У межах цієї архітектури формується централізований рівень обробки бізнес-логіки, що охоплює операції з фільмами, сеансами та бронюваннями. Мінімалістична структура Express.js дозволяє зменшити накладні витрати на

конфігурацію та забезпечує гнучкість при подальшому розширенні функціональності системи, що є характерним для MVP-орієнтованих рішень.

Рівень збереження даних реалізується на основі SQLite як вбудованої реляційної системи керування базами даних, що не потребує окремого серверного розгортання та забезпечує автономність функціонування застосунку. Такий підхід є доцільним для навчальних і малих виробничих систем, оскільки поєднує простоту інтеграції з достатнім рівнем продуктивності та підтримкою транзакційної обробки [9].

Архітектурна організація системи базується на клієнт-серверній моделі, у межах якої фронтенд-компонент взаємодіє з бекендом через стандартизований REST API, що забезпечує уніфікований підхід до обміну даними та розмежування відповідальності між рівнями системи [10]. Така структурна декомпозиція дозволяє ізолювати бізнес-логіку від інтерфейсного шару, спрощуючи супровід, тестування та подальше масштабування програмного продукту. Загальна схема взаємодії компонентів представлена на рисунку 2.1.

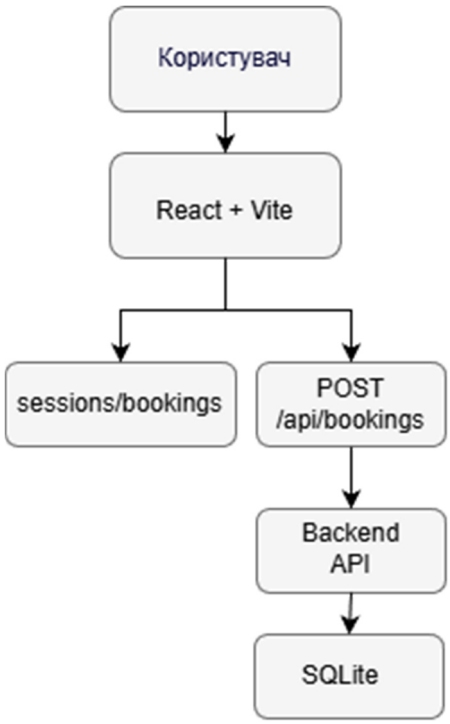

Рисунок 2.1 – Загальна архітектура системи взаємодії клієнта, сервера та бази даних

Джерело: розроблено автором

Вибір технологічного стеку детермінується не лише функціональними характеристиками окремих компонентів, а й рівнем їхньої підтримки у професійній спільноті, обсягом доступної технічної документації та стабільністю застосування в умовах промислової розробки програмного забезпечення. У цьому контексті React і Express.js розглядаються як усталені інструменти сучасної вебінженерії, що системно застосовуються при побудові масштабованих інформаційних систем і підтверджуються практикою реалізації комерційних та високонавантажених сервісів [12].

Застосування обраного набору технологій забезпечує узгоджену реалізацію функціональних вимог розроблюваної системи, створюючи архітектурно цілісне середовище для побудови продуктивного вебзастосунку з прогнозованою поведінкою компонентів. Структурна сумісність клієнтської та серверної частин, доповнена використанням реляційної бази даних, формує основу для стабільного виконання операцій бронювання, обробки запитів та управління станами сутностей у межах єдиної інформаційної моделі.

Подальше розширення функціональності системи може здійснюватися без необхідності фундаментальної зміни архітектури, оскільки закладена модульна структура дозволяє інтеграцію додаткових підсистем, зокрема платіжних сервісів, механізмів автентифікації користувачів та аналітичних модулів для обробки даних бронювань. Така еволюційна масштабованість забезпечується розділенням відповідальності між рівнями системи та використанням стандартизованого API-взаємодії, що відповідає сучасним практикам побудови веборієнтованих інформаційних платформ.

2.2 Структурно-функціональна організація об’єкта проєктування та опис алгоритмів обробки

Структурно-функціональна організація розроблюваної системи онлайн-кінотеатру та бронювання квитків базується на класичній клієнт-серверній архітектурі, яка широко використовується у сучасних інформаційних системах

завдяки своїй масштабованості, модульності та простоті супроводу [3]. Такий підхід дозволяє чітко розділити відповідальність між інтерфейсом користувача, бізнес-логікою та рівнем збереження даних, що відповідає принципам побудови сучасних вебзастосунків [10].

Фронтенд частина системи реалізує взаємодію користувача із застосунком та відповідає за відображення інформації про фільми, сеанси, доступні місця та стан бронювання. Для побудови інтерфейсу використовується компонентна модель React, яка дозволяє розбити інтерфейс на незалежні логічні блоки та забезпечити повторне використання компонентів [8]. Така структура значно спрощує підтримку та розширення функціоналу системи, оскільки зміни в одному компоненті не впливають на інші частини інтерфейсу.

Серверна частина системи побудована на основі Express.js, яка реалізує REST API (рис. 2.2) для взаємодії з клієнтом [5]. Вона виконує обробку запитів, пов’язаних із отриманням списку фільмів і сеансів, перевіркою зайнятих місць та створенням нових бронювань. Логіка сервера відповідає принципам REST-архітектури, що передбачає використання стандартних HTTP-методів для роботи з ресурсами [13].

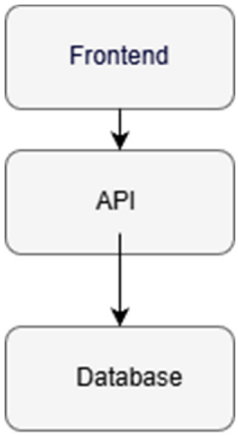

Рисунок 2.2 – Структурно-функціональна схема системи

Джерело: розроблено автором

Рівень збереження даних реалізовано на основі SQLite, яка функціонує як компактна вбудована реляційна система керування базами даних і не потребує

окремого серверного розгортання [6]. У межах системи вона використовується для структурованого зберігання інформації про фільми, кінозали, сеанси та бронювання, забезпечуючи цілісність даних і підтримку транзакційної обробки. Завдяки внутрішній архітектурі SQLite досягається стабільність виконання операцій читання та запису, що є критично важливим у сценаріях конкурентного доступу до спільних ресурсів. Оптимізація взаємодії з базою даних реалізується через бібліотеку better-sqlite3, яка забезпечує синхронну модель доступу до даних і підвищену продуктивність виконання SQL-запитів порівняно з асинхронними альтернативами [7].

Загальна структурна організація системи передбачає трирівневу модель, що включає клієнтський рівень, серверний рівень та рівень збереження даних. На клієнтському рівні здійснюється формування запитів користувача та відображення актуального стану системи, тоді як серверний рівень виконує обробку бізнес-логіки, включаючи валідацію запитів, управління бронюваннями та координацію взаємодії з базою даних. Рівень збереження даних забезпечує довготривале зберігання інформації та гарантує її цілісність у межах транзакційної моделі. Обмін даними між рівнями реалізується через HTTP-запити відповідно до принципів REST API, що відповідає сучасним підходам до побудови розподілених інформаційних систем .

Алгоритм функціонування системи бронювання формалізується як послідовність взаємопов’язаних операцій, що ініціюються отриманням користувачем переліку доступних фільмів і сеансів. Після вибору конкретного сеансу здійснюється звернення до серверного рівня для отримання актуального стану зайнятості місць, що забезпечує синхронізацію клієнтського представлення з фактичними даними системи. Подальший етап передбачає вибір вільних місць користувачем та формування запиту на створення бронювання, який передається на сервер для обробки. На серверному рівні виконується перевірка доступності обраних місць із подальшим створенням запису в базі даних у разі відсутності конфліктів. У випадку виявлення зайнятості хоча б одного місця ініціюється повернення помилки, що унеможливлює порушення

цілісності даних і дублювання бронювань. Така модель узгоджується з принципами транзакційної обробки інформації та забезпечення консистентності стану системи [14].

Після успішного створення бронювання йому присвоюється початковий статус RESERVED, що відображає його проміжний стан до завершення платіжної операції. Подальша зміна статусу на PAID здійснюється після підтвердження виконання умов оплати, що реалізується на рівні клієнтської логіки та синхронізується із серверною частиною. Такий механізм дозволяє формалізувати життєвий цикл бронювання та забезпечити чітке розмежування між активними і завершеними операціями, що відповідає практикам побудови систем електронної комерції та онлайн-сервісів бронювання [2].

Окремий аспект функціонування системи пов’язаний із синхронізацією стану між локальним сховищем і серверною частиною. Частина даних тимчасово зберігається на стороні клієнта з метою зменшення кількості мережевих запитів і підвищення швидкодії інтерфейсу, тоді як усі критичні операції обов’язково підтверджуються сервером. Такий підхід дозволяє поєднати оперативність взаємодії з користувачем із гарантією коректності та узгодженості даних, що зберігаються в централізованій базі [15].

Узагальнена структурно-функціональна організація системи забезпечує чіткий розподіл відповідальності між рівнями обробки даних, стабільність виконання операцій та надійність збереження інформації. Використання поєднання React, Express.js та SQLite формує технологічну основу, що дозволяє реалізувати продуктивний, масштабований і структурно узгоджений вебзастосунок із можливістю подальшого розширення функціональності.

2.3 Проєктування та фізична реалізація моделі бази даних

Проєктування бази даних є одним із визначальних етапів розробки системи онлайн-кінотеатру та бронювання квитків, оскільки саме на цьому рівні формується структурована модель зберігання інформації, від якої залежить

коректність виконання транзакцій, цілісність даних та можливість подальшого масштабування програмного продукту. У межах інформаційної системи забезпечення узгодженості даних набуває особливого значення через наявність конкурентного доступу до спільних ресурсів, зокрема місць у кінозалі.

Концептуальна модель базується на принципах реляційної організації даних, що передбачає подання інформації у вигляді взаємопов'язаних таблиць із чітко визначеними зв'язками між ними. Такий підхід дозволяє нормалізувати структуру даних, мінімізувати надлишковість та забезпечити логічну узгодженість між сутностями, що особливо важливо для транзакційно-орієнтованих систем [3]. Реляційна модель у цьому випадку виступає основою для формалізації предметної області та забезпечує передбачувану поведінку при виконанні операцій додавання, оновлення та вибірки даних.

У процесі аналізу предметної області виділено основні сутності, що відображають ключові об'єкти системи: фільми, кінозали, сеанси та бронювання. Кожна сутність реалізується у вигляді окремої таблиці, що містить набір атрибутів, необхідних для її однозначного опису та подальшої взаємодії з іншими компонентами моделі даних. Таблиця фільмів використовується для збереження інформації про кінематографічні твори, включаючи базові характеристики, що ідентифікують контент. Таблиця кінозалів формалізує параметри фізичних або логічних залів, зокрема їх місткість та конфігурацію. Таблиця сеансів забезпечує зв'язок між фільмами та залами, визначаючи часові інтервали показів. Таблиця бронювань фіксує результати взаємодії користувача із системою, включаючи обрані місця, статуси операцій та посилання на відповідні сеанси.

Фізична реалізація бази даних здійснюється із застосуванням SQLite як вбудованої реляційної системи керування даними, що функціонує без необхідності окремого серверного розгортання та забезпечує автономність роботи застосунку [6]. Така архітектурна особливість сприяє спрощенню процесу розгортання та зменшенню залежностей від зовнішньої інфраструктури, що є доцільним для навчальних і MVP-рішень. Взаємодія з базою даних

реалізується через бібліотеку better-sqlite3, яка забезпечує синхронну модель виконання SQL-запитів і підтримку транзакційної обробки, що позитивно впливає на продуктивність операцій читання та запису [16].

Окрему увагу приділено забезпеченню цілісності даних на рівні фізичної моделі. У таблиці бронювань запроваджено унікальне обмеження, що охоплює комбінацію ідентифікатора сеансу, номера ряду та номера місця, що унеможливлює виникнення ситуацій подвійного резервування одного й того самого місця різними користувачами.

Узагальнена структура бази даних, сформована на основі реляційної моделі (рис. 2.3) та підтримувана вбудованою СУБД, забезпечує узгоджену взаємодію між сутностями предметної області, стабільність транзакційних операцій та можливість подальшого розширення інформаційної моделі без порушення існуючих зв’язків.

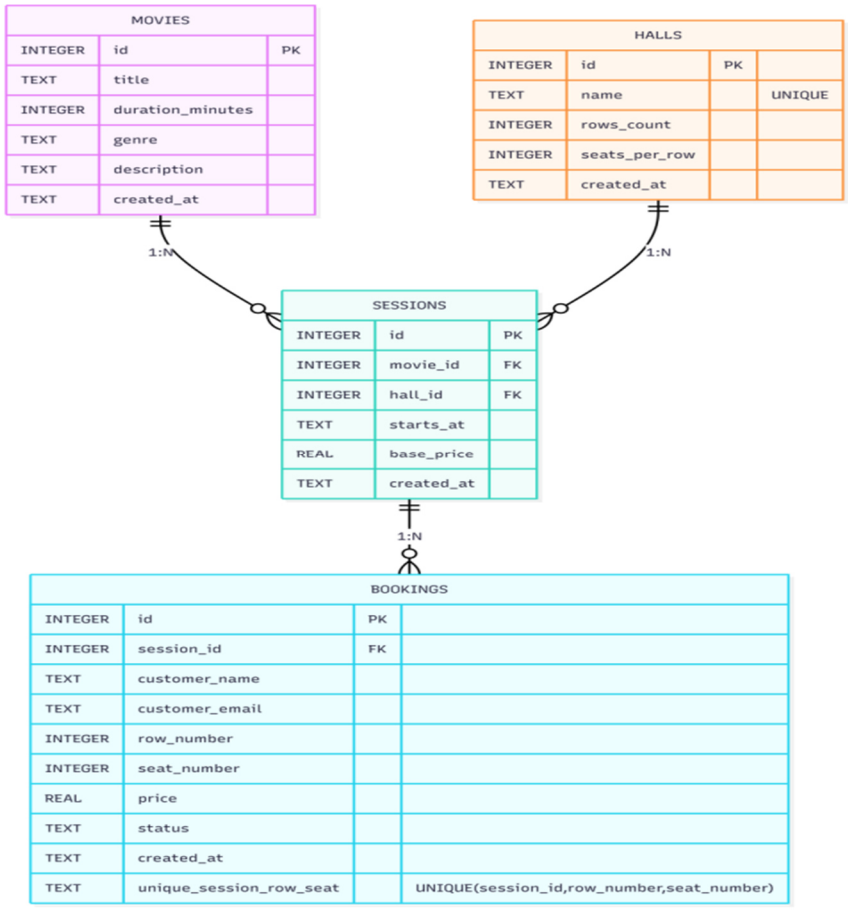

Рисунок 2.3 – ER-діаграма бази даних системи

Джерело: розроблено автором

Логічна структура бази даних сформована відповідно до ієрархічної моделі взаємозв’язків між сутностями, що забезпечує відображення особливостей організації предметної області та ефективну взаємодію між основними компонентами системи, у якій один фільм може бути пов’язаний із множиною сеансів, тоді як кожен окремий сеанс агрегує множину бронювань, сформованих користувачами системи. Така модель забезпечує узгоджене представлення даних, мінімізує надлишковість інформації та створює передумови для ефективного виконання SQL-запитів, спрямованих на вибірку, фільтрацію та агрегацію даних у межах визначених зв’язків між сутностями.

Фізична реалізація бази даних організована таким чином, що її ініціалізація здійснюється автоматично під час запуску застосунку на основі підготовлених SQL-скриптів. У процесі ініціалізації формується структура таблиць, визначаються первинні та зовнішні ключі, а також встановлюються обмеження цілісності, що регламентують взаємозв’язки між сутностями. Додатково виконується заповнення початковими тестовими даними (seed data), що дозволяє забезпечити готовність системи до функціонального тестування без необхідності попереднього ручного конфігурування або наповнення сховища інформації.

Узагальнено обрана модель організації бази даних характеризується поєднанням структурної простоти та достатнього рівня функціональної повноти, що є доцільним для реалізації навчальних систем і рішень рівня MVP. Використання SQLite у поєднанні з Node.js дозволяє уникнути необхідності розгортання окремої серверної інфраструктури, одночасно зберігаючи підтримку ключових властивостей реляційних баз даних, зокрема транзакційної обробки, забезпечення цілісності даних та виконання стандартизованих запитів .

РОЗДІЛ 3
РОЗРОБКА ВЕБЗАСТОСУНКУ ОНЛАЙН-КІНОТЕАТРУ ТА СИСТЕМИ БРОНЮВАННЯ КВИТКІВ

3.1 Практична реалізація об’єкта проектування. Побудова блок-схем модулів програми

Програмне забезпечення вебкласу, орієнтоване на організацію онлайн-кінотеатру та системи бронювання квитків, потребує чіткої структуризації всіх функціональних компонентів для забезпечення стабільної роботи, масштабованості та зручності подальшого супроводу. Оскільки система працює з динамічними даними, такими як розклад сеансів, доступність місць, історія бронювань та обробка замовлень, особливо важливим є правильний розподіл відповідальності між інтерфейсною частиною, серверною логікою та рівнем збереження даних.

Для реалізації проєкту було обрано сучасний клієнт-серверний підхід, який дозволяє розділити систему на незалежні функціональні рівні. Така архітектура забезпечує зручність розробки, спрощує тестування окремих модулів та дозволяє поступово розширювати функціонал без суттєвого перепроєктування всієї системи. Основу клієнтської частини становить React із використанням TypeScript, що дозволяє реалізувати гнучкий, швидкий та адаптивний інтерфейс користувача. Серверна частина побудована на платформі Express.js, яка забезпечує обробку API-запитів, бізнес-логіку бронювання та взаємодію з базою даних. Для локального збереження інформації використовується SQLite через бібліотеку better-sqlite3, що дозволяє забезпечити стабільну роботу системи без необхідності окремого серверного середовища баз даних.

Загальна архітектура системи побудована за принципом поділу відповідальності між трьома основними рівнями: користувацьким інтерфейсом, логічним шаром обробки та рівнем збереження даних. Такий підхід дозволяє ізолювати кожну частину системи, що значно зменшує складність супроводу та покращує контроль над програмною логікою.

Верхній рівень представлений frontend-частиною, яка відповідає за безпосередню взаємодію з користувачем. До цього рівня входять сторінки перегляду фільмів, списку сеансів, оформлення бронювання, checkout та розділ перегляду історії замовлень. Особливістю реалізації є використання компонентного підходу, при якому кожна функціональна частина інтерфейсу реалізується окремим незалежним компонентом. Це дозволяє повторно використовувати елементи інтерфейсу, зменшувати дублювання коду та спрощувати внесення змін у майбутньому.

Система навігації між сторінками реалізована без повного перезавантаження сторінки, що забезпечує високу швидкість відгуку інтерфейсу та комфортну взаємодію користувача із застосунком (рис. 3.1). Користувач може швидко переглядати доступні фільми, переходити до вибору сеансу, обирати місця в залі та завершувати оформлення замовлення без затримок і втрати стану системи. На цьому ж рівні виконується первинна валідація введених даних, зокрема перевірка коректності платіжної інформації та правильності введення персональних даних.

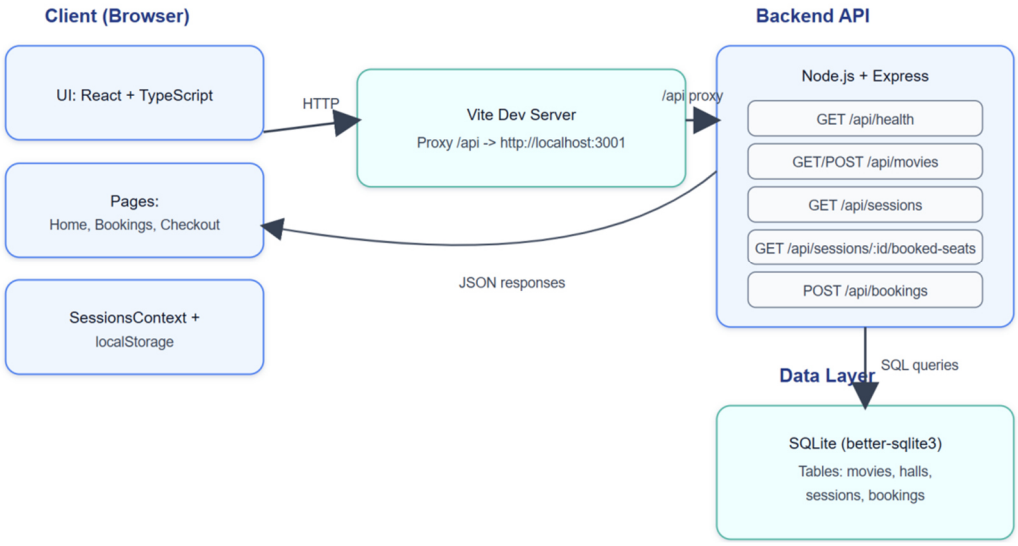

Рисунок 3.1 – Загальна архітектура взаємодії компонентів системи

Джерело: розроблено автором

Середній рівень виконує роль центральної сполучної ланки та містить основну бізнес-логіку системи. Саме тут відбувається обробка запитів користувача, перевірка доступності місць, формування бронювань та контроль правильності виконання операцій. Серверна частина приймає HTTP-запити від клієнта, обробляє їх та повертає відповідь у форматі JSON. Такий підхід дозволяє забезпечити чіткий поділ між візуальною частиною системи та логікою роботи з даними.

Особливо важливою є реалізація алгоритму перевірки зайнятості місць. Перед створенням нового бронювання система виконує перевірку, чи не було вже заброньовано обране місце іншим користувачем. Якщо місце вже зайняте, сервер повертає відповідне повідомлення про помилку, що унеможливлює подвійне резервування. Такий механізм є критично важливим для забезпечення коректної роботи всієї системи бронювання.

Окремо реалізовано логіку поділу бронювань за статусами. Після вибору місця користувач отримує бронювання зі статусом RESERVED, а після завершення процесу оплати бронювання змінює свій стан на PAID. Це дозволяє системі вести повну історію замовлень та надавати користувачеві можливість переглядати як активні, так і завершені бронювання.

Нижній рівень системи відповідає за постійне збереження інформації та забезпечує роботу з локальною базою даних. Для цього використовується SQLite, яка є вбудованою реляційною системою керування базами даних.

Її використання є доцільним для MVP-проєктів та навчальних систем, оскільки вона не потребує окремого адміністрування, має високу швидкість роботи та забезпечує надійне збереження інформації.

У структурі бази даних реалізовано чотири основні таблиці: movies, halls, sessions та bookings. Таблиця movies містить інформацію про фільми, таблиця halls відповідає за зали кінотеатру, sessions зберігає дані про конкретні сеанси, а bookings фіксує всі створені бронювання. Така структура дозволяє чітко розділити сутності предметної області та забезпечити логічну узгодженість усіх операцій.

Для підвищення безпеки роботи з базою даних усі SQL-запити виконуються через підготовлені механізми взаємодії, що знижує ризик помилок та забезпечує стабільну роботу серверної частини. Додатково реалізовано автоматичне створення таблиць при першому запуску застосунку, а також початкове заповнення бази тестовими даними. Це дозволяє запускати систему без додаткового ручного налаштування та значно спрощує процес тестування.

Особливістю проєкту є також гібридна модель збереження даних, де частина інформації працює через локальний mock-шар на frontend-рівні, а критичні операції бронювання виконуються через реальну взаємодію з сервером і базою даних. Такий підхід дозволяє одночасно забезпечити швидкість розробки та надійність збереження ключових даних.

Таким чином, розроблена архітектура системи поєднує сучасний підхід до побудови вебзастосунків, чіткий розподіл відповідальності між рівнями системи та ефективну модель взаємодії між користувачем, сервером і базою даних. Це створює надійну програмну основу для стабільної роботи онлайн-кінотеатру та подальшого розширення функціональних можливостей системи.

Автоматична перевірка наявності таблиць особливо актуальна в умовах багаторазового запуску серверної частини, коли база даних уже може містити необхідну інформацію. Завдяки цьому система зберігає цілісність раніше записаних даних та не потребує ручного втручання адміністратора для повторного налаштування структури зберігання. Це суттєво спрощує процес підтримки програмного продукту та зменшує ймовірність технічних помилок під час його експлуатації.

Окрім створення самої структури таблиць, у лістингу 3.1 також реалізовано механізм первинного заповнення бази даних, відомий як seed-ініціалізація.

Його принцип полягає в перевірці вмісту таблиць після їх створення: якщо система визначає, що таблиці порожні, до них автоматично додаються стартові записи. Це можуть бути початкові облікові записи користувачів,

адміністраторські профілі, базові категорії фільмів, перші кіносеанси або інші службові дані, необхідні для коректного функціонування вебзастосунку.

Наявність такого механізму значно спрощує перший запуск системи після її розгортання, оскільки застосунок стає готовим до роботи без необхідності ручного наповнення бази даних. Адміністратору не потрібно самостійно створювати початкові записи або виконувати додаткові SQL-запити, що особливо зручно під час тестування, демонстрації функціоналу та розгортання програмного забезпечення на новому сервері чи локальному середовищі розробки.

У лістингу 3.1 наведено технічну реалізацію процесу початкової ініціалізації локальної бази даних SQLite, яка виконується автоматично під час запуску серверної частини програмного застосунку. У фрагменті коду чітко видно використання механізму створення таблиць за допомогою SQL-конструкції CREATE TABLE IF NOT EXISTS, що дозволяє системі безпечно перевіряти наявність необхідних структур даних перед їх створенням. Такий підхід є важливим для забезпечення стабільної роботи програмного забезпечення, оскільки запобігає виникненню помилок, пов’язаних із повторним створенням уже існуючих таблиць, а також виключає ризик дублювання службової інформації.

Лістинг 3.1 – Програмне створення та початкова ініціалізація таблиць БД

```
import Database from "better-sqlite3";
db.pragma("foreign_keys = ON");
db.exec(`
  CREATE TABLE IF NOT EXISTS movies (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    duration_minutes INTEGER NOT NULL CHECK(duration_minutes > 0),
    genre TEXT,
    description TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
  );
  CREATE TABLE IF NOT EXISTS halls (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL UNIQUE,
    rows_count INTEGER NOT NULL CHECK(rows_count > 0),
    seats_per_row INTEGER NOT NULL CHECK(seats_per_row > 0),
```

```
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
  );
export default db;
```

кінець лістингу 3.1

Таким чином, представлений підхід до автоматизованої ініціалізації SQLite-бази даних забезпечує не лише створення необхідної структури зберігання інформації, але й формує базове середовище для стабільної та безперервної роботи всієї системи. Це підвищує надійність програмного продукту, покращує зручність його адміністрування та сприяє ефективному супроводу програмного забезпечення на всіх етапах його життєвого циклу.

У фрагменті коду також визначається порт запуску сервера з використанням змінних середовища, що забезпечує гнучкість конфігурації застосунку в різних середовищах розгортання. Такий підхід дозволяє адаптувати систему як до локального середовища розробки, так і до тестових або продуктивних серверів без внесення змін у вихідний код, що відповідає сучасним принципам конфігураційної незалежності програмного забезпечення.

Окрему увагу слід звернути на підключення middleware cors(), який реалізує механізм Cross-Origin Resource Sharing і забезпечує безпечну взаємодію між фронтенд- та бекенд-частинами застосунку, навіть у випадку їх розміщення на різних доменах або портах. Це є особливо важливим у процесі розробки вебзастосунків, оскільки дозволяє уникнути обмежень браузера, пов’язаних із політикою однакового походження (Same-Origin Policy).

Крім того, у конфігурації використовується middleware express.json(), який забезпечує автоматичний розбір вхідних HTTP-запитів із JSON-тілом. Це суттєво спрощує обробку даних, що надходять від клієнта, та є необхідним для реалізації операцій створення, оновлення та передачі структурованих даних, таких як користувацькі профілі, фільми або бронювання квитків.

У лістингу 3.2 наведено технічну реалізацію первинної конфігурації серверної частини застосунку, яка є ключовим етапом ініціалізації REST API та формування базової архітектури взаємодії між клієнтською і серверною

складовими системи. На цьому етапі здійснюється створення екземпляра веб-сервера на основі фреймворку Express, що забезпечує структуровану обробку HTTP-запитів і дозволяє реалізувати модульний підхід до побудови серверної логіки.

Лістинг 3.2 – Налаштування серверного застосунку та базових middleware

```
import cors from "cors";
import express from "express";
import db from "./db.js";
const app = express();
const PORT = process.env.API_PORT || 3001;
app.use(cors());
app.use(express.json());
app.get("/api/health", (_req, res) => {
  res.json({ ok: true, message: "Cinema API is running" });
});
app.get("/api/movies", (_req, res) => {
  const movies = db.prepare("SELECT * FROM movies ORDER BY id
DESC").all();
  res.json(movies);
});
app.listen(PORT, () => {
  console.log(`Cinema API started on http://localhost:${PORT}`);
});
```

кінець лістингу 3.2

Таким чином представлено базову серверну конфігурацію, яка формує фундамент для подальшої реалізації бізнес-логіки застосунку, забезпечує масштабованість архітектури та створює умови для стабільної та безпечної взаємодії між усіма компонентами системи.

Реалізація модуля розкладу сеансів є важливою частиною вебсистеми бронювання квитків, оскільки саме через нього користувач отримує актуальну інформацію про доступні покази фільмів. Для забезпечення швидкого доступу до даних було реалізовано endpoint для читання розкладу сеансів у зручному для клієнта форматі.Замість виконання кількох окремих запитів до різних таблиць бази даних використовується один SQL-запит із застосуванням оператора JOIN, що дозволяє об’єднати інформацію з таблиць сеансів, фільмів та кінозалів.

Передача вже підготовлених даних із сервера дозволяє уникнути дублювання логіки на клієнтській стороні та покращує підтримуваність програмного коду.

Додатковою перевагою є сортування результатів за полем starts_at, завдяки чому всі сеанси відображаються у хронологічному порядку від найближчих до найпізніших. Це значно покращує зручність користування системою, оскільки користувач одразу бачить актуальні покази без необхідності додаткового сортування в інтерфейсі браузера.

У лістингу 3.3 представлено фрагмент серверної логіки, який формує розширений набір даних для відображення розкладу. Запит повертає ідентифікатор сеансу, дату та час початку показу, базову вартість квитка, назву фільму, а також назву залу, в якому відбувається сеанс. Такий підхід значно зменшує навантаження на систему, оскільки виключає необхідність виконання додаткових запитів з боку клієнта та спрощує подальшу обробку інформації на фронтенді.Використання єдиного SQL-запиту також позитивно впливає на продуктивність застосунку, особливо для сторінки афіші та сторінки розкладу сеансів, де користувач очікує швидкого завантаження актуальної інформації.

Лістинг 3.3 – Отримання списку сеансів із приєднанням пов’язаних сутностей

```
app.get("/api/sessions", (_req, res) => {
const sessions = db.prepare(`
SELECT s.id, s.movie_id, s.hall_id, s.starts_at, s.base_price,
m.title AS movie_title, h.name AS hall_name
FROM sessions s
JOIN movies m ON m.id = s.movie_id
JOIN halls h ON h.id = s.hall_id
ORDER BY s.starts_at ASC
`).all();
```

кінець лістингу 3.3

Така реалізація також створює основу для подальшого розширення функціоналу, зокрема додавання фільтрації за датою, залом, жанром фільму або ціновим діапазоном, що підвищує гнучкість і масштабованість усього програмного продукту.

3.2 Тестування та налагодження платформи

Перевірка розробленого вебзастосунку для онлайн-кінотеатру та системи бронювання квитків мала на меті встановити працездатність усіх ключових компонентів програми, надійність взаємодії між серверною логікою, користувацьким інтерфейсом та базою даних, а також підтвердити коректність виконання операцій, пов’язаних із резервуванням місць. Оскільки застосунок охоплює не лише збереження інформації у базі даних, а й обробку клієнтських запитів, створення розкладу сеансів, управління фільмами, перевірку доступності місць та оформлення бронювань, процес тестування охоплював декілька рівнів перевірки функціональності системи.

Особлива увага була зосереджена на перевірці роботи основних таблиць бази даних: movies, halls, sessions та bookings, які формують основу логіки програмного продукту. Було протестовано процеси додавання нових фільмів до каталогу, створення сеансів, перегляду доступних місць, виконання бронювання, зміни статусу замовлення та скасування резервування. Окремо перевірялася коректність відображення цих даних у frontend-частині застосунку, а також правильність синхронізації між клієнтською та серверною частинами системи.

У процесі тестування особливо важливою була перевірка механізму захисту від подвійного бронювання місць. Для цього в таблиці bookings реалізовано унікальне обмеження за полями session_id, row_number та seat_number, що унеможливлює повторне резервування одного й того самого місця. Під час перевірки система коректно повертала помилку при спробі дублювання бронювання, що підтвердило правильність реалізованої логіки та забезпечило цілісність даних.

У лістингу 3.4 представлено приклад ініціалізації тестового сховища даних, його заповнення початковими записами та перевірку обмежень цілісності.

Такий підхід дозволив змоделювати реальні сценарії використання системи, включаючи перевірку зв’язків між таблицями, роботу зовнішніх ключів, контроль унікальності записів та запобігання внесенню неповних або

некоректних даних. Це значно підвищило стабільність роботи системи ще на етапі її розробки.

Лістинг 3.4 – Ініціалізація тестової бази даних та перевірки таблиць

```
import Database from 'better-sqlite3';
function initTestDb() {
const db = new Database('./test_cinema.db');
db.pragma("foreign_keys = ON");
db.exec(`
CREATE TABLE users (
id INTEGER PRIMARY KEY,
email TEXT NOT NULL UNIQUE
);
CREATE TABLE movies (
id INTEGER PRIMARY KEY,
title TEXT NOT NULL,
duration_minutes INTEGER CHECK(duration_minutes > 0)
);
return db;
}
```

кінець лістингу 3.4

Додатково було перевірено механізм каскадного видалення записів, який забезпечує автоматичне видалення пов’язаних даних при видаленні основного елемента.

Наприклад, у разі видалення сеансу система автоматично видаляє всі пов’язані бронювання, що дозволяє уникнути накопичення застарілих або некоректних записів у базі даних. Така перевірка є важливою для підтримання цілісності інформації та спрощення подальшого супроводу програмного забезпечення.

Під час розробки програмного забезпечення активно використовувався процес налагодження (debugging), що дозволяло своєчасно виявляти та усувати помилки на різних етапах реалізації системи. Особлива увага приділялася перевірці логіки серверних запитів, правильності SQL-запитів до бази даних, а також коректності обробки відповідей на стороні клієнта. Для діагностики використовувалися стандартні засоби логування, зокрема console.log у

середовищі Node.js, що дозволяло відстежувати послідовність виконання коду та контролювати значення ключових змінних у реальному часі.

Додатково активно застосовувалися можливості середовища розробки Visual Studio Code, зокрема breakpoint-налагодження через вбудований debugger, покрокове виконання коду (step over, step into, step out), а також перегляд стеку викликів. Це значно спростило пошук логічних помилок та проблем інтеграції між frontend і backend модулями системи.

Для перевірки API-ендпоінтів використовувався інструмент Postman, який дозволяв тестувати коректність HTTP-запитів, перевіряти структуру JSON-відповідей та аналізувати статус-коди сервера. Це було особливо важливо при перевірці ендпоінтів створення бронювання, отримання списку сеансів та перевірки зайнятих місць. Також активно використовувалася консольна перевірка SQL-запитів у середовищі SQLite через бібліотеку better-sqlite3.

Окремо у процесі розробки застосовувалася система контролю версій Git із розміщенням проєкту на платформі GitHub. Це дозволило відстежувати зміни у кодовій базі, працювати з комітами, створювати окремі версії проєкту та, за необхідності, повертатися до стабільних станів системи за допомогою механізму rollback. Для роботи з репозиторієм використовувався Git CLI та інтеграція з Visual Studio Code, що значно спростило процес командної та індивідуальної розробки.

Проведені контрольні сценарії використання включали вибір фільму, перегляд доступних сеансів, вибір місць у залі, оформлення бронювання, зміну статусу замовлення та перегляд історії бронювань. Результат успішного проходження повного процесу оформлення замовлення представлено на рисунку 3.2, де показано сторінку checkout після завершення бронювання або розділ «Мої бронювання» із підтвердженим статусом замовлення. Це дозволило наочно підтвердити коректність роботи всього механізму резервування.

Усі перевірки продемонстрували стабільну роботу системи та відповідність основним функціональним вимогам.

Особливо важливо було підтверджено коректну взаємодію між frontend, backend та базою даних, а також надійність механізму резервування місць.

Таким чином, проведене тестування та налагодження дозволили своєчасно виявити та усунути помилки на різних рівнях системи, підвищити стабільність роботи застосунку та забезпечити надійну взаємодію між усіма його компонентами.

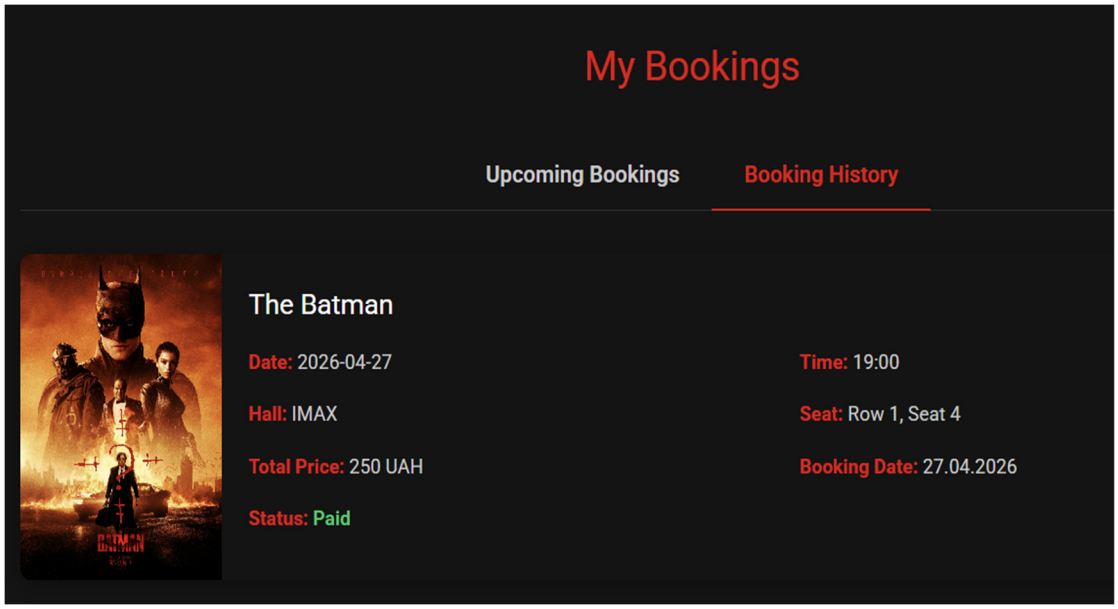

Рисунок 3.2 – Результат тестування процесу бронювання місць у системі

Джерело: розроблено автором

Реалізована система демонструє коректну роботу механізмів бронювання, збереження даних та зручність користування, що підтверджує готовність програмного продукту до подальшого розвитку та практичного використання.

3.3 Посібник з інсталяції та використання програмного забезпечення

Розроблений вебзастосунок онлайн-кінотеатру та системи бронювання квитків є клієнт-серверним програмним продуктом, який не потребує складної інсталяції на стороні кінцевого користувача. Основною умовою для його використання є наявність сучасного веббраузера та підключення до локального або віддаленого сервера, на якому розгорнуто backend-частину системи.

Процес інсталяції програмного продукту передбачає попереднє налаштування середовища виконання та встановлення необхідних залежностей для обох частин системи – серверної та клієнтської. На стороні серверної частини необхідно встановити середовище виконання Node.js, яке забезпечує роботу JavaScript-коду поза межами браузера та виступає основою функціонування backend-компонентів системи. Після цього виконується інсталяція програмних залежностей за допомогою пакетного менеджера npm, включаючи бібліотеки Express.js для організації серверної логіки та better-sqlite3 для забезпечення взаємодії з локальною базою даних SQLite. Після завершення інсталяції залежностей запускається серверна частина застосунку, яка за замовчуванням функціонує на визначеному мережевому порту та забезпечує доступ до REST API для обробки клієнтських запитів, виконання операцій бронювання та взаємодії з базою даних.

Фронтенд-частина реалізована на основі бібліотеки React із використанням мови TypeScript та сучасного інструменту збірки Vite, що забезпечує високу швидкість компіляції та оптимізований цикл розробки. Для запуску користувацького інтерфейсу необхідно виконати встановлення залежностей проєкту та ініціалізувати dev-сервер за допомогою відповідної npm-команди. У результаті застосунок стає доступним через локальну адресу у браузері, забезпечуючи можливість інтерактивної взаємодії із системою в режимі реального часу. Архітектурна організація клієнтської частини дозволяє здійснювати динамічне оновлення інтерфейсу без повного перезавантаження сторінки, що позитивно впливає на швидкодію та загальну плавність користувацького досвіду.

Після запуску програмного продукту користувач потрапляє на головну сторінку, де відображається перелік доступних фільмів із короткою інформацією про кожну позицію. Для кожного фільму можуть бути представлені такі атрибути, як назва, жанр, тривалість, вікові обмеження, рейтинг та постер. Організація структури відображення контенту побудована таким чином, щоб забезпечити швидке сприйняття інформації та мінімізувати кількість дій,

необхідних для переходу до процесу бронювання. Після вибору фільму користувач отримує доступ до сторінки сеансів, де відображається розклад показів, дата та час проведення сеансу, а також інформація про кінозал і доступність місць.

Процес бронювання квитків реалізований як багатокрокова процедура, що складається з послідовних етапів взаємодії користувача із системою. На початковому етапі користувач обирає потрібний сеанс, після чого здійснюється перехід до інтерактивної схеми залу. Візуальне представлення місць дозволяє швидко оцінити їх доступність та вибрати оптимальні позиції. Заброньовані або недоступні місця позначаються окремими візуальними індикаторами, що унеможливлює їх повторний вибір та забезпечує коректність процесу бронювання. Після завершення вибору місць користувач переходить до сторінки оформлення замовлення (checkout), де виконується перевірка коректності введених даних, підтвердження вибраних позицій та формування остаточного запису бронювання.

Після успішного завершення процедури бронювання система автоматично змінює статус замовлення на PAID, а відповідна інформація зберігається у базі даних SQLite для подальшої обробки та відображення. Реалізована система керування замовленнями дозволяє користувачу переглядати історію власних бронювань, аналізувати попередні транзакції та отримувати інформацію про активні або завершені замовлення. Такий підхід забезпечує додатковий рівень зручності взаємодії та підвищує рівень прозорості функціонування системи.

У випадку необхідності скасування бронювання система дозволяє видаляти записи, що перебувають у статусі RESERVED, після чого відповідні місця автоматично повертаються до категорії доступних для інших користувачів. Водночас логіка обробки підтверджених бронювань із статусом PAID може передбачати окремі обмеження або спеціальні сценарії скасування, що визначаються бізнес-логікою застосунку та правилами функціонування сервісу. Така організація дозволяє забезпечити узгодженість станів системи та уникнути конфліктних ситуацій у процесі конкурентного доступу до даних.

Інтерфейс застосунку спроєктовано з урахуванням принципів ергономіки, когнітивної узгодженості та адаптивності, унаслідок чого забезпечується інтуїтивна зрозумілість взаємодії без необхідності попереднього навчання користувача.

Організація навігації реалізована за моделлю односторінкового застосунку (SPA), що передбачає динамічну зміну представлень без повного перезавантаження сторінки. Завдяки цьому досягається суттєве зниження латентності відгуку інтерфейсу та підвищення швидкості виконання користувацьких сценаріїв. Основні дії користувача супроводжуються системою візуальних повідомлень та індикаторів стану, включаючи повідомлення про успішне виконання операцій, попередження та повідомлення про помилки, що забезпечує своєчасний зворотний зв’язок із системою.

Логіка побудови інтерфейсу передбачає використання уніфікованих компонентів, що забезпечують консистентність відображення даних, стандартизацію елементів керування та передбачуваність поведінки інтерфейсу. Подібний підхід сприяє зменшенню когнітивного навантаження на користувача та підвищує швидкість виконання типових операцій. Взаємодія з динамічними даними реалізована із застосуванням асинхронних механізмів завантаження та оновлення інформації, що дозволяє уникнути блокування інтерфейсу під час виконання запитів до сервера. Додатково реалізовано адаптивний дизайн, який забезпечує коректне відображення елементів інтерфейсу на різних типах пристроїв та екранів.

Процес інсталяції програмного продукту організовано з орієнтацією на мінімізацію початкових витрат часу та ресурсів. Використання автоматизованих механізмів встановлення залежностей та ініціалізації середовища дозволяє швидко підготувати систему до експлуатації без необхідності виконання складних конфігураційних операцій. Такий підхід забезпечує можливість оперативного розгортання застосунку як у середовищі розробки, так і в умовах тестування або демонстрації функціональних можливостей системи.

Підготовка середовища виконання передбачає інсталяцію необхідних пакетів через систему керування залежностями Node Package Manager, після чого автоматично формується структура робочого середовища, необхідна для запуску клієнтської та серверної частин застосунку. Використання сучасних інструментів збірки та розробки забезпечує оптимізацію процесів компіляції, гарячого оновлення компонентів та швидкого перезапуску сервера під час внесення змін до програмного коду. Це скорочує тривалість циклу розробки та підвищує ефективність тестування окремих функціональних модулів.

Ініціалізація бази даних виконується автоматично при першому запуску серверної частини системи. У процесі запуску створюються необхідні таблиці, формуються зв’язки між сутностями та додаються тестові дані, що дозволяє одразу перейти до перевірки основного функціоналу без ручного наповнення сховища даних. Подібна організація значно спрощує процес перенесення застосунку між різними середовищами виконання та забезпечує відтворюваність структури даних.

Експлуатаційні характеристики програмного продукту орієнтовані на спрощення процесів супроводу та подальшого розвитку системи. Модульна організація програмного коду забезпечує можливість локалізованого внесення змін без порушення цілісності архітектури застосунку, а використання сучасних засобів типізації та статичного аналізу коду сприяє підвищенню підтримуваності програмного забезпечення.

Компонентний підхід до побудови клієнтської частини забезпечує логічне розділення функціональних елементів інтерфейсу, що спрощує повторне використання окремих модулів та зменшує складність подальшої модифікації системи. На серверному рівні реалізовано структуроване розділення маршрутів, сервісної логіки та механізмів взаємодії з базою даних, унаслідок чого підвищується читабельність програмного коду та спрощується процес його супроводу.

Передбачуваність поведінки системи, структурованість компонентів та розділення відповідальностей між окремими модулями формують основу для

подальшого масштабування функціональних можливостей застосунку та інтеграції додаткових сервісів. Архітектурна модель допускає розширення функціоналу шляхом підключення систем електронної оплати, механізмів автентифікації користувачів, модулів аналітики або сервісів сповіщень без необхідності повного перепроєктування програмної структури.

Додатковою перевагою реалізованого підходу є можливість адаптації системи до різних сценаріїв використання. Завдяки незалежності окремих компонентів та використанню стандартизованих інтерфейсів взаємодії забезпечується спрощення процесу інтеграції із зовнішніми API та сторонніми інформаційними сервісами. Це створює передумови для подальшої трансформації навчального або MVP-рішення у повноцінний програмний продукт із підтримкою багатокористувацького режиму, централізованого адміністрування та розширених механізмів обробки даних.

РОЗДІЛ 4
СПЕЦІАЛЬНІ РОЗРАХУНКИ

У розроблюваній системі онлайн-кінотеатру та бронювання квитків питання інформаційної безпеки інтерпретується як багаторівнева сукупність організаційно-технічних заходів, спрямованих на захист персональних даних користувачів, забезпечення коректності транзакцій бронювання та обмеження доступу до адміністративного функціоналу. Забезпечення конфіденційності, цілісності та доступності даних реалізується у межах узгодженої клієнт-серверної архітектури, де кожен рівень системи виконує власні функції захисту, взаємодіючи з іншими компонентами через контрольовані інтерфейси обміну даними.

Механізми автентифікації та авторизації побудовані з урахуванням необхідності мінімізації ризиків компрометації облікових записів, у зв’язку з чим застосовується попередня обробка паролів перед їх збереженням у базі даних. Перетворення пароля у хеш-значення із додаванням випадкового сольового параметра унеможливлює його відновлення у початковому вигляді навіть за умов несанкціонованого доступу до сховища, а також суттєво ускладнює реалізацію атак, пов’язаних із перебором або використанням попередньо сформованих словників. Такий підхід забезпечує підвищення криптостійкості системи без необхідності ускладнення логіки автентифікації.

Подальша взаємодія користувача із системою реалізується через механізм токенізації доступу, за якого після успішної автентифікації формується унікальний маркер сесії, що використовується для ідентифікації запитів до захищених ресурсів. Застосування токенів дозволяє реалізувати гнучку модель контролю доступу, в межах якої визначаються права виконання операцій залежно від ролі користувача, що забезпечує розмежування доступу до функціональних можливостей системи та обмеження виконання критичних дій. Додатково реалізовано механізми перевірки терміну дії токенів та можливість

примусового завершення сесії у випадку виявлення підозрілої активності або повторної авторизації з іншого пристрою.

Передавання даних між клієнтською та серверною частинами організовано із врахуванням вимог до захисту мережевої взаємодії, де базова реалізація передбачає використання HTTP-протоколу з можливістю подальшого переходу до захищеного каналу зв’язку на основі HTTPS. Така архітектурна гнучкість дозволяє адаптувати систему до умов продуктивного середовища, забезпечуючи шифрування переданих даних та зменшення ризиків їх перехоплення або модифікації під час передачі. Використання TLS-протоколів у перспективі дозволяє реалізувати захищений механізм обміну конфіденційною інформацією, включаючи персональні дані користувачів та службові параметри сесії.

Захист на рівні взаємодії з базою даних забезпечується шляхом використання параметризованих запитів, що виключає можливість інтерпретації користувацького вводу як частини SQL-команд. Такий підхід дозволяє ефективно протидіяти атакам типу SQL-ін’єкції, формуючи безпечний механізм доступу до даних та запобігаючи несанкціонованим змінам структури або вмісту бази даних. Додатково застосування підготовлених запитів сприяє підвищенню продуктивності виконання операцій за рахунок повторного використання попередньо скомпільованих інструкцій. Структура бази даних також передбачає використання обмежень цілісності, зовнішніх ключів та механізмів каскадного оновлення, що забезпечує узгодженість взаємопов’язаних записів.

Забезпечення цілісності бізнес-логіки бронювання реалізується шляхом введення обмежень на рівні схеми бази даних, що регламентують допустимі стани системи та унеможливлюють виникнення логічних конфліктів. Унікальність комбінацій параметрів, що ідентифікують конкретне місце у межах сеансу, гарантує відсутність дублювання бронювань навіть за умов одночасного виконання великої кількості запитів. Транзакційна модель обробки даних, доповнена механізмами блокування, забезпечує узгодженість змін і запобігає виникненню неконсистентних станів у процесі конкурентного доступу. Для зменшення ризику втрати даних додатково можуть застосовуватись механізми

резервного копіювання та журналювання транзакцій, що забезпечують можливість відновлення працездатності системи після критичних збоїв.

Окремий аспект безпеки пов’язаний із захистом від перевантаження системи, що може виникати внаслідок інтенсивного або зловмисного використання API. Архітектура серверної частини передбачає можливість інтеграції механізмів обмеження частоти запитів, використання проміжного програмного забезпечення для фільтрації трафіку та впровадження стратегій керування навантаженням. Навіть у спрощеній реалізації закладено структурні передумови для подальшого розширення засобів протидії DoS та DDoS-атакам без необхідності суттєвої перебудови системи. Додатково передбачено можливість інтеграції систем логування та моніторингу активності, що дозволяє оперативно виявляти аномальні навантаження та потенційні спроби несанкціонованого втручання.

Механізми контролю навантаження мають особливе значення для систем онлайн-бронювання, оскільки пікові періоди активності користувачів можуть супроводжуватися різким зростанням кількості одночасних запитів до серверної частини. У таких умовах перевантаження окремих компонентів інфраструктури здатне призводити до збільшення часу відповіді або часткової недоступності функціоналу. Процес формування хеш-значення представлено у формулі 4.1.

Відомо, що

$$H = h(P \oplus S) \tag{4.1}$$

де P – пароль користувача;

S – випадкове сольове значення;

h – криптографічна хеш-функція;

H– результат хешування, що зберігається у базі даних.

Використання сольового параметра забезпечує унікальність хеш-значень навіть у випадках збігу початкових паролів, унаслідок чого ускладнюється реалізація атак із використанням таблиць відповідності та методів масового

перебору такий підхід забезпечує підвищення криптостійкості системи без необхідності ускладнення логіки автентифікації.

Валідація даних виконується як на стороні клієнта, так і на серверному рівні, що дозволяє реалізувати багаторівневий механізм перевірки коректності введеної інформації. На клієнтській стороні здійснюється перевірка форматів введення, наявності обов'язкових полів та допустимості значень, унаслідок чого скорочується кількість помилкових запитів та покращується загальна стабільність взаємодії користувача із системою. Серверна перевірка, своєю чергою, виконує функцію остаточного контролю достовірності отриманих даних, запобігаючи обходу клієнтських механізмів валідації або передачі модифікованих запитів.

Значна увага приділяється також питанням забезпечення безпеки користувацької сесії та збереження конфіденційної інформації у браузерному середовищі. Умова коректності бронювання представлена у формулі 4.2.

Відомо, що

$$\forall(s_i, m_j) \in B\colon \exists!\ r_k \tag{4.2}$$

де s_i– конкретний сеанс;

m_j – місце у кінозалі;

B – множина бронювань;

r_k – єдиний допустимий запис бронювання для відповідної комбінації параметрів.

Подібний підхід забезпечує відсутність дубльованих транзакцій та підтримує узгодженість станів системи навіть у процесі конкурентного доступу до ресурсів. Додатково передбачено очищення локальних службових даних у випадках завершення сесії або втрати авторизаційного статусу, що знижує ймовірність використання залишкових даних у браузерному середовищі.

Окремого значення набуває забезпечення цілісності транзакційних операцій під час роботи із системою бронювання. У процесі обробки запитів

виконується контроль послідовності зміни станів бронювання, перевірка доступності місць та фіксація результатів операцій у базі даних лише після успішного завершення всіх етапів перевірки. Подібний підхід мінімізує ризики виникнення логічних конфліктів, пов’язаних із паралельним доступом користувачів до одного ресурсу, а також підвищує надійність функціонування системи в умовах конкурентного виконання запитів.

Додатковим елементом захисту є використання централізованої обробки помилок на серверному рівні. Усі виняткові ситуації, пов’язані з порушенням правил доступу, помилками автентифікації або некоректними параметрами запитів, обробляються через єдиний механізм формування відповідей API. Це дозволяє уніфікувати реакцію системи на помилки, спростити журналювання подій та забезпечити контрольоване інформування користувача без розкриття внутрішньої структури серверної логіки.

Сукупність реалізованих заходів формує цілісну модель інформаційної безпеки, що охоплює криптографічний захист облікових даних, токенізацію доступу, контроль прав користувачів, захищену взаємодію з базою даних і механізми забезпечення цілісності транзакцій. Реалізовані механізми безпеки інтегровані в загальну архітектуру застосунку та функціонують узгоджено на всіх рівнях системи, включаючи клієнтський інтерфейс, серверну логіку та рівень збереження даних.

Сформована архітектура забезпечує базовий рівень захисту інформації та створює передумови для подальшого впровадження більш складних засобів безпеки, включаючи багатофакторну автентифікацію, централізоване керування доступом, системи журналювання подій, автоматизований аудит безпеки та розширені механізми моніторингу активності користувачів у системі.

ВИСНОВКИ

У процесі виконання кваліфікаційної роботи вирішено задачу розробки веборієнтованої системи бронювання квитків у кінотеатрі з використанням сучасних технологій веброзробки. Актуальність теми зумовлена розвитком цифрових сервісів у сфері дозвілля, що супроводжується підвищенням вимог до швидкості роботи системи, зручності користувацького інтерфейсу та надійності збереження даних.

На етапі постановки задачі виконано аналіз предметної області та досліджено існуючі системи онлайн-бронювання квитків, у результаті чого виявлено типові архітектурні підходи, функціональні обмеження та експлуатаційні недоліки. Отримані результати стали основою для формування вимог до розроблюваної системи, включаючи визначення ключових функціональних та нефункціональних характеристик, що охоплюють процеси перегляду афіші, вибору сеансів, бронювання місць і ведення історії замовлень із забезпеченням узгодженості та цілісності даних.

На основі сформованих вимог обґрунтовано вибір технологічного стеку, орієнтованого на побудову сучасного вебзастосунку з розділенням клієнтської та серверної логіки. Клієнтська частина реалізована з використанням компонентної архітектури, що забезпечує структурованість інтерфейсу та повторне використання програмних модулів, а застосування статичної типізації підвищує формальну коректність коду та знижує ймовірність виникнення помилок на етапі розробки.

Архітектура системи побудована за клієнт-серверною моделлю, у межах якої інтерфейс користувача, серверна логіка та рівень збереження даних функціонально розділені, що відповідає сучасним принципам проєктування розподілених інформаційних систем. Така організація забезпечує масштабованість, спрощує супровід програмного продукту та підвищує гнучкість його подальшої модифікації без порушення загальної структури.

У процесі реалізації створено функціональний вебзастосунок, що забезпечує виконання основних сценаріїв взаємодії користувача із системою, включаючи перегляд фільмів, вибір сеансів, бронювання місць та керування власними замовленнями. Інтерфейс характеризується адаптивністю та динамічним оновленням даних, тоді як логіка взаємодії побудована з урахуванням асинхронної обробки запитів і маршрутизації станів застосунку.

Серверна частина реалізує бізнес-логіку системи, забезпечуючи обробку HTTP-запитів, перевірку доступності місць та створення бронювань із контролем цілісності даних. Особлива увага приділена механізмам запобігання конфліктам при одночасному доступі користувачів, що досягається шляхом реалізації унікальності комбінацій ключових атрибутів бронювання та перевірки стану ресурсів перед їх резервуванням.

Проєктування бази даних здійснено на основі реляційної моделі, що забезпечує формалізований опис предметної області та встановлення логічних зв’язків між сутностями. Така структура дозволяє ефективно організувати зберігання даних і підтримувати їх узгодженість у межах транзакційних операцій. Використана система керування базами даних забезпечує достатній рівень продуктивності та спрощує процес розгортання застосунку.

Застосування гібридного підходу до організації даних, що поєднує локальне збереження стану та серверну обробку критичних операцій, дозволило досягти балансу між швидкодією інтерфейсу та надійністю централізованого збереження інформації. Така модель взаємодії підвищує ефективність користувацького досвіду без втрати контролю над цілісністю даних.

У процесі тестування перевірено коректність функціонування основних модулів системи, включаючи сценарії бронювання, обробки запитів та взаємодії між клієнтською і серверною частинами. Отримані результати засвідчили стабільність роботи застосунку та відповідність реалізованої логіки поставленим вимогам.

Практична значущість роботи полягає у можливості використання розробленого програмного продукту як базової платформи для створення

повноцінних систем онлайн-бронювання квитків із подальшим розширенням функціональності. Подальший розвиток системи може бути спрямований на інтеграцію платіжних сервісів, впровадження повноцінної системи авторизації, розширення адміністративного функціоналу та реалізацію аналітичних модулів.

Перспективи вдосконалення пов’язані з еволюцією архітектури у напрямі масштабованих серверних рішень, впровадження мікросервісного підходу та розширення інтеграційних можливостей відповідно до вимог сучасних інформаційних систем. У підсумку досягнуто сформульованої мети та забезпечено виконання визначених завдань у повному обсязі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Інформаційні системи та технології. URL: https://journals.uran.ua/vsed_oneu/article/view/21455 (дата звернення: 04.02.2026).

2. Електронна комерція та цифрові сервіси в сучасному суспільстві. URL: https://visnyk-ekon.uzhnu.edu.ua/article/view/278441 (дата звернення: 06.02.2026).

3. E-commerce. URL: https://doi.org/ (дата звернення: 09.02.2026).

4. Conceptual foundations of electronic commerce systems. URL: https://www.sciencedirect.com/topics/computer-science/e-commerce (дата звернення: 20.02.2026).

5. React Documentation. URL: https://react.dev/ (дата звернення: 25.02.2026).

6. Express.js Documentation. URL: https://expressjs.com/ (дата звернення: 05.04.2026).

7. SQLite Doc. URL:https://www.sqlite/docs.html (дата звернення: 06.03.2026).

8. better-sqlite3 Doc. URL: https://github.com/WiseLibs/better-sqlite3 (дата звернення: 07.03.2026).

9. React Docu. URL: https://reactrouter.com/ (дата звернення: 12.03.2026).

10. TanStack Query Documentation. URL: https://tanstack.com/query/latest (дата звернення: 12.03.2026).

11. Tailwind CSS Documentation. URL: https://tailwindcss.com/docs (дата звернення: 13.04.2026).

12. MDN Web Docs. Web APIs and JavaScript Guide. URL: https://developer.mozilla.org/ (дата звернення: 20.04.2026).

13. Nielsen J. Usability Engineering. Morgan Kaufmann. URL: https://www.nngroup.com/books/usability-engineering/ (дата звернення: 29.04.2026).

14. Elements of Reusable Object-Oriented. URL: https://refactoring.guru/design-patterns/book (дата звернення: 30.04.2026).

15. W3C. Web Content Accessibility Guidelines (WCAG) 2.2. URL: https://www.w3.org/TR/WCAG22/ (дата звернення: 15.04.2026).

16. D3.js Documentation. URL: https://d3js.org/ (дата звернення: 16.04.2026).

ДОДАТКИ

ДОДАТОК А

Сертифікат про участь у міжнародній науково-практичній конференції

ДОДАТОК Б

База даних веб сайту

```
import Database from "better-sqlite3";

const db = new Database("cinema.db");
db.pragma("foreign_keys = ON");

db.exec(`
  CREATE TABLE IF NOT EXISTS movies (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    title TEXT NOT NULL,
    duration_minutes INTEGER NOT NULL CHECK(duration_minutes > 0),
    genre TEXT,
    description TEXT,
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
  );

  CREATE TABLE IF NOT EXISTS halls (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL UNIQUE,
    rows_count INTEGER NOT NULL CHECK(rows_count > 0),
    seats_per_row INTEGER NOT NULL CHECK(seats_per_row > 0),
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
  );

  CREATE TABLE IF NOT EXISTS sessions (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    movie_id INTEGER NOT NULL,
    hall_id INTEGER NOT NULL,
    starts_at TEXT NOT NULL,
    base_price REAL NOT NULL CHECK(base_price >= 0),
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (movie_id) REFERENCES movies(id) ON DELETE CASCADE,
    FOREIGN KEY (hall_id) REFERENCES halls(id) ON DELETE CASCADE
  );

  CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    email TEXT NOT NULL UNIQUE,
    name TEXT NOT NULL,
    password TEXT NOT NULL,
    role TEXT NOT NULL DEFAULT 'user' CHECK(role IN ('user', 'admin')),
    created_at TEXT DEFAULT CURRENT_TIMESTAMP
  );

  CREATE TABLE IF NOT EXISTS bookings (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    session_id INTEGER NOT NULL,
```

```
    user_id INTEGER,
    customer_name TEXT NOT NULL,
    customer_email TEXT NOT NULL,
    row_number INTEGER NOT NULL CHECK(row_number > 0),
    seat_number INTEGER NOT NULL CHECK(seat_number > 0),
    price REAL NOT NULL CHECK(price >= 0),
    status TEXT NOT NULL DEFAULT 'booked' CHECK(status IN ('booked',
'cancelled')),
    created_at TEXT DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (session_id) REFERENCES sessions(id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(id) ON DELETE SET NULL,
    UNIQUE(session_id, row_number, seat_number)
  );

  CREATE INDEX IF NOT EXISTS idx_sessions_movie_id ON sessions(movie_id);
  CREATE INDEX IF NOT EXISTS idx_sessions_starts_at ON
sessions(starts_at);
  CREATE INDEX IF NOT EXISTS idx_bookings_session_id ON
bookings(session_id);
`);

const hallsCount = db.prepare("SELECT COUNT(*) AS count FROM
halls").get().count;
if (!hallsCount) {
  const seedHall = db.prepare(`
    INSERT INTO halls (name, rows_count, seats_per_row)
    VALUES (?, ?, ?)
  `);
  seedHall.run("Hall A", 8, 12);
}

const usersCount = db.prepare("SELECT COUNT(*) AS count FROM
users").get().count;
if (!usersCount) {
  const seedUser = db.prepare(`
    INSERT INTO users (email, name, password, role)
    VALUES (?, ?, ?, ?)
  `);
  seedUser.run("user@example.com", "Regular User", "user123", "user");
  seedUser.run("admin@example.com", "Admin User", "admin123", "admin");
}

const moviesCount = db.prepare("SELECT COUNT(*) AS count FROM
movies").get().count;
if (!moviesCount) {
  const seedMovie = db.prepare(`
    INSERT INTO movies (title, duration_minutes, genre, description)
    VALUES (?, ?, ?, ?)
  `);
  const duneId = seedMovie.run(
    "Dune: Part Two",
```

```
    166,
    "Sci-Fi",

    "Epic science fiction adventure."
  ).lastInsertRowid;

  const hallId = db.prepare("SELECT id FROM halls LIMIT 1").get().id;
  db.prepare(`
    INSERT INTO sessions (movie_id, hall_id, starts_at, base_price)
    VALUES (?, ?, ?, ?)
  `).run(duneId, hallId, new Date(Date.now() + 24 * 60 * 60 *
1000).toISOString(), 230);
}

export default db;
```