

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
(повне найменування факультету)

Кафедра комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка
до дипломного проєкту (роботи)
бакалавра
(ступінь вищої освіти (освітній ступінь))

на тему **РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ЗАМОВЛЕННЯ**
КВИТКІВ ДЛЯ КІНОТЕАТРУ

Виконав: студент 4 курсу, групи **КНТ-520**
спеціальності _____

123 «Комп'ютерна інженерія»

(код і найменування спеціальності)

Освітня програма (спеціалізація) _____

«Комп'ютерна інженерія»

ВЕРТМАН М.А

(ПРИЗВИЩЕ та ініціали)

Керівник **СКРУПСЬКИЙ С.Ю.**

(ПРИЗВИЩЕ та ініціали)

Рецензент **КОЗИНА Г.Л.**

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування вищого навчального закладу)

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ
Завідувач кафедри КУДЕРМЕТОВ Р.К.
«14» жовтня 2024 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ВЕРТМАНА Миколи Андрійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка комп'ютерної системи замовлення квитків для кінотеатру

керівник проєкту (роботи) к. т. н., доцент, СКРУПСЬКИЙ Степан Юрійович

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом вищого навчального закладу від «01» квітня 2024 року № 105

2. Строк подання студентом проєкту (роботи) 01 червня 2024 року

3. Вихідні дані до проєкту (роботи) наявні методи та алгоритми бронювання квитків та роботи з базами даних

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

1) Аналіз предметної області

2) Моделювання системи;

3) Розробка системи;

4) Випробування комп'ютерної системи

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

ER-модель бази даних;

Інтерфейс десктопної версії застосунку;

Інтерфейс мобільної версії застосунку.

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	СКРУПСЬКИЙ С. Ю., к.т.н., доцент		
нормоконтроль	ПОЛЬСЬКА О. В., ст. викл.		

7. Дата видачі завдання 25.02.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз наявних методів та алгоритмів пошуку об'єктів, перегляд отриманих результатів, виділення недоліків та переваг	17.03.2024 р.	
2	Розробка структури системи	10.04.2024 р.	
3	Розробка алгоритму роботи системи	15.04.2024 р.	
4	Реалізація системи уточненого розпізнавання	19.04.2024 р.	
5	Дослідження системи уточненого розпізнавання	22.04.2024 р.	
6	Оформлення отриманих результатів у ПЗ	28.04.2024 р.	
7	Оформлення графічного матеріалу	10.05.2024 р.	
8	Оформлення допоміжного матеріалу	20.05.2024 р.	

Студент (ка)



(підпис)

Микола ВЕРТМАН

(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)



(підпис)

Степан СКРУПСЬКИЙ

(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

ПЗ: 100 стор., 47 рис., 12 табл., 1 додаток, 28 джерел.

ВЕБСАЙТ, БРОНЮВАННЯ КВИТКІВ, КЛІЄНТ, СЕРВЕР, NODE.JS, POSTGRESQL, EXPRESS, REACT.JS

Розроблений вебсайт з замовлення квитків для кінотеатру, що містить серверну і клієнтську частини, а дані зберігаються в базі даних.

На стороні клієнта відбувається взаємодія з контентом застосунку, функціонал яких обробляє серверна частина застосунку що отримує запити, обробляє їх та відправляє до бази даних, або звіряється з нею.

Реалізована клієнт-серверна архітектура використовує усі сучасні інструменти для побудови реактивних застосунків та надає адміністрації структури легкий інструмент для обробки замовлень та зберігання інформації про них, а для клієнтів – легкий та дистанційний доступ до можливості створити замовлення.

ЗМІСТ

Вступ.....	7
1. Аналіз предметної області	9
1.1 Поняття про електронну комерцію	9
1.2 Відмінні властивості та основний функціонал веб-застосунків для продажу квитків.....	9
1.3 Приклади подібних застосунків	10
1.4 Вибір архітектури комп'ютерної системи	13
1.5 Вибір технологій для серверної частини застосунку	15
1.6 Вибір технологій для клієнтської частини застосунку	20
1.7 Вибір СКБД	27
1.8 Висновки за розділом.....	32
2. Моделювання системи	33
2.1 Загальні вимоги	33
2.2 Вимоги до апаратного забезпечення	33
2.3 Функціональні вимоги до системи та розподілення ролей.....	34
2.4 Діаграма послідовностей	35
2.5 Моделювання бази даних	36
3. Розробка системи	41
3.1 Створення бази даних	41
3.2 Розробка серверної частини застосунку	41
3.3 Розробка клієнтської частини застосунку.....	53
4. Випробування комп'ютерної системи	64
4.1 Випробування форми авторизації та реєстрації	64
4.2 Випробування можливостей адміністратора	67
4.3 Випробування функції замовлення квитків.....	73
4.4 Випробування адаптивності інтерфейсу.....	77
Висновки.....	85

Перелік джерел посилання	86
Додаток А. Коди до модулів.....	88

ВСТУП

Головні переваги використання інтернет технологій між споживачем та надавачем послуг в сучасному світі – це економія часу, дистанційність та простота. Там де раніше з боку клієнта вимагалось необхідність долати відстані, відстоювати черги та вести соціальної взаємодії, а з боку керуючого бізнесом – необхідність вести облікові фізичні документи, витрачати матеріальні ресурси та розширювати персонал, що призводило до більших затрат, зараз відбувається просто: в декілька натискань на електронному пристрої зі сторони клієнта, та економії на ресурсах, завдяки автоматизації процесу, та переводу до електронного вигляду документованої частини бізнесу, через перехід на ведення бізнесу, завдяки сучасним програмним рішенням.

Одна з галузей, котра суттєво розвинулася та збільшила свої прибутки за рахунок розвитку комп'ютерних технологій - є галузь продажу квитків для кінотеатру, на прикладі якої можна прослідити ефективні сторони інтегрування комп'ютерних систем у бізнесі та дослідити принципи їх створення та використання.

За час виконання роботи мною буде виконано наступні задачі:

- проведення аналізу предметної області та огляд існуючих веб-ресурсів в даному напрямку;
- обґрунтування вибору технологій для створення веб-застосунку з порівнянням відносно альтернативних рішень;
- проведення попереднього моделювання системи та визначення основних принципів роботи;
- розробка веб-застосунку;
- випробування програмного забезпечення та підбиття підсумків з виконаної роботи за їх результатами.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття про електронну комерцію

Електронна комерція — це купівля та продаж товарів або послуг в Інтернеті. Вона охоплює широкий спектр даних, систем та інструментів для онлайн-покупців і продавців, включаючи мобільні покупки та шифрування онлайн-платежів [1].

Більшість компаній, які працюють в Інтернеті, використовують онлайн-магазин і/або платформу для маркетингу електронної комерції та продажів, а також для контролю за логістикою та виконанням [1].

За даними eMarketer, у 2022 році глобальні роздрібні продажі електронної комерції вперше перевищать 5 трильйонів доларів, що становитиме понад п'яту частину загального обсягу роздрібного продажу. А за прогнозами до 2025 року загальні витрати перевищать 7 трильйонів доларів, незважаючи на уповільнення зростання [1].

1.2 Відмінні властивості та основний функціонал веб-застосунків для продажу квитків

Програмне забезпечення для продажу квитків дозволяє компаніям продавати та розповсюджувати квитки на події через кілька каналів, таких як соціальні мережі, бізнес-сайти та платформи трансляції подій. Підприємства можуть використовувати онлайн-систему для продажу квитків на свої особисті, віртуальні та змішані заходи. Для організаторів і менеджерів подій цей інструмент допомагає відстежувати реєстрацію подій, збирати інформацію про гостей і легко керувати бронюваннями. Крім того, це підвищує зручність для відвідувачів, спрощуючи весь процес — від покупки квитків до відвідування події [2].

Загальні переваги програмного забезпечення для продажу квитків:

- система автоматизує процес онлайн-продажу, що робить його зручнішим для організаторів. Зручний інтерфейс для онлайн-продажу квитків скорочує період очікування та пропонує цілодобовий доступ до продажу квитків. В сучасних реаліях велика кількість платформ продажу квитків також можуть адаптуватися до мобільних пристроїв, або навіть пропонують окремі мобільні додатки, що дозволяє клієнтам зручно купувати квитки за допомогою своїх смартфонів і планшетів будь-де та будь-коли. Клієнти можуть в реальному часі перевіряти наявності квитків, варіантів місць і вартості квитків [2];

- програмне забезпечення зменшує кількість різноманітних адміністративних завдань, зокрема сам продаж квитків і, як наслідок, управління запасами та друк квитків, для менеджерів закладів та організаторів заходів. Це також допомагає компаніям автоматично збирати інформацію про учасників, зокрема їх імена, контактні дані та дані квитків, без необхідності втручання вручну. Програмне забезпечення також автоматизує обробку платежів для користувачів, дозволяючи їм безпечно приймати платежі за допомогою різних методів оплати, таких як дебетові картки, кредитні картки та цифрові гаманці [2];

- інструменти продажу квитків дозволяють продавати квитки на події через різні платформи, наприклад веб-сайти онлайн-продажу квитків, кіоски, каси та канали соціальних мереж. Організатори подій також можуть використовувати платформу для розміщення квитків на популярних онлайн-ринках, які мають ширшу базу користувачів. Це допомагає підприємствам охопити ширшу аудиторію, залучаючи різні демографічні та клієнтські сегменти [2].

1.3 Приклади подібних застосунків

На прикладі двох українських кінотеатрів, а саме «Кінотеатр ім. Довженка» [3] та «Планета кіно» [4], можна побачити як може бути реалізована робота

подібних застосунків, виділити загальний необхідний базовий функціонал, та прослідити відмінності в додаткових параметрах та рішеннях між веб-застосунком мультирегіональної корпорації та локального кінотеатру.

1.3.1 Кінотеатр ім. Довженка

Кінотеатр ім. Довженка – кінотеатр міста Запоріжжя, котрий існує з 1964 року та має в своєму розпорядженні всього дві зали для перегляду фільмів.

Веб-сайт даного кінотеатру має простий базовий функціонал, а саме:

- із стартової сторінки користувач може одразу обрати фільм;
- на сторінці фільму передивитися інформацію про нього та обрати сеанс;
- перейти на сторінку замовлення, обрати місця та забронювати їх;
- є можливість подивитися контактні дані кінотеатру.

Головну сторінку сайту представлено на рис. 1.1.

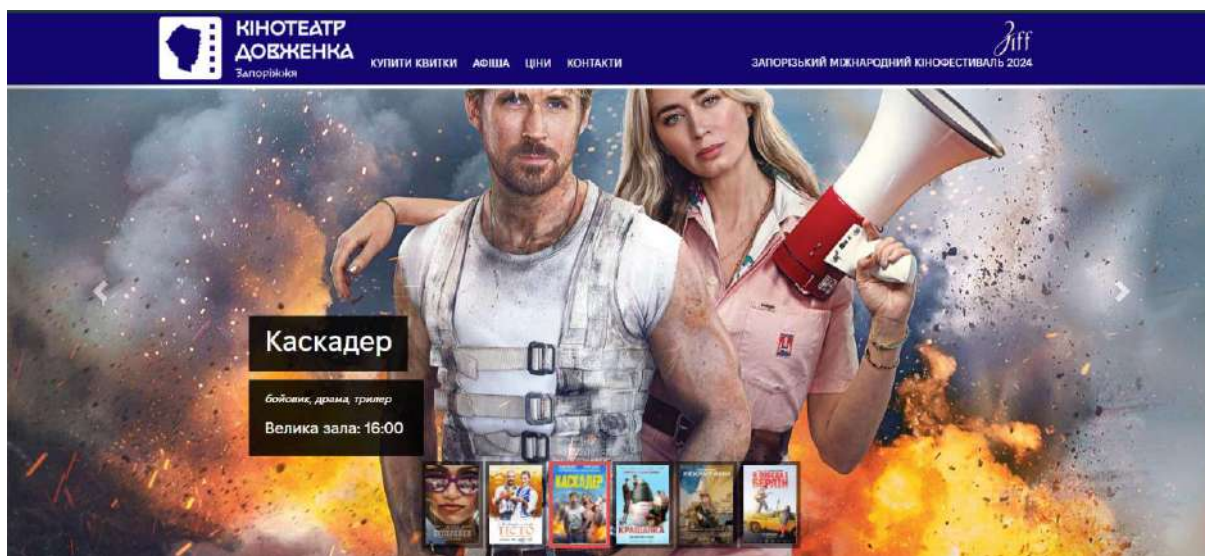


Рисунок 1.1 – Головна сторінка сайту «Кінотеатр ім. Довженка»

1.3.2 Планета Кіно

Планета Кіно – мережа кінотеатрів, що відкрилась в 2008 році у м. Київ та на даний момент в своєму розпорядженні налічує 9 кінотеатрів [5].

Даний кінотеатр для нашої країни в галузі розвитку взаємодії з користувачем за допомогою інтернету є знаковим, оскільки це перший серед кінотеатрів України, що почав продавати квитки онлайн через сайт та перший у

світі розробив процедуру доступу до кінозалу без друку квитка — його можна показувати з екрану смартфона. Також, у 2011 році запустили перший серед кінотеатрів країн східної Європи мобільний додаток. Він дозволяє купувати квитки онлайн, переглядати інформацію про фільми, сеанси, розклад. Не потрібно роздруковувати квитки — вони зберігаються у вигляді штрих-коду на смартфоні. Має сповіщення про початок прокату фільмів та продаж квитків. Також варто зазначити, що це перший в країні кінотеатр, що реалізував можливість купувати квитки пільговим категоріям відвідувачів: студентам, школярам, пенсіонерам та іншим категоріям, — зі знижкою на сайті [5].

В додаток до базового функціоналу сайту «Кінотеатру ім. Довженка», в застосунку «Планети Кіно», реалізовано наступні можливості:

- наявність особистого кабінету, де можна подивитись інформацію про переглянуті фільми, залишені коментарі та де також можна завантажити придбані квитки;
- вести діалог з адміністрацією кінотеатру через чат;
- перейти до соціальних мереж кінотеатру через кнопки-посилання, що розташовані збоку на головному екрані;
- можливість переглянути новини та анонси акцій кінотеатру;
- подивитися анонси фільмів, котрі буде показано в майбутньому;
- одна з найцікавіших функцій – це можливість подивитися інформацію про фільм (оцінка, актори, опис, коментарі до фільму тощо), котрий вже знятий з прокату, через пошук за його назвою.

Ці фактори дозволяють інтегруватися в дозвілля користувача, дають можливість залучати нових та заохочувати вже існуючих клієнтів і далі користуватися їх послугами через програми лояльності, котрі, не в останню чергу, впроваджуються завдяки розширеним функціям веб-застосунку, що виводить бізнес на вищий рівень заробітку.

Головну сторінку сайту можна побачити на рис. 1.2.

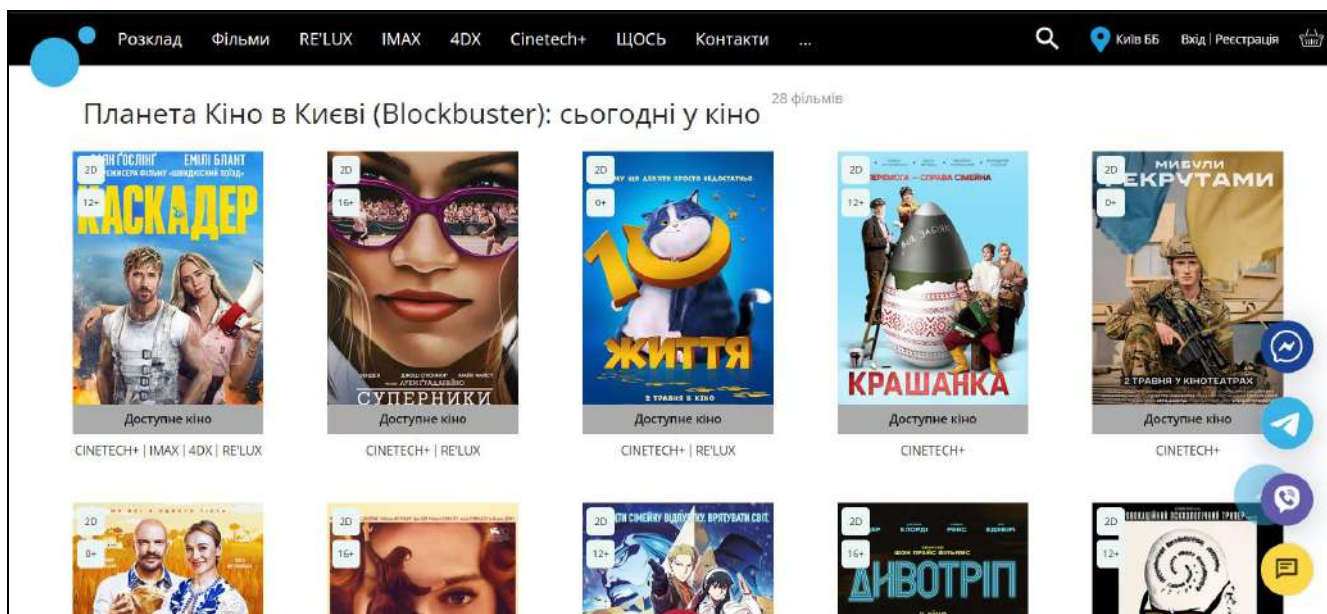


Рисунок 1.2 – Головна сторінка сайту кінотеатру «Планета Кіно»

1.4 Вибір архітектури комп'ютерної системи

Перший етап створення комп'ютерної системи – це вибір її архітектури.

Архітектура програмного забезпечення – це структурна концепція та організація програми, яка визначає загальну форму, компоненти, взаємозв'язки та принципи роботи програмного забезпечення.

Процес вибору архітектури програмного забезпечення – це процес визначення набору апаратних і програмних компонентів та їхніх інтерфейсів для створення основи для розробки комп'ютерної системи [6].

Основні цілі побудови архітектури програмного забезпечення включають:

- чіткий розподіл завдань і функцій між компонентами для забезпечення прозорості та керованості;
- розбиття (декомпозиція) системи на невеликі, легко підтримувані модулі;
- створення компонентів, які можна повторно використовувати в різних частинах проекту або в інших завданнях;

- створення системи, яка здатна адаптуватися до змін у вимогах і легко масштабуватися у відповідь на зростання бізнесу;
- управління ресурсами так, щоб процеси були ефективними та забезпечували необхідну продуктивність [7].

Основними моделями архітектури програмного забезпечення є:

- клієнт-сервісна;
- монолітна;
- мікросервісна.

1.4.1 Клієнт-серверна архітектура

Програми побудовані на клієнт-серверній архітектурі дотримуються розподіленої обчислювальної моделі, де функціональні можливості програми розподілені між двома основними компонентами: клієнтом і сервером. Клієнт відповідає за представлення інтерфейсу кожного окремого користувача та обробку взаємодії між усіма користувацькими інтерфейсами, а сервер відповідає за обробку запитів і керуванням зберіганням даних [8].

Характеристики:

- клієнти та сервери спілкуються через мережу;
- централізоване зберігання даних відбувається на сервері;
- клієнти можуть бути легкими та працювати на різних пристроях (наприклад, комп'ютер, мобільний);
- кілька клієнтів можуть взаємодіяти з одним сервером одночасно [8].

Випадки використання:

Клієнт-серверні програми зазвичай використовуються для веб-додатків, служб електронної пошти, платформ обміну файлами та багатьох інших сценаріїв, коли кільком користувачам потрібен доступ до спільних ресурсів [8].

1.4.2 Монолітна архітектура

Монолітна архітектура — це традиційний підхід, коли всі компоненти програми тісно інтегровані в єдину кодову базу та розгорнуті як єдине ціле [8].

Переваги даної системи:

- простий у розробці та тестуванні на початкових етапах;

- легко обслуговування в простих застосунках [8].

Недоліки:

- важко масштабувати систему в складних проектах;
- зміни в одній частині програми можуть вплинути на весь моноліт, через що, виникають складності при підтримці проекту [8].

1.4.3 Мікросервісна архітектура

Архітектура мікросервісів — це підхід, при якому додаток розбивається на невеликі незалежні сервіси, кожен з яких орієнтований на конкретну бізнес-можливість [8].

Дана архітектура має наступні переваги:

- відокремлені та незалежні послуги;
- служби спілкуються один з одним через чітко визначені API;
- кожен сервіс може розроблятися і підтримуватися окремими командами;
- простіша масштабованість і гнучкість, оскільки сервіси можна масштабувати незалежно;
- легше виконувати технічне обслуговування та оновлення, оскільки можна вносити зміни в окремі служби, не впливаючи на всю програму [8].

Недоліки:

- вимагає керування розподіленою системою, що може ускладнити зв'язок і моніторинг;
- додаткові накладні витрати на керування кількома службами [8].

1.5 Вибір технологій для серверної частини застосунку

Серверну частину застосунку прийнято називати бекендом (backend). Він відповідає за обробку даних, здійснення бізнес-логіки, взаємодію з базою даних та іншими зовнішніми сервісами, а також за підготовку та відправку відповідей на клієнтські запити.

Розробка бекенду відбувається на одній або кількох технологіях (мовах програмування) та їх фреймворках, кожна з яких має свої відмінні переваги та недоліки, котрі визначаються наявними ресурсами, часом та критеріями масштабності проекту.

Найпопулярнішими мовами програмування бекенду є:

- JavaScript;
- Java;
- PHP;
- Python [9].

1.5.1 JavaScript

JavaScript - це мова програмування, яка використовується для розробки веб-додатків та веб-сторінок. Вона є однією з найпоширеніших мов програмування в Інтернеті і, в своїй базовій версії, використовується для динамічного змінення вмісту веб-сторінок та інтерактивності з користувачем.

Node.js - це відкрите середовище виконання, яке дозволяє працювати JavaScript на серверному боці. В основі Node.js лежить двигун V8, який також використовується в браузері Google Chrome для виконання JavaScript [10]. Node.js надає зручні і потужні інструменти для розробки серверних програм, веб-додатків та мережевих додатків.

Node.js має багато переваг, але також і деякі недоліки, які варто враховувати.

Переваги:

- Node.js використовує асинхронний, неблокуючий ввід/вивід, що дозволяє ефективно обробляти багато процесів одночасно;
- має велику кількість сторонніх модулів, які дозволяють розширювати функціональність додатків і прискорюють розробку;
- Node.js має велику активну спільноту розробників, що сприяє розвитку екосистеми, розробці нових інструментів та вирішенню типових проблем проблем;

- Node.js може бути використаний як для розробки серверних додатків, так і для створення веб-додатків, API, мікрослужб та інших додатків.

Недоліки Node.js:

- використання Node.js для обробки великих обчислювальних завдань (CPU-інтенсивних) може призвести до затримок в роботі програми через однопотоковий режим виконання;

- під великим навантаженням програма на Node.js може швидко збільшувати використання пам'яті, що може призвести до перевантаження сервера.

1.5.2 Java

Java — це багатоплатформна, об'єктно-орієнтована та мережево-орієнтована мова, яку можна використовувати як платформу саму по собі. Це швидка, безпечна та надійна мова програмування для кодування будь-чого: від мобільних додатків і корпоративного програмного забезпечення до програм для великих даних і серверних технологій [12].

Переваги Java:

- Java працює на основі віртуальної машини Java (JVM), що дозволяє виконувати код на різних платформах без змін;

- Java має вбудовані механізми безпеки, такі як управління пам'яттю та вбудована обробка винятків, що дозволяє розробникам створювати безпечні додатки;

- Java використовує об'єктно-орієнтований підхід, який дозволяє створювати модульний та легко зрозумілий код.

Недоліки Java для backend:

- високі вимоги до ресурсів - запуск JVM може вимагати великих обсягів пам'яті та процесорних ресурсів, що може бути проблемою для обмежених середовищ або великої кількості одночасних підключень;

- запуск Java-додатків може бути повільним через процес ініціалізації JVM та завантаження бібліотек;

- Java може виглядати складною, в порівнянні з аналогами, через свою велику кількість ключових слів, паттернів та концепцій;

- написання коду на Java може займати більше часу порівняно з іншими мовами, оскільки вона має строгу типізацію та вимагає більше коду для досягнення тих самих результатів.

1.5.3 PHP

PHP — це серверна мова програмування з відкритим вихідним кодом, яку можна використовувати для створення веб-сайтів, програм, систем управління взаємовідносинами з клієнтами тощо. Це широко використовувана мова загального призначення, яку можна вставляти в HTML. Ця функція з HTML означає, що мова PHP залишається популярною серед розробників, оскільки вона допомагає спростити код HTML [13].

PHP залишається популярною мовою програмування протягом майже трьох десятиліть завдяки численним перевагам, які вона пропонує користувачам і розробникам [13]. Найзначніші з них:

- PHP можна запускати на будь-якій основній операційній системі, тому вам не потрібно турбуватися про сумісність;

- PHP може допомогти спростити проекти на HTML;

- як мову, її відносно легко вивчити, завдяки простоті написання, ніж інші альтернативи;

- оскільки продуктивність веб-сайту все більше і більше залежить від швидкості, швидке завантаження PHP дійсно може мати визначну роль при її виборі[13].

Недоліки PHP:

- однією з поширених скарг більшості розробників на PHP є те, що вона має обмежені інструменти налагодження. Вона погано обробляє помилки, особливо в порівнянні з іншими мовами сценаріїв;

- PHP як мова сценаріїв не допускає будь-яких модифікацій або змін у рамках основної поведінки веб-додатків. Це означає, що у розробника будуть певні обмеження в спробах досягти нових результатів;

- оскільки РНР є відкритим кодом, це також означає, що він має текстовий файл ASCII, до якого легко отримати доступ. По суті, це означає, що який би код не написали розробники, широка публіка може легко побачити його разом із усіма помилками, які містить код. Це також означає, що вони можуть використати ці помилки проти розробника [14].

1.5.4 Python

Python — це мова комп'ютерного програмування, яка часто використовується для створення веб-сайтів і програмного забезпечення, автоматизації завдань і аналізу даних. Python є мовою загального призначення, тобто її можна використовувати для створення різноманітних програм і не спеціалізується на конкретних проблемах. Ця універсальність, а також зручність для початківців зробили її однією з найбільш використовуваних мов програмування сьогодні [15].

Переваги Python:

- є чудовою платформою для початківців. Це багатослівна мова, оскільки читання на Python дуже схоже на читання англійською. Мові не потрібні фігурні дужки, щоб визначити блоки. Однак відступи є обов'язковими, що додатково сприяє читабельності коду;

- Python надзвичайно гнучкий і може бути розширений іншими мовами. Розробники можуть писати код мовами C і C++ і створювати нові функції мовою динамічної типізації;

- містить великий набір бібліотек і код для різних цілей, наприклад: веб-програмування, редактура зображень, CGI, робота з виразами тощо;

- Python є розширюваним, і більшість його кодів можна навіть написати на інших мовах, таких як C++. Це дозволяє додати можливості створення сценаріїв до коду Python іншою мовою;

- Python є масштабованим, і на ринку представлено багато програм, які підтверджують це. Instagram і Pinterest є двома найпопулярнішими програмами, які успішно отримують мільйони запитів користувачів;

- Python — правильний вибір, коли потрібні веб-проекти, інтегровані з алгоритмами машинного навчання [16].

Недоліки Python:

- Python повільний порівняно з іншими некомпільованими мовами, оскільки вимагає великої обчислювальної потужності. Отже, це те, на що потрібно звернути увагу, перш ніж вибрати Python;

- Python не є 100% безпечним. Потрібно вжити необхідних заходів для забезпечення безпеки коду. Однак виконання правильного тестування якості може вирішити цю проблему;

- перш ніж працювати з Python, необхідно налаштувати робоче середовище. Багато проблем може виникнути під час налаштування середовища, що вплине на досвід програмування початківців;

- вимагає значного використання пам'яті, і під час проекту слід ретельно підходити до цього. Він використовує підрахунок посилань у збиранні сміття, що часто призводить до потенційних втрат пам'яті;

- багатопотоковість у Python не зовсім справжня, через її глобальне блокування інтерпретатора (GIL). Тут один потік може одночасно містити GIL, що чітко означає, що це не точна багатопотоковість [16].

1.6 Вибір технологій для клієнтської частини застосунку

При програмуванні клієнтської сторони, котру прийнято називати фронтенд (frontend), не стає питання вибору мови програмування, оскільки всі сучасні рішення, так чи інакше, працюють виключно на трьох технологіях:

- HTML (розшифровується як HyperText Markup Language – Мова Гіпертекстової Розмітки) - це стандартна мова розмітки для створення веб-сторінок. Він дозволяє створювати та структурувати розділи, абзаци та посилання

за допомогою елементів HTML (будівельних блоків веб-сторінки), таких як теги та атрибути [17];

- CSS (Cascading Style Sheets – Таблиці Каскадних Стилів) - це мова таблиць стилів, яка використовується для опису вигляду та форматування документа, написаного на HTML. Це важливо для веб-дизайну, оскільки дозволяє розробникам контролювати типографічні характеристики, властивості кольору та загальний макет веб-сторінок. CSS можна використовувати внутрішньо в елементі стилю, вбудованого в елементи HTML або зовні через зв'язані файли CSS. З моменту його розробки в 1996 році він зазнав удосконалень, щоб відповідати зростаючим вимогам Інтернету, забезпечуючи послідовний вигляд у різних браузерах і пристроях. Незважаючи на складність деяких функцій CSS, його основною функцією є відокремлення вмісту від презентації, що робить веб-сайти більш доступними та естетично привабливими [18];

- JavaScript, котрий був докладніше розглянутий при виборі мови програмування для бекенду, в контексті програмування клієнтської сторони сайтів – це мова програмування, яка використовується, щоб зробити веб-сайти та програми динамічними та інтерактивними. Він унікальний, оскільки його можна запускати безпосередньо у браузері, а не лише на сервері [19].

В чому ж питання дійсно стає – це вибір архітектури веб-застосунку, де постає вибір між двома варіантами: MPA (Multi-Page Application – багатосторінкові застосунки) та SPA (Single-Page Application – односторінкові застосунки).

MPA - це тип веб-програми, яка завантажує нову сторінку щоразу, коли користувач виконує певну дію. Ці програми вважаються традиційними веб-програмами, оскільки вони існують довше та функціонують як традиційні веб-сайти [20].

Коли користувач виконує певну дію в MPA (наприклад, натискає кнопку чи гіперпосилання), програма переспрямовує його на нову сторінку та завантажує цю сторінку з нуля. З цієї причини MPA складаються з кількох

окремих статичних веб-сторінок HTML, на відміну від однієї сторінки, яка динамічно оновлюється щоразу, коли виконується певна дія [20].

Компанії, які продають декілька різних продуктів і послуг одночасно на одному сайті, зазвичай використовують архітектуру MPA, оскільки вона дає їм змогу персоналізувати вигляд і відчуття кожної статичної сторінки відповідно до специфіки конкретної частини бізнесу [20].

SPA — це веб-додаток або веб-сайт, який працює у веб-браузері та завантажує все з однієї сторінки. Це означає, що користувач взаємодіє з усім у додатку на одному екрані, а не переходить з однієї сторінки на іншу, як на традиційному веб-сайті чи MPA. Коли користувач відкриває SPA, програма використовує динамічне перезаписування в JavaScript, щоб поступово оновлювати те, що зображено на екрані, залежно від певних дій, які виконуються [20].

Соціальні медіа-платформи, такі як Facebook і X (раніше Twitter), є хорошими прикладами SPA; коли користувач прокручує свій канал Facebook або X, програма постійно оновлюватиметься та показуватиме йому більше вмісту, не перезавантажуючи сторінку чи не перенаправляючи його на нову. Ця ж логіка застосовується, коли відбувається перехід на інші сторінки програми. Замість повного перезавантаження програми SPA перезавантажить лише розділ основного вмісту програми, зберігаючи той самий верхній колонтитул, меню та нижній колонтитул, що й на будь-якій іншій сторінці [20].

При створенні веб-застосунку з архітектурою SPA зазвичай використовується одна з трьох технологій: Angular, Vue.js або React.

1.6.1 Angular

Angular - це веб-фреймворк, розроблений компанією Google, який дозволяє розробляти клієнтські веб-додатки та односторінкові додатки. Він базується на мові програмування TypeScript і використовується для створення динамічних та інтерактивних веб-інтерфейсів.

Переваги:

- Angular надає структуру, яка сприяє чітко визначеній та організованій методології веб-розробки. Програмна система дотримується архітектурного

шаблону Model-View-Controller (MVC), який підтримує чіткий розподіл обов'язків. Це полегшує здатність розробників ефективно структурувати та підтримувати свій код, що призводить до створення більш оптимізованих та адаптованих програм;

- вбудований фреймворк залежностей Angular полегшує безперебійне керування та впровадження взаємодії між різними компонентами. Ця практика покращує можливість багаторазового використання та тестування коду, уможливорюючи заміну залежностей за потреби;

- двостороння архітектура прив'язки даних Angular сприяє автоматизованій синхронізації між моделлю та його виводі в додатку. Це означає, що зміни, внесені до моделі, негайно відображаються в поданні, і, навпаки, мінімізується потреба в маніпуляціях з об'єктною моделлю документа (DOM) розробником;

- інтерфейс командного рядка Angular (CLI) — це комплексний інструмент, який спрощує процес ініціалізації, розробки та тестування проекту. Інструмент створює стандартизований шаблон коду та пропонує ряд інструкцій для створення компонентів, модулів, служб та інших елементів, таким чином підвищуючи ефективність розробників і зменшуючи їх робоче навантаження;

- Angular розроблено за допомогою TypeScript, який є статично типізованим розширенням JavaScript. TypeScript полегшує виявлення помилок протягом усього процесу розробки та пропонує розширену підтримку інструментів, отже, спрощуючи підтримку та масштабованість проектів [21].

Недоліки:

- крива навчання Angular більш виражена в порівнянні з альтернативами. Широкий набір функцій і складний дизайн можуть збентежити, що може призвести до тривалого періоду адаптації;

- програми Angular іноді мають більшу кількість шаблонного коду порівняно з іншими фреймворками, що призводить до більшої кодової бази, яку може бути складніше підтримувати. Незважаючи на те, що інтерфейс командного рядка (CLI) допомагає генерувати код, він, тим не менш, може створити певні недоліки для деяких розробників;

- використання двостороннього зв'язування даних Angular і широких можливостей фреймворку може призвести до збільшення продуктивності, особливо під час роботи з великими програмами. Для вирішення цієї проблеми часто потрібна ретельна оптимізація та налаштування продуктивності;

- процес оновлення між основними версіями Angular може спричинити труднощі через потенційну потребу в суттєвих модифікаціях коду. Це явище може призвести до проблем із сумісністю із зовнішніми бібліотеками та потребує додаткових зусиль з обслуговування;

- додатки Angular іноді демонструють більший розмір комплекту, що призводить до подовження початкового часу завантаження для клієнтів. Це може бути не оптимальним для ініціатив, які мають суворі критерії ефективності [21].

1.6.2 Vue.js

Vue.js - це прогресивний JavaScript-фреймворк для створення користувацьких інтерфейсів (UI) та односторінкових додатків. Він використовується для розробки веб-додатків та мобільних додатків з використанням компонентної архітектури.

Переваги:

- дуже важливою особливістю, яку підкреслюють багато розробників Vue.js, є його малий розмір і продуктивність. Основний пакет gzipped важить лише 18 Кб. Фреймворк розроблений таким чином, щоб бути продуктивним без додаткової оптимізації. Крім того, він надає вбудовані директиви, такі як v-once та v-memo, які допомагають підвищити продуктивність;

- Vue легко зрозуміти та засвоїти. Крива навчання менша, а концепції розвитку прості для розуміння. Це допомагає розвивати масштабні проекти. Справа в тому, що Vue перейняв найкращі концепції React і Angular.

- згідно з попереднім пунктом, найбільш корисним у вивченні Vue і процесі розробки є велика документація;

- Vue легко інтегрувати на відміну від інших фреймворків JavaScript – його можна використовувати не лише для створення програм, але й для включення

компонентів у існуючі програми. Існують бібліотеки, створені, щоб допомогти цьому процесу та зробити його ще простішим.

- Vue 3 представляє нову функцію - Composition API, яка схожа на React Hooks. Vue 3 повністю переробив реактивність, яка тепер передає дані за посиланням і створює автономні обчислені значення. Новий API можна використовувати в кожному фрагменті коду за допомогою бібліотеки `@vue/reactivity` [22].

Недоліки:

- однією з проблем Vue є те, що його найбільша спільнота розташована в Китаї, що може бути проблематичним у випадку деяких пакетів або досліджень. Є багато пакетів, зроблених китайськими розробниками Vue.js, які пишуть документацію рідною мовою. Численні дослідження предметів Vue можуть призвести до китайських форумів і дискусій. Це проблематично для англomовних розробників. Варто згадати той факт, що ця проблема повільно вирішується в міру зростання глобальних спільнот Vue;

- великі проекти можуть призвести до проблем із невідповідністю коду. У цій ситуації розробникам Vue.js потрібно витратити час на узгодження коду, інакше різні підходи можуть призвести до більш серйозних проблем, таких як різні збої в роботі додатків і через що, як наслідок, складніше зайти в розробку проекту;

- порівняно з React і Angular, розробники Vue.js можуть відчувати нестачу плагінів і бібліотек [22].

1.6.3 React

React - це бібліотека JavaScript для розробки інтерфейсів користувача, створена компанією Facebook. Вона використовується для побудови ефективних та динамічних веб-додатків з використанням компонентної архітектури.

React має наступні переваги:

- React дозволяє швидко писати код без великого шаблону та багатьох складних бібліотек. Це важливо, коли розробник хоче за короткий час створити невелику програму MVP;

- після кожного оновлення React API залишається майже незмінним, тобто зберігається сумісність версій. Навіть якщо вносяться значні зміни, буде виставлено попередження протягом тривалого часу, тому розробник може мати достатньо часу щоб змінити свій код;

- розробники React можуть створювати повторно використовувані компоненти. Це дозволяє створювати невеликі фрагменти інтерфейсу користувача та включати їх у будь-яке місце програми. Завжди можна поєднати багато менших компонентів, щоб створити більший складний інтерфейс користувача. Досить часто розробники створюють додаток, використовуючи готові компоненти інтерфейсу користувача, наприклад, MaterialUI;

- Virtual DOM (Document Object Model) — відповідає за підтримку абстракції програми React і синхронізацію в актуальному стані з реальним DOM (це те, що дійсно показано на екрані). Це дозволяє підвищити продуктивність і швидкість додатка, оскільки реалізація React Virtual DOM порівнює абстракцію додатка між змінами та повторно відтворює лише змінені частини додатка [23].

Недоліки:

- у React немає роздільної логіки компонента та перегляду за замовчуванням (як це є в Angular);

- програми React засновані на сторонніх бібліотеках. Це одночасно і перевага, і недолік. React — це бібліотека, а не фреймворк, і тому немає офіційних бібліотек для обробки загальних функцій інтерфейсних програм, як, наприклад, маршрутизація, HTTP-запити тощо. Це може бути перевагою, оскільки є свобода до вибору компонентів, через що є можливість вибрати найкращі інструменти для своїх потреб. Але це все ж недолік, тому що React не має багатьох базових інструментів із коробки. Навіть якщо розробник використовує лише найпопулярніші та найпоширеніші бібліотеки, у нього може виникнути конфлікт версій React між версією бібліотеки та версією програми React (здебільшого йдеться про застарілі програми);

- React не нав'язує жодних домовленостей щодо розробки. Ознайомлення нових розробників із застарілим проектом займає більше часу, оскільки дуже

часто розробники JavaScript створюють програми React різними способами, і існує висока ймовірність того, що відбудеться зіткнення із застарілим проектом з умовами кодування, які були в React значно раніше і вже застаріли. Це змушує команду розробників витрачати час на обговорення деяких загальних правил розробки в проекті. Коли команда змінюється багато разів, це може призвести до безладного коду, оскільки всі мають різні звички та дотримуються різних умов.

1.7 Вибір СКБД

Вибір системи керування базами даних (СКБД) залежить від багатьох факторів, таких як тип додатку, обсяг даних, вимоги до продуктивності, доступність та вартість рішення. Мною було розглянуто наступні варіанти СКБД: MySQL, MongoDB, PostgreSQL.

1.7.1 MySQL

MySQL є однією з найбільш авторизованих систем баз даних у всьому світі. За даними Data Base Journal Research, він займає 44,04% відносної частки ринку інструментів керування базами даних [24].

Переваги MySQL:

- MySQL — це програмне забезпечення з відкритим вихідним кодом, що означає, що воно доступне для вільного використання, модифікації та розповсюдження. Ця характеристика робить його доступним для широкого кола користувачів, від індивідуальних розробників до великих підприємств, без сплати великих ліцензійних зборів;

- MySQL пропонує параметри масштабованості, що дозволяє користувачам легко обробляти зростаючі обсяги даних і зростаючі вимоги до робочого навантаження. Завдяки таким функціям, як шардинг, реплікація та кластеризація, MySQL може ефективно масштабуватися як вертикально (додаючи більше

ресурсів на один сервер), так і горизонтально (розподіл даних між кількома серверами);

- MySQL відомий своєю високою продуктивністю, особливо в програмах, які потребують великої кількості читання. Він використовує різні методи оптимізації, такі як індексування, кешування та оптимізація запитів, щоб забезпечити швидкий час відповіді навіть із великими наборами даних;

- MySQL відомий своєю надійністю та стабільністю. Він забезпечує цілісність даних завдяки таким функціям, як відповідність ACID (Atomicity, Consistency, Isolation, Durability), підтримку транзакцій і механізми відновлення після збою, що робить його придатним для критично важливих програм [25].

Недоліки:

- незважаючи на те, що MySQL має багато функцій, у ній може бути відсутнім деякі розширені функції порівняно з іншими базами даних корпоративного рівня. Наприклад, він може не пропонувати вбудовану підтримку складних типів даних, повнотекстовий пошук або розширені аналітичні можливості без додаткових плагінів або розширень;

- горизонтальне масштабування, може створити складнощі, пов'язані з узгодженістю та синхронізацією даних;

- нативна реплікація MySQL дотримується єдиної головної моделі, де один сервер (головний) обробляє операції запису, а інші (підлеглі) асинхронно копіюють дані для операцій читання. Ця архітектура може створювати проблемні місця та окремі точки відмови, особливо в налаштуваннях із високою доступністю;

- хоча сам MySQL має відкритий вихідний код, для деяких розширених функцій, інструментів керування та служб підтримки підприємства можуть знадобитися платні підписки або ліцензії від постачальників MySQL, таких як Oracle.

1.7.2 MongoDB

MongoDB - це документ-орієнтована система керування базами даних, яка призначена для зберігання, запитів та аналізу різних по об'єму обсягів даних.

MongoDB використовує JSON-подібні документи з динамічними схемами та відноситься до сімейства баз даних NoSQL, оскільки вона не використовує традиційні реляційні таблиці.

Переваги:

- MongoDB не вимагає попередньо визначених схем та пропонує більшу гнучкість, ніж традиційні реляційні бази даних, які суворо вимагають схем;
- документоорієнтований підхід MongoDB узгоджується з динамічними запитами. Це дозволяє виконувати гнучкі та різноманітні операції із запитами на основі характеру документів, на відміну від статичних запитів на основі таблиць RDBMS;
- порівняно з реляційними базами даних, оптимізація продуктивності MongoDB є порівняно простішою завдяки її архітектурі та можливості внутрішнього керування даними;
- MongoDB використовує внутрішню пам'ять для зберігання даних. Таким чином, він дуже швидко отримує доступ до даних і підвищує загальну продуктивність;
- за допомогою MongoDB можна горизонтально розширювати базу даних, розподіляючи дані між кількома серверами (шардинг). Шардинг забезпечує поділ даних і рівномірний розподіл між сегментами, що дуже ефективно для пошуку даних;
- зазвичай вважається, що MongoDB легше підтримувати, ніж традиційні бази даних, завдяки своїй гнучкій схемі та простішим процесам оптимізації;
- роблячи копії даних і розподіляючи роботу між різними частинами, MongoDB гарантує, що інформація завжди доступна, а система працює дуже швидко. Це відбувається тому, що завдання розподіляються між багатьма місцями, а не лише в одному, що робить роботу швидшою та надійнішою [26].

Недоліки:

- у MongoDB транзакції працюють із кожним фрагментом даних (документом), але вони не повністю охоплюють ситуації, коли потрібно робити

кілька речей одночасно з великою кількістю даних. Це може бути негативним фактором для додатків, в яких необхідно, щоб усе відбувалося одночасно;

- хоча MongoDB пропонує атомарність, узгодженість, ізоляцію та довговічність (ACID) на рівні документа, вона не забезпечує повної відповідності ACID для кількох документів або колекцій. Це обмеження може бути складним для програм, які вимагають суворих і складних гарантій транзакцій;

- MongoDB зберігає імена ключів з кожною парою значень, що спричиняє деяку надлишковість даних через обмеження об'єднань. Ця надмірність може призвести до збільшення використання пам'яті порівняно з аналогами;

- MongoDB накладає обмеження на максимальний розмір документа 16 Мб. Більші документи, можливо, доведеться обробляти інакше або розділити на менші документи, щоб відповідати цьому обмеженню;

- вкладеність документів у MongoDB можлива, але обмежена максимум 100 рівнями. Це обмеження може вплинути на те, наскільки глибоко є можливість організувати та структурувати дані в документах [26].

1.7.3 PostgreSQL

PostgreSQL — це об'єктно-реляційна система керування базами даних (ORDBMS), яка є однією з найбільш розповсюджених і високо оцінених відкритих СКБД у світі. Вона надає широкий спектр можливостей для зберігання та обробки даних з використанням реляційної моделі даних.

Переваги:

- PostgreSQL надає транзакційний DDL. Це не просто INSERT, UPDATE або DELETE для транзакції, це буквально можливість створення нової таблиці, таблиці відкидання, автоматичної таблиці тощо. Це дуже відрізняється від того, як працює Oracle;

- на відміну від інших програм баз даних, PostgreSQL надає коментарі до коду. Коментарі допомагають побачити, що конкретний код робить, перш ніж заблокувати його для програми. Це означає, що простіше зрозуміти дизайн і стежити за інженерним процесом. Це дає безпеку, якість і здатність розвивати спільноту, оскільки більше людей можуть бачити, що відбувається з кодом;

- у багатьох системах баз даних є можливість встановити параметри середовища, але вони використовуються на рівні всієї бази даних, в той час як PostgreSQL надає змогу налаштовувати гнучкі параметри до кожного елементу бази даних окремо;

- розширюваність — це цілий набір різних аспектів PostgreSQL. Це дозволяє створювати функції, типи даних, мови, зміни, і надалі інсталиувати їх на сервер бази даних, просто завдяки команді створення розширення, а потім PostgreSQL автономно зробить зміни;

- PostgreSQL має функції, пов'язані з безпекою, а також розширення, які можна використовувати для підвищення безпеки. З точки зору безпеки параметрів, якщо є необхідність заблокувати свою систему бази даних, вона надає конфігурації на рівні ОС, які можна налаштувати для блокування середовища навколо бази даних. З точки зору безпеки програми, вона забезпечує безпеку на основі привілеїв користувача, розділяючи облікові записи як лише для читання, читання/запису або інші дії залежно від категорії. Окрім простого надання дозволу певному користувачеві на доступ до елементів, також можна і робити заборони на використання певних частин функціоналу [27].

Недоліки:

- під час пошуку запиту PostgreSQL через свою структуру реляційної бази даних має починати з першого рядка, а потім прочитати всю таблицю, щоб знайти відповідні дані. Тому вона працює повільніше, особливо коли в рядках і стовпцях таблиці зберігається велика кількість даних, що містить багато полів додаткової інформації для порівняння [27];

- для кожного підключення клієнта в PostgreSQL створює новий процес із виділенням пам'яті 10 МБ. Це може здатися небагато. Але, коли використовується кілька з'єднань, сервер може вийти з ладу, особливо якщо він має обмежений обсяг пам'яті [28].

1.8 Висновки за розділом

Після аналізу предметної області, дослідження типових архітектур та порівняння мов програмування, я визначився яку структуру та програмні засоби буду використовувати при виконанні практичної частини.

Сам проект «Комп'ютерної системи для продажу квитків для кінотеатру» матиме підсумковий результат рівний за функціоналом до веб-застосунку «Кінотеатру ім. Довженка», в якому можливо дослідити базові принципи побудови подібних застосунків та перевірити накопичені знання, отримані під час навчання по спеціальності.

Задля забезпечення максимальної зрозумілості та можливості контролювати елементи застосунку роздільно та комплексно, мною була обрана архітектура з принципом «клієнт-сервер».

Оскільки система з продажу квитків не є надто складною для математичних обчислень і, через що, не займає багато місця в пам'яті, враховуючи усі переваги: Node.js, написана на мові JavaScript – є хорошим вибором для розробки серверної частини застосунку.

Для створення клієнтської сторони проекту, мною була обрана бібліотека React, мова якої є JavaScript та, котра є оптимальною за функціоналом, оскільки не вимагає дуже складних графічних та різних за направленістю багатосторінкових рішень (для котрих більше підходить Angular), та спрощує процес створення відносно варіанту з використання стандартної зв'язки HTML-CSS-JavaScript.

В якості СКБД я обрав PostgreSQL, оскільки при її використанні можна спрощено, завдяки бібліотекам Node.js, користуватися не SQL-запитами, а простими командами на мові JavaScript, та передивлятися таблиці в класичному вигляді завдяки підпрограмі pg Admin 4.

2 МОДЕЛЮВАННЯ СИТЕМИ

2.1 Загальні вимоги

Головна мета створення комп'ютерної системи з замовлення квитків для кінотеатру – є надання швидкого, дистанційного та зручного доступу користувача до фільмів та сеансів, задля оформлення бронювання місця в кінотеатрі. Застосунок матиме назву “СМooVI”, із звучною аналогією “See movie” (Дивитися фільм). Головні вимоги, котрим повинна відповідати дана система наступні:

- мати зрозумілий інтерфейс;
- безпечно передавати дані користувача по мережі та безпечно зберігати дані у базі даних;
- швидко працювати та стабільно виконувати роботу;
- коректно відображатися на різних пристроях (мати адаптивність).

2.2 Вимоги до апаратного забезпечення

Оскільки система розгортається в інтернеті, то для коректної роботи користувачеві необхідний лише доступ до браузеру, наприклад, Google Chrome, Edge, Safari, Firefox, Atom тощо. Мінімальні системні вимоги для цих браузерів для стаціонарних десктопних пристроїв наведено в таблиці 2.1.

Таблиця 2.1 – Системні вимоги для роботи системи

Параметр	Вимога
Процесор	32-розрядний (x86) або 64-розрядний (x64) із тактовою частотою 1 ГГц.
Оперативна пам'ять	512 Мб
Диск	20 Гб (x64) або 16 Гб (x86), звичайний або твердотільний, що містить не менше 15 Гб вільного простору
Графічний адаптер	З підтримкою DirectX 9 та 32 Мб пам'яті.

У випадку мобільних пристроїв рекомендовано мати смартфон з версією android 8.0 та вище, або iOS версії 7 та вище.

2.3 Функціональні вимоги до системи та розподілення ролей

Увесь функціонал, котрий буде використовувати застосунок, можна розбити на три види доступу, котрі залежать від ролі користувача, а саме:

- адміністратор – представник кінотеатру, котрий має можливість додавати та змінювати контент доступний на сайті;
- авторизований користувач – має розширені функції використання сайту, та збережену історію взаємодії з ним;
- неавторизований користувач – може виконувати мінімум задач.

Неавторизований користувач має наступні можливості:

- обирати необхідний фільм та сеанс;
- оформити замовлення, використовуючи як ідентифікатор ім'я та електронну пошту.

Авторизований користувач має наступні можливості:

- всі, що має неавторизований користувач;
- при оформленні замовлення квитка може не вводити свої дані, вони автоматично підтягуються з бази даних;
- має доступ до особистого кабінету, де зберігаються всі його минулі замовлення і де він може завантажити PDF-версію свого квитка.

Адміністратор має наступні можливості:

- всі, що має авторизований користувач;
- може додавати жанр, фільм, сеанс до фільму та видаляти фільм з бази даних.

2.4 Діаграма послідовностей

Для того щоб прослідити всі функціональні можливості застосунку та взаємодію його окремих елементів, мною було створено діаграму послідовностей (рис. 2.1).

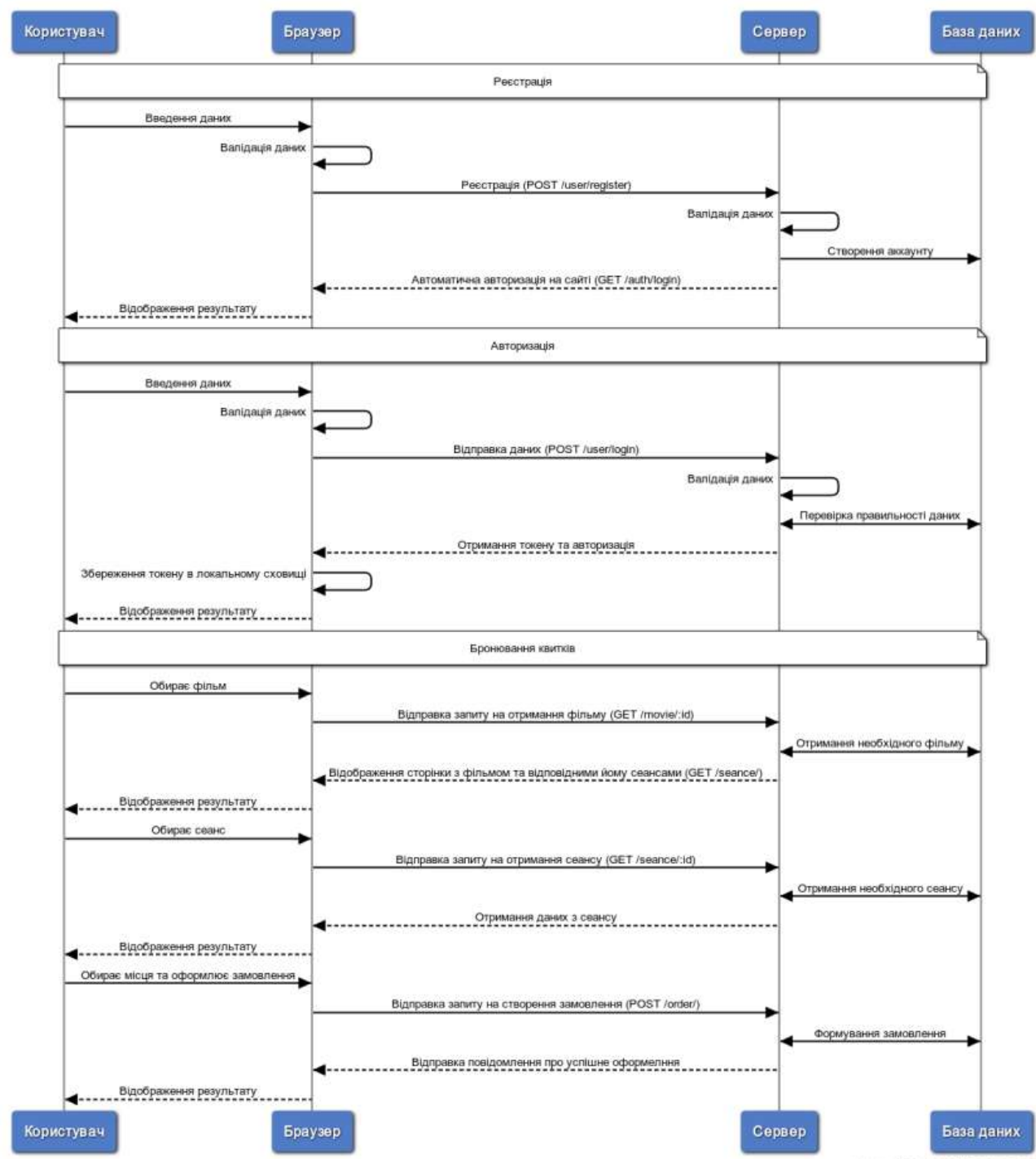


Рисунок 2.1 – Діаграма послідовностей застосунку

2.5 Моделювання бази даних

Головним етапом планування роботи застосунку є створення логічної моделі бази даних та побудова правильної взаємодії між її елементами.

Для правильної роботи бази даних я відокремив та прописав наступні сутності:

- користувач (User) – зберігаються дані користувача;
- фільм (Movie) – дані про фільм;
- зала (Hall) – зали в яких проводяться сеанси, відрізними особливостями яких, є колір;
- місце (Seat) – дані про місця в кінотеатрі;
- сеанс (Seance) – дані про сеанси, котрі відносяться до фільмів;
- замовлення (Order) – дані про замовлення користувача;
- жанр (Genre) – дана сутність відноситься до фільму, і створюється по причині того, що різні фільми можуть мати одні й ті ж жанри;
- жанр-фільм (GenreMovie) – створюється для зберігання співвідношення фільму та жанрів, оскільки один фільм може мати багато жанрів;
- місце-сеанс (SeatSeance) – створюється по причині того, що в багатьох сеансів використовуються одні й ті ж місця та задля додавання такого елемента, як статус місця, атрибут, котрий означає що місце заброньоване або вільне, задля створення зрозумілих умов використання застосунку;
- замовлення-місце-сеанс (OrderSeatSeance) – створюється по причині того, що один і той самий користувач може в одне своє замовлення внести декілька місць.

Всі зв'язки одних сутностей пов'язаних до інших, є використання первинного ключа однієї, в якості зовнішнього ключа до іншої.

Всі сутності та значення їх атрибутів представлено в таблицях 2.2-2.11.

Таблиця 2.2 – Сутність «Користувач»

Атрибут	Тип даних	Значення за умовчанням	Обов'язковість	Унікальність	Ключ
Код користувача	Лічильник	-	Так	Так	Первинний
Ім'я користувача	Текст	-	Так	Так	-
Пошта	Текст	-	Так	Так	-
Роль	Текст	-	Так	Так	-

Таблиця 2.3 – Сутність “Місце”

Атрибут	Тип даних	Значення за умовчанням	Обов'язковість	Унікальність	Ключ
Код місця	Лічильник	-	Так	Так	Первинний
Код залу	Число	-	Так	Ні	Зовнішній
Номер ряду	Текст	-	Так	Ні	-
Номер місця	Текст	-	Так	Ні	-
Ціна	Число	-	Так	Ні	-

Таблиця 2.4 – Сутність “Фільм”

Атрибут	Тип даних	Значення за умовчанням	Обов'язковість	Унікальність	Ключ
Код фільму	Лічильник	-	Так	Так	Первинний
Назва в оригіналі	Текст	-	Ні	Ні	-
Назва на українській	Текст	-	Так	Ні	-
Режисер	Текст	-	Так	Ні	-
Актори	Текст	-	Так	Ні	-
Рік виходу	Число	-	Так	Ні	-

Продовження таблиці 2.4

Атрибут	Тип даних	Значення за умовчанням	Обов'язковість	Унікальність	Ключ
Країна	Текст	-	Так	Ні	-
Тривалість	Число	-	Так	Ні	-
Опис	Текст	-	Так	Ні	-
Зображення	Текст	-	Так	Так	-

Таблиця 2.5 – Сутність “Жанр”

Атрибут	Тип даних	Значення за умовчанням	Обов'язковість	Унікальність	Ключ
Код жанру	Лічильник	-	Так	Так	Первинний
Назва жанру	Текст	-	Так	Так	-

Таблиця 2.6 – Сутність “Зала”

Атрибут	Тип даних	Значення за умовчанням	Обов'язковість	Унікальність	Ключ
Код залу	Лічильник	-	Так	Так	Первинний
Колір залу (назва)	Текст	-	Так	Так	-

Таблиця 2.7 – Сутність “Сеанс”

Атрибут	Тип даних	Значення за умовчанням	Обов'язковість	Унікальність	Ключ
Код сеансу	Лічильник	-	Так	Так	Первинний
Час початку сеансу	Дата та час	-	Так	Ні	-
Час кінця сеансу	Дата та час	-	Так	Ні	-

Продовження таблиці 2.7

Атрибут	Тип даних	Значення за умовчанням	Обов'язковість	Унікальність	Ключ
Актуальність	Текстовий	-	Так	Ні	-
Код фільму	Число	-	Так	Ні	Зовнішній
Код залу	Число	-	Так	Ні	Зовнішній

Таблиця 2.8 – Сутність “Замовлення”

Атрибут	Тип даних	Значення за умовчанням	Обов'язковість	Унікальність	Ключ
Код замовлення	Лічильник	-	Так	Так	Первинний
Ім'я замовника	Текст	-	Так	Ні	-
Пошта замовника	Текст	-	Так	Ні	-
Код користувача	Число	-	Ні	Ні	Зовнішній
Код сеансу	Число	-	Так	Ні	Зовнішній
Час створення	Дата і час	Поточний час	Так	Ні	-

Таблиця 2.9 – Сутність “Місце-Сеанс”

Атрибут	Тип даних	Значення за умовчанням	Обов'язковість	Унікальність	Ключ
Код місця сеансу	Лічильник	-	Так	Так	Первинний
Код місця	Число	-	Так	Ні	Зовнішній
Код сеансу	Число	-	Так	Ні	Зовнішній
Статус	Текстовий	-	Так	Ні	-
Час замовлення	Дата і час	Поточний час	Так	Ні	-

Таблиця 2.10 – Сутність “Жанр-Фільм”

Атрибут	Тип даних	Значення за умовчанням	Обов'язковість	Унікальність	Ключ
Код жанру фільму	Лічильник	-	Так	Так	Первинний
Код жанру	Число	-	Так	Ні	Зовнішній
Код фільму	Число	-	Так	Ні	Зовнішній

Таблиця 2.11 – Сутність “Замовлення-Місце-Сеанс”

Атрибут	Тип даних	Значення за умовчанням	Обов'язковість	Унікальність	Ключ
Код змовлення-місця-сеансу	Лічильник	-	Так	Так	Первинний
Код замовлення	Число	-	Так	Ні	Зовнішній
Код місця сеансу	Число	-	Так	Ні	Зовнішній

Взаємозв'язок усіх сутностей показано на реляційній моделі бази даних (рис. 2.2)

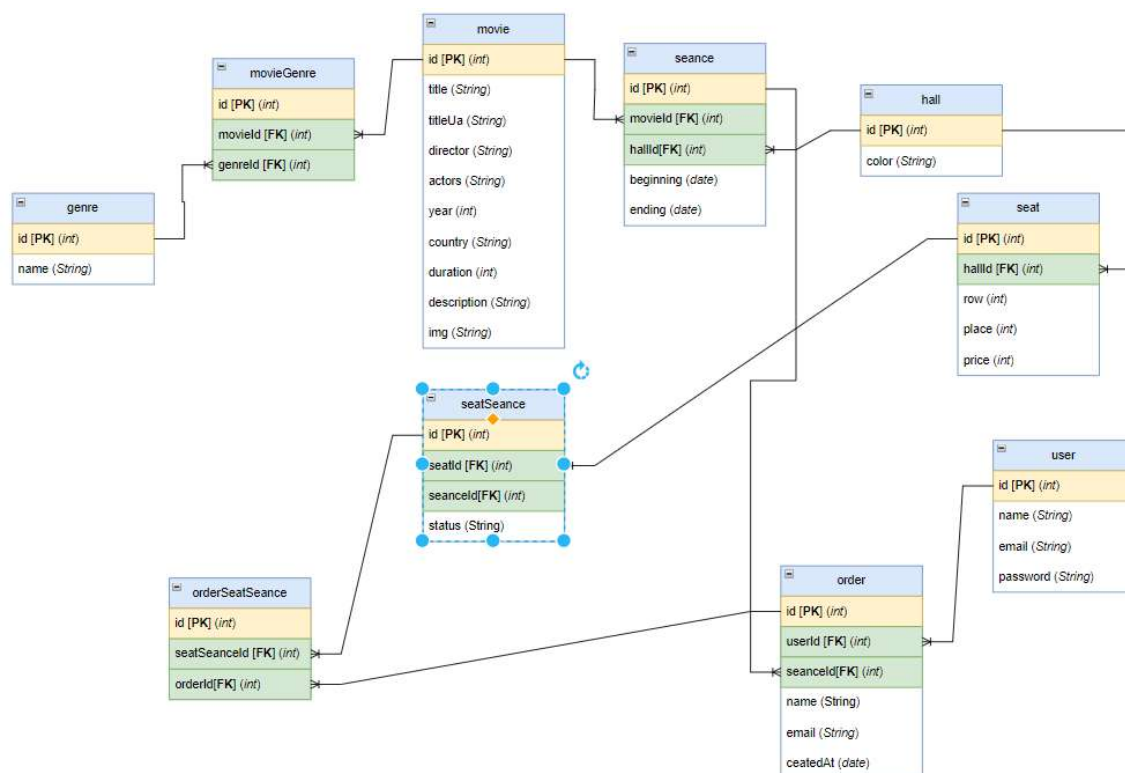


Рисунок 2.2 – Реляційна модель бази даних застосунку

3 РОЗРОБКА СИСТЕМИ

3.1 Створення бази даних

Для створення бази даних в PostgreSQL, я використовував підпрограму pgAdmin 4, яка надає спрощені інтерфейсні рішення взаємодії з базою даних, без необхідності використовувати SQL-запити.

Для того щоб створити базу даних кінотеатру в pgAdmin 4, необхідно просто ініціалізувати нову порожню базу даних, в моєму випадку це «смоові» (рис. 3.1). Всі елементи та моделі встановлюються в базу при першому з'єднанні з сервером.

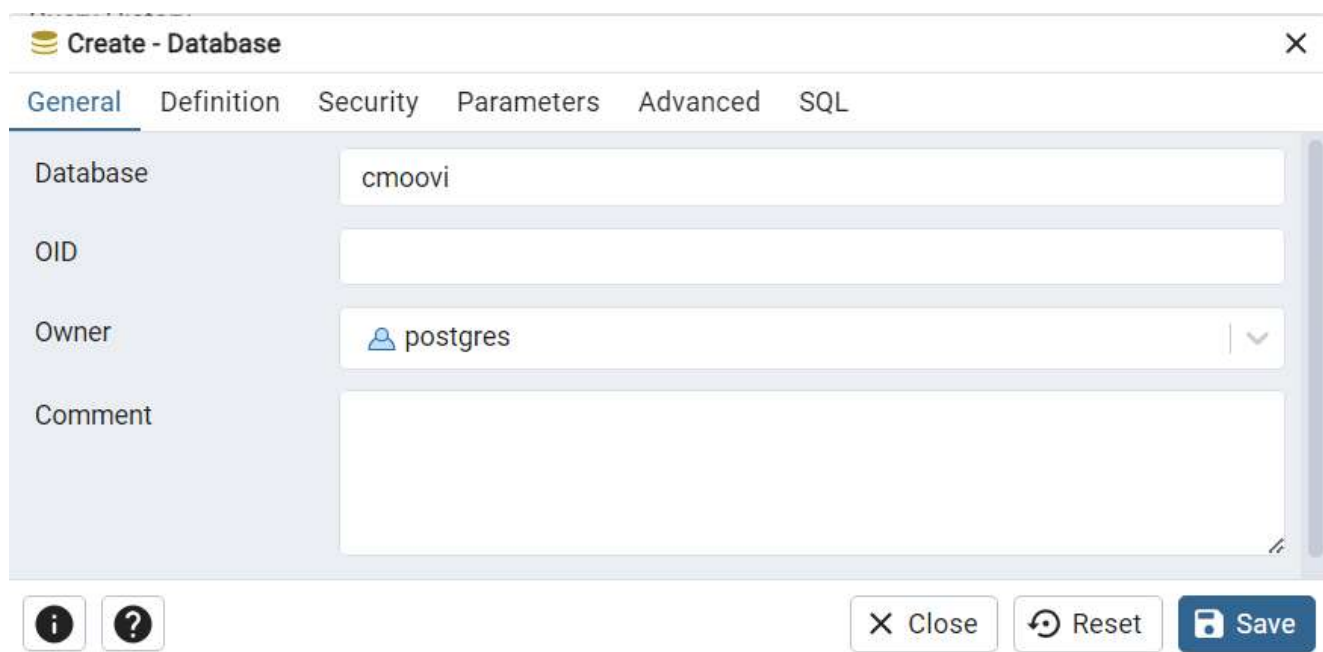


Рисунок 3.1 – Створення нової бази даних в pgAdmin 4

3.2 Розробка серверної частини застосунку

3.2.1 Ініціалізація та встановлення залежностей

Для встановлення усіх необхідних для проекту зовнішніх модулів та бібліотек, використовується `prtm`.

Пакетний менеджер npm (Node Package Manager) — це менеджер для мови програмування JavaScript. Він є стандартним менеджером пакетів для середовища виконання Node.js. Він дозволяє легко встановлювати, оновлювати та управляти залежностями у проєктах.

Перше, що необхідно зробити при створенні застосунку на базі node.js – це проініціалізувати проєкт. Робиться це завдяки введенню в термінал команди “npm init -y”, після чого автоматично створюється файл package.json, де позначено інформацію про проєкт та його залежності.

Наступний крок – встановити залежності проєкту. Робиться це через введення в термінал назв необхідних компонентів (лістинг 3.1).

Лістинг 3.1 – Встановлення залежностей для проєкту

```
npm install express pg pg-hstore sequelize cors dotenv
```

Для свого проєкту я встановив наступні бібліотеки:

- Express — це мінімалістичний і гнучкий веб-фреймворк для Node.js, який спрощує розробку веб-додатків та API. Він надає базову структуру для створення веб-додатків, забезпечуючи необхідні інструменти для обробки HTTP-запитів, маршрутизації, обробки середовища виконання;

- pg — це офіційний клієнт для роботи з PostgreSQL в Node.js. Він забезпечує інтерфейс для взаємодії з базою даних PostgreSQL, дозволяючи виконувати SQL-запити, керувати з'єднаннями та працювати з результатами запитів;

- pg-hstore — це модуль для Node.js, який дозволяє зручно працювати з типом даних hstore в PostgreSQL. Тип даних hstore в PostgreSQL використовується для зберігання пар ключ-значення. pg-hstore дозволяє конвертувати об'єкти JavaScript у формат hstore та навпаки;

- Sequelize — це об'єктно-реляційний маппер (ORM) для Node.js, який спрощує взаємодію з базами даних. Він підтримує різні СКБД, такі як PostgreSQL, MySQL, SQLite та MariaDB, і дозволяє працювати з базою даних за допомогою об'єктів JavaScript замість написання SQL-коду;

- CORS (Cross-Origin Resource Sharing) — це механізм, що дозволяє веб-сторінкам запитувати ресурси з іншого домену, ніж той, з якого була завантажена сторінка. Це є важливим аспектом веб-безпеки, оскільки за замовчуванням браузері забороняють веб-сторінкам запитувати ресурси з доменів, відмінних від того, з якого була завантажена сама сторінка, для запобігання потенційно небезпечних дій;

- dotenv — це модуль для Node.js, який використовується для завантаження змінних середовища з .env файлу в process.env. Це дозволяє легко зберігати конфіденційні дані, такі як паролі, ключі API, конфігураційні налаштування та інші змінні середовища, у файлі замість того, щоб писати напрямку їх у коді додатка.

3.2.2 Налаштування вхідного файлу

Для під'єднання серверу до мережі, мною було створено вхідний файл index.js. Цей файл забезпечує повноцінну конфігурацію та запуск веб-сервера на базі Express з підтримкою CORS, обробки файлів, маршрутизації та обробки помилок, а також підключенням до бази даних через Sequelize. Код файлу index.js викладено в лістингу 3.2.

Лістинг 3.2 – index.js

```
require('dotenv').config()
const express = require('express')
const sequelize = require('./db')
const cors = require('cors')
const fileUpload = require('express-fileupload')
const router = require('./routes/index')
const errorHandler =
require('./middleware/errorHandlingMiddleware')
const path = require('path')

const PORT = process.env.PORT || 7000

const app = express()
app.use(cors())
app.use(express.json())
app.use(express.static(path.resolve(__dirname, 'static')))
app.use(fileUpload({}))
app.use('/api', router)
```

```
//Обробка помилок, останній Middleware
app.use(errorHandler)

const start = async () => {
  try {
    await sequelize.authenticate()
    await sequelize.sync()
  } catch (e) {
    console.log(e)
  }
}

start().then(() =>
  app.listen(PORT, () => console.log(`Server started on port:
${PORT}`))
)
```

В index.js для роботи застосунку використовуються наступні параметри:

- завантажує змінні середовища з файлу .env в process.env, щоб вони були доступні у всьому додатку (лістинг 3.3);

Лістинг 3.3 - .env

```
PORT=7000
DB_NAME=cmoovi
DB_USER=postgres
DB_PASSWORD=fromser2007
DB_HOST=localhost
DB_PORT=5432
SECRET_KEY=super_secret_key199
```

- імпорти компонентів розглянутих в пункті 3.2.1;
- імпортує модуль express-fileupload, який дозволяє обробляти файли, завантажені через форми у веб-додатку;
- імпортує маршрутизатор, який визначає різні маршрути для API;
- імпортує middleware для обробки помилок, який перехоплює та обробляє всі помилки в додатку;
- імпортує вбудований модуль path, який надає утиліти для роботи з шляхами файлової системи;

- встановлює порт, на якому буде працювати сервер. Значення порту береться зі змінної середовища PORT, або використовується порт 7000, якщо змінна не встановлена;

- створює екземпляр додатку Express;

- включає middleware cors, дозволяючи обробляти CORS-запити;

- включає middleware для автоматичного парсингу JSON-даних у вхідних запитах;

- включає middleware для обслуговування статичних файлів (в моєму випадку зображень) з директорії static;

- включає middleware для обробки завантаження файлів;

- включає маршрутизатор, який обробляє всі запити, що починаються з /api;

- включає middleware для обробки помилок, який виконується в самому кінці обробки запитів;

- await sequelize.authenticate() – перевіряє з'єднання з базою даних;

- await sequelize.sync() – синхронізує всі визначені моделі з базою даних;

- викликає функцію start для запуску сервера. Якщо успішно, сервер починає слухати на вказаному порту і виводить повідомлення в консоль.

3.2.3 Підключення до бази даних

Код, представлений в лістингу 3.4, ініціалізує та експортує екземпляр Sequelize, налаштований для підключення до бази даних PostgreSQL з використанням конфігурації, що зберігається у змінних середовища (значення змінних представлено в лістингу 3.3).

Лістинг 3.4 – db.js

```
const {Sequelize} = require('sequelize')

module.exports = new Sequelize(
  process.env.DB_NAME,
  process.env.DB_USER,
  process.env.DB_PASSWORD,
  {
    dialect: 'postgres',
    host: process.env.DB_HOST,
    port: process.env.DB_PORT
```

```
    }  
  )
```

Для підключення передаються наступні дані:

- назва бази даних;
- ім'я користувача;
- пароль користувача;
- діалект (індивідуальний синтаксис окремої СКБД);
- адреса хосту;
- номер порта.

Після запуску додатку, під'єднання до бази даних відбувається автоматично.

3.2.4 Розробка моделі бази даних

Для успішного додавання сутностей до бази даних та для керування ними в межах застосунку, мною було створено файл `models.js` повний код якої, представлено в Лістингу А.1.

На прикладі `Movie` можна побачити, як саме ініціалізуються сутність завдяки `Sequelize`:

- надається ім'я сутності;
- ініціалізується ім'я моделі в базі даних ("movie").

В полі атрибутів проставляються наступні значення:

- назва атрибуту;
- тип даних;
- якщо він є ключем, то прописується `"primaryKey: true"`;
- якщо значення не повинно бути порожнім, прописується `"allowNull: true"`;
- також можна додавати специфічні параметри, що підходять під вимоги конкретного атрибуту та сутності.

На прикладі фрагменту коду, представленому в лістингу 3.5, можна побачити як встановлюється зв'язок між сутностями (в даному випадку багатьох сеансів до одного фільму). Через те, що видалення фільму повинно автоматично видаляти і його сеанси, прописано параметр `{onDelete: 'CASCADE'}`.

Лістинг 3.5 – Зв’язок між сутностями Movie та Séance

```
Movie.hasMany(Séance, { onDelete: 'CASCADE' })
  Séance.belongsTo(Movie)
```

Зв’язок «багато до багатьох» показано на прикладі SeatSéance (лістинг 3.6), де для спрощеної роботи між ними створюються асоціативні ім’я “seats” та “séances”.

Лістинг 3.6 – Зв’язок SeatSéance

```
Séance.belongsToMany(Seat, { through: SeatSéance, as: 'seats'
})
  Seat.belongsToMany(Séance, { through: SeatSéance, as:
'séances' })
```

3.2.5 Побудова маршрутів застосунку

Для побудови API-запитів я розділив логіку через створення файлу index.js (лістинг 3.7), де позначено вхідні точки до кожного маршруту та на окремі роутери для кожної з сутностей (через посилання до них завдяки require) з їх власними методами.

Лістинг 3.7 – index.js

```
const Router = require('express')
const router = new Router()
const userRouter = require('./userRouter')
const movieRouter = require('./movieRouter')
const genreRouter = require('./genreRouter')
const séanceRouter = require('./séanceRouter')
const orderRouter = require('./orderRouter')
const hallRouter = require('./hallRouter')

router.use('/user', userRouter)
router.use('/movie', movieRouter)
router.use('/genre', genreRouter)
router.use('/séance', séanceRouter)
router.use('/order', orderRouter)
router.use('/hall', hallRouter)

module.exports = router
```

На лістингу 3.8 представлено код маршрутів для надсилання запитів сутності “Фільми”.

Лістинг 3.8 – movieRouter.js

```
const express = require('express')
const router = express.Router()
const movieController = require('../controllers/movieController')
const checkRole = require('../middleware/checkRoleMiddleware')

router.post('/', checkRole('ADMIN'), movieController.create)
//checkRole('ADMIN'),
router.get('/', movieController.getAll)
router.get('/:id', movieController.getOne)
router.delete('/:id', movieController.deleteOne)

module.exports = router
```

Докладніше про те, як працюють CRUD-запити даного коду:

- POST /: Додає новий фільм. Перед виконанням контролера movieController.create викликається middleware checkRole('ADMIN'), який перевіряє, чи має користувач роль "ADMIN";
- GET /: Отримує всі фільми, викликаючи метод getAll з movieController;
- GET /:id: Отримує інформацію про один фільм за його ID, викликаючи метод getOne з movieController;
- DELETE /:id: Видаляє один фільм за його ID, викликаючи метод deleteOne з movieController.

3.2.6 Робота контролерів

Принцип роботи контролерів можна розібрати на прикладі контролеру роботи з фільмами movieController.js (повний код викладено в лістингу A.2).

В лістингу 3.9 представлено фрагмент коду що відповідає за створення нового екземпляру фільму.

Лістинг 3.9 – Функція create

```
async create(req, res, next) {
  try {
```

```

        let {title, titleUa, director, actors, year,
country, duration, description, genreId} = req.body

        const {img} = req.files
        let fileName = uuid.v4() + ".jpg"
        await      img.mv(path.resolve(__dirname,      '..',
'static', fileName))

        const movie = await Movie.create({
            title,
            titleUa,
            director,
            actors,
            year,
            country,
            duration,
            description,
            img: fileName
        })
        genreId = JSON.parse(genreId)
        const movieGenre = genreId.forEach(i =>
            MovieGenre.create({
                movieId: movie.id,
                genreId: i.genreId
            })
        )

        return res.json(movieGenre)
    } catch (e) {
        return next(ApiError.internal('Необхідно заповнити
усі поля'));
    }
}

```

В асинхронній функції для створення нового екземпляру фільму можна виділити наступний принцип роботи:

- приймає дані фільму з req.body та зображення з req.files;
- генерує унікальне ім'я для файлу зображення завдяки генератору випадкових імен uuid і переміщує його в папку static;
- створює новий запис у таблиці Movie;
- додає записи в таблицю MovieGenre для зв'язку фільму з жанрами;
- повертає створені зв'язки або помилку.

Відмінні властивості `getOne` (запит на отримання одного фільму), `getAll` (запит на отримання усіх фільмів), `deleteOne` (запит на видалення фільму) полягають в тому, що:

- `findOne` використовує метод `findOne`, котрий за ключем фільму знаходить його, та де знаходиться зв'язок з таблицями сеансів та жанрів;
- `getAll` – завдяки `findAndCountAll` знаходить усі фільми з додатковими параметрами (кількістю на одній сторінці, та при необхідності за жанром);
- `deleteOne` – знаходить фільм за `id` (`findByIdPk()`) та завдяки методу `destroy()` видаляє фільм.

3.2.7 Безпека персональних даних користувача

Оскільки безпека даних користувача має велике значення при взаємодії “Надавач послуг - Клієнт”, адже через невиконання принципів захисту інформації відбуваються фінансові та репутаційні втрати, їй слід приділити окрему увагу та впевнитися, що всі дані зберігаються безпечно.

Задля того, щоб до даних користувача не могли дібратися зловмисники навіть після отримання доступу до бази даних, пароль необхідно хешувати (реформувати пароль завдяки математичним операціям).

Принцип роботи хешування паролю представлено в методі реєстрації користувача в `UserController.js`, код якого представлено в лістингу 3.10 (повний код представлено в лістингу А.3).

Лістинг 3.10 – Контролер реєстрації

```
async registration(req, res, next) {
    const { name, email, password, role } = req.body;

    const candidateEmail = await User.findOne({ where: {
email } }));

    if (candidateEmail) {
        return next(ApiError.badRequest('Користувач з таким
email вже існує'))
    }

    const candidateName = await User.findOne({ where: {
name } }));

    if (candidateName) {
        return next(ApiError.badRequest("Користувач з таким
ім'ям вже існує"))
    }
}
```

```

    const hashPassword = await bcrypt.hash(password, 5);

    const user = await User.create({
      name,
      email,
      role,
      password: hashPassword,
    });

    const token = generateJwt(user.id, user.name,
user.email, user.role);
    return res.json({ token });
  }

```

Функція реєстрації використовує наступні параметри:

- generateJwt: Функція для створення JSON Web Token (JWT). JWT - це стандарт для створення токенів, які можуть бути використані для безпечної передачі інформації між сторонами як JSON об'єкт. Цей токен є компактним, самодостатнім і зручним для використання в веб-застосунках для автентифікації та авторизації.

- jwt.sign: Створює підписаний JWT з об'єктом, що містить id, name, email, role, використовуючи секретний ключ з змінної середовища SECRET_KEY, з терміном дії 24 години;

- витягує ім'я, email, пароль та роль з тіла запиту;
- шукає користувача в базі даних за email та ім'ям;
- якщо користувач з таким email або ім'ям вже існує, викликає повідомлення з відповідною помилкою;

- хешує пароль користувача з використанням bcrypt (бібліотека для хешування паролів, яка забезпечує високий рівень безпеки) з рівнем складності 5, при якому відбувається хешування 32 рази;

- створює новий запис користувача в базі даних з хешованим паролем;
- створює JWT токен для нового користувача;
- повертає згенерований токен у відповіді на запит;

Для перевірки токена в процесі роботи додатку, мною було створено файл-посередник (middleware), котра використовується під час обробки HTTP-запитів (лістинг 3.11).

Лістинг 3.11 – authMiddleware.js

```
const jwt = require('jsonwebtoken')

module.exports = function (req, res, next) {
  if (req.method === "OPTIONS") {
    next()
  }
  try {
    const token = req.headers.authorization.split(' ')[1]

    if (!token) {
      return res.status(401).json({message: "Не авторизований"})
    }

    req.user = jwt.verify(token, process.env.SECRET_KEY)
    next()
  } catch (e) {
    res.status(401).json({message: "Не авторизований"})
  }
};
```

Пояснення роботи middleware:

- отримання токена – токен витягується з заголовку Authorization HTTP-запиту. Якщо заголовок не містить токен, повертається помилка з кодом 401 (Не авторизований);
- перевірка токена – токен перевіряється за допомогою методу jwt.verify з використанням секретного ключа, збереженого в змінній середовища SECRET_KEY;
- присвоєння користувача запиту – якщо токен дійсний, декодовані дані користувача з токена присвоюються req.user;
- обробка помилок – якщо виникає помилка під час перевірки токена, повертається помилка з кодом 401 (не авторизований);

Щоб додати middleware до GET-запиту, необхідно прописати його другим параметром в відповідному запиті (лістинг 3.12).

Лістинг 3.12 – Додавання middleware до запиту

```
router.get('/auth', authMiddleware, userController.check)
```

3.3 Розробка клієнтської частини застосунку

3.3.1 Ініціалізація проекту та встановлення залежностей

Для розгортання можливостей React.js в межах середовища розробки клієнтського застосунку необхідно в терміналі ввести команду “`npm create-react-app`”, після чого створяться базові компоненти та бібліотеки.

Завдяки команді “`npm i axios react-router-dom mobx mobx-react-lite bootstrap`” я встановив наступні залежності:

- Axios — це бібліотека для здійснення HTTP-запитів. Вона дозволяє легко відправляти запити GET, POST, PUT, DELETE та інші до серверу;
- React Router DOM — це бібліотека для реалізації маршрутизації в додатках на React. Вона дозволяє створювати маршрути і керувати переходами між різними компонентами в залежності від URL;
- MobX — це бібліотека для управління станом додатків на JavaScript. Вона забезпечує реактивне управління станом, дозволяючи автоматично оновлювати компоненти при зміні стану;
- MobX React Lite — це бібліотека, яка забезпечує інтеграцію MobX з функціональними компонентами React;
- Bootstrap — це популярний CSS-фреймворк для створення адаптивних і сучасних веб-сторінок. Він надає готові компоненти та стилі для швидкої розробки інтерфейсів.

3.3.2 Структура та компоненти клієнтського застосунку

Для кращого контролю розробки React-застосунку, мною було прийнято рішення декомпонувати застосунок на різні модулі, кожен з яких відповідає за окремий функціонал. Структура React-застосунку представлено на рис. 3.2.

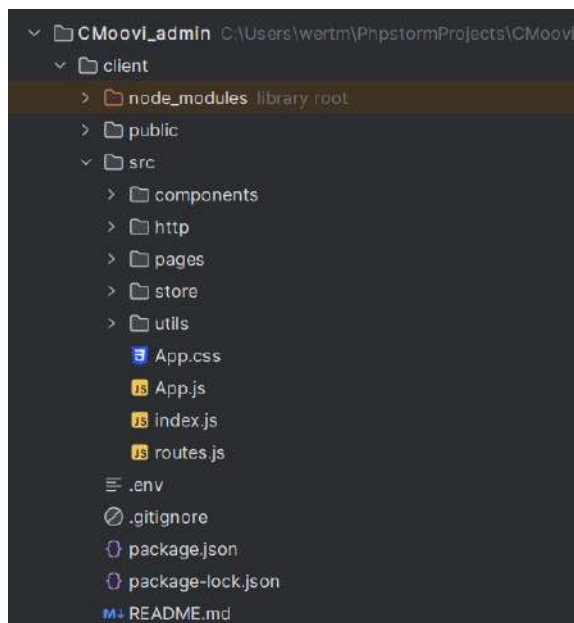


Рисунок 3.2 – Структура клієнтського застосунку

Можна виділити наступний функціонал, що несе кожний з модулів:

- node-modules – це файл, що автоматично створюється при ініціалізації React.js та в якому містяться усі встановлені бібліотеки;
- public – папка, яка містить статичні файли, такі як HTML, зображення та інші ресурси, які не змінюються динамічно;
- src – головна папка, де знаходиться весь вихідний код додатку.

Вміст папки src:

- Components містить у собі наступні елементи кнопок, шрифтів та модальних вікон;
- http – містить функції для здійснення HTTP-запитів до API;
- pages – папка для компонентів сторінок, кожна з яких представляє окремий маршрут у додатку;

- store – містить налаштування глобального стану додатка завдяки використанню MobX;
- utils – папка для допоміжних функцій, які використовуються у додатку (містить константи);
- App.css – файл стилів для головного компоненту додатку;
- App.js – головний компонент додатку з в якому прописана базова структура проекту;
- index.js – точка входу в додаток, яка рендерить головний компонент “App” в DOM. Document Object Model - програмний інтерфейс для HTML і XML документів. Він представляє структуру документа у вигляді дерева об'єктів, що дозволяє програмам і скриптам динамічно отримувати доступ до документу та змінювати його вміст, структуру та стиль в процесі розробки та роботи;
- routes.js – містить налаштування маршрутів додатку.

3.3.3 Налаштування точки входу

В реактивних додатках React, як і в інших додатках що використовують HTTP, використовується точка входу у додаток, де розміщено всі елементи з яких починається відтворення сторінки (лістинг 3.13)

Лістинг 3.13 – Вхідний файл index.js

```
import React, {createContext} from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import UserStore from './store/UserStore';
import MovieStore from './store/MovieStore';

export const Context = createContext(null)

const root =
ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <Context.Provider value={{
    user: new UserStore(),
    movie: new MovieStore(),
  }}>
    <App />
  </Context.Provider>
```

);

В даному коді реалізовано створення контексту, який є механізмом в React, що дозволяє передавати дані через дерево компонентів без необхідності передавати пропси на кожному рівні. Він створюється з початковим нульовим значенням, а вже в дочірніх елементах додатку через значення `value` в `Context.Provider` передається екземпляри `UserStore` та `MovieStore`, які містять налаштування глобальних станів додатку.

`ReactDOM.createRoot` створює кореневий елемент для рендерингу `React`-додатку.

Все це наслідує файл `App.js`, код якого представлено в лістингу A.4.

На прикладі фрагменту коду, що представлено в лістингу 3.14, можна побачити як відокремлено генеруються елементи застосунку, а саме навігаційна панель, футер та контент сторінки рендеряться незалежно одне від одного, що дає змогу при, наприклад, переході користувача на іншу сторінку застосунку, не рендерити повторно навігаційну панель, що в свою чергу впливає на швидкість роботи та зручність взаємодії окремих елементів застосунку.

Лістинг 3.14 – Відокремлене генерування елементів

```
<BrowserRouter>
  <div className="app-container">
    <NavBar />
    <div className="main-content-container">
      <AppRouter />
    </div>
    <Footer />
  </div>
</BrowserRouter>
```

Хуки в React дозволяють використовувати стан та інші можливості компонентів без необхідності створення класів, що спрощує та робить більш зручним розробку компонентів. Вони допомагають підтримувати чистоту і зрозумілість коду, а також сприяють повторному використанню логіки між різними компонентами.

В App.js є хук useEffect, котрий викликає функцію check, яка перевіряє jwt-токен, і налаштовує стан користувача на основі результату цієї перевірки. Якщо перевірка успішна, користувач встановлюється як авторизований, і, якщо необхідно, як адміністратор. Якщо перевірка не вдалася, користувач встановлюється як неавторизований. В будь-якому випадку, після завершення перевірки, стан завантаження змінюється на false. Хук використовується один раз після першого рендеру компонента, оскільки масив залежностей порожній.

3.3.4 Написання реактивної сторінки в React.js

Приклад написання реактивної сторінки в React.js, можна побачити на прикладі сторінки з замовленням квитків на сеансі Seance.js, повний код якої представлено в лістингу А.5.

Перше, що необхідно виділити це використання хука useState, який дозволяє додавати локальний стан до функціональних компонентів. Синтаксис написання представлено в листингу 3.15

Лістинг 3.15 – Хук useState

```
const [state, setState] = useState(initialState);
```

Пояснення:

- state – змінна, яка зберігає поточний стан;
- setState - функція, яка використовується для оновлення стану;
- initialState – початкове значення стану, що може мати фіксоване або порожнє значення, також може бути масивом.

В випадку Seance.js, хук useState отримує дані (лістинг 3.16), необхідні для оформлення замовлення та завдяки функції submitOrder відправляє їх до серверу (лістинг 3.17).

Лістинг 3.16 – Використання хуку useState

```
const [user, setUser] = useState({ id: 0 });
const [seance, setSeance] = useState(null);
const { id } = useParams();
const navigate = useNavigate();
```

```

const [seatSeanceId, setSeatSeanceId] = useState([]);
const [selectedSeats, setSelectedSeats] = useState([]);
const [totalPrice, setTotalPrice] = useState(0);
const [showModal, setShowModal] = useState(false);
const [name, setName] = useState('');
const [email, setEmail] = useState('');

```

Лістинг 3.17 – Формування та відправка замовлення до серверу

```

const submitOrder = async (name, email) => {
  if (!name || !email) {
    alert('Будь ласка, заповніть усі поля.');
```

```

    return;
  }
  try {
    const formData = new FormData();
    formData.append('name', name);
    formData.append('email', email);
    formData.append('userId', user.id || 0);
    formData.append('seanceId', seance.id);
    formData.append('seatSeanceId',
JSON.stringify(seatSeanceId));
    await createOrder(formData);
    alert("Вітаємо, місця заброньовані!");
    navigate(`/movie/${seance.movie.id}`);
    // Update the seance to reflect the booked seats
    fetchOneSeance(id).then(data => setSeance(data));
  } catch (e) {
    alert(e.response.data.message);
  }
};

```

Для зручності відображення місць для користувача, мною була створена функція `groupSeatsByRow` (лістинг 3.18), в якій передається параметр з бази даних `seat`, з відповідними характеристиками рядків та місць, тому що використовує `reduce` для створення об'єкта, де ключі – номери рядків, а значення – масиви місць у цих рядках, після чого відбувається сортування місця в кожному ряду за номерами.

Лістинг 3.18 – `groupSeatsByRow()`

```

const groupSeatsByRow = (seats) => {
  const grouped = seats.reduce((rows, seat) => {
    if (!rows[seat.row]) {
      rows[seat.row] = [];
    }
    rows[seat.row].push(seat);
  });

```

```

        return rows;
    }, {}));
    Object.keys(grouped).forEach(row => {
        grouped[row].sort((a, b) => a.place - b.place);
    });
    return grouped;
};

```

Код функції обробки кліку на місце `handleSeatClick`, представлено в лістингу 3.19.

Лістинг 3.19 – `handleSeatClick()`

```

const handleSeatClick = (seat) => {
    const seatSeance = seat.seatSeances.find(ss =>
ss.seanceId === seance.id);
    if (seatSeance.status === 'BOOKED') return;

    setSeatSeanceId(prev => {
        if (prev.includes(seatSeance.id)) {
            setSelectedSeats(selectedSeats.filter(s => s.id
!== seat.id));

            setTotalPrice(totalPrice - seat.price);
            return prev.filter(id => id !== seatSeance.id);
        } else {
            setSelectedSeats([...selectedSeats, seat]);
            setTotalPrice(totalPrice + seat.price);
            return [...prev, seatSeance.id];
        }
    });
};

```

Дана функція має наступні інструкції:

- якщо статус місця значиться “BOOKED” (заброньовано), то місце не буде активним і його не можна буде додати в замовлення;
- в випадку коли місце не заброньоване, то при натисканні відбувається вивід даних про замовлення на сторінці сайту, підраховується загальна сума замовлення та зберігається id обраного місця, задля додавання його до бази даних, в якості інформації про заброньоване місце.

Функція `addOrder` (лістинг 3.20) має наступні інструкції:

- перевіряє, чи користувач авторизований;

- якщо користувач неавторизований, відкриває модальне вікно для введення ім'я та електронної пошти;
- якщо користувач авторизований, викликає submitOrder.

Лістинг 3.20 – addOrder()

```
const addOrder = async () => {
  if (!user.id) {
    setShowModal(true);
    return;
  }
  submitOrder(user.name, user.email);
};
```

Функція renderSeats (лістинг 3.2z) має наступні інструкції:

- рендерить кнопки для місць з відповідними класами і обробкою кліків;
- кожна кнопка має унікальний ключ і клас, залежно від статусу та ціни місця;
- показує ціну і статус місця при наведенні;
- використовує обробник подій “onClick”, що дозволяє визначити, яка функція повинна бути виконана, коли користувач натискає на елемент (в випадку renderSeats – це виклик функції handleSeatClick з параметром seat).

Лістинг 3.21 – renderSeats()

```
const renderSeats = (seats) => {
  return seats.map(seat => (
    <button
      key={seat.id}
      className={`seat-btn ${seat.price === 140 ?
getSeatStatus(seat) === 'BOOKED' ? 'BOOKED' : 'price-140' :
getSeatStatus(seat)}
${seat.seanceId.includes(seat.seatSeances.find(ss => ss.seanceId ===
seance.id).id) ? 'selected' : ''}`}
      title={` ${seat.price} грн,
${getSeatStatus(seat)} `}
      onClick={() => handleSeatClick(seat)}
    >
      {seat.place}
    </button>
  ));
};
```

Формотування дати та часу для зручного відображення, відбувається завдяки константі `formatDateTime`:

- використовує `parseISO` для перетворення рядка дати в об'єкт дати;
- використовує `format` для форматування дати і часу з урахуванням українського локального часу;
- розділяє форматовані дати і часи на окремі змінні.

Компонент `return` – це JSX елемент, що є основним шаблоном для компонента, що відображає інформацію про кіносеанс, місця в залі, а також обробку замовлень квитків. Має наступну структуру:

- `<div className="seance-container">`: Основний контейнер для всього контенту компонента. Використовується для загального стилю і структурування;
- `<div className="seance-header">`: Контейнер для заголовка сеансу, що включає постер фільму та основну інформацію;
- `<Image>`: Компонент з React Bootstrap для відображення зображення (постера фільму);
- `className='seance-poster'`: Клас для стилізації постера;
- `src`: Джерело зображення. Якщо є постер фільму, використовує URL з базовим API URL;
- `<div className="seance-info">`: Контейнер для текстової інформації про сеанс;
- `<div className="seats-container">`: Контейнер для всіх місць у залі;
- `{Object.keys(seatsByRow).map(row => (`: Проходить по всіх рядах, щоб відобразити місця в кожному ряду;
- `<div key={row} className="seat-row">`: Контейнер для кожного ряду місць;
- `<h3>Ряд {row}</h3>`: Відображає номер ряду;
- `<div className="seat-row-buttons">`: Контейнер для кнопок місць у конкретному ряду;
- `{renderSeats(seatsByRow[row])}`: Викликає функцію `renderSeats` для відображення місць у ряду.

3.3.5 Формування електронного квитка

В наслідок дій користувача з використання функціоналу комп'ютерної системи з замовлення квитків кінотеатру, як результат, повинен бути створений електронний квиток в форматі PDF, в якому позначену всю інформацію по бронюванню. Реалізація виконання формування та завантаження електронного квитка, виконує функція `downloadPDF` (лістинг 3.22), що знаходиться в класі `User.js`, що є особистим кабінетом користувача, в якому зберігаються всі його замовлення, код якого представлено в лістингу А.6.

Лістинг 3.22 – функція `downloadPDF`

```
const downloadPDF = async (order) => {
  const doc = new jsPDF();

  // Добавление шрифта
  doc.addFileToVFS("Roboto-Regular.ttf", robotoFont);
  doc.addFont("Roboto-Regular.ttf", "Roboto", "normal");
  doc.setFont("Roboto");

  // Загрузка изображения и преобразование в base64
  const imgData = await fetch(order.seance.movie?.img ?
process.env.REACT_APP_API_URL + order.seance.movie.img : '')
    .then(res => res.blob())
    .then(blob => new Promise((resolve, reject) => {
      const reader = new FileReader();
      reader.onloadend = () =>
resolve(reader.result);
      reader.onerror = reject;
      reader.readAsDataURL(blob);
    }));

  doc.addImage(imgData, 'JPEG', 10, 10, 50, 75);

  doc.text(`Номер брони: ${order.id}`, 10, 100);
  doc.text(`Користувач: ${user.name}`, 10, 110);
  doc.text(`Фільм:   ${order.seance.movie.titleUa}`, 10,
120);
  doc.text(`Початок                               сеансу:
${formatSeanceTime(order.seance.beginning)}`, 10, 130);
  doc.text(`Кінець                               сеансу:
${formatSeanceTime(order.seance.ending)}`, 10, 140);
  doc.text(`Дата                               замовлення:
${formatOrderDate(order.createdAt)}`, 10, 150);

  let y = 160;
  order.seatSeances.forEach((seatSeance, index) => {
```

```

                doc.text(`Місце      ${index      +      1}:      Ряд
${seatSeance.seat.row},      Місце      ${seatSeance.seat.place},      Ціна
${seatSeance.seat.price} грн`, 10, y);
                y += 10;
        });

        const      totalPrice      =
calculateTotalPrice(order.seatSeances);
        doc.text(`Загальна сума:  ${totalPrice} грн`, 10, y +
10);

        doc.save(`order-${order.id}.pdf`);
    };

```

Пояснення до коду:

- addFileToVFS: Додає шрифт у віртуальну файлову систему jsPDF, задля коректного відображення українських символів;
- addFont: Додає шрифт у документ;
- setFont: Встановлює шрифт для використання у документі;
- fetch: Завантажує зображення постера фільму;
- blob: Перетворює відповідь на Blob-об'єкт;
- FileReader: Використовується для читання Blob як Data URL (base64);
- reader.readAsDataURL(blob): Читає Blob і перетворює його у Data URL;
- reader.onloadend: Викликається після завершення читання. Результат зберігається у reader.result;
- reader.onerror: Викликається у разі помилки;
- text: Додає текст у документ, завдяки використанню значень змінних об'єктів та додатковим параметрів координат;
- calculateTotalPrice: Розраховує загальну суму за всі місця у замовленні.
- save: Зберігає документ як PDF-файл;
- order-\${order.id}.pdf: Ім'я файлу, що зберігається.

Після завантаження квитка, клієнт може пред'явити його зі свого мобільного пристрою адміністрації кінотеатру, що буде підтвердженням заброньованого ним місця.

4 ВИПРОБУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

4.1 Випробування форми авторизації та реєстрації

Як правило, першим етапом перевірки роботи користувацького інтерфейсу з сервером, а того в свою чергу з базою даних, прийнято вважати перевірку роботи форм авторизації та реєстрації, оскільки це базова складова частина персональної безпеки та захисту інформації користувача.

На рис. 4.1 – 4.6 можна побачити різні варіанти валідації даних користувача при різних сценаріях помилкового внесення даних при авторизації та реєстрації, а на рис. 4.7 – 4.8 – результат успішного введення даних авторизації.

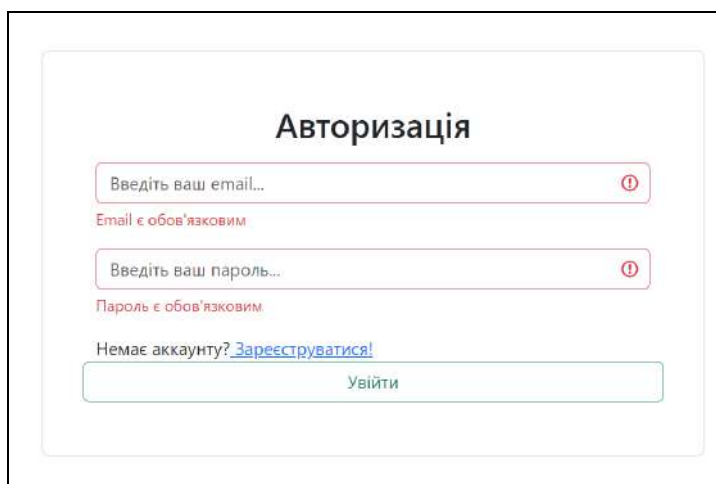


Рисунок 4.1 – Користувач не ввів даних в жодне поле

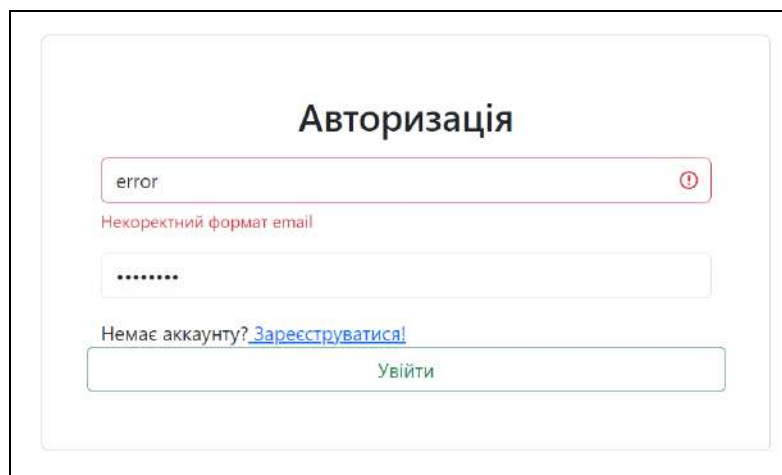


Рисунок 4.2 – Користувач ввів пошту в невірному форматі

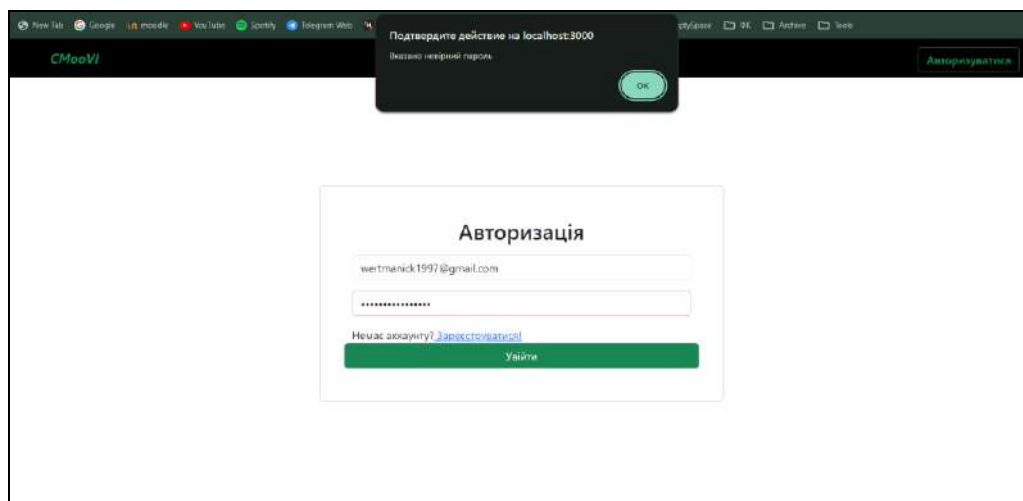


Рисунок 4.3 – Користувач ввів невірний пароль

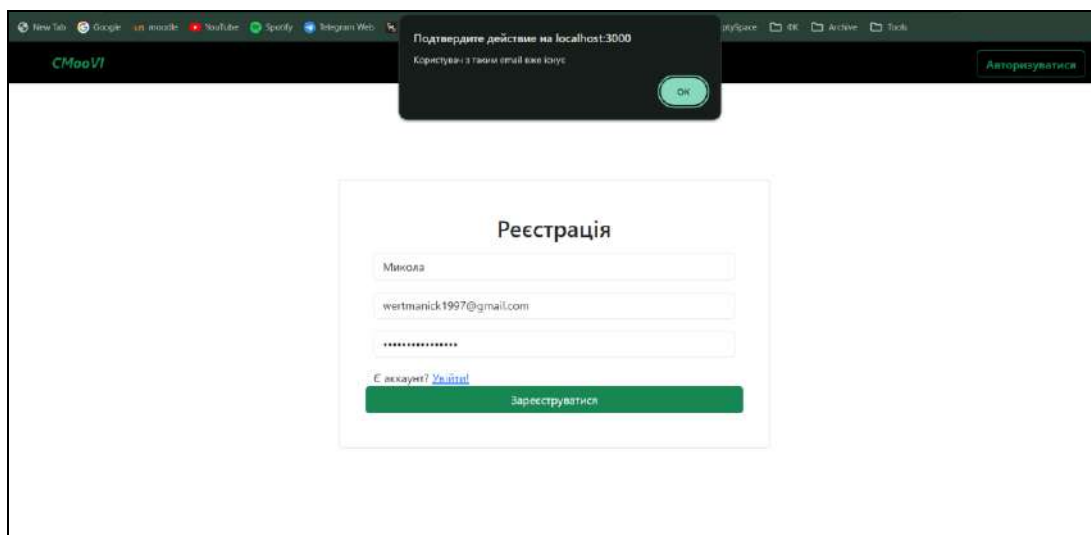


Рисунок 4.4 – Користувач ввів пошту, котра вже є в базі даних користувачів

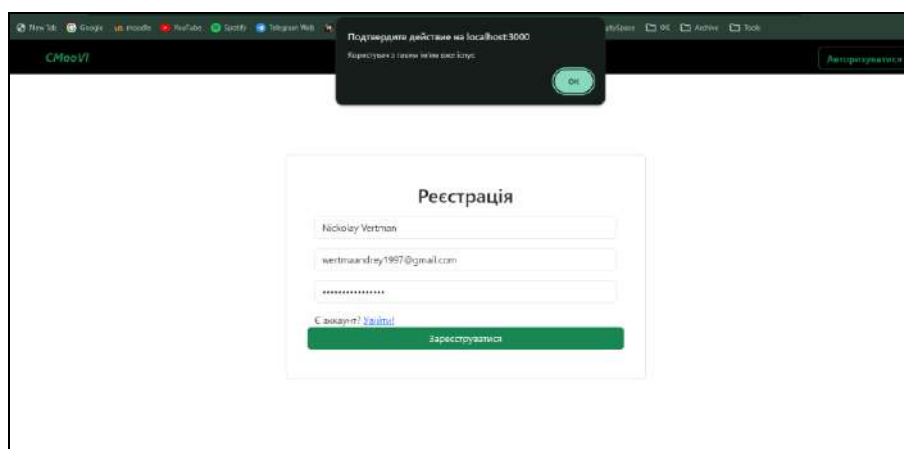


Рисунок 4.5 – Користувач ввів ім'я, котре вже закріплене за іншим користувачем

CMooVI Авторизуватися

Реєстрація

Nickolay Vertman

wertmaandrey1997@gmail.com

Пароль повинен містити принаймні 4 літери

Є акаунт? [Ввійти!](#)

[Зареєструватися](#)

Рисунок 4.6 – Користувач ввів недостатню кількість літер

id [PK] Integer	name character varying (255)	email character varying (255)	password character varying (255)	role character varying (255)
1	2 Nickolay Vertman	wertmanick1997@gmail.com	\$2b\$05\$G.H4s7OfkvRBQUXabtdxE.LLrHREFzQ2J3Y5ZjcbmLB5SZ3l/e...	ADMIN
2	3 Sergey Vertman	wertmann2007@gmail.com	\$2b\$05\$vTGYi1jp6Z63MmJ6067eV.e9CZ7tRpKVNfFApQnjK25lbrNC.f...	USER

Рисунок 4.7 – Додавання нового користувача до бази даних

CMooVI Особистий кабінет Панель адміністратора Вийді

Зараз в прокаті

Вічне сяйво чистого розуму
Eternal Sunshine of the Spotless Mind

Форрест ґамп
Forrest Gump

Інтерстеллар
Interstellar

Вартові Галактики 2
Guardians of the Galaxy Vol. 2

All rights reserved
Тел: +380-66-273-36-59

Рисунок 4.8 – Після успішної авторизації, користувач потрапляє на головну сторінку застосунку

4.2 Випробування можливостей адміністратора

4.2.1 Навігаційні кнопки

Щоб поетапно, розподілено та згруповано показати взаємодію користувача веб-застосунком з окремими елементами моделі бази даних, випробування комп'ютерної системи, я вважаю доцільним, розпочати з випробування можливостей адміністратора додатка.

Перша відмінна властивість інтерфейсу адміністратора, від інтерфейсу звичайного користувача – це наявність в навігаційній панелі додаткової кнопки «Панель адміністратора» (рис. 4.9). У звичайного користувача є тільки два маршрути в навігаційній панелі: можливість входу в особистий кабінет та клавiша виходу з аккаунта (рис. 4.10). Неавторизований користувач має в навігаційній панелі тільки кнопку, що направляє його на сторінку авторизації (рис. 4.11).

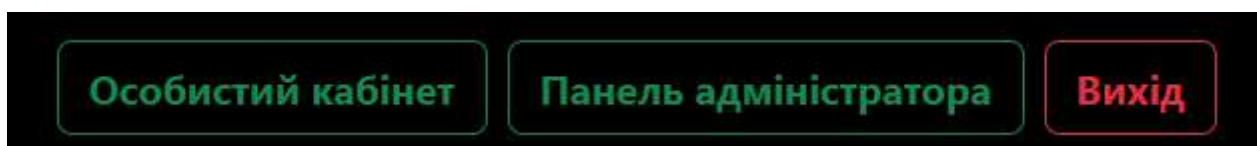


Рисунок 4.9 – Навігаційні клавiші адміністратора

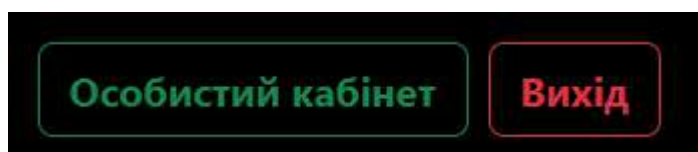


Рисунок 4.10 – Навігаційні клавiші авторизованого користувача



Рисунок 4.11 – Навігаційні можливості неавторизованого користувача

4.2.2 Панель адміністратора. Загальний вигляд

Сторінку панелі адміністратора з функціональними клавішами для додавання жанру, додавання фільму, видалення фільму та додавання сеансу можна побачити на рис. 4.12. Це проста сторінка, без зайвого функціоналу, яка надає моментальний доступ до необхідних елементів для роботи з контентом у застосунку.

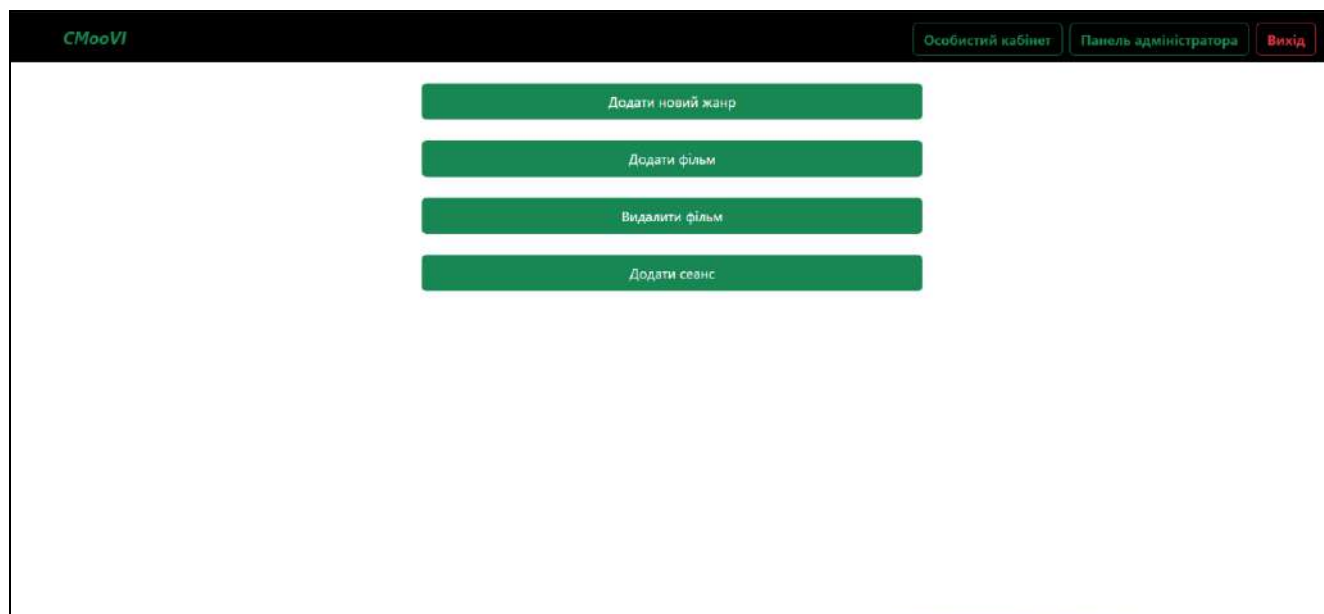


Рисунок 4.12 – Сторінка «Панель адміністратора»

4.2.3 Робота модальних вікон з керування контентом для застосунку

Модальне вікно для додавання нових жанрів – це просте вікно в котрому необхідно ввести назву жанру, після чого він додається в базу даних.

На рисунку 4.13 представлено вихідну базу даних, до проведення випробування роботи модального вікна з додавання жанрів.

На рисунку 4.14 представлено варіант валідації даних, при варіанті розвитку подій коли адміністратор не ввів жодних даних.

На рисунку 4.15 представлено введення даних у вікні.

На рисунку 4.16 представлено результат додавання нової позиції до бази даних.

	id [PK] integer	name character varying (255)
1	1	Комедія
2	2	Драма

Рисунок 4.13 – Вихідна таблиця даних

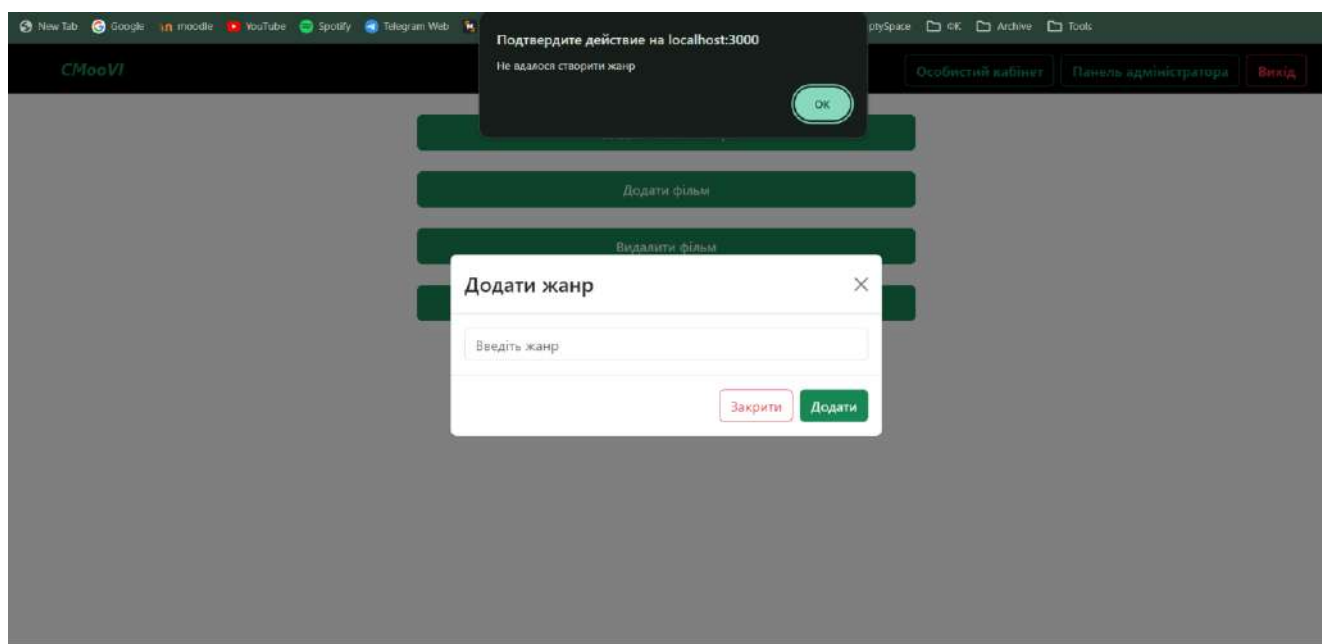


Рисунок 4.14 – Результат валідації неправильних даних

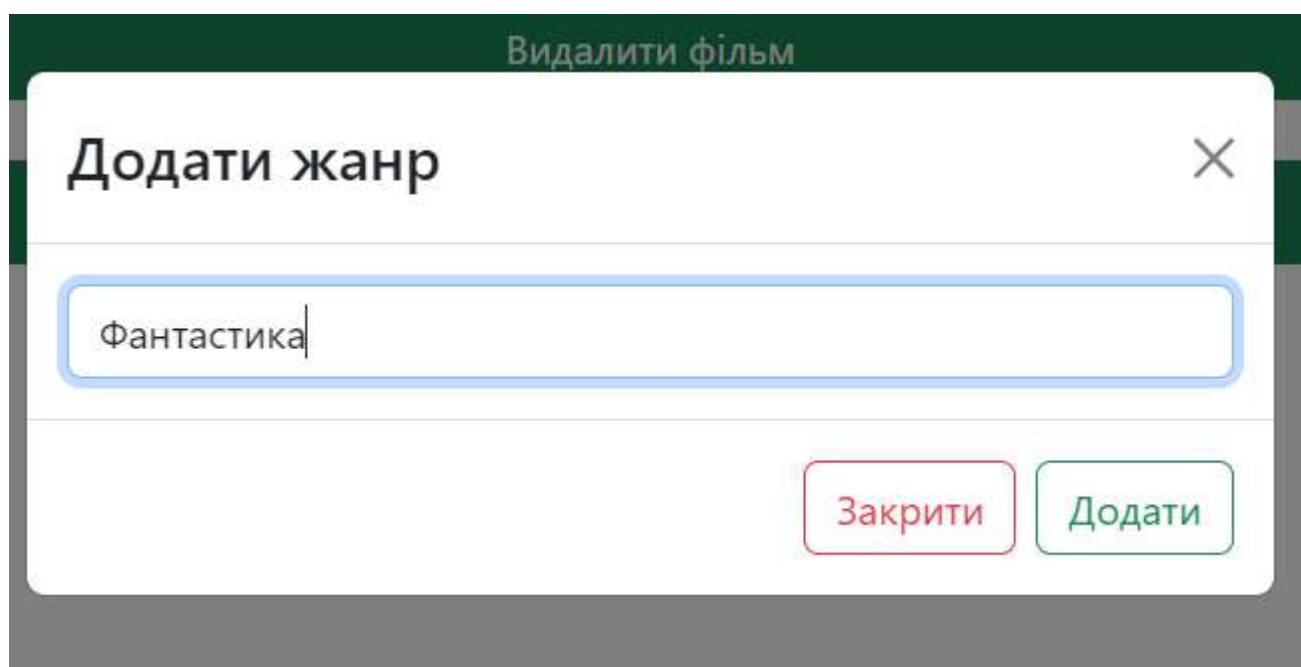


Рисунок 4.15 – Введення даних у модальне вікно

	id [PK] integer	name character varying (255)
1	1	Комедія
2	2	Драма
3	3	Фантастика

Рисунок 4.16 – Результат додавання нового жанру до бази даних

Принцип роботи інших модальних вікон ідентичний, але складніший, через додавання нових типів даних до кожної з позиції бази даних (наприклад, зображення для таблиці з фільмами та роботи з датами для таблиці з сеансами).

Принцип заповнення даних для таблиці з фільмами зображено на рис. 4.17, результат додавання даних до таблиці на рис 4.18, а на рис 4.19 його поява на головній сторінці сайту.

Принцип заповнення даних для таблиці з сеансами зображено на рис. 4.20, результат додавання даних до таблиці на рис 4.21, а поява сеансу на сторінці з фільмом показано на рис 4.22.

Рисунок 4.17 – Введені дані для додання нового фільму до бази даних

id [PK] integer	title character varying (255)	titleUa character varying (255)	director character varying (255)	actors character varying (255)	year integer	country character varying (255)	duration integer	description text	img character varying (255)
1	Eternal Sunshine of the Spotless Mind	Вічне сяйво чистого розуму	Мішель Гондрі	Джим Керрі, Кейт Вінсл...	2004	США	106	Технічний прог...	04051a07-b8...
2	Forrest Gump	Форрест ґамп	1	1	1	1	1	1	b2987a57-e6...
5	Interstellar	Інтерстеллар	1	1	1	1	1	1	2908d992-cfe...
6	Guardian of the Galaxy vol.2	Вартові Галактики 2	1	1	1	1	1	1	f766f9af-1a08...
7	Bad Boys: Ride or Die	Погані хлопці: Все або нічого	Адиль Ель Арбі	Вілл Сміт, Мартін Лоуре...	2024	США	110	Детективи Май...	b415c2e7-570...

Рисунок 4.18 – Створений новий екземпляр таблиці Movie

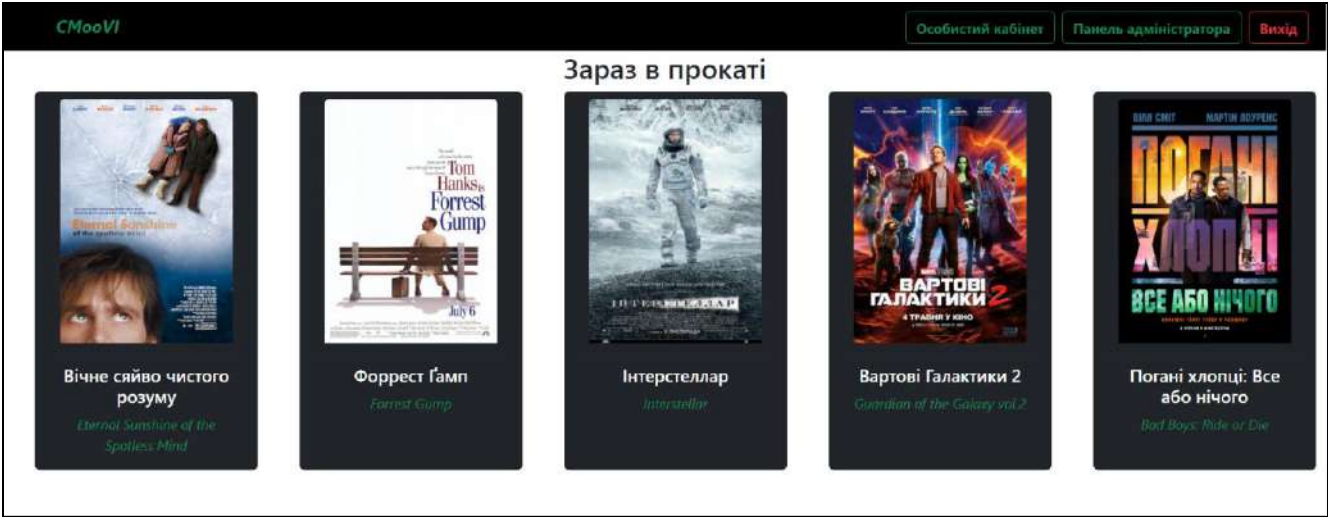


Рисунок 4.19 – Результат додання фільму до веб-застосунку

Додати сеанс

Оберіть фільм

Погані хлопці: Все або нічого

Оберіть залу

Зелена

Початок сеансу

01.06.2024 12:00

Кінець сеансу

01.06.2024 13:49

Закрити

Додати

Рисунок 4.20 – Введені дані для додавання нового сеансу до бази даних

	id [PK] integer	beginning timestamp with time zone	ending timestamp with time zone	actual boolean	movielid integer	hallid integer
1	1	2024-06-01 14:00:00+03	2024-06-01 15:47:00+03	true	1	1
2	2	2024-06-10 12:35:00+03	2024-06-10 13:37:00+03	true	2	1
3	3	2024-06-01 12:00:00+03	2024-06-01 13:49:00+03	true	7	1

Рисунок 4.21 – Створений новий екземпляр таблиці Séance

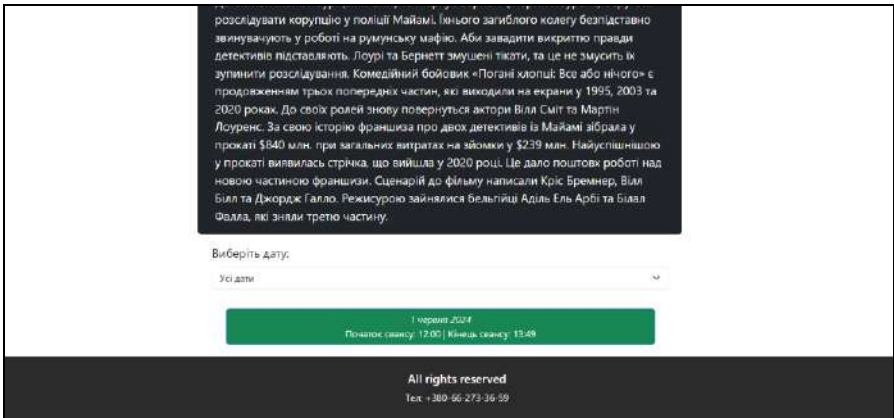


Рисунок 4.22 – Відображення створеного сеансу

В модульному вікні для видалення фільму, необхідно просто обрати необхідний за назвою та підтвердити його видалення (рис. 4.23 – 4.25). Також за принципом взаємопов’язаності (каскаду) було видалено сеанс, пов’язаний з цим фільмом.

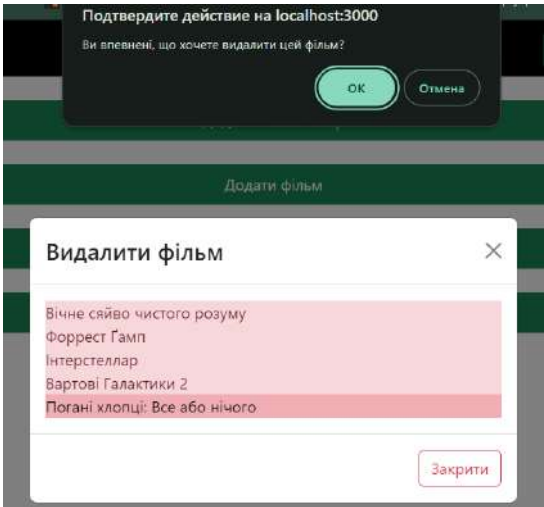


Рисунок 4.23 – Вибір фільму на видалення

id [PK] integer	title character varying (255)	titleUa character varying (255)	director character varying (255)	actors character varying (255)
1	Eternal Sunshine of the Spotless Mind	Вічне сяйво чистого розуму	Мішель Гондрі	Джим Керрі, Кейт Вінслет, Джеррі Роберт Берн, Елайджа
2	Forrest Gump	Форрест Гамп	1	1
5	Interstellar	Інтерстеллар	1	1
6	Guardian of the Galaxy vol.2	Вартові Галактики 2	1	1

Рисунок 4.24 – Результат видалення фільму з бази даних

	id [PK] integer	beginning timestamp with time zone	ending timestamp with time zone	actual boolean	movioid integer	hallid integer
1	1	2024-06-01 14:00:00+03	2024-06-01 15:47:00+03	true	1	1
2	2	2024-06-10 12:35:00+03	2024-06-10 13:37:00+03	true	2	1

Рисунок 4.25 – Результат видалення пов'язаного сеансу з бази даних

4.3 Випробування функції замовлення квитків

При випробування можливості оформлювати онлайн-замовлення квитків були виконані наступні кроки:

- користувач обирає сеанс на сторінці з фільмом, з використанням фільтру по даті (рис. 4.26);

- користувач обирає необхідні йому місця на сторінці сеансу (рис. 4.27). Як можна побачити, місця з різними цінками, відображаються різними кольорами, а заброньовані місця мають сірий колір та не можуть бути додані в замовлення. Також необхідно зазначити що в додатку показується кінцева сума оформлення замовлення;

- повідомлення про успішне оформлення замовлення (рис. 4.28);

- після оформлення замовлення можна побачити, що місця котрі обирав користувач (перший ряд, третє місце та четвертий ряд, перше місце) є заброньованими та недоступними до оформлення нового замовлення (рис. 4.29);

- на рис. 4.30 можна побачити додавання замовлення до таблиці order;

- підтягування інформації про замовлення в особистий кабінет користувача (рис. 4.31);
- завантажена PDF-версія квитка (рис. 4.32)

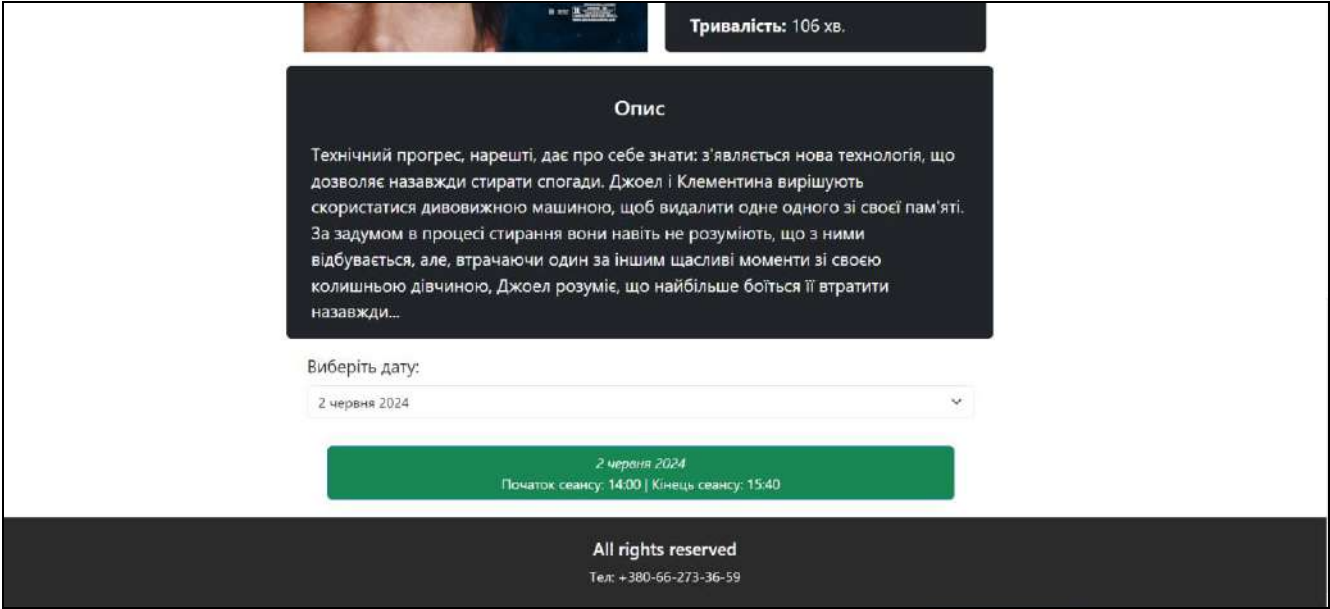


Рисунок 4.26 – Вибір сеансу через фільтр

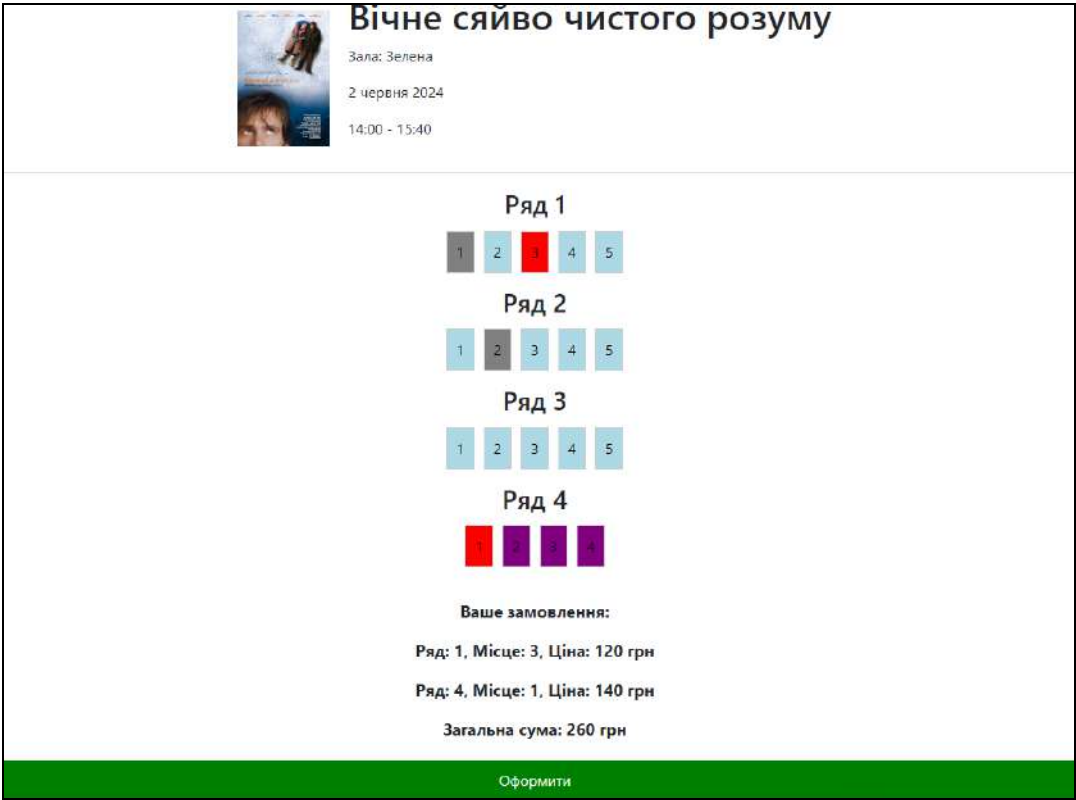


Рисунок 4.27 – Оформлення замовлення користувачем

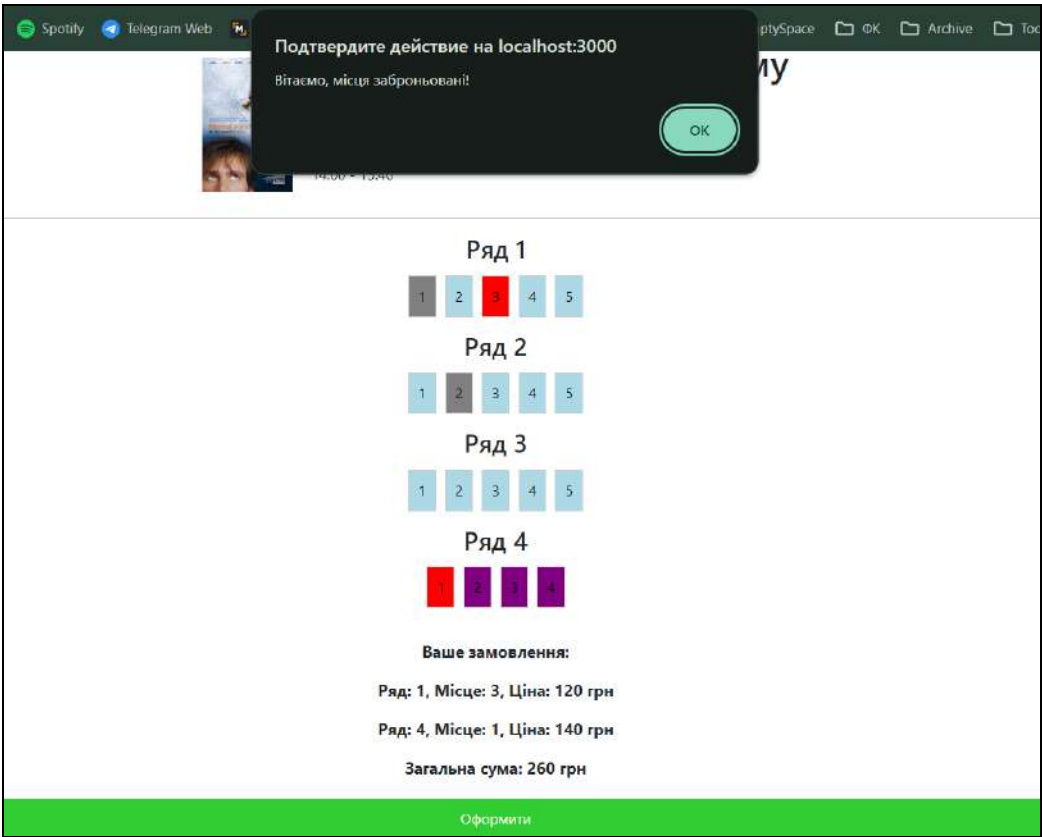


Рисунок 4.28 – Повідомлення про успішне оформлення замовлення

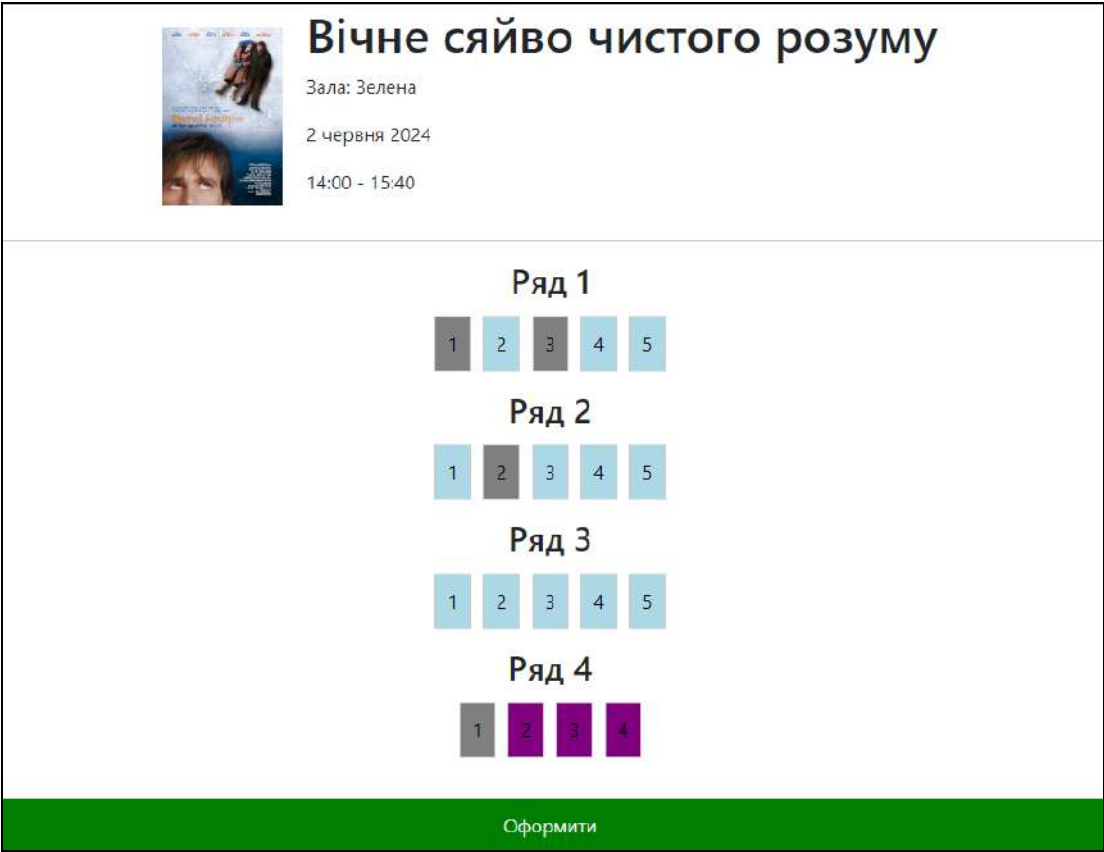


Рисунок 4.29 – Оновлені заброньовані місця

id	name	email	userId	seanceId	createdAt	updatedAt
[PK] integer	character varying (255)	character varying (255)	integer	integer	timestamp with time zone	timestamp with time zone
10000005	Nickolay Vertman	wertmanick1997@gmail.com	2	4	2024-06-01 20:33:30.021+03	2024-06-01 06:33:30.021+03

Рисунок 4.30 – Заовлення доане до бази даних

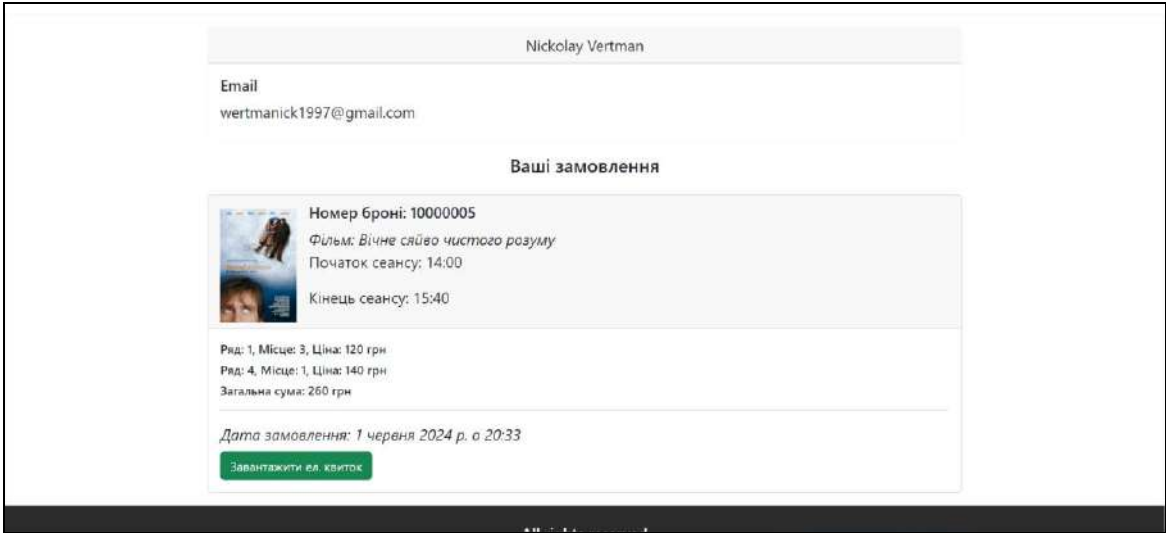


Рисунок 4.31 – Відображення замовлення в особистій сторінці користувача

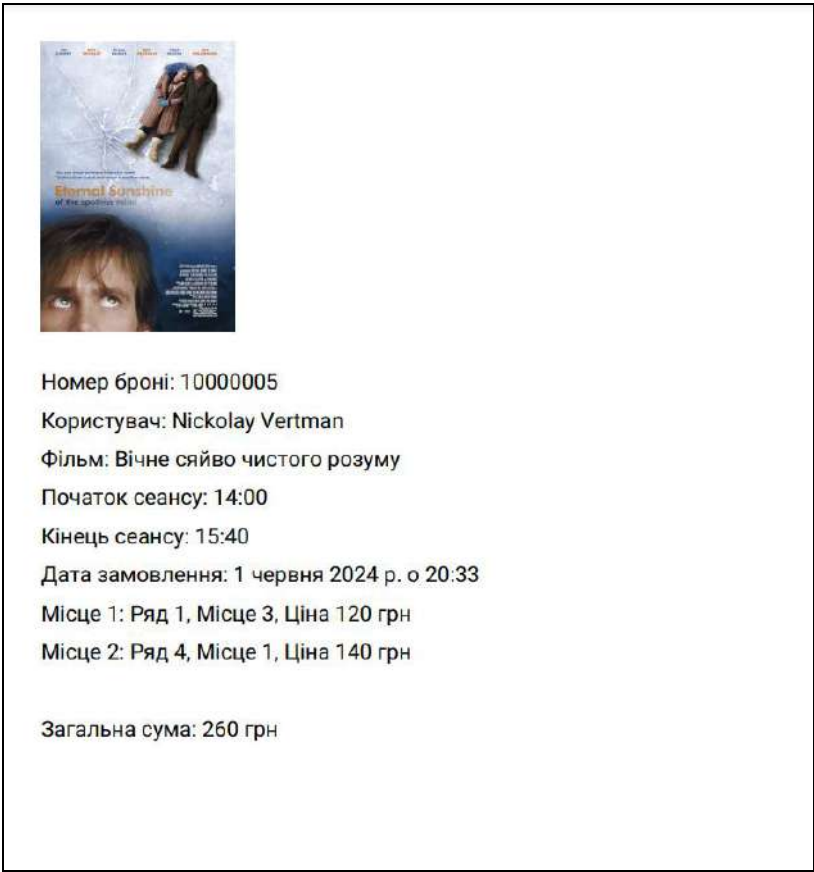


Рисунок 4.32 – Завантажена електронна версія квитка

4.4 Випробування адаптивності інтерфейсу

Останнім етапом випробування застосунку, як готового продукту, є перевірка відображення користувацького інтерфейсу на пристроях з різними характеристиками та розмірами екранів. Випробування робилось мною на мобільній версії застосунку iPhone SE з розширенням екрану 375x667 пікселів. Результати виконання випробування можна побачити на рис. 4.33 – 4.42.



Рисунок 4.33 – Головна сторінка сайту

The screenshot shows the login interface of the CMooVI application. At the top, there is a black header with the green text "CMooVI" on the left and a green menu icon on the right. The main content area is white and contains a light gray rounded rectangle. Inside this rectangle, the title "Авторизація" is centered. Below the title are two input fields: the first contains the email "wertmanick1997@gmail.com" and the second contains masked characters ".....". Below these fields is a link "Немає аккаунту? [Зареєструватися!](#)" and a button labeled "Увійти".

Рисунок 4.34 – Форма авторизації

The screenshot shows the registration interface of the CMooVI application. At the top, there is a black header with the green text "CMooVI" on the left and a green menu icon on the right. The main content area is white and contains a light gray rounded rectangle. Inside this rectangle, the title "Реєстрація" is centered. Below the title are three input fields: the first contains the name "Nickolay Vertman", the second contains the email "wertmanick1997@gmail.com", and the third contains masked characters ".....". Below these fields is a link "Є аккаунт? [Увійти!](#)" and a button labeled "Зареєструватися".

Рисунок 4.35 – Форма реєстрації

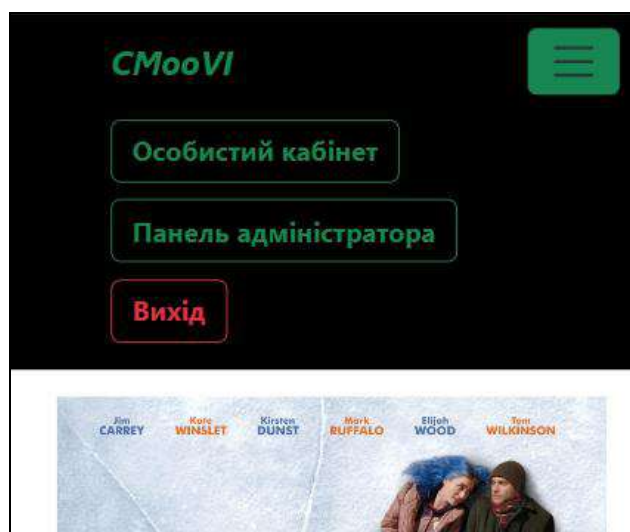


Рисунок 4.36 – Розгорнута навігаційна панель з функціональними клавiшами



Рисунок 4.37 – Сторінка з фільмом

Клементина вирішують
скористатися дивовижною
машиною, щоб видалити одне
одного зі своєї пам'яті. За
задумом в процесі стирання вони
навіть не розуміють, що з ними
відбувається, але, втрачаючи
один за іншим щасливі моменти
зі своєю колишньою дівчиною,
Джоел розуміє, що найбільше
боїться її втратити назавжди...

Виберіть дату:

Усі дати

1 червня 2024
Початок сеансу: 14:00 | Кінець сеансу:
15:47

All rights reserved
Тел: +380-66-273-36-59

Рисунок 4.38 – Доступні сеанси



Вічне сяйво чистого розуму

Зала: Зелена

1 червня 2024

14:00 - 15:47

Ряд 1

1	2	3	4	5
---	---	---	---	---

Ряд 2

1	2	3	4	5
---	---	---	---	---

Ряд 3

1	2	3	4	5
---	---	---	---	---

Ряд 4

1	2	3	4
---	---	---	---

[Оформити](#)

Рис 4.39 – Сторінка формування замовлення

Nickolay Vertman

Email

wertmanick1997@gmail.com

Ваші замовлення



Номер броні:
10000000

Фільм: Вічне сяйво
чистого розуму

Початок сеансу: 14:00

Кінець сеансу: 15:47

Ряд: 1, Місце: 1, Ціна: 120 грн

Ряд: 1, Місце: 3, Ціна: 120 грн

Загальна сума: 240 грн

Дата замовлення: 2 червня 2024
р. о 08:27

Завантажити ел. квиток

Рисунок 4.40 – Особистий кабінет користувача

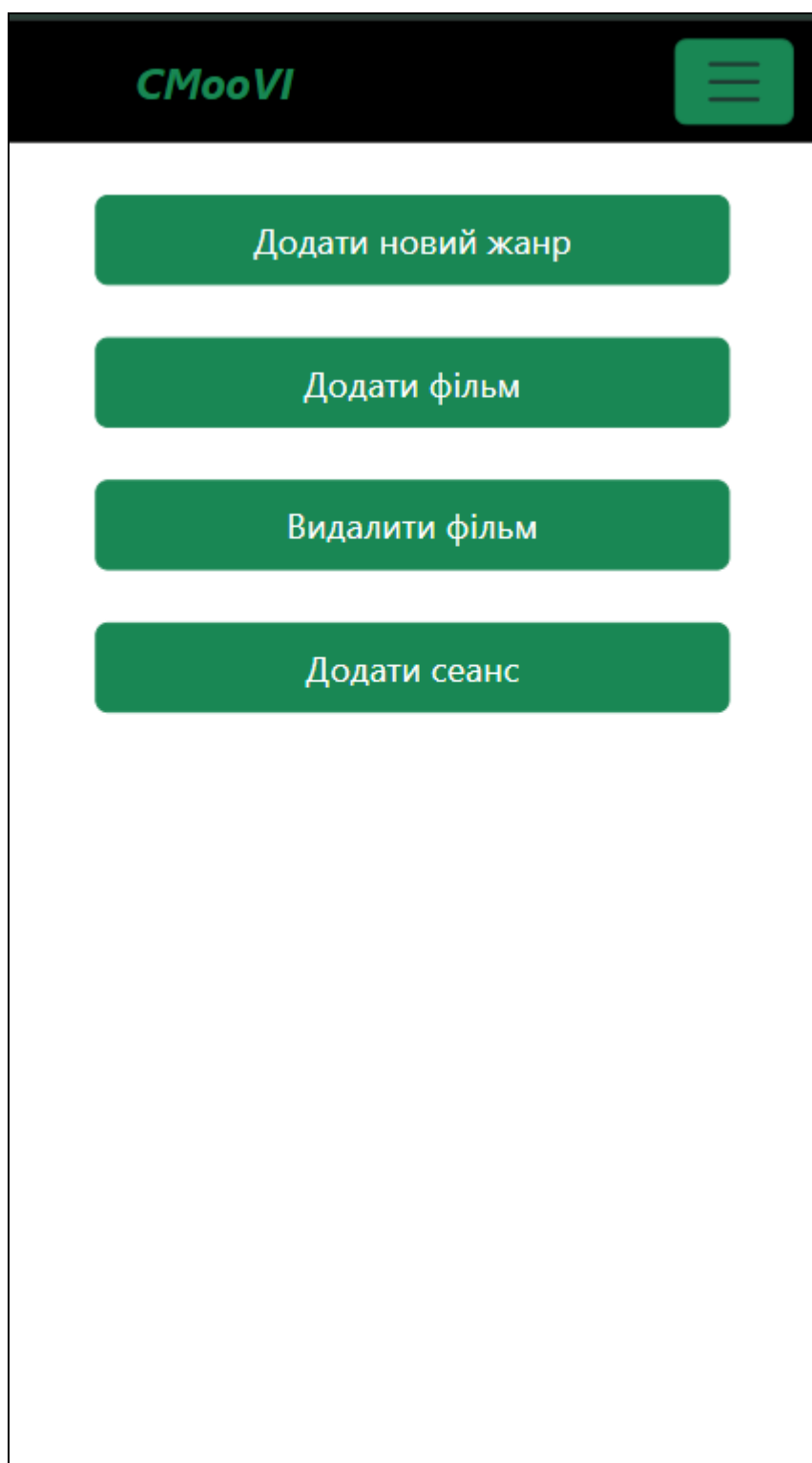


Рисунок 4.41 – Панель адміністратора

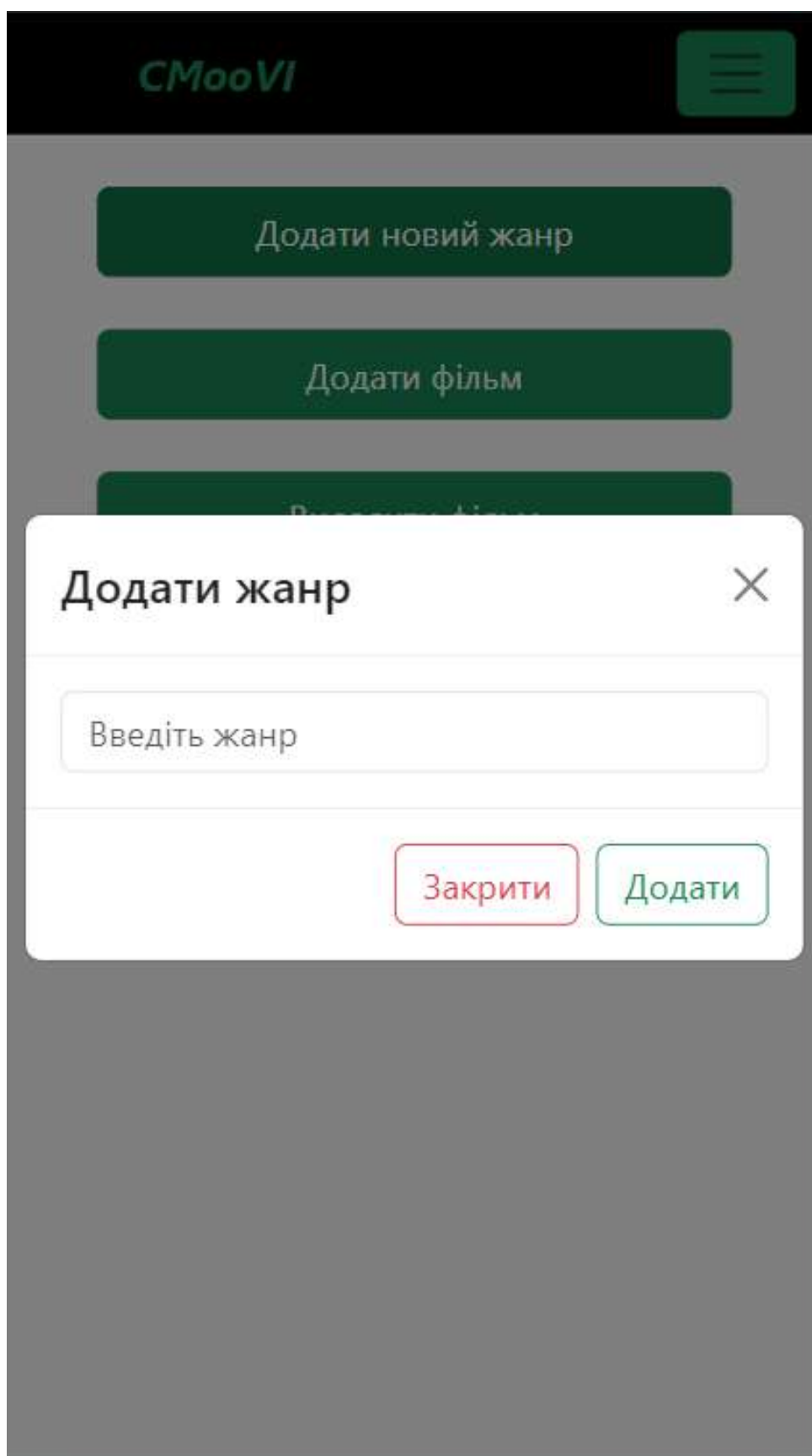


Рисунок 4.42 – Модальне вікно з додаванням жанру

ВИСНОВКИ

Під час виконання дипломної роботи мною була створена комп'ютерна система замовлення квитків для кінотеатру. Для реалізації даного проекту мною було проведено:

- аналіз предметної області;
- первинне моделювання системи та встановлення основних вимог до неї;
- створена база даних на базі PostgreSQL;
- створена серверна частина застосунку з використанням програмної платформи node.js;
- створена клієнтська частина застосунку з використанням бібліотеки динамічного відтворення контенту React.js;
- успішно проведене випробування системи в різних умовах, по результатах якої, можна констатувати, що застосунок відповідає умовам поставленим при моделювання системи.

Як результат дипломної роботи, отримано комп'ютерну систему, що має зрозумілий дизайн, повноцінний функціонал для роботи з замовленнями, має потенціал для розширення та вже може бути використана для полегшення адміністративної завантаженості роботи кінотеатру.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is Ecommerce? URL: <https://www.bigcommerce.com/articles/ecommerce/> (Дата звертання: 29.04.2024).
2. Ticketing Software Buyers Guide. - URL: <https://www.capterra.com/ticketing-software/buyers-guide/> (Дата звертання 27.04.2024).
3. Кінотеатр Довженка: вебсайт. URL: <http://dov.zp.ua/> (Дата звертання: 04.05.2024).
4. Планета кіно: вебсайт. URL: <https://planetakino.ua/> (Дата звертання: 04.05.2024).
5. Про Планету Кіно. URL: <https://planetakino.ua/about/> (Дата звертання 04.05.2024).
6. IEEE Standard Glossary of Software Engineering Terminology. URL: <https://ieeexplore.ieee.org/document/159342> (Дата звертання: 29.04.2024).
7. Що краще моноліт чи мікросервіси? Як обрати архітектуру проєкту? URL: <https://iampm.club/ua/blog/shho-krashhe-monolit-chi-mikroservisi-yak-obrati-arhitekturu-projektu/> (Дата звертання: 29.04.2024).
8. Client-Server Applications vs Monolithic vs Microservice. URL: https://www.linkedin.com/posts/cloud-sheger_client-server-applications-vs-monolithic-activity-7093080165974462465-aLbp (Дата звертання: 30.04.2024).
9. Рейтинг мов програмування 2023. URL: <https://dou.ua/lenta/articles/language-rating-2023/> (Дата звертання: 29.04.2024).
10. The Good and the Bad of Node.js Web App Programming. URL: <https://www.altexsoft.com/blog/the-good-and-the-bad-of-node-js-web-app-development/> (Дата звертання: 29.04.2024).
12. What is Java? URL: <https://aws.amazon.com/what-is/java> (Дата звертання: 30.04.2024).

13. A Comprehensive Guide to PHP Programming: What You Need to Know. URL: <https://codeinstitute.net/global/blog/what-is-php-programming/> (Дата звертання: 30.04.2024).

14. Advantages and disadvantages of PHP you should know when starting a new project. URL: <https://anywhere.epam.com/en/blog/pros-and-cons-of-php> (Дата звертання: 30.04.2024).

15. What Is Python Used For? A Beginner's Guide. URL: <https://anywhere.epam.com/en/blog/pros-and-cons-of-php> (Дата звертання: 01.05.2024).

16. Pros and Cons of Python Programming Language. URL: <https://www.pixelcrayons.com/blog/software-development/python-pros-and-cons/> (Дата звертання: 01.05.2024).

17. What Is HTML? Hypertext Markup Language Basics Explained. URL: <https://www.hostinger.com/tutorials/what-is-html> (Дата звертання: 01.05.2024).

18. What is CSS: An Introduction to Cascading Style Sheets. URL: <https://blog.lewagon.com/skills/what-is-css/> (Дата звертання: 01.05.2024).

19. What Is JavaScript & What Is It Used For? A Basic Guide to JS. URL: <https://www.semrush.com/blog/javascript/> (Дата звертання: 01.05.2024).

20. SPA Vs. MPA Explained: Differences Between Single-Page Applications and Multi-Page Applications. URL: <https://www.orientsoftware.com/blog/spa-vs-mpa/> (Дата звертання: 01.05.2024).

21. Pros and Cons of Angular Development. URL: <https://agiliway.com/pros-and-cons-of-angular-development/> (Дата звертання: 01.05.2024).

22. Using Vue: Pros and Cons. URL: <https://thecodest.co/blog/pros-and-cons-of-vue/> (Дата звертання: 01.05.2024).

23. Pros and Cons of React. URL: <https://thecodest.co/blog/pros-and-cons-of-react/> (Дата звертання: 01.05.2024).

24. The Advantages and Disadvantages of MySQL Database. URL: <https://www.remoteplatz.ch/blog/advantages-and-disadvantages-of-mysql> (Дата звертання: 02.05.2024).

25. The Importance of MySQL Database: Advantages, Disadvantages, and Influence in AI and Data. URL: <https://www.linkedin.com/pulse/importance-mysql-database-advantages-disadvantages-influence-bk-xhfce> (Дата звертання: 02.05.2024).

26. MongoDB Advantages & Disadvantages, URL: <https://www.geeksforgeeks.org/mongodb-advantages-disadvantages/> (Дата звертання: 02.05.2024).

27. PostgreSQL Advantages and Disadvantages. URL: <https://aalpha.net/blog/pros-and-cons-of-using-postgresql-for-application-development/> (Дата звертання: 02.05.2024).

28. SQLite vs MySQL vs PostgreSQL – The Search For The “Best” Relational Database Management System. URL: <https://runcloud.io/blog/sqlite-vs-mysql-vs-postgresql> (Дата звертання: 02.05.2024).

ДОДАТОК А КОДИ ДО МОДУЛІВ

Лістинг А.1 – models.js

```

const sequelize = require('../db')
const { DataTypes } = require('sequelize')

const User = sequelize.define('user', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  name: { type: DataTypes.STRING, unique: true, allowNull: false },
  email: { type: DataTypes.STRING, unique: true, allowNull: false },
  password: { type: DataTypes.STRING, allowNull: false },
  role: { type: DataTypes.STRING, defaultValue: "ADMIN" }
}, { timestamps: false })

const Movie = sequelize.define('movie', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  title: { type: DataTypes.STRING },
  titleUa: { type: DataTypes.STRING, allowNull: false },
  director: { type: DataTypes.STRING, allowNull: false },
  actors: { type: DataTypes.STRING, allowNull: false },
  year: { type: DataTypes.INTEGER, allowNull: false },
  country: { type: DataTypes.STRING, allowNull: false },
  duration: { type: DataTypes.INTEGER, allowNull: false },
  description: { type: DataTypes.TEXT, allowNull: false, maxLength: 100 },
  img: { type: DataTypes.STRING, allowNull: false },
})

const Genre = sequelize.define('genre', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  name: { type: DataTypes.STRING, unique: true, allowNull: false },
}, { timestamps: false })

const Hall = sequelize.define('hall', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  color: { type: DataTypes.STRING, allowNull: false },
}, { timestamps: false })

const Seat = sequelize.define('seat', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  row: { type: DataTypes.INTEGER, allowNull: false },
  place: { type: DataTypes.INTEGER, allowNull: false },
  price: { type: DataTypes.INTEGER, allowNull: false },
}, { timestamps: false })

const Seance = sequelize.define('seance', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  beginning: { type: DataTypes.DATE, allowNull: false },
  ending: { type: DataTypes.DATE, allowNull: false },
  actual: { type: DataTypes.BOOLEAN, defaultValue: true }
}, { timestamps: false })

const Order = sequelize.define('order', {
  id: {
    type: DataTypes.INTEGER,
    primaryKey: true,
    unique: true,
  },

```

```

    name: { type: DataTypes.STRING, allowNull: false },
    email: { type: DataTypes.STRING, allowNull: false },
    userId: { type: DataTypes.INTEGER, allowNull: true },
    seanceId: { type: DataTypes.INTEGER, allowNull: false }
  }, { timestamps: true })

Order.beforeCreate(async (order) => {
  const lastOrder = await Order.findOne({
    order: [['id', 'DESC']],
  });

  const lastId = lastOrder ? lastOrder.id : 9999999;
  order.id = lastId + 1;
});

const MovieGenre = sequelize.define('movieGenre', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true }
}, { timestamps: false })

const SeatSeance = sequelize.define('seatSeance', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  status: { type: DataTypes.STRING, allowNull: false, defaultValue:
"EMPTY" },
}, { timestamps: false });

User.hasMany(Order)
Order.belongsTo(User)

Movie.hasMany(Seance, { onDelete: 'CASCADE' })
Seance.belongsTo(Movie)

Order.belongsToMany(SeatSeance, { through: 'OrderSeatSeances' });
SeatSeance.belongsToMany(Order, { through: 'OrderSeatSeances' });

Hall.hasMany(Seat)
Seat.belongsTo(Hall)

Hall.hasMany(Seance)
Seance.belongsTo(Hall)

Seance.hasMany(Order)
Order.belongsTo(Seance)

SeatSeance.belongsTo(Seat);
Seat.hasMany(SeatSeance);

SeatSeance.belongsTo(Seance);
Seance.hasMany(SeatSeance);

Seance.belongsToMany(Seat, { through: SeatSeance, as: 'seats' })
Seat.belongsToMany(Seance, { through: SeatSeance, as: 'seances' })

Seance.hasMany(SeatSeance, { foreignKey: 'seanceId' })
Seat.hasMany(SeatSeance, { foreignKey: 'seatId' })

Movie.belongsToMany(Genre, { through: MovieGenre, as: 'genres' })
Genre.belongsToMany(Movie, { through: MovieGenre, as: 'movies' })

Movie.hasMany(MovieGenre, { foreignKey: 'movieId' })
Genre.hasMany(MovieGenre, { foreignKey: 'genreId' })

module.exports = {
  User,
  Movie,

```

```

Genre,
MovieGenre,
Seat,
Seance,
SeatSeance,
Hall,
Order
}

```

Лістинг А.2 – movieController.js

```

const uuid = require('uuid')
const path = require('path')
const {Movie, MovieGenre, Review, User, Genre, Seance} =
require('../models/models')
const ApiError = require('../error/apiError')
const {Op} = require("sequelize")

class MovieController {
  async create(req, res, next) {
    try {
      let {title, titleUa, director, actors, year, country, duration,
description, genreId} = req.body

      const {img} = req.files
      let ilename = uuid.v4() + ".jpg"
      await img.mv(path.resolve(__dirname, '..', 'static', fileName))

      const movie = await Movie.create({
        title,
        titleUa,
        director,
        actors,
        year,
        country,
        duration,
        description,
        img: fileName
      })
      genreId = JSON.parse(genreId)
      const movieGenre = genreId.forEach(€ =>
        MovieGenre.create({
          movieId: movie.id,
          genreId: i.genreId
        })
      )

      return res.json(movieGenre)
    } catch € {
      return next(ApiError.internal('Необхідно заповнити усі поля'));
    }
  }

  async getAll(req, res) {
    let { genreId, limit, page, search } = req.query
    page = page || 1
    limit = limit || 6
    let offset = page * limit - limit
    let movies

    if (genreId) {
      movies = await Movie.findAndCountAll({
        include: [{ model: MovieGenre, where: { genreId } }],

```

```

        limit,
        offset,
        order: [['createdAt', 'DESC']],
    })
} else if (search) {
    movies = await Movie.findAndCountAll({
        where: {
            [Op.or]: [
                { title: { [Op.iLike]: `%${search}%` } },
                { titleUa: { [Op.iLike]: `%${search}%` } },
            ],
        },
        limit,
        offset,
        order: [['createdAt', 'DESC']],
    })
} else {
    movies = await Movie.findAndCountAll({ limit, offset })
}

return res.json(movies)
}

async getOne(req, res) {
    const {id} = req.params
    const movie = await Movie.findOne(
        {
            where: {id},
            include: [
                {
                    model: €,
                    as: 'seances'
                },
                {
                    model: Genre,
                    as: 'genres'
                },
            ],
        }
    )

    return res.json(movie)
}

async deleteOne(req, res, next) {
    const { id } = req.params;
    try {
        const movie = await Movie.findByPk(id);
        if (!movie) {
            return next();
        }

        await movie.destroy();

        return res.json({ message: 'Фільм було успішно видалено' });
    } catch (error) {
        return next(ApiError.internal('Помилка при видаленні фільму'));
    }
}

module.exports = new MovieController()

```

ЛІСТИНГ А.3 – userController.js

```

const ApiError = require('../error/apiError')
const bcrypt = require('bcrypt')
const jwt = require('jsonwebtoken')
const validator = require('validator')
const {User} = require('../models/models')

const generateJwt = (id, name, email, role) => {
  return jwt.sign(
    {id, name, email, role},
    process.env.SECRET_KEY,
    {expiresIn: '24h'}
  )
}

class UserController {
  async registration(req, res, next) {
    const { name, email, password, role } = req.body;

    const candidateEmail = await User.findOne({ where: { email } });

    if (candidateEmail) {
      return next(ApiError.badRequest('Користувач з таким email вже існує'))
    }

    const candidateName = await User.findOne({ where: { name } });

    if (candidateName) {
      return next(ApiError.badRequest("Користувач з таким ім'ям вже існує"))
    }

    const hashPassword = await bcrypt.hash(password, 5);

    const user = await User.create({
      name,
      email,
      role,
      password: hashPassword,
    });

    const token = generateJwt(user.id, user.name, user.email, user.role);
    return res.json({ token });
  }

  async login(req, res, next) {
    try {
      const {email, password} = req.body
      const user = await User.findOne({where: {email}})

      if (!user) {
        return next(ApiError.internal('Такого користувача не існує, перевірте введені дані'))
      }

      if (!validator.isEmail(email)) {
        return next(ApiError.internal('Некорректний формат email'))
      }
    }
  }
}

```

```

        let comparePassword = bcrypt.compareSync(password,
user.password)

        if (!comparePassword) {
            return next(ApiError.internal('Вказано невірний пароль'))
        }

        const token = generateJwt(user.id, user.name, user.email,
user.role)
        return res.json({token})
    } catch (e) {
        return next(ApiError.internal('Такого користувача не існує,
перевірте введені дані'));
    }

    }

    async check(req, res) {
        const token = generateJwt(req.user.id, req.user.name,
req.user.email, req.user.role)
        return res.json({token})
    }
}

module.exports = new UserController()

```

Лістинг А.4 – App.js

```

import React, { useContext, useEffect, useState } from 'react';
import { BrowserRouter } from 'react-router-dom';
import AppRouter from './components/AppRouter';
import NavBar from './components/NavBar';
import { observer } from 'mobx-react';
import { Context } from './index';
import { check } from './http/userAPI';
import Footer from './components/Footer';
import './App.css';

const App = observer(() => {
    const { user } = useContext(Context);
    const [loading, setLoading] = useState(true);

    useEffect(() => {
        check().then((decodedToken) => {
            user.setUser(true);
            user.setIsAuth(true);
            user.setIsAdmin(decodedToken.isAdmin);
            if (decodedToken.role === 'ADMIN') {
                user.setIsAdmin(true);
            } else {
                user.setIsAdmin(false);
            }
        })
        .catch(() => {
            user.setUser(false);
            user.setIsAuth(false);
            user.setIsAdmin(false);
        })
        .finally(() => setLoading(false));
    }, []);

    if (loading) {

```

```

        return (
            <div className="spinner-grow text-warning justify-content-
center" role="status">
                <span className="visually-hidden">Loading...</span>
            </div>
        );
    }

    return (
        <BrowserRouter>
            <div className="app-container">
                <NavBar />
                <div className="main-content-container">
                    <AppRouter />
                </div>
                <Footer />
            </div>
        </BrowserRouter>
    );
});

export default App;

```

Лістинг А.5 – Seance.js

```

import React, { useEffect, useState } from 'react';
import { useParams, useNavigate } from "react-router-dom";
import { createOrder, fetchOneSeance } from "../http/movieAPI";
import { check } from "../http/userAPI";
import { Modal, Button, Form, Image } from "react-bootstrap";
import "../Seance.css";
import { format, parseISO } from "date-fns";
import { uk } from "date-fns/locale";

const Seance = () => {
    const [user, setUser] = useState({ id: 0 });
    const [seance, setSeance] = useState(null);
    const { id } = useParams();
    const navigate = useNavigate();
    const [seatSeanceId, setSeatSeanceId] = useState([]);
    const [selectedSeats, setSelectedSeats] = useState([]);
    const [totalPrice, setTotalPrice] = useState(0);
    const [showModal, setShowModal] = useState(false);
    const [name, setName] = useState('');
    const [email, setEmail] = useState('');

    useEffect(() => {
        check()
            .then((data) => setUser(data))
            .catch((error) => {
                if (error.response && error.response.status === 401) {
                    setUser({ id: 0 });
                } else {
                    console.error("Unexpected error during authorization
check:", error);
                }
            });
    }, []);

    useEffect(() => {
        fetchOneSeance(id).then(data => setSeance(data));
    }, [id]);

```

```

if (!seance) {
  return <div>Loading...</div>;
}

const groupSeatsByRow = (seats) => {
  const grouped = seats.reduce((rows, seat) => {
    if (!rows[seat.row]) {
      rows[seat.row] = [];
    }
    rows[seat.row].push(seat);
    return rows;
  }, {});
  Object.keys(grouped).forEach(row => {
    grouped[row].sort((a, b) => a.place - b.place);
  });
  return grouped;
};

const handleSeatClick = (seat) => {
  const seatSeance = seat.seatSeances.find(ss => ss.seanceId ===
seance.id);
  if (seatSeance.status === 'BOOKED') return;

  setSeatSeanceId(prev => {
    if (prev.includes(seatSeance.id)) {
      setSelectedSeats(selectedSeats.filter(s => s.id !==
seat.id));
      setTotalPrice(totalPrice - seat.price);
      return prev.filter(id => id !== seatSeance.id);
    } else {
      setSelectedSeats([...selectedSeats, seat]);
      setTotalPrice(totalPrice + seat.price);
      return [...prev, seatSeance.id];
    }
  });
};

const addOrder = async () => {
  if (!user.id) {
    setShowModal(true);
    return;
  }
  submitOrder(user.name, user.email);
};

const submitOrder = async (name, email) => {
  if (!name || !email) {
    alert('Будь ласка, заповніть усі поля.');
```

return;

```

  }
  try {
    const formData = new FormData();
    formData.append('name', name);
    formData.append('email', email);
    formData.append('userId', user.id || 0);
    formData.append('seanceId', seance.id);
    formData.append('seatSeanceId', JSON.stringify(seatSeanceId));
    await createOrder(formData);
    alert("Вітаємо, місця заброньовані!");
    navigate(`/movie/${seance.movie.id}`);
    // Update the seance to reflect the booked seats
    fetchOneSeance(id).then(data => setSeance(data));
  } catch (e) {
    alert(e.response.data.message);
  }
};

```

```

    }
  };

  const getSeatStatus = (seat) => {
    const seatSeance = seat.seatSeances.find(ss => ss.seanceId ===
seance.id);
    return seatSeance.status;
  };

  const renderSeats = (seats) => {
    return seats.map(seat => (
      <button
        key={seat.id}
        className={`seat-btn ${seat.price === 140 ?
getSeatStatus(seat) === 'BOOKED' ? 'BOOKED' : 'price-140' : getSeatStatus(seat)}
${seatSeanceId.includes(seat.seatSeances.find(ss => ss.seanceId === seance.id).id)
? 'selected' : ''}`}
        title={`${seat.price} ррн, ${getSeatStatus(seat)}`}
        onClick={() => handleSeatClick(seat)}
      >
        {seat.place}
      </button>
    ));
  };

  const formatDate = (dateTime) => {
    const date = parseISO(dateTime);
    const formattedDate = format(date, 'd MMMM yyyy', { locale: uk });
    const formattedTime = format(date, 'HH:mm');
    return { formattedDate, formattedTime };
  };

  const { formattedDate, formattedTime: startTime } =
formatDate(seance.beginning);
  const { formattedTime: endTime } = formatDate(seance.ending);

  const seatsByRow = groupSeatsByRow(seance.hall.seats);

  return (
    <div className="seance-container">
      <div className="seance-header">
        <Image
          className="seance-poster"
          src={seance.movie?.img ? process.env.REACT_APP_API_URL +
seance.movie.img : ''}
        />
        <div className="seance-info">
          <h1>{seance.movie?.titleUa}</h1>
          <p>Зала: {seance.hall?.color}</p>
          <p>{formattedDate}</p>
          <p>{startTime} - {endTime}</p>
        </div>
      </div>
      <hr/>
      <div className="seats-container">
        {Object.keys(seatsByRow).map(row => (
          <div key={row} className="seat-row">
            <h3>Ряд {row}</h3>
            <div className="seat-row-buttons">
              {renderSeats(seatsByRow[row])}
            </div>
          </div>
        ))}
      </div>
    </div>
  );

```

```

</div>
{selectedSeats.length > 0 && (
  <div className="order-summary">
    <p>Ваше замовлення:</p>
    {selectedSeats.map(seat => (
      <p key={seat.id}>Ряд: {seat.row}, Місце:
{seat.place}, Ціна: {seat.price} грн</p>
    ))}
    <p>Загальна сума: {totalPrice} грн</p>
  </div>
)}

<Modal show={showModal} onHide={() => setShowModal(false)}>
  <Modal.Header closeButton>
    <Modal.Title>Заповніть ваші дані</Modal.Title>
  </Modal.Header>
  <Modal.Body>
    <Form>
      <Form.Group controlId="formName">
        <Form.Label>Ім'я</Form.Label>
        <Form.Control
          type="text"
          placeholder="Введіть ваше ім'я"
          value={name}
          onChange={(e) => setName(e.target.value)}
        />
      </Form.Group>
      <Form.Group controlId="formEmail">
        <Form.Label>Email</Form.Label>
        <Form.Control
          type="email"
          placeholder="Введіть ваш email"
          value={email}
          onChange={(e) => setEmail(e.target.value)}
        />
      </Form.Group>
    </Form>
  </Modal.Body>
  <Modal.Footer>
    <Button variant="secondary" onClick={() =>
setShowModal(false)}>
      Скасувати
    </Button>
    <Button variant="primary" onClick={() =>
submitOrder(name, email)}>
      Оформити замовлення
    </Button>
  </Modal.Footer>
</Modal>
<div className="btn-seance-container">
  <div className="btn-seance"
onClick={addOrder}>Оформити</div>
</div>
</div>
);
};

export default Seance;

```

Лістинг А.6 – User.js

```

import React, { useEffect, useState } from 'react';
import { Container, Image, Row, Col, Button } from "react-bootstrap";
import { useParams } from "react-router-dom";
import { check } from "../http/userAPI";
import { fetchUserOrders } from "../http/movieAPI";
import jsPDF from 'jspdf';
import robotoFont from '../components/fonts/roboto.js';

const User = () => {
  const [, setMovie] = useState('');
  const [user, setUser] = useState('');
  const [orders, setOrders] = useState([]);
  const { id } = useParams();

  useEffect(() => {
    check(id).then((data) => setUser(data));
  }, [id]);

  useEffect(() => {
    if (user.id && !isNaN(Number(user.id))) {
      const userId = Number(user.id);
      fetchUserOrders(userId).then((data) => setOrders(data));
    }
  }, [user.id]);

  const formatOrderDate = (dateString) => {
    const options = { year: 'numeric', month: 'long', day: 'numeric',
hour: '2-digit', minute: '2-digit' };
    const date = new Date(dateString);
    return date.toLocaleDateString('uk-UA', options);
  };

  const formatSeanceTime = (dateString) => {
    const options = { hour: '2-digit', minute: '2-digit' };
    const date = new Date(dateString);
    return date.toLocaleTimeString('uk-UA', options);
  };

  const calculateTotalPrice = (seatSeances) => {
    return seatSeances.reduce((total, seatSeance) => total +
seatSeance.seat.price, 0);
  };

  const downloadPDF = async (order) => {
    const doc = new jsPDF();

    // Добавление шрифта
    doc.addFileToVFS("Roboto-Regular.ttf", robotoFont);
    doc.addFont("Roboto-Regular.ttf", "Roboto", "normal");
    doc.setFont("Roboto");

    // Загрузка изображения и преобразование в base64
    const imgData = await fetch(order.seance.movie?.img
process.env.REACT_APP_API_URL + order.seance.movie.img : '')
    .then(res => res.blob())
    .then(blob => new Promise((resolve, reject) => {
      const reader = new FileReader();
      reader.onloadend = () => resolve(reader.result);
      reader.onerror = reject;
      reader.readAsDataURL(blob);
    }));
  };
};

```

```

    doc.addImage(imgData, 'JPEG', 10, 10, 50, 75);

    doc.text(`Номер брони: ${order.id}`, 10, 100);
    doc.text(`Користувач: ${user.name}`, 10, 110);
    doc.text(`Фільм: ${order.seance.movie.titleUa}`, 10, 120);
    doc.text(`Початок сеансу: ${formatSeanceTime(order.seance.beginning)}`, 10, 130);
    doc.text(`Кінець сеансу: ${formatSeanceTime(order.seance.ending)}`, 10, 140);
    doc.text(`Дата замовлення: ${formatOrderDate(order.createdAt)}`, 10, 150);

    let y = 160;
    order.seatSeances.forEach((seatSeance, index) => {
      doc.text(`Місце ${index + 1}: Ряд ${seatSeance.seat.row}, Місце ${seatSeance.seat.place}, Ціна ${seatSeance.seat.price} грн`, 10, y);
      y += 10;
    });

    const totalPrice = calculateTotalPrice(order.seatSeances);
    doc.text(`Загальна сума: ${totalPrice} грн`, 10, y + 10);

    doc.save(`order-${order.id}.pdf`);
  };

  return (
    <Container className='border border-light-subtle mt-3' fluid>
      <Row className="justify-content-center">
        <Col lg={8} md={10} sm={12}>
          <div className="card border-light fs-5 mt-3">
            <div className="card-header text-center">
              {user.name}
            </div>
            <div className="card-body">
              <h5 className="card-title">Email</h5>
              <p className="card-text mb-2">{user.email}</p>
            </div>
          </div>
        </Col>
        <Col lg={8} md={10} sm={12} id="orders-section">
          {orders && orders.length > 0 ? (
            <h4 className="text-center mt-3">Ваші замовлення</h4>
            <Row className="justify-content-center">
              {orders.map((order) => (
                <Col lg={12} md={12} sm={12}
                  className="mb-3" key={order.id}>
                  <div className="card fs-5 mt-3"
                    id={`order-${order.id}`}>
                    <div className="card-header d-
                      flex align-items-center">
                        <Image
                          className='seance-poster
                            me-3'
                          src={order.seance.movie?.img ? process.env.REACT_APP_API_URL +
                            order.seance.movie.img : ''}
                          width={100}
                        />
                        <div className="flex">
                          <h5>Номер брони:
                            {order.id}</h5>

```

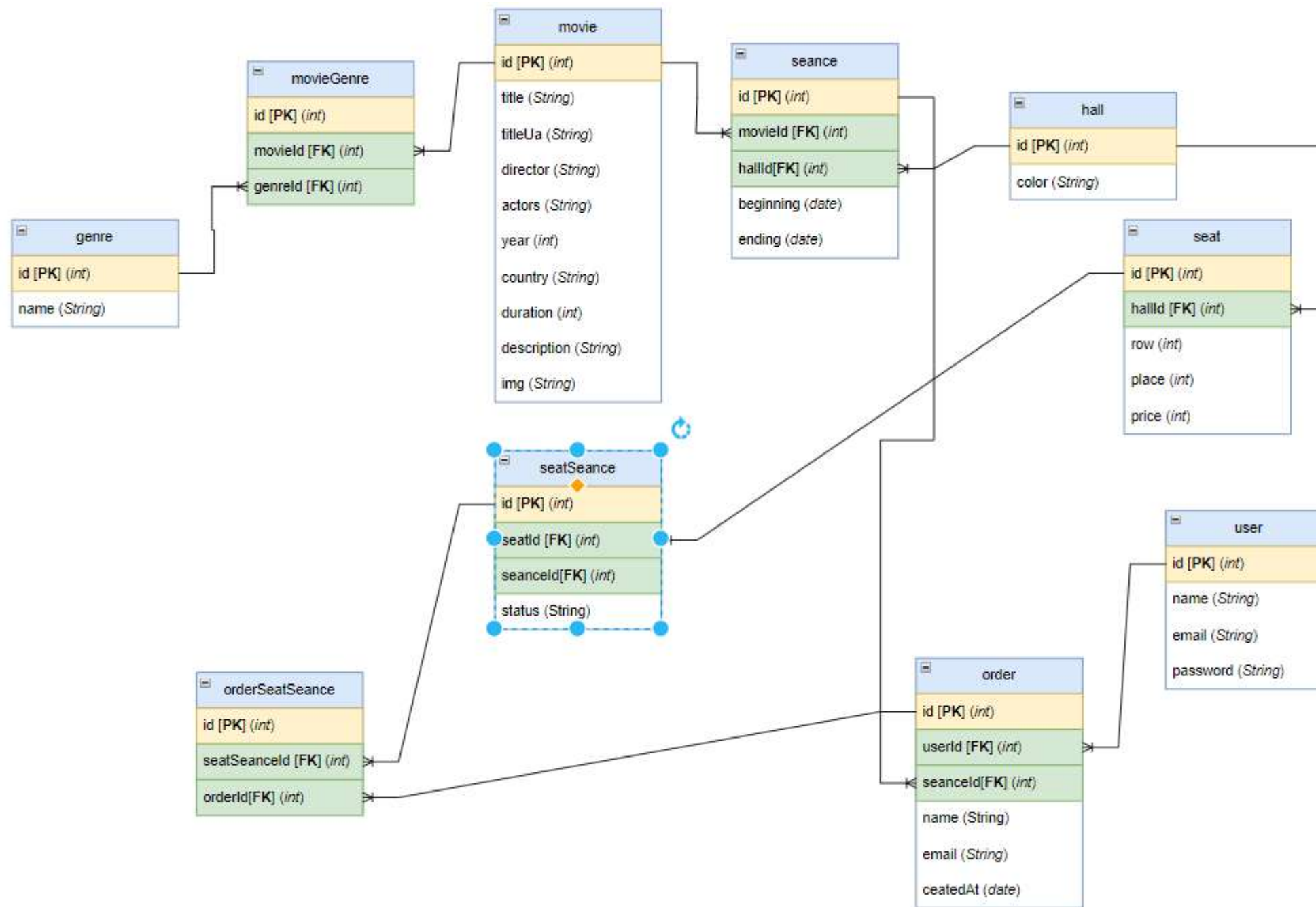
```

{order.seance.movie.titleUa}</em>
{formatSeanceTime(order.seance.beginning)}</p>
{formatSeanceTime(order.seance.ending)}</p>
</div>
</div>
<div className="card-body">

{order.seatSeances.map((seatSeance, index) => (
  <div key={index}>
    <h6 className="card-
title">
      Ряд:
      Місце: {seatSeance.seat.place},
      Ціна:
      {seatSeance.seat.price} грн
    </h6>
  </div>
))}
<p>
  <h6>Загальна сума:
  {calculateTotalPrice(order.seatSeances)} грн</h6>
</p>
<hr/>
<p className="card-title">
  <em>Дата замовлення:
  {formatOrderDate(order.createdAt)}</em>
</p>
<Button variant="success"
  onClick={ () =>
    downloadPDF(order)}>Завантажити ел.
    квиток</Button>
  </div>
</div>
</Col>
))}
</Row>
</>
) : (
  <h4 className="text-center mt-3">Ви ще не робили
замовлень</h4>
  </Col>
</Row>
</Container>
);
};

export default User;

```



Зам.	Лист.	№ докум.	Підп.	Дата
Розроб.		Вертман М.А.		
Перев.		Скрупський С.Ю.		
Т.контр.				
Н.контр.		Польська О. В.		
Затв.		Кудерметов Р.К.		

13.02070849.00042 ПЛ1

Розробка комп'ютерної системи
замовлення квитків для
кінотеатру
ER-модель бази даних

Лист.	Маса	Масшт.
Лист 1	Листів 1	

НУ «Запорізька політехніка»
КНТ-520

