

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

ПОЯСНЮВАЛЬНА ЗАПИСКА

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

на тему:

«Онлайн сервіс бронювання квитків у кінотеатрах»

Виконав студент групи ПЗ-19-1
ДРЖЕВЕЦЬКИЙ Микита Олегович

Керівник роботи:
ЛОБАНЧИКОВА Надія Миколаївна

Рецензент:
МОРОЗОВ Андрій Васильович

Житомир – 2023

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

«ЗАТВЕРДЖУЮ»

В.о. зав. кафедри інженерії
програмного забезпечення
Андрій МОРОЗОВ
«14» лютого 2023 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **Држевецький Микита Олегович**

Керівник роботи: **Лобанчикова Надія Миколаївна**

Тема роботи: **«Онлайн сервіс бронювання квитків у кінотеатрах»,**
затверджена Наказом закладу вищої освіти від **«15» лютого 2023 р., №70с**

Вихідні дані для роботи: використання технологій, структур, засобів та методів для розробки веб-додатку, методи та засоби створення веб-додатку з використанням стеку MERN. Орієнтовний перелік фільмів, приклад опису, перелік залів, розклад роботи кінотеатрів та сеансів перегляду фільмів.

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

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю.М.	16.02.2023р.	16.02.2023р.
2	Єфремов Ю.М.	16.02.2023р.	16.02.2023р.
3	Єфремов Ю.М.	16.02.2023р.	16.02.2023р.

РЕФЕРАТ

Кваліфікаційна робота присвячена розробці «Онлайн сервісу бронювання квитків у кінотеатрах» та пояснювальної записки. Пояснювальна записка до кваліфікаційної роботи містить 62 сторінки, 23 ілюстрації та 5 таблиць.

Метою роботи є розробка веб-додатку за допомогою якого можна бронювати квитки з різних кінотеатрів, що зареєстровані у системі.

В роботі визначено основні завдання на розробку системи, проаналізовано аналоги розробленої системи. Наведено сценарії роботи веб-додатку, його загальну структуру, опис алгоритмів взаємодії окремих модулів системи, опис об'єктної структури веб-додатку. Також наводяться результати тестування додатку.

КЛЮЧОВІ СЛОВА: AXIOS, ANT, CRUD, EXPRESS, NODE.JS, MERN, REACT, WEB APPLICATION, WEB SERVICE, ВЕБ-ДОДАТОК, ВЕБ-СЕРВІС.

					ІПЗ.КР.Б – 121 – 23 – ПЗ			
Зм.	Арк.	№ докум.	Підпис	Дата	Онлайн сервіс бронювання квитків у кінотеатрах Пояснювальна записка	Літ.	Арк.	Аркушів
Розроб.		М.О. Держевецький					3	62
Керівник		Н.М. Лобанчикова						
Рецензент		А.В.Морозов				Житомирська політехніка, група ІПЗ-19-1		
Зав. каф.		А.В.Морозов						

ABSTRACT

The qualification work is dedicated to the development of an "Online Ticket Booking Service for Cinemas" and its explanatory memorandum. The explanatory memorandum for the qualification work consists of 62 pages, 23 illustrations, and 5 tables.

The aim of this work is to develop a web application that allows users to book tickets from various registered cinemas within the system.

The work defines the main tasks for the development of the system, analyzes existing analogues of the developed system, presents scenarios for the operation of the web application, outlines its overall structure, describes the algorithms for the interaction of individual system modules, and provides an overview of the object structure of the web application. Additionally, the testing results of the application are presented.

KEYWORDS: AXIOS, ANT, CRUD, EXPRESS, NODE.JS, MERN, REACT, WEB APPLICATION, WEB SERVICE.

					ПЗ.КР.Б – 121 – 23 – ПЗ	
						4

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ ТА ОСОБЛИВОСТЕЙ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ОНЛАЙН СЕРВІСУ БРОНЮВАННЯ КВИТКІВ У КІНОТЕАТРАХ	9
1.1 Постановка задачі.....	9
1.2 Аналіз аналогів програмного продукту	10
1.3 Вибір архітектури веб-додатку	15
1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення	17
Висновки до першого розділу.....	20
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ	21
2.1 Визначення варіантів використання та структури системи.....	21
2.2 Розробка бази даних системи.....	25
2.3 Проєктування алгоритмів роботи системи.....	29
2.4 Реалізація веб-додатку онлайн сервіса бронювання квитків у кінотеатрах	31
Висновки до другого розділу	36
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ.....	38
3.1 Порядок встановлення та налаштування системи.....	38
3.2 Структура інтерфейсу. Інтерфейс та порядок роботи з веб-додатком.....	39
Висновки до третього розділу	46
ВИСНОВКИ.....	47
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	48
ДОДАТКИ.....	50

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

- API – Application Programming Interface (Прикладний програмний інтерфейс).
- CRUD – Create, Read, Update, Delete (створення, читання, оновлення, видалення).
- HTTP – HyperText Transfer Protocol (протокол передачі гіпертексту).
- JSON – JavaScript Object Notation (формат збереження даних).
- MERN – Mongo Express React Node (Це JavaScript-стек, що складається з 4 компонентів із відкритим кодом, які забезпечують комплексне середовище розробки).
- UML – Unified Modeling Language (уніфікована мова моделювання).
- БД – База даних.

ВСТУП

Актуальність теми. Протягом останніх декількох років сфера кіноіндустрії дуже сильно набуває популярності. Багато людей обирають кінотеатри як спосіб відпочинку та розваг. Замість того, щоб витратити час на візит до каси кінотеатру, людина може зручно забронювати квитки в будь-який зручний для неї час з будь-якого місцерозташування, використовуючи мобільний пристрій або комп'ютер. Отримання квитка в кінотеатрі може вимагати значних зусиль і забрати багато часу. Щоб уникнути таких проблем, кінотеатри розробляють власні онлайн системи бронювання квитків, що надає клієнтам зручність та доступність.

Мета даної кваліфікаційної роботи полягає у детальному вивченні та аналізі онлайн-кінотеатрів та систем бронювання квитків для розробки онлайн сервісу бронювання квитків у кінотеатрах.

Встановлена мета обумовлює наступні завдання:

- Аналіз можливих задач додатка: провести дослідження та виявити основні задачі, які має вирішувати інформаційний додаток для онлайн бронювання квитків у кінотеатрах.
- Вивчення аналогів: дослідити наявні онлайн сервіси бронювання квитків у кінотеатрах та оцінити їхні недоліки та переваги. Визначити конкурентні переваги та унікальність розробленого додатка.
- Визначити структуру та алгоритм додатка: забезпечити функціонал для перегляду розкладу сеансів, бронювання квитків, процесу оплати та вибору фільмів.
- Розробка додатка: реалізувати інформаційний додаток згідно з визначеною структурою та алгоритмом. Забезпечити зручний та інтуїтивно зрозумілий інтерфейс для користувачів.
- Опис роботи додатка: детально описати функціонал та можливості інформаційного додатка, його особливості та переваги в порівнянні з іншими аналогами.

- Тестування інформаційного додатка: провести тестування для перевірки функціональності, надійності та коректності роботи додатка. Виявити та виправити можливі помилки та проблеми.

Для досягнення поставленої мети необхідно вирішити ряд завдань:

- вивчити сучасні технології та проаналізувати інструменти для проєктування сайтів;
- складання вимог до сайту;
- проєктування та впровадження бази даних;
- реалізувати логіку серверної частини;
- дизайн-архітектури сайту; налагодження та тестування сайту.

Об’єктом дослідження є процес проєктування веб-застосунку з онлайн замовленням квитків у кінотеатрах.

Предметом дослідження є методи, засоби, алгоритми та технології реалізації веб-додатку за допомогою стеку MERN.

Розроблений веб-додаток можна використати як централізовану систему кінотеатрів, де користувач зможе забронювати квиток у будь-якому кінотеатрі, що зареєстрований у системі.

Кваліфікаційна робота складається з вступу, 3-х розділів, висновків, списку використаних джерел з найменувань, додатків, таблиць та ілюстрацій по тексту. Повний обсяг роботи становить 62 сторінки, з них 49 сторінок основного тексту.

РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ ТА ОСОБЛИВОСТЕЙ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ОНЛАЙН СЕРВІСУ БРОНЮВАННЯ КВИТКІВ У КІНОТЕАТРАХ

1.1 Постановка задачі

Основною метою розробки є створення зручної платформи для користувачів, яка дозволить їм легко бронювати квитки на фільми у кінотеатрах. Крім того, система має функціонал, що дозволяє додавати фільми у систему та створювати сеанси у потрібну дату та час. Отже, основна мета полягає у спрощенні процесу бронювання квитків, розроблений веб-додаток можна використовувати як загальну систему кінотеатрів, де користувачі зможуть забронювати квитки на сеанси у будь-якому з кінотеатрів, що входять до системи.

Основними етапами реалізації цього завдання є:

- Створення клієнтської частини:
 - створення структури папок;
 - створення таблиць стилів;
 - створення завантажувачів за допомогою ReduxToolkit [11];
 - Використання бібліотеки Ant з використанням готових компонентів.
- Створення серверної частини:
 - створення структури папок;
 - створення ядра серверу;
 - проєктування бази даних;
 - налаштування бази даних MongoDB [14];
 - створення необхідних моделей бази даних з відповідними полями:
 - модель користувачів;
 - модель кінотеатрів;
 - модель сеансів;
 - модель фільмів;

- модель бронювання.
- Створення адмін панелі:
 - CRUD фільмів;
 - встановлення статусу кінотеатру.
- Створення панелі користувача:
 - перегляд фільмів;
 - бронювання квитка;
 - перегляд сторінки особистих бронювання;
 - надавання запиту адміністратору на додавання кінотеатру у систему;
 - CRUD сеансів (якщо користувач отримав підтвердження на додавання власного кінотеатру);
 - реєстрація у системі;
 - автентифікація у системі.

У майбутньому додатку повинна бути додана роль консультанта.

1.2 Аналіз аналогів програмного продукту

Початковий аналіз показав, що типовою та необхідною інформацією для розробки веб-сайту кінотеатрів з онлайн замовленням квитків, крім програмної частини, є архітектура створеного веб-сервісу, принцип його роботи, оформлення навігаційного меню тощо. Також важливо дізнатися інформацію про каталоги фільмів і послуг, розташування, експозицію тощо, а також новини, пов'язані з продукцією кінотеатру, присвячену певній тематиці, наявність певних онлайн-сервісів (наприклад, онлайн-оплата та замовлення доставки товарів та послуг), посилення на контактну інформацію, інформацію про кінотеатр та методи його роботи. Крім усього перерахованого вище, в веб-сервісах відображаються статті та новини, пов'язані з проданими квитками та послугами.

Давайте детальніше розглянемо кілька великих веб-сервісів.

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						10

Vkino – найпопулярніший сервіс продажу квитків у кіно. Він сучасний, підтримує всі популярні платформи і багато кінотеатрів, (рис. 1.1).

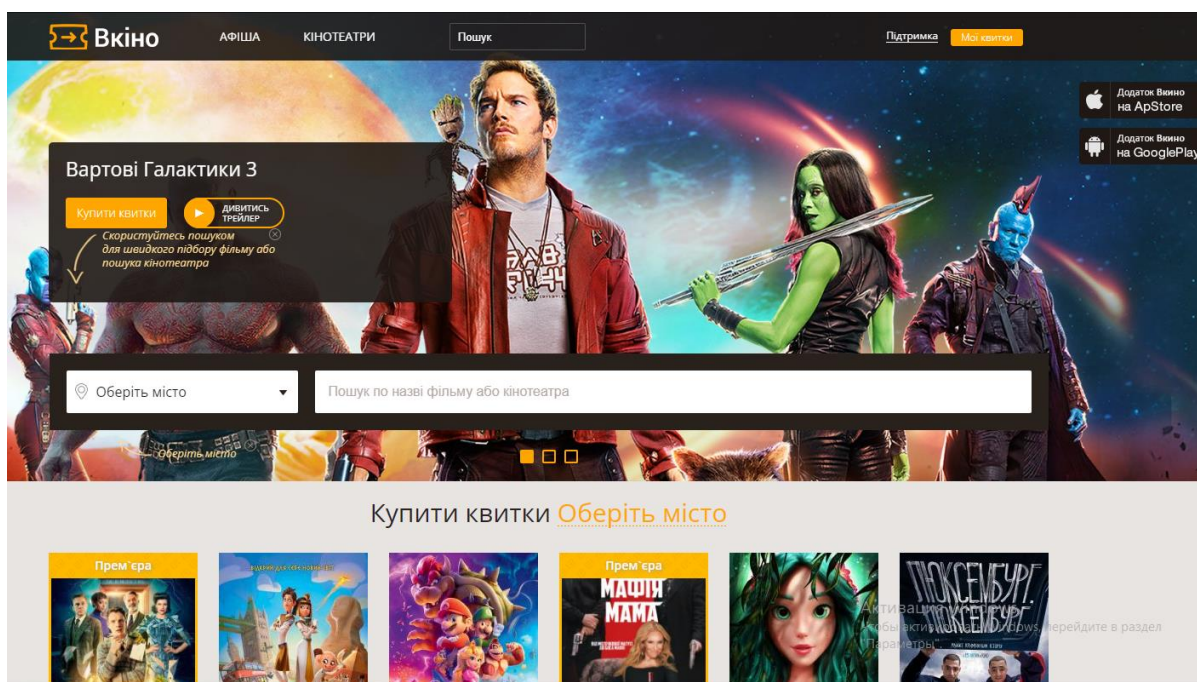


Рисунок 1.1 – Веб-сайт «Vkino»

До основних переваг слід віднести:

- наявність веб-інтерфейсу;
- програми для iOS та Android;
- підтримка кількох театрів по всій країні;
- офіс користувача;
- фільтр для кінотеатрів за містом;
- пошук фільмів;
- дивіться трейлер;
- онлайн-оплата;
- повернення.

Сервіс міг би бути найкращим, якби не недоліки в мобільному додатку. Мобільні телефони сьогодні дуже популярні, і більшість користувачів віддають перевагу мобільним додаткам, а не веб-сайтам, оскільки мобільні телефони завжди з ними. На жаль, мобільний додаток Vkino дуже застарів, має певну

кількість багів, не адаптується до останніх моделей смартфонів і регулярно дає збої. Їх 2 зірки з 5 і багато негативних відгуків лише підтверджують висновки.

Сервіс Kinoland містить мобільні додатки для iOS та Android і використовується локальними мережами кінотеатрів, (рис. 1.2).

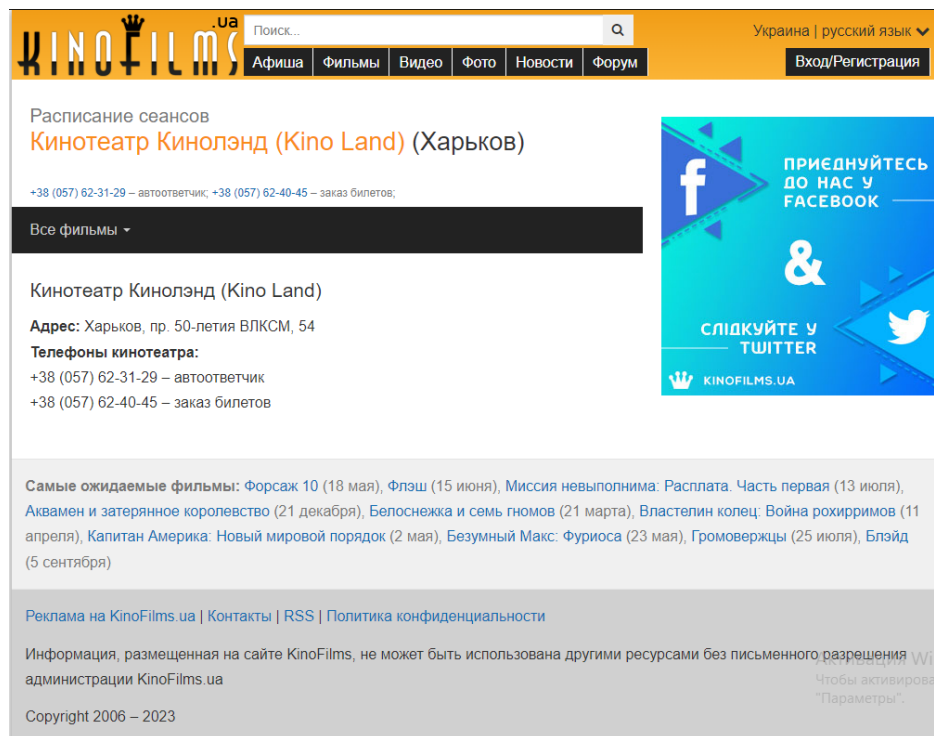


Рисунок 1.2 – Веб-сайт «Kinoland»

Основними перевагами є:

- програма сумісна з найпопулярнішими мобільними операційними системами – iOS та Android;
- покупка квитків здійснюється безпосередньо в додатку;
- зручний інтерфейс;
- офіс користувача, який може генерувати штрих-код, що дозволяє вам ходити в кінотеатр абсолютно самостійно;
- система бонусів.

До недоліків системи можна віднести інтерфейс, який не адаптується до нових екранів смартфонів, і малу кількість кінозалів (5 одиниць).

Сервіс Online Tickets System пропонує оренду програмного забезпечення та роботу через, а також пряму покупку програмного забезпечення з можливістю його встановлення на власний сервер, (рис 1.3).

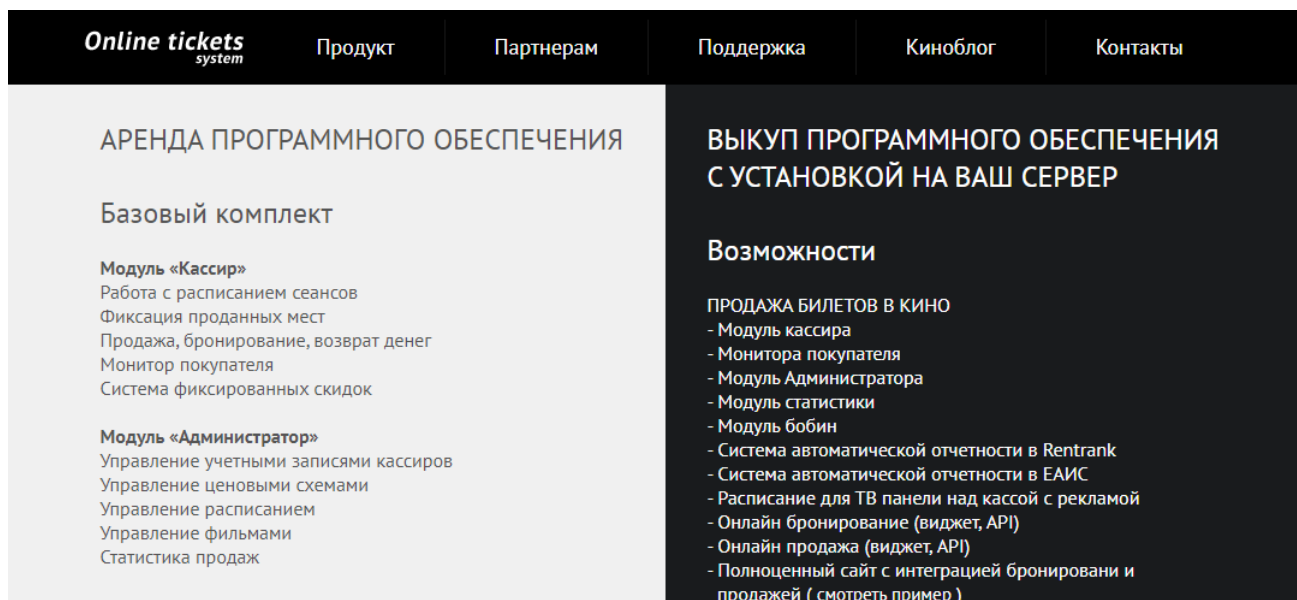


Рисунок 1.3 – Веб-сайт «Online tickets system»

Сервіс надає наступні основні функції:

- Модуль «Каса»:
 - керувати розкладом зустрічей;
 - забезпечення проданих місць;
 - моніторинг покупців;
 - налагоджена система знижок.
- Модуль «Адміністратор»:
 - служба охорони каси;
 - управління ціновими схемами;
 - управління розкладом;
 - управління фільмами;
 - статистика продажів.

Недоліками системи є відсутність графічного інтерфейсу (оплачується окремо), неможливість клієнта придбати квитки онлайн, відсутність дисконтної програми в базовому пакеті, відсутність мобільного додатку.

Його основні переваги:

- програми працюють з найпопулярнішими мобільними операційними системами iOS та Android;

- зручний інтерфейс;
- офіс користувача, який може генерувати штрих-код, що дозволяє вам ходити в кінотеатр абсолютно самостійно;
- система бонусів.

До недоліків системи можна віднести інтерфейс, який не адаптується до нових екранів смартфонів, і малу кількість кінозалів (5 одиниць).

Kino-teatr.ua – найпопулярніший сервіс продажу квитків у кіно, інтерфейс якого представлено нижче, (див. рис.1.4).

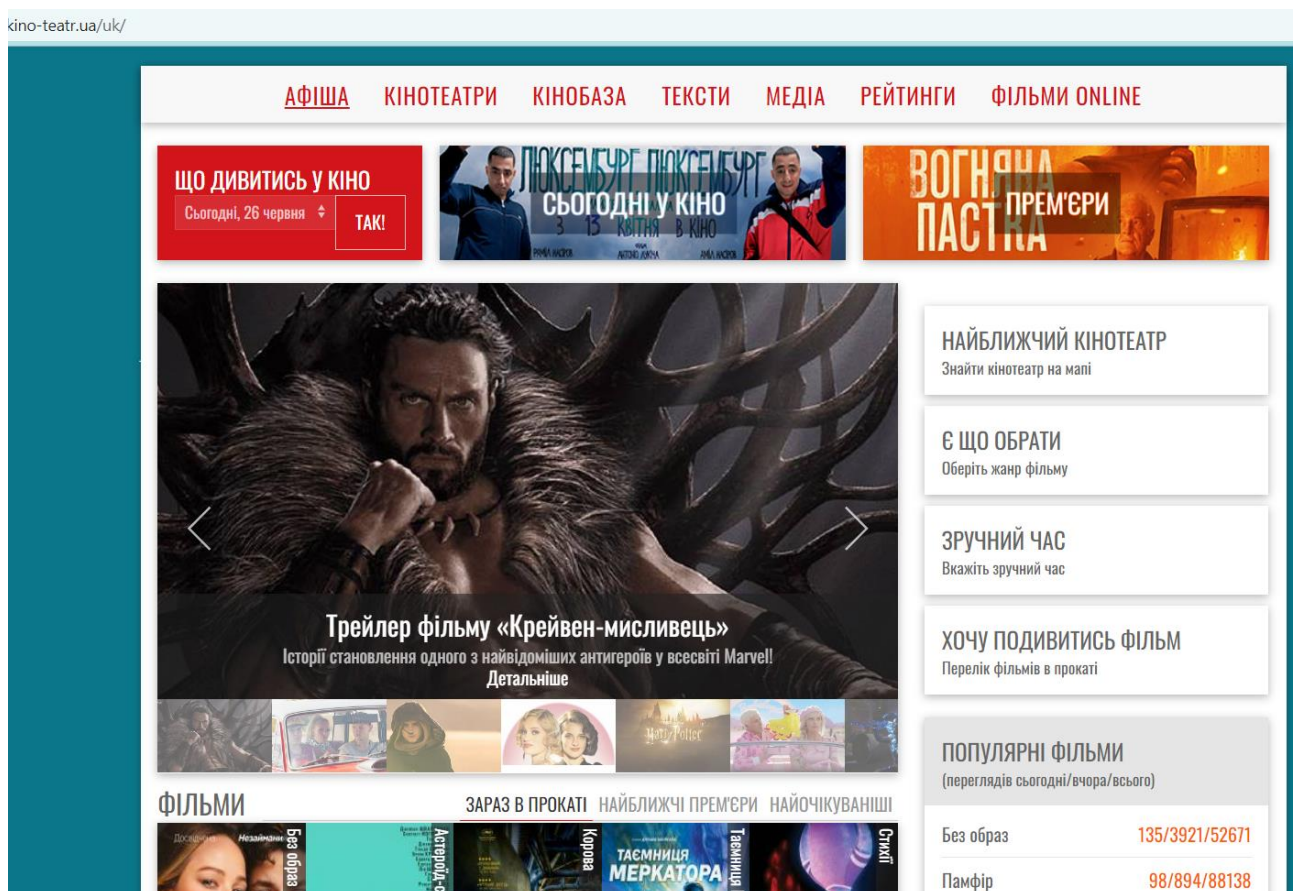


Рисунок 1.4 – Веб-сайт «kino-teatr.ua»

Його основні переваги:

- бронювання квитків у будь-який кінотеатр в Україні.
- коментування фільмів та ставлення їм рейтингу.
- отримання новин і трендів у світі кіно.
- інформація про акторів у фільмах.
- рейтинг фільмів, базований на оцінках користувачів.

Недоліки:

					ІПЗ.КР.Б – 121 – 23 – ПЗ	14

- російська мова за замовчуванням.
- занадто багато реклами.

Таким чином визначено переваги та недоліки сайту kino-teatr.ua для бронювання квитків у кінотеатрах в Україні.

Ознайомившись з інформацією про найпопулярніші платформи автоматичної купівлі квитків у кіно, можна виділити деякі плюси та мінуси. Деякі платформи мають лише один WEB-інтерфейс, чого в сучасних умовах явно недостатньо. Крім того, не кожен має обліковий запис користувача, який дозволяє клієнтам самостійно піти на зустріч, не звертаючись до касира. Та сама платформа з мобільним додатком і обліковим записом або морально і технічно застаріла, або була розроблена та експлуатується виключно місцевою мережею кінотеатрів.

1.3 Вибір архітектури веб-додатку

При розробці веб-додатків, архітектура є дуже важливим аспектом. Архітектура допомагає організувати код і забезпечити його розширюваність та ефективне управління. Для реалізації додатку було обрано клієнт-серверну архітектуру. У цьому розділі обґрунтовано вибір цієї архітектури та визначені її переваги.

Клієнт-серверна [3] архітектура є моделлю у веб-розробці широкого використання, яка передбачає розділення системи на дві основні складові:

- клієнтську частину – відповідає за взаємодію з користувачем;
- серверну частину – обробляє запити клієнта та забезпечує доступ до даних і бізнес-логіки.

Основні причини вибору клієнт-серверної архітектури для даного проекту включають:

- 1) Розділення відповідальностей. Використання клієнт-серверної архітектури дозволяє чітко розділити відповідальності між клієнтом і сервером. Клієнтська частина відповідає за взаємодію з користувачем, відображення

інтерфейсу, а також обробку користувацьких дій. Серверна частина, зі свого боку, забезпечує обробку запитів, доступ до бази даних, бізнес-логіку та інші серверні операції. Це розділення спрощує управління проектом, полегшує розробку та підтримку коду.

2) Масштабованість. Клієнт-серверна архітектура дозволяє “гнучко” масштабувати систему за потребою. Збільшення навантаження з боку користувачів може бути вирішено шляхом масштабування серверної частини, додавання додаткових серверів або використання хмарних рішень. Це дозволяє підтримувати високу продуктивність та доступність сервісу навіть при зростанні кількості користувачів.

3) Розширення та модульність. Клієнт-серверна архітектура дозволяє розділити функціональність на незалежні модулі. Це полегшує розробку та підтримку коду, оскільки різні частини системи можуть бути незалежно розроблені, тестовані та модифіковані.

Схема шаблону клієнт-сервер зображено нижче (див. рис.1.5).



Рисунок 1.5 – Схема шаблону клієнт-сервер

Наступним етапом є обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.

1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Після проведення аналізу програмних продуктів та з'ясування потрібних вимог додатку та вибору архітектури наступним кроком є обґрунтування вибору інструментів та встановлення вимог до апаратного забезпечення.

Для розробки серверної частини було обрано фреймворк Express [2] для Node.js [6]. Він використовується для створення серверної частини вашого веб-застосунку і обробки HTTP-запитів.

Express – є доволі простим та інтуїтивно зрозумілим для розробки веб-застосунків. Express дозволяє швидко створювати веб-додатки за допомогою простого і інтуїтивно зрозумілого API. Він забезпечує основні функціональні можливості для маршрутизації, обробки запитів і відповідей, роботи зі статичними файлами тощо.

Для програмної реалізації застосунку планується використання шаблону middleware.

Middleware [10] – це певний стиль функцій, які ви налаштовуєте для використання вашим додатком. Вони можуть виконувати будь-який код, який вам подобається, але зазвичай вони направлені на обробку вхідних запитів, надсилання відповідей і обробку помилок. Вони є будівельними блоками кожного додатка Express.

Для розробки клієнтської частини планується використання бібліотеки React [5]. React – це бібліотека, яка використовується для розробки користувацьких інтерфейсів, тобто frontend частини. В React використовуються компоненти, які можна розділити на два види: класові функціональні компоненти та класові компоненти. Кожен компонент має свій стан та структуру, а також вони можуть обмінюватися даними з іншими компонентами, що полегшує процес розробки. React надає підтримку використання JSX (JavaScript XML), що дозволяє комбінувати JavaScript з HTML-подібним синтаксисом. Це дозволяє зручно створювати компоненти та використовувати їх в додатку. JSX

спрощує розробку інтерфейсу, оскільки забезпечує зрозумілу структуру та відображення компонентів відповідно до їх функціональності. Це дає можливість зручно змішувати логіку JavaScript [22] з розміткою HTML, що полегшує розробку та підтримку веб-інтерфейсу. В React існує велика кількість бібліотек, які полегшують та пришвидшують процес розробки. Приклади бібліотек, які плануються до використання в програмному продукту:

1) react-router [2] – бібліотека яка використовується для маршрутизації.

2) axios [13] – бібліотека дозволяє легко взаємодіяти з сервером. Використовується для здійснення HTTP-запитів з браузера або середовища Node.js. Вона надає простий та зручний інтерфейс для взаємодії з веб-серверами та отримання даних з них.

Для розробки веб застосунку для бронювання квитків у кінотеатрі було обрано базу даних MongoDB [14]. MongoDB – це документно-орієнтована система управління базами даних (СУБД) з відкритим вихідним кодом, яка не вимагає опису схеми таблиці. MongoDB має хорошу продуктивність, зручні запити для взаємодії з базою даних, хороший обсяг зберігання даних, запити для взаємодії з базою підходять для онлайн-ігор або невеликих додатків.

До переваг MongoDB слід віднести:

– Гнучкість схеми даних. MongoDB дозволяє гнучко визначати схему даних без строгих вимог до структури. Це дозволяє зручно працювати зі змінюваними та нерегулярними даними.

– Масштабованість. MongoDB розроблена для горизонтального масштабування, що означає, що ви можете легко розширювати свою базу даних, додавши нові сервери.

– Швидкодія. MongoDB пропонує швидку роботу з даними, особливо для операцій читання, завдяки своїй архітектурі та підтримці індексів.

За допомогою Redux Toolkit [11] планується створення завантажувачів (loaders) для керування станом завантаження даних у веб-додатку. Завантажувачі допомагають спростити та уніфікувати логіку завантаження

даних з сервера, забезпечуючи кращу керованість та зручність у взаємодії з асинхронними операціями.

Переваги:

- Спрощена конфігурація. Redux Toolkit надає стандартні шаблони та утиліти для створення редукторів, дій та магазину Redux. Це дозволяє швидко налаштувати основні елементи Redux і уникнути повторюваного шаблону коду.

- Автоматична генерація дій. Redux Toolkit автоматично генерує дії для вас на основі визначених редукторів. Це дозволяє уникнути ручного написання багато коду і скорочує час розробки.

- Використання createSlice. Redux Toolkit надає createSlice, який дозволяє вам описувати редуктори, дії та початковий стан в одному місці. Це спрощує створення та керування станом в Redux.

Для відправки даних сервер планується використання Axios. Axios – це клієнтська бібліотека HTTP, заснована на промесах. Це спрощує надсилання асинхронних HTTP-запитів до кінцевих точок REST і допомагає виконувати операції CRUD.

AntD – це фреймворк, розроблений на базі React, який надає готові компоненти та стилізацію для швидкої і зручної розробки користувацького інтерфейсу. Ось переваги використання Ant Design [9] в контексті розробки веб-застосунків:

- Готові компоненти. Ant Design надає широкий набір готових компонентів, які можна використовувати для швидкої побудови користувацького інтерфейсу. Це включає такі елементи як кнопки, форми, таблиці, модальні вікна, навігаційні панелі та багато іншого. Використання готових компонентів дозволяє ефективно використовувати час розробки та покращує інтерфейс веб-додатку.

- Інтуїтивний та простий API. Ant Design має добре задокументований та інтуїтивно зрозумілий API для використання компонентів. Це дозволяє розробникам швидко засвоїти веб-фреймворк і працювати з ним ефективно.

Компоненти Ant Design легко налаштовуються та адаптуються до потреб проекту.

– Оптимізована продуктивність. Ant Design працює на базі React та використовує оптимізований підхід до рендерингу компонентів. Він використовує віртуальний DOM (Virtual DOM) [15] та розумні алгоритми оновлення, що сприяє покращенню продуктивності веб-додатку та ефективному використанню ресурсів.

Висновки до першого розділу

При обґрунтуванні актуальності теми роботи, було визначено, що розробка веб-додатку для онлайн сервісу бронювання квитків у кінотеатрах має на меті створення зручної та функціональної платформи для користувачів. Цей додаток спрощує процес бронювання квитків, надаючи зручний інтерфейс для вибору фільмів та сеансів.

Проведено аналіз аналогів програмного продукту на прикладі кінотеатрів, таких як: Kino-teatr.ua, Online Tickets, System, V kino, Kinoland. Це дає можливість для розробки конкурентоспроможного продукту з більш широким функціоналом та більшим вибором фільмів та кінотеатрів.

Було розглянуто вибір клієнт-серверної архітектури для веб-додатку і наведено основні переваги цієї архітектури. Клієнт-серверна архітектура дозволяє чітко розділити відповідальності між клієнтом і сервером, спрощує управління проектом, полегшує розробку та підтримку коду, має гнучку масштабованість та підтримує розширення та модульність.

Також було обґрунтовано вибір інструментальних засобів та встановлено вимоги до апаратного забезпечення.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ

2.1 Визначення варіантів використання та структури системи

Для забезпечення безперервного виконання функцій системи та відповідності всім стандартам та специфікаціям, нижче наведено високорівневі вимоги до онлайн веб-орієнтованої системи для бронювання квитків у кінотеатрах. Описано умови, які користувач повинен виконувати для роботи з системою, а також основні переваги цієї системи. Нижче приведено можливості системи, обмеження та показники якості.

Бізнес-вимоги:

1. Основною метою проекту є створення системи, яка дає можливість відображати дані про певний кінотеатр, його, сеанси, місця, квитки та каталог всіх фільмів які є у системі. У програмі передбачено використання механізму пошуку за різними параметром, таким як фільми

2. Для кожного окремого замовлення надається можливість розрахунку нормативного часу виконання, при цьому замовлення виконується безпосередньо після отримання оплати.

3. Проект реалізований у формі веб-додатку, який має зручний інтерфейс та містить всю необхідну інформацію, пов'язану з даною тематикою.

Вимоги користувачів:

- аутентифікація та авторизація користувачів;
- перегляд каталогу фільмів;
- перегляд всіх сеансів які є у системі;
- можливість забронювати квитки у кінотеатрі;
- переглянути інформацію бронювання квитка;
- мати можливість запросити дозвіл у адміністратора на додавання нового кінотеатру;
- додати сеанс у кінотеатр (якщо адміністратор дав дозвіл на додавання кінотеатру у систему).

Внутрішній користувач – Адміністратор

- CRUD фільмів.
- Визначення статусу кінотеатру(адміністратор визначає чи буде він додавати кінотеатр у систему.

Функціональні вимоги:

- адміністративна частина має форму для додавання фільму;
- адміністративна частина має інформацію про всі кінотеатри, які додані у систему;
- клієнтська частина має форму для додавання сеансу (якщо адміністратор підтвердив статус кінотеатру);
- клієнтська частина має інформацію про всі кінотеатри, які були додані у систему користувачем;
- підключення сторонніх API для реалізації оплати.

Нефункціональні вимоги:

- інтерфейси клієнтської та адміністративної частини додатку мають бути зручними для використання для клієнта та для адміністратора відповідно;
- додаток має працювати в сучасних браузерях;
- легкість до масштабування.

Системні вимоги:

Система має відповідати вимогам, що наведені нижче, і працювати на комп'ютерах або смартфонах з мінімальною конфігурацією.

Вимоги до середовища виконання:

- підтримка браузера Microsoft Edge версії 12+;
- підтримка браузера Safari версії 9+;
- підтримка браузера Google Chrome версії 45+;
- підтримка браузера Opera версії 30+;
- підтримка браузера Mozilla Firefox версії 38+.

За допомогою аналізу вимог користувачів та функціональних вимог були виявлені ключові сутності, що сприятимуть реалізації програмного комплексу

системи. Давайте розглянемо, як звичайний працівник та адміністратор можуть взаємодіяти з адміністративною частиною веб-додатку (див. рис. 2.1).

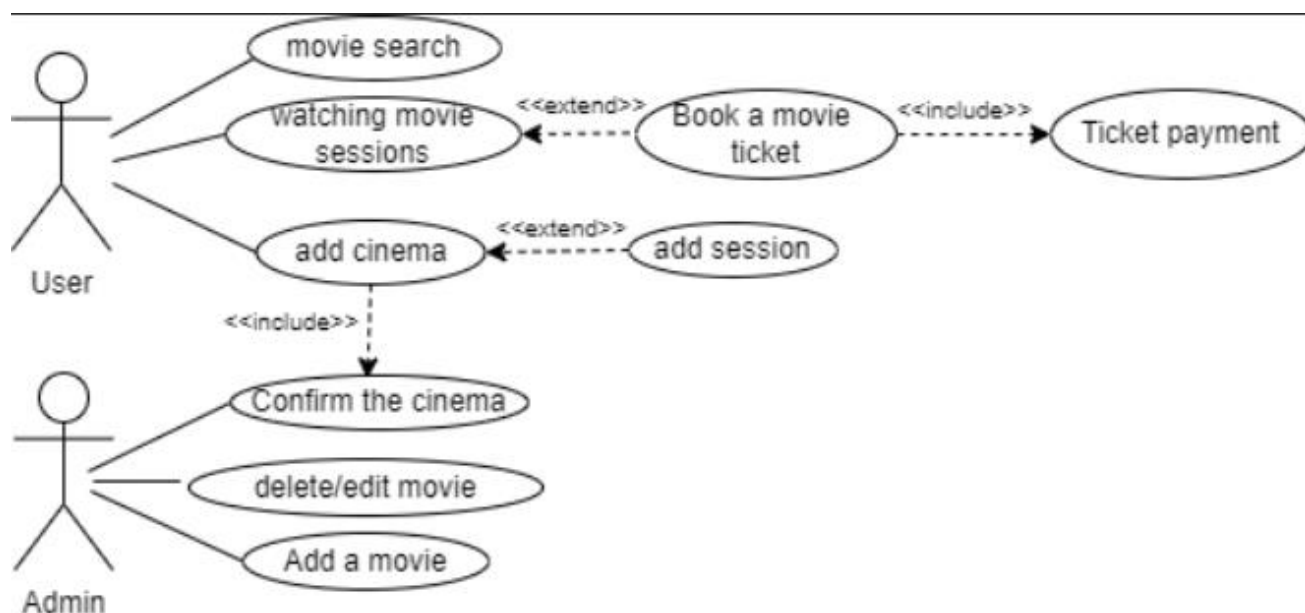


Рисунок 2.1 – Діаграма варіантів використання адміністративної частини

Об'єктно-орієнтована модель системи. Після аналізу вимог до системи та створення діаграми використання, наступним кроком є побудова діаграми класів, яка буде розроблена за допомогою інструменту StarUML [12]. Діаграма класів є важливим етапом, вона допомагає візуалізувати структуру системи, її основні сутності та зв'язки між ними. Побудова діаграми класів передбачає ідентифікацію та визначення класів системи, їх атрибутів, методів та зв'язків. Кожен клас відображає певну сутність системи або компонент, який має свої властивості та поведінку. Основними елементами діаграми класів є класи, взаємозв'язки між класами (такі як асоціації, композиції, агрегації) та методи, які визначають операції, що можуть бути виконані над об'єктами цих класів.

Діаграма класів (англ. Class diagram) [19] – статичне представлення структури моделі в UML. Відображає статичні (декларативні) елементи, такі як: класи, типи даних, їх зміст та відношення. Діаграма класів може містити позначення для пакетів та може містити позначення для вкладених пакетів. На

(рис.2.2) створюється внутрішня структура системи, описується спадкування й взаємне положення класів один відносно другого.

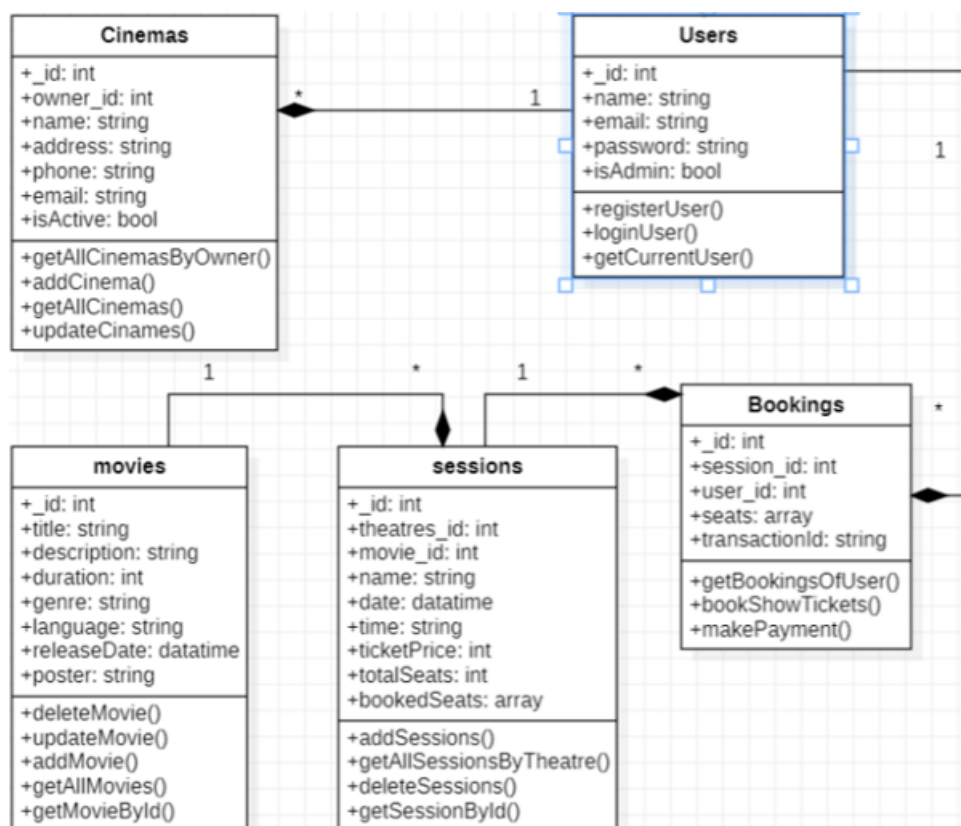


Рисунок 2.2 – Діаграма класів

На даному рисунку є наступні класи:

User – клас містить методи для роботи з користувачем.

Методи:

- registerUser() – реєстрація користувача;
- getCurrentUser() – встановлення ролі користувача;
- loginUser() - авторизація користувача.

Cinemas – клас містить методи для кінотеатрів, а саме:

- getAllCinemasByOwner() – список кінотеатрів власника
- addCinema() – додати кінотеатр;
- getAllCinemas() – отримати всі кінотеатри;
- updateCinemas() – оновити всі кінотеатри.

Movies:

- getAllMovies() – клас містить методи для фільмів;
- DeleteMovie() – видалити фільм;
- UpdateMovie() – оновити фільм;
- addMovie() – додати фільм;
- getAllMovies() – отримати всі фільми для відображення.

Sessions – клас містить методи для сеансів:

- addSessions() – отримати сеанси;
- getAllSessionsByTheatre() – отримати сеанси;
- deleteSessions() – видалити сеанси;
- getSessionsById() – отримати конкретний сеанс.

Booking() – клас містить методи для бронювання:

- getBookingsOfUser() – сеанси;
- bookShowTickets() – відобразити білети в особистому кабінеті;
- makePayment() – виконати оплату.

2.2 Розробка бази даних системи

У даному пункті було проведено детальний опис структури бази даних, яка була розроблена у веб-застосунку "Онлайн сервіс бронювання квитків у кінотеатрах". Для реалізації було обрано базу даних MongoDB.

В першу чергу, MongoDB була обрана через її гнучкість в роботі з даними. База даних розроблена з використанням моделей, які включають такі сутності, як користувачі (users), кінотеатри (cinemas), сеанси (sessions), фільми (movies) та бронювання (bookings) (див. табл. 2.1-2.5). Кожна модель представлена відповідною схемою, де кожному полю встановлено відповідний тип даних та налаштування.

Наприклад, модель користувачів (users) містить поля для імені, електронної пошти, пароля та ролі користувача. Використовуються типи даних,

такі як string, boolean, та date для збереження необхідної інформації про користувачів.

Крім того, MongoDB дозволяє використовувати зв'язки між колекціями за допомогою зовнішніх ключів (foreign key). Наприклад, у моделі кінотеатрів (cinemas) присутнє поле owner, яке є посиланням на користувача, який володіє кінотеатром. Такі зв'язки дозволяють ефективно організувати зв'язані дані та здійснювати зв'язані запити.

Також, MongoDB забезпечує підтримку для індексування даних, що дозволяє прискорити пошук та запити до бази даних. Наприклад, для швидкого пошуку фільмів за жанром (genre) можна створити відповідний індекс.

За допомогою MongoDB є можливість масштабування бази даних, що є важливим аспектом для веб-застосунків з високою навантаженістю. За необхідності, можна легко розширити кількість серверів бази даних та розподілити навантаження між ними. Отже, обрання MongoDB для розробки бази даних у веб-застосунку "Онлайн-сервіс замовлення квитків та послуг "Кінотеатр" було обґрунтовано гнучкістю роботи з даними, підтримкою зв'язків між сутностями, можливістю індексування та масштабованістю. Ці функціональні можливості допомагають забезпечити ефективне зберігання та обробку даних для потреб розробленого веб-застосунку.

Таблиця "users" містить інформацію про користувачів, табл.2.1.

Таблиця 2.1

Структура таблиці "users"

Назва	Тип даних	РК	FK	Опис поля
name	string	—	—	ім'я користувача
email	string	—	—	електронна пошта
password	string	—	—	пароль
isAdmin	boolean	—	—	права адміністратора

Таблиця "cinemas" містить інформацію про кінотеатри, табл.2.2.

Таблиця 2.2

Структура таблиці “cinemas”

Назва	Тип даних	PK	FK	Опис поля
name	string	—	—	назва кінотеатру
address	string	—	—	адреса кінотеатру
phone	string	—	—	телефон власника кінотеатру
email	string	—	—	пошта власника кінотеатру
owner	objectId (users)	—	+	власник кінотеатру
isActive	boolean	—	—	статус кінотеатру

Таблиця “sessions” містить інформацію про всі сеанси в кінотеатрі, табл.2.3.

Таблиця 2.3

Структура таблиці “sessions”

Назва	Тип даних	PK	FK	Опис поля
name	string	—	—	назва сеансу
date	date	—	—	дата сеансу
time	string	—	—	час сеансу
movie	objectId(movies)	—	+	фільм сеансу
ticketPrice	number	—	—	ціна квитка
totalSeats	number	—	—	загальна кількість місць
bookedSeats	array	—	—	заброньовані місця
cinema._id	objectId(cinemas)	—	+	кінотеатр

Таблиця “movies” містить інформацію про всі фільми, зокрема з їх описом, тривалістю фільму, жанром, мовою, датою випуску, постером, що дає змогу отримати інформацію для прийняття рішення, табл.2.4.

Таблиця 2.4

Структура таблиці “movies”

Назва	Тип даних	PK	FK	Опис поля
title	string	—	—	назва фільму
description	string	—	—	опис фільму
duration	number	—	—	тривалість фільму
genre	string	—	—	жанр фільму
language	string	—	—	мова фільму
releaseDate	date	—	—	дата випуску фільму
poster	string	—	—	постер фільму

Таблиця “ bookings ” – містить інформацію про бронювання, табл.2.5.

Таблиця 2.5

Структура таблиці “bookings”

Назва	Тип даних	PK	FK	Опис поля
session_id	objectId(shows)	—	+	id сеансу
user_id	objectId(users)	—	+	id користувача
seats	array	—	—	місця
transactionId	string	—	—	id транзакції

Наступним етапом є проектування додатку, розробка та реалізація алгоритмів роботи системи.

2.3 Проектування алгоритмів роботи системи

Веб-додаток є простим у використанні, має всі основні функції, всі дії користувачів описані вище. Алгоритм роботи веб-сайту системи представлено на рис 2.3 – рис.2.5.

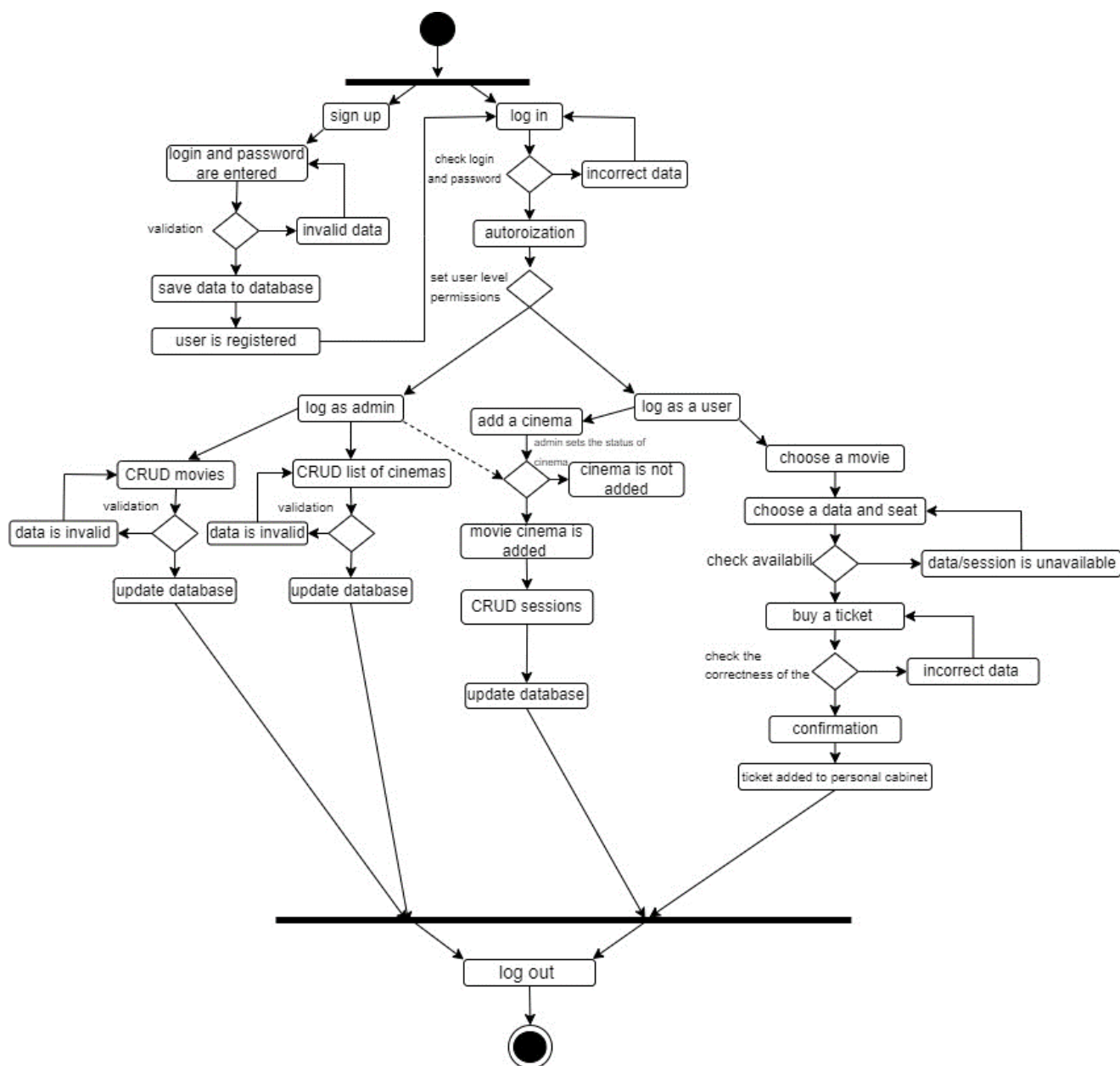


Рисунок 2.3 – Діаграма активності

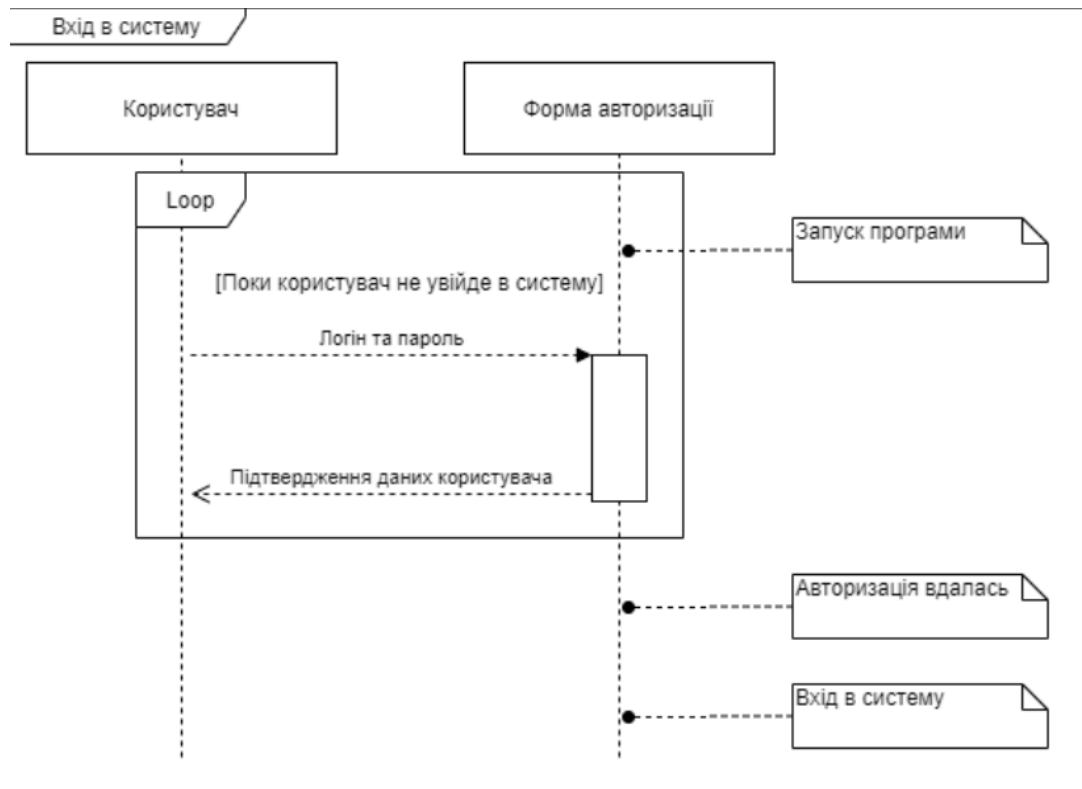


Рисунок 2.4 – Діаграма послідовності входу в додаток

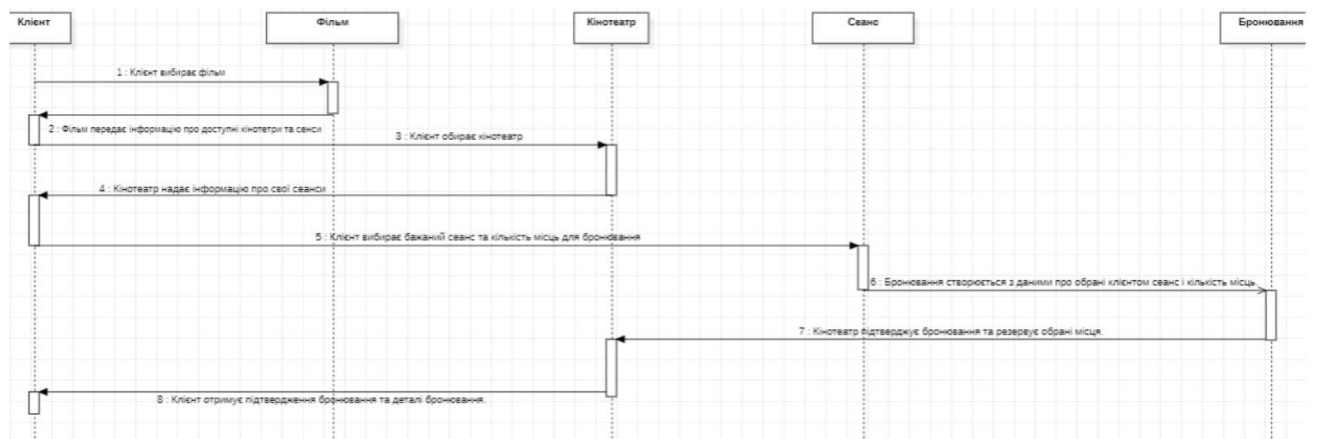


Рисунок. 2.5 – Діаграма послідовності бронювання квитка

Переходимо до реалізації запропонованих рішень.

2.4 Реалізація веб-додатку онлайн сервіса бронювання квитків у кінотеатрах

Компонент Register.js використовується для сторінки реєстрації користувача в онлайн-сервісі "Кінотеатр". Він взаємодіє з бібліотеками Ant Design, реакт-рутера та Redux [8]. Після успішної реєстрації користувача дані відправляються на сервер. Є також перевірка, чи вже є збережений токен авторизації в localStorage. Компонент містить реєстраційну форму з правилами валідації для кожного поля, кнопку "Реєструватись" та повідомлення про результат реєстрації. Компонент експортується для використання в інших частинах додатка.

Register.js

```
function Register() {
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const onFinish = async (values) => {
    try {
      dispatch(ShowLoading());
      const response = await RegisterUser(values);
      dispatch(HideLoading());

      if (response.success) {
        message.success(response.message);
      } else message.error(response.message);
    } catch (error) {
      dispatch(HideLoading());
      message.error(error.message);
    }
  };

  useEffect(() => {
    if (localStorage.getItem("token")) {
      navigate("/");
    }
  }, []);

  return (
    <>
    <div className="container">
      <div className="registerForm">
        <h1 className="text-register">Регістрація</h1>
        <Form onFinish={onFinish}>
```

```

    <Form.Item
      label="Name"
      name="name"
      rules={[{ required: true, message: "Введіть своє Ім'я" }]}
    >
      <Input className="registerForm__input"></Input>
    </Form.Item>
    <Form.Item
      label="Email"
      name="email"
      rules={[{ required: true, message: "Введіть свій email" }]}
    >
      <Input className="registerForm__input"></Input>
    </Form.Item>
    <Form.Item
      label="Password"
      name="password"
      rules={[{ required: true, message: "Введіть свій пароль" }]}
    >
      <Input className="registerForm__input"></Input>
    </Form.Item>
    <div className="btn_wrapper">
      <Button type="primary" title="Реєструватись" htmlType="submit" />
      <Link to="/login">Ви вже маєте аккаунт? Login</Link>
    </div>
  </Form>
</div>
</div>
</>
);
}

export default Register;

```

Код представлений нижче є модулем, який експортує функцію з middleware для перевірки автентифікаційного токена користувача. Основна мета цього middleware – перевірити та підтвердити достовірність токена перед продовженням обробки запиту.

```

const jwt = require("jsonwebtoken");

module.exports = function (req, res, next) {
  try {
    const token = req.headers.authorization.split(" ")[1];
    const decoded = jwt.verify(token, process.env.jwt_secret);
    req.body.userId = decoded.userId;
    next();
  }
}

```

```

} catch (error) {
  res.status(401).send({ success: false, message: "Недійсний токен" });
}

```

Компонент Login відповідає за сторінку входу в систему. Він містить форму з полями для введення email та пароля, кнопку для відправки форми та посилання на сторінку реєстрації. Коли користувач натискає кнопку "Login", дані з форми відправляються на сервер для авторизації. Після успішної авторизації користувача перенаправляють на головну сторінку, а у випадку помилки відображається відповідне повідомлення.

```

function Login() {
  const dispatch = useDispatch();
  const navigate = useNavigate();
  const onFinish = async (values) => {
    try {
      dispatch(ShowLoading());
      const response = await LoginUser(values);
      dispatch(HideLoading());
      if (response.success) {
        message.success(response.message);
        localStorage.setItem("token", response.data);
        window.location.href = "/";
      } else {
        message.error(response.message);
      }
    } catch (error) {
      dispatch(HideLoading());
      message.error(error.message);
    }
  };

  useEffect(() => {
    if (localStorage.getItem("token")) {
      navigate("/");
    }
  }, []);

  return (
    <>
      <div className="container">
        <div className="registerForm">
          <h1 className="text-register">Логін</h1>
          <Form onFinish={onFinish}>
            <Form.Item
              label="Email"
              name="email"

```

```

        rules=[[{ required: true, message: "Введіть свій email" }]]
      >
      <Input></Input>
    </Form.Item>
    <Form.Item
      label="Password"
      name="password"
      rules=[[{ required: true, message: "Введіть свій пароль" }]]
    >
      <Input></Input>
    </Form.Item>
    <div className="btn_wrapper">
      <Button type="primary" title="Login" htmlType="submit" />
      <Link to="/register">Ви вже маєте аккаунт? Register</Link>
    </div>
  </Form>
</div>
</div>
</>
);
}

export default Login;

```

Код описаний нище зроблений за допомогою Redux Toolkit для створення центрального зберігання даних (стору) у додатку. Він має два розділи (reducers): "users" та "loaders".

В розділі "users" (usersReducer.js) визначений редуктор, який контролює стан користувачів. Він починається зі значення null для поля "user", а потім може бути змінений дією "SetUser", коли значення користувача передається через action.payload.

У файлі store.js створюється стор за допомогою configureStore з двома редукторами: loadersReducer для керування станом завантаження і usersReducer для керування станом користувачів.

Розділ "loaders" (loadersReducer.js) має редуктор, який відповідає за стан завантаження. Він починається зі значення false для поля "loading". Дії "ShowLoading" та "HideLoading" встановлюють значення "loading" відповідно на true і false.

Отже, цей код дозволяє налаштувати стор з двома розділами: "users" для управління користувачами та "loaders" для управління станом завантаження. Використання Redux Toolkit спрощує процес створення дій та редукторів.

loadersReducer.js

```
import {createSlice} from '@reduxjs/toolkit';

const loadersReducer = createSlice({
  name: 'loaders',
  initialState: {
    loading : false,
  },
  reducers: {
    ShowLoading : (state) => {
      state.loading = true;
    },
    HideLoading : (state) => {
      state.loading = false;
    }
  }
});

export const {ShowLoading, HideLoading} = loadersReducer.actions;
export default loadersReducer.reducer;
```

store.js

```
import { configureStore } from "@reduxjs/toolkit";
import loadersReducer from "./loadersReducer";
import usersReducer from "./usersReducer";

const store = configureStore({
  reducer: {
    loaders: loadersReducer,
    users: usersReducer,
  },
});

export default store;
```

usersReducer.js

```
import {createSlice} from "@reduxjs/toolkit"

const usersReducer = createSlice({
  name: "users",
```

```

    initialState: {
      user: null,
    },
    reducers: {
      SetUser: (state, action) => {
        state.user = action.payload
      }
    }
  });

export const { SetUser } = usersReducer.actions;
export default usersReducer.reducer;

```

Висновки до другого розділу

На даному етапі розробки дипломної роботи було проведено аналіз та визначено вимоги до розроблюваної системи. Крім того, були встановлені мінімальні системні вимоги, необхідні для належної роботи додатку. З метою більш детального уявлення про функціонал системи, були створені діаграми варіантів використання, які дозволили описати різні сценарії взаємодії з системою.

Для покращення розуміння та визначення взаємодії об'єктів веб-сайту лікарні, була розроблена об'єктно-орієнтована модель. Ця модель надала загальне уявлення про структуру системи та взаємозв'язки між її складовими.

Додатково, був розроблений загальний алгоритм, за яким буде виконуватися робота веб-сайту. Цей алгоритм був відображений у вигляді діаграми діяльності, що уявляє послідовність дій та переходів між ними.

Також була розроблена структура бази даних, яка описує схему зберігання і взаємозв'язки даних, необхідних для функціонування системи.

У заключному пункті детально розглядаються певні аспекти реалізації системи. Описується процес передачі даних між клієнтом та сервером, включаючи протоколи та формати обміну даними. Також згадується про виконання валідації даних та інші суттєві аспекти, що впливають на функціональність та надійність системи.

					ІПЗ.КР.Б – 121 – 23 – ПЗ	
						36

Було проведено аналіз потенційних сценаріїв використання системи та встановлено основні цілі її функціонування. Структура системи була розроблена з урахуванням потреб користувачів та можливостями адміністрування.

Розроблено структуру бази даних, що включає необхідні таблиці та зв'язки для зберігання інформації про фільми, сеанси, квитки, бронювання, користувачів.

Проведено проектування та реалізацію алгоритмів роботи системи, що включають процеси пошуку сеансів, обробку замовлень квитків, резервацію місць та інші важливі функції.

Здійснено реалізацію самого онлайн-сервісу для бронювання квитків, де було враховано всі попередні етапи та було створено зручний та функціональний інтерфейс для користувачів.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ

3.1 Порядок встановлення та налаштування системи

Оскільки веб-додаток має клієнт-серверну архітектуру, є можливість розмістити серверну, клієнтську та адміністративну частини окремо. Мінімальні вимоги для розгортання серверної частини додатку залежать від очікуваного обсягу трафіку, проте рекомендується враховувати їх з додатковим запасом. Тому, для ефективного розгортання серверної частини вашого додатку, рекомендується використовувати сервер із наступними характеристиками:

- двоядерний процесор з тактовою частотою 2 ГГц або вище.
- мінімум 2 ГБ оперативної пам'яті (ОЗУ).
- мінімум 1 ГБ вільного місця на жорсткому диску.

Щоб запустити серверну частину додатку потрібно виконати наступний ряд дій:

1. Встановіть Node.js на сервері.
 2. Перейдіть у папку server та запустіть програму за допомогою команди `npm install`, щоб встановити залежності додатка, які зазначені у файлі `package.json`.
 3. Запустіть сервер, щоб додаток почав прослуховувати вхідні з'єднання.
- Для встановлення та підключення бази даних MongoDB, потрібно виконати такі дії:

1. Встановіть MongoDB та MongoDB Compass [23] на свій комп'ютер. Ви можете завантажити їх з офіційного сайту MongoDB.
2. Запустіть MongoDB сервер на своєму комп'ютері. Це можна зробити, виконавши команду `mongod` в командному рядку. При необхідності ви можете налаштувати параметри сервера, такі як порт або шлях до директорії з даними.
3. Відкрийте MongoDB Compass і виконайте наступні кроки:
4. Натисніть на кнопку "New Connection" або "Create New Connection".

5. Введіть необхідні дані підключення, такі як хост (localhost, якщо MongoDB запущено на вашому комп'ютері), порт (за замовчуванням 27017) та інші параметри, які ви використовували при налаштуванні бази даних MongoDB.

6. Натисніть кнопку "Connect" або "Connect to Host" для підключення до бази даних.

7. Переконайтеся, що ви успішно підключилися до бази даних MongoDB за допомогою MongoDB Compass. У вас буде змога переглядати колекції, документи та виконувати різні операції з базою даних.

8. Потрібно створити файл .env, та написати наступний рядок

```
mongo_url = "mongodb+srv://admin:admin123@cluster5.dflk9y.mongodb.net/booking-system-database"
```

Дані паролю та адміна потрібно написати згідно вашим даним які ви створили за допомогою MongoDB Compass.

3.2 Структура інтерфейсу. Інтерфейс та порядок роботи з веб-додатком

При вході у веб-застосунок, користувач зобов'язаний ввести свої облікові дані (логін та пароль) на сторінці входу. Після введення даних користувач натисне на кнопку "Увійти", щоб авторизуватись в системі. В разі успішної аутентифікації користувача, йому буде надано доступ до всіх функцій та можливостей веб-застосунку. (рис 3.1).

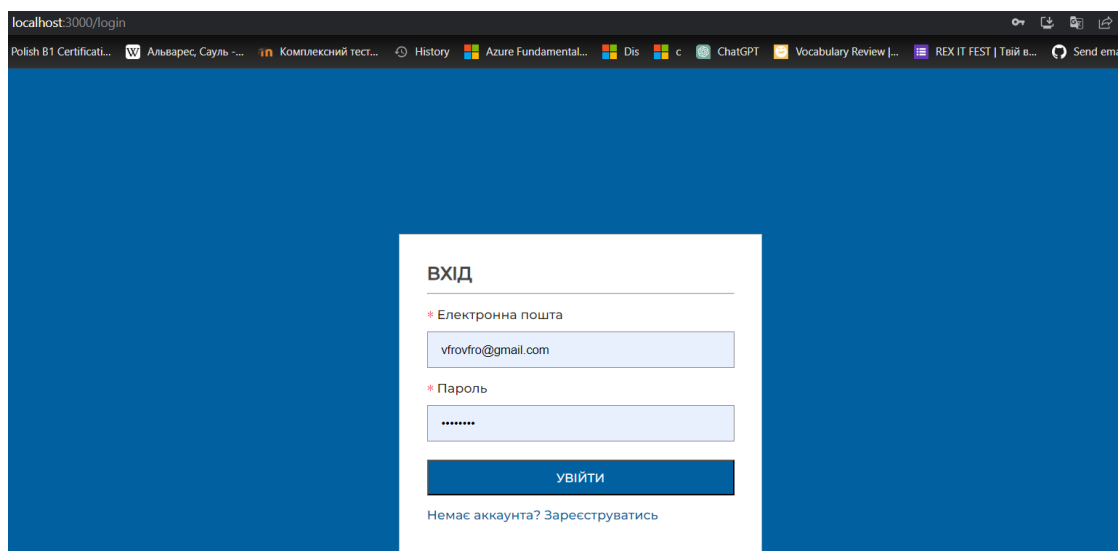


Рисунок 3.1 – Сторінка входу у веб-додаток

Якщо користувач ще не має облікового запису у системі, він буде перенаправлений на сторінку реєстрації. На цій сторінці користувач буде запрошений заповнити реєстраційну форму, де він повинен буде ввести особисті дані (ім'я, прізвище, електронну пошту тощо) та вибрати унікальний логін та пароль. Після заповнення форми користувач натисне на кнопку "Зареєструватись" для створення облікового запису, (рис 3.2).

localhost:3000/register

Polish B1 Certificati... Альварес, Сауль -... Комплексний тест... History Azure Fundamental... Dis c ChatGPT Vocabulary Review J... REX IT FEST | Твій в... Send email v

SHEYMOVIES - РЕЄСТРАЦІЯ

* Ім'я

* Електронна пошта

* Пароль

ЗАРЕЄСТРУВАТИСЬ

[Вже маєте акаунт? Увійти](#)

Рисунок 3.2 – Сторінка реєстрації

Після успішної реєстрації або входу в систему, користувач буде перенаправлений на головну сторінку веб – сайту "Бронювання квитків". Інтерфейс головної сторінки буде містити каталог фільмів, де користувач зможе обрати бажаний фільм для бронювання квитка у кінотеатрі, (рис 3.3).

При вході на сторінку профілю, користувач має можливість дати запит адміністратору на додавання кінотеатру у систему.

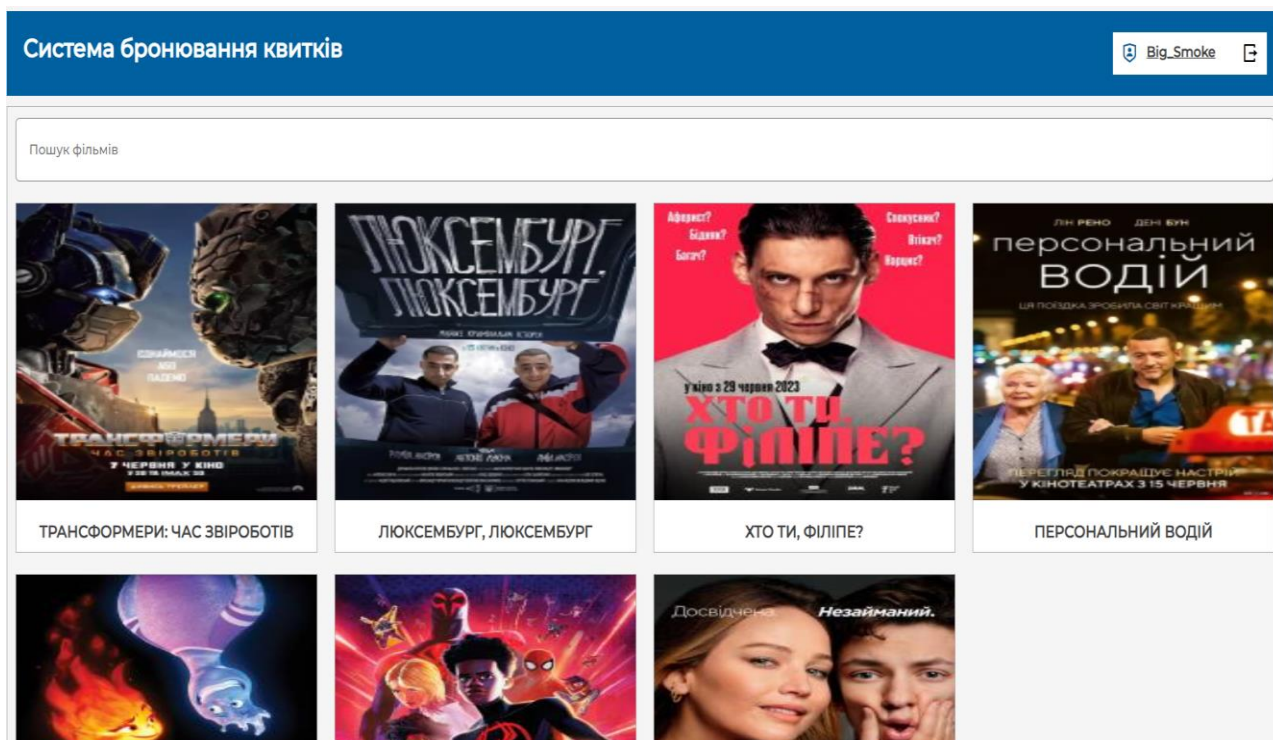


Рисунок 3.3 – Головна сторінка, каталог фільмів

На сторінці ви можете побачити, що було створено декілька кінотеатрів щоб додати їх у систему, (рис 3.4).

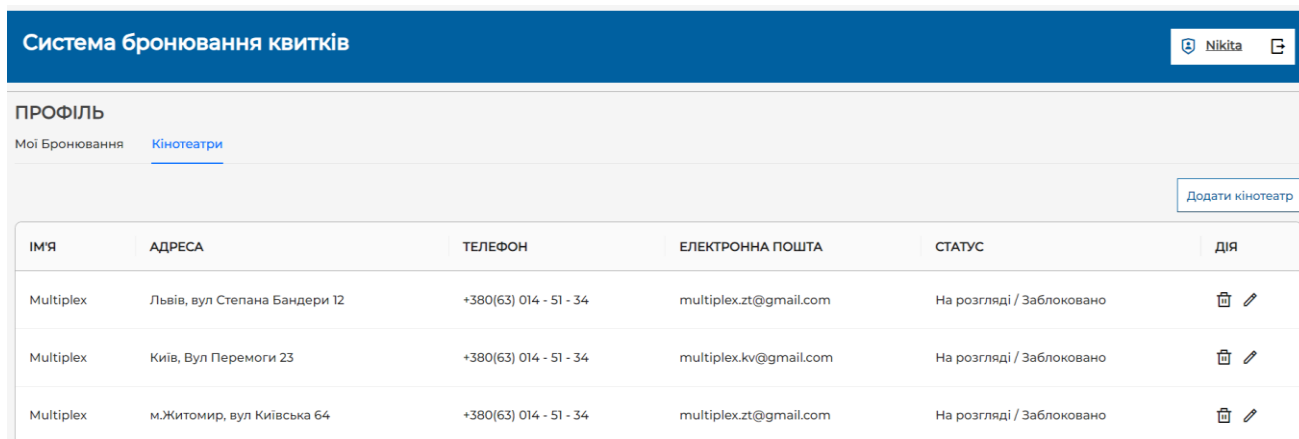


Рисунок 3.4 – Профіль користувача, студ кінотеатрів

Користувач має можливість додати кінотеатр, щоб адмін додав його до системи. Користувач має ввести такі дані як: назва фільму, опис фільму, тривалість фільму (хв), мова, дата виходу фільму, жанр, посилання на постер, (рис 3.5).

ДОДАТИ ФІЛЬМ



Назва фільму

Опис фільму

Тривалість фільму (хв)

Мова

Виберіть мову

▼

Дата виходу фільму

дд.мм.гггг

📅

Жанр

Виберіть жанр

▼

Посилання на постер

Скасувати

Зберегти

Рисунок 3.5 – Форма для додавання кінотеатру

Адміністратор має можливість визначити статус кінотеатру, а саме додати кінотеатр чи відхилити можливість, щоб адміністратор додав його до системи. Користувач має ввести такі дані як: назва фільму, опис фільму, тривалість фільму (хв), мова, дата виходу фільму, жанр, посилання на постер, (рис 3.6).

Система бронювання квитків						
ADMIN						
Фільми Кінотеатри						
ІМ'Я	АДРЕСА	ТЕЛЕФОН	ЕЛЕКТРОННА ПОШТА	ВЛАСНИК	СТАТУС	ACTION
Multiplex	Львів, вул Степана Бандери 12	+380(63) 014 - 51 - 34	multiplex.zt@gmail.com	Nikita	Очікується / Заблоковано	Додати кінотеатр
Multiplex	Київ, Вул Перемоги 23	+380(63) 014 - 51 - 34	multiplex.kv@gmail.com	Nikita	Очікується / Заблоковано	Додати кінотеатр
Multiplex	м.Житомир, вул Київська 64	+380(63) 014 - 51 - 34	multiplex.zt@gmail.com	Nikita	Очікується / Заблоковано	Додати кінотеатр
Multiplex	м.Житомир, вул Київська 64	+380630145134	multiplex_kiev@gmail.com		Очікується / Заблоковано	Додати кінотеатр

Рисунок 3.6 – Адміністративна панель, сторінка з кінотеатрами

Після того, як користувач заповнив форму для додавання кінотеатру у систему, адміністратор має можливість визначити статус кінотеатру. Розглянемо варіант, коли адміністратор додав кінотеатр до системи.

У профілі користувача можна побачити, що в нього додалась можливість додати сеанс, (рис 3.7).

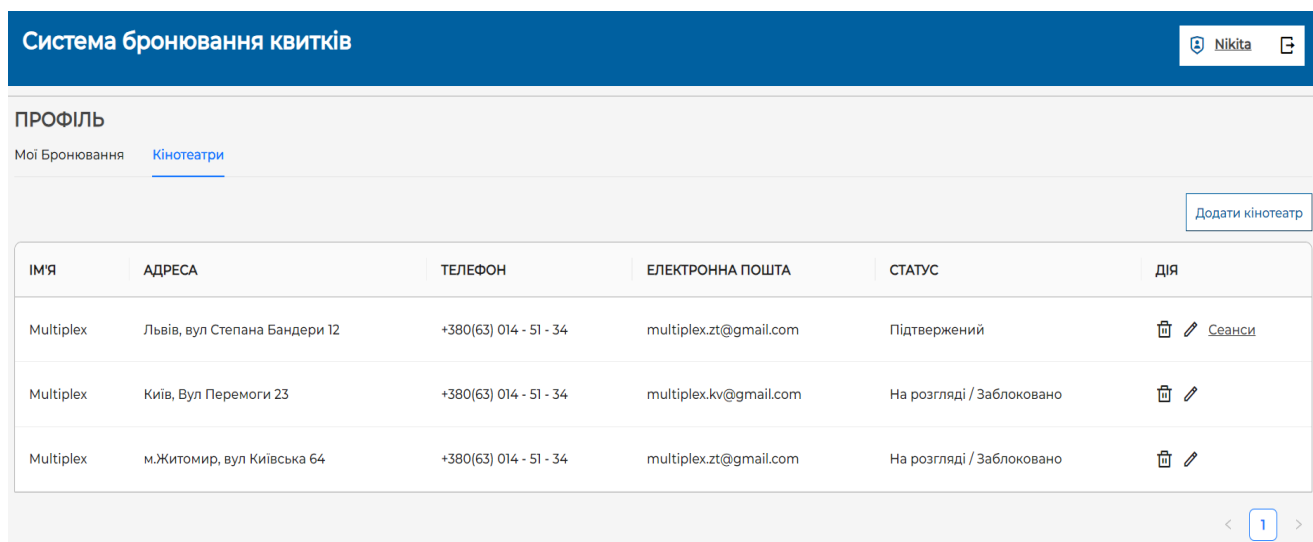


Рисунок 3.7 – Панель користувача, сторінка з кінотеатрами

По-скільки статус кінотеатру для приєднання до системи було підтверджено, користувач має можливість створити сеанси, (рис 3.8).

THEATRE : MULTIPLEX ×

ДОДАТИ СЕАНС

Назва сеансу

ТРАНСФОРМЕРИ: ЧАС ЗВІРОБОТІВ

Дата

30.06.2023

Час

23:30

Фільм

Трансформери: Час Звіроботів

Ціна квитка

230

Загальна кількість місць

110

Скасувати

ЗБЕРЕГТИ

Рисунок. 3.8 – Панель користувача, сторінка для додавання сеансу у кінотеатрі

Обравши фільм та дату можна побачити, що відображається фільм у кінотеатрах де користувач створив сеанси, (рис 3.9).

ADMIN

Фільми

Кінотеатри

Додати фільм

ПОСТЕР	НАЗВА	ОПИС	ТРИВАЛІСТЬ	ЖАНР	МОВА	ДАТА ВИПУСКУ	ДІЯ
	Трансформери: Час Звіроботів	Культова франшиза продовжитьс! В гру вступають нові види трансформерів: Максимили, Предакони и Тероркони. Коли світ опиниться за крок до прірви, людству доведеться об'єднати сили в глобальному протистоянні.	200	Action	Ukranian	28-06-2023	
	Люксембург, Люксембург	Є чимало гарних місць для смерті. Люксембург – не одне з них. Але гарний чи ні, Люксембург робить смерть дуже дорогою. Коли брати-близнюки дізнаються, що їхній давно зниклий батько захворів у Люксембурзі, вони вирушають у подорож, аби побачити його востаннє. Та чи буде чоловік, якого вони там зустрінуть, таким же поганим татом, як вони його пам'ятають?	180	Drama	Ukranian	28-06-2023	
	Хто ти, Філіпе?	Драма «Хто ти, Філіпе?» заснована на романі з автобіографічними мотивами польського письменника єврейського походження Леопольда Тирманда. Стрічка без пафосу і героїзації розповідає про прихований особистий спротив нацистському режиму під час Другої світової. Її головний герой Філіп – суперечливий і неоднозначний персонаж. Польський єврей, утікач із Варшавського гетто, він працює офіціантом в одному із вишуканих ресторанів Франкфурта та вдає із себе француза, щоби мститися нацистам і не викликати у них зайвих підозр.	160	Drama	Ukranian	28-06-2023	

Рисунок 3.12 – CRUD фільмів

ДОДАТИ ФІЛЬМ

Назва фільму

Хто ти, Філіпе?

Опис фільму

Драма «Хто ти, Філіпе?» заснована на романі з автобіографічними мотивами польського письменника єврейського походження Леопольда Тирманда. Стрічка без пафосу і героїзації розповідає про прихований особистий спротив нацистському режиму під час Другої світової. Її головний герой Філіп – суперечливий і неоднозначний персонаж. Польський єврей, утікач із Варшавського гетто, він працює офіціантом в одному із вишуканих ресторанів Франкфурта та вдає

Тривалість фільму (хв)

Мова

Дата виходу фільму

120

Українська

02.07.2023

Жанр

Посилання на постер

Драма

https://d2q8nf5aywi2aj.cloudfront.net/uploads/resize/cinemaFilm/poster

Скасувати

Зберегти

Рисунок 3.13 – CRUD фільмів

У формі для додавання фільму потрібно вказати такі дані як: опис, фільму, тривалість, фільму, мова, дата виходу фільму, жанр, постер на кінотеатр, (рис. 3.13).

Висновки до третього розділу

1. Описано порядок встановлення та налаштування параметрів системи, що є необхідним для правильної роботи продукту.

2. Розглянуто структуру інтерфейсу та порядок взаємодії з онлайн-сервісом. Зазначено, що клієнтська частина представляє собою додаток, що використовує бібліотеки для побудови користувацького інтерфейсу та налаштування логіки оновлення даних та сторінок. Користувач, який є адміністратором, переноситься на вікно адмін-панелі для виконання відповідних функцій.

ВИСНОВКИ

Аналіз аналогів програмного продукту, вибір архітектури веб-додатку та обґрунтування вибору інструментальних засобів та вимог до апаратного забезпечення проведено. В результаті аналізу були виявлені аналогічні програмні продукти, які використовуються для замовлення квитків та послуг у кінотеатрах. Були розглянуті різні архітектурні варіанти для веб-додатку, і на основі обґрунтування було зроблено вибір певної архітектури. Також були визначені вимоги до інструментальних засобів та апаратного забезпечення для успішної реалізації проекту.

Було проведено визначення варіантів використання та структури системи. Було розроблено детальний проект бази даних веб-додатку, що включає в себе необхідні таблиці та зв'язки між ними. Також був проведений аналіз аналогів програмного продукту, що дозволив виявити переваги та недоліки існуючих рішень. На основі цього аналізу та врахування вимог проекту було розроблено концепцію та архітектуру веб-додатку. Окрема увага була приділена реалізації онлайн-сервісу бронювання квитків у кінотеатрах, включаючи відповідні функції та інтерфейс користувача.

Також було описано порядок встановлення та налаштування системи для правильної роботи веб-додатку. Була розглянута структура інтерфейсу та описано порядок роботи з веб-додатком. Це включає в себе взаємодію користувача з різними функціями додатку, такими як вибір фільму, дати та часу шоу, вибір місця та оформлення замовлення квитків.

Розділи 1-3 роботи охоплюють певні аспекти цього процесу, зокрема аналіз аналогів, вибір архітектури, проектування бази даних, аналіз аналогів та реалізацію веб-додатку. Ці розділи детально розглядають відповідні етапи роботи над проектом та надають необхідну основу для подальшої реалізації та розвитку онлайн-сервісу бронювання квитків у кінотеатрах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. React-Router-Dom [Електронний ресурс] – Режим доступу до ресурсу: <https://reactrouter.com/en/main>
2. Express [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/>
3. What is Client Server Architecture? [Електронний ресурс] – Режим доступу до ресурсу: <https://intellipaat.com/blog/what-is-client-server-architecture/?US>
4. Advantages of MongoDB [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mongodb.com/advantages-of-mongodb>
5. ReactJS [Електронний ресурс] – Режим доступу до ресурсу: <https://www.javatpoint.com/pros-and-cons-of-react>
6. Node.js Modules [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/nodejs/nodejs_modules.asp
7. Mongoose getting Started [Електронний ресурс] – Режим доступу до ресурсу: <https://mongoosejs.com/docs/index.html>
8. React Redux Quick Start [Електронний ресурс] – Режим доступу до ресурсу: <https://react-redux.js.org/tutorials/quick-start>
9. Ant Design [Електронний ресурс] – Режим доступу до ресурсу: <https://ant.design/docs/spec/introduce>
10. Middleware [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/en/guide/using-middleware.html>
11. Redux Toolkit: – [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/en/guide/using-middleware.html>
12. StarUML [Електронний ресурс] – Режим доступу до ресурсу: <https://staruml.io/>
13. Axios [Електронний ресурс] – Режим доступу до ресурсу: <https://axios-http.com/docs/intro>
14. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: <https://www.mongodb.com/docs/>

- 15.VirtualDom - [Электронный ресурс] – Режим доступа до ресурсу:
<https://legacy.reactjs.org/docs/faq-internals.html>
- 16.Site Kino-land [Электронный ресурс] – Режим доступа до ресурсу:
<https://kino-land.com/>
- 17.Site Kino-teatr [Электронный ресурс] – Режим доступа до ресурсу:
<https://kino-teatr.ua/uk/>
- 18.Site V kino [Электронный ресурс] – Режим доступа до ресурсу:
<https://vkino.com.ua/ua>
- 19.Class Diagram [Электронный ресурс] – Режим доступа до ресурсу:
<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>.
- 20.Activity Diagram [Электронный ресурс] – Режим доступа до ресурсу:
<https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/>.
- 21.How to Build an Authentication API with JWT Token in Node.js [Электронный ресурс] – Режим доступа до ресурсу: <https://www.section.io/engineering-education/how-to-build-authentication-api-with-jwt-token-in-nodejs/>
- 22.Modern JavaScript book [Электронный ресурс] – Режим доступа до ресурсу:
<https://uk.javascript.info/>
- 23.MongoDB Compass [Электронный ресурс] – Режим доступа до ресурсу:
<https://www.mongodb.com/docs/compass/current/>

ДОДАТКИ

					ПЗ.КР.Б – 121 – 23 – ПЗ	
						50

Моделі баз даних

BookingModel.js

```
const mongoose = require("mongoose");

const bookingSchema = new mongoose.Schema(
  {
    show: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "shows",
    },
    user: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "users",
    },
    seats: {
      type: Array,
      required: true,
    },
    transactionId: {
      type: String,
      required: true,
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model("bookings", bookingSchema);

const mongoose = require("mongoose");
```

MovieModel.js

```
const movieSchema = new mongoose.Schema(
  {
    title: {
      type: String,
      required: true,
    },
    description: {
      type: String,
      required: true,
    },
    duration: {
      type: Number,
      required: true,
    },
    genre: {
      type: String,
```

```

        required: true,
    },
    language: {
        type: String,
        required: true,
    },
    releaseDate: {
        type: Date,
        required: true,
    },
    poster: {
        type: String,
        required: true,
    },
    },
    { timestamps: true }
);

```

```
module.exports = mongoose.model("movies", movieSchema);
```

SessionModel.js

```

const mongoose = require('mongoose');

const SessionSchema = new mongoose.Schema({
    name : {
        type: String,
        required: true
    },
    date : {
        type: Date,
        required: true
    },
    time : {
        type: String,
        required: true
    },
    movie : {
        type: mongoose.Schema.Types.ObjectId,
        ref: 'movies',
        required: true
    },
    ticketPrice : {
        type: Number,
        required: true
    },
    totalSeats : {
        type: Number,
        required: true
    },
    },

```

```

    bookedSeats : {
      type: Array,
      default: []
    },
    theatre : {
      type: mongoose.Schema.Types.ObjectId,
      ref: 'theatres',
      required: true
    },
  }, { timestamps: true });

const Show = mongoose.model('session', SessionSchema);

module.exports = Show;

```

TheatreModel.js

```

const mongoose = require("mongoose");

const theatreSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    address: {
      type: String,
      required: true,
    },
    phone: {
      type: String,
      required: true,
    },
    email: {
      type: String,
      required: true,
    },
    owner: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "users",
    },
    isActive: {
      type: Boolean,
      default: false,
    },
  },
  { timestamps: true }
);

module.exports = mongoose.model("theatres", theatreSchema);

```

UserModel.js

```
const mongoose = require("mongoose");

const userSchema = new mongoose.Schema(
  {
    name: {
      type: String,
      required: true,
    },
    email: {
      type: String,
      required: true,
      unique: true,
    },
    password: {
      type: String,
      required: true,
    },
    isAdmin: {
      type: Boolean,
      required: true,
      default: false,
    },
  },
  {
    timestamps: true,
  }
);

module.exports = mongoose.model("users", userSchema);
```

Маршрути API

Bookings.js

```

const { axiosInstance } = require(".");

// Add a new movie
export const AddMovie = async (payload) => {
  try {
    const response = await axiosInstance.post("/api/movies/add-movie",
payload);
    return response.data;
  } catch (error) {
    return error.response;
  }
}

// get all movies
export const GetAllMovies = async () => {
  try {
    const response = await axiosInstance.get("/api/movies/get-all-movies");
    return response.data;
  } catch (error) {
    return error.response;
  }
}

// update a movie
export const UpdateMovie = async (payload) => {
  try {
    const response = await axiosInstance.post("/api/movies/update-movie",
payload);
    return response.data;
  } catch (error) {
    return error.response;
  }
}

// delete a movie
export const DeleteMovie = async (payload) => {
  try {
    const response = await axiosInstance.post("/api/movies/delete-movie",
payload);
    return response.data;
  } catch (error) {
    return error.response;
  }
}

```

```
// get a movie by id
export const GetMovieById = async (id) => {
  try {
    const response = await axiosInstance.get(`/api/movies/get-movie-by-id/${id}`);
    return response.data;
  } catch (error) {
    return error.response;
  }
}
```

Theatres.js

```
import { axiosInstance } from ".";

// Add a new theatre
export const AddTheatre = async (payload) => {
  try {
    const response = await axiosInstance.post(
      "/api/theatres/add-theatre",
      payload
    );
    return response.data;
  } catch (error) {
    return error.response;
  }
};

// get all theatres
export const GetAllTheatres = async () => {
  try {
    const response = await axiosInstance.get("/api/theatres/get-all-theatres");
    return response.data;
  } catch (error) {
    return error.response;
  }
};

// get all theatres by owner
export const GetAllTheatresByOwner = async (payload) => {
  try {
    const response = await axiosInstance.post(
      "/api/theatres/get-all-theatres-by-owner",
      payload
    );
    return response.data;
  } catch (error) {
    return error.response;
  }
}
```

```

};

// update theatre
export const UpdateTheatre = async (payload) => {
  try {
    const response = await axiosInstance.post(
      "/api/theatres/update-theatre",
      payload
    );
    return response.data;
  } catch (error) {
    return error.response;
  }
};

// delete theatre
export const DeleteTheatre = async (payload) => {
  try {
    const response = await axiosInstance.post(
      "/api/theatres/delete-theatre",
      payload
    );
    return response.data;
  } catch (error) {
    return error.response;
  }
};

// add show
export const AddSession = async (payload) => {
  try {
    const response = await axiosInstance.post(
      "/api/theatres/add-show",
      payload
    );
    return response.data;
  } catch (error) {
    return error.response;
  }
};

// get all shows
export const GetAllShowsByTheatre = async (payload) => {
  try {
    const response = await axiosInstance.post(
      "/api/theatres/get-all-shows-by-theatre",
      payload
    );
    return response.data;
  } catch (error) {

```

```

        return error.response;
    }
};

// delete show
export const DeleteSession = async (payload) => {
    try {
        const response = await axiosInstance.post(
            "/api/theatres/delete-show",
            payload
        );
        return response.data;
    } catch (error) {
        return error.response;
    }
};

// get all theatres for a movie
export const GetAllTheatresByMovie = async (payload) => {
    try {
        const response = await axiosInstance.post(
            "/api/theatres/get-all-theatres-by-movie",
            payload
        );
        return response.data;
    } catch (error) {
        return error.response;
    }
};

// get show by id
export const GetSessionById = async (payload) => {
    try {
        const response = await axiosInstance.post(
            "/api/theatres/get-show-by-id",
            payload
        );
        return response.data;
    } catch (error) {
        return error.response;
    }
};

```

Users.js

```

const { axiosInstance } = require(".");

// Реєстрація нового користувача
export const RegisterUser = async (payload) => {
    try {
        const response = await axiosInstance.post("/api/users/register", payload);
    }
};

```

```

return response.data;
} catch (error) {
return error.response;
}
};

// Вхід користувача
export const LoginUser = async (payload) => {
try {
const response = await axiosInstance.post("/api/users/login", payload);
return response.data;
} catch (error) {
return error.response;
}
}

// Отримати поточного користувача
export const GetCurrentUser = async () => {
try {
const response = await axiosInstance.get("/api/users/get-current-user");
return response.data;
} catch (error) {
return error;
}
}

```

Movies.js.

```

const { axiosInstance } = require(".");

// Додати новий фільм
export const AddMovie = async (payload) => {
try {
const response = await axiosInstance.post("/api/movies/add-movie", payload);
return response.data;
} catch (error) {
return error.response;
}
}

// Отримати всі фільми
export const GetAllMovies = async () => {
try {
const response = await axiosInstance.get("/api/movies/get-all-movies");
return response.data;
} catch (error) {
return error.response;
}
}

```

```

// Оновити фільм
export const UpdateMovie = async (payload) => {
  try {
    const response = await axiosInstance.post("/api/movies/update-movie", payload);
    return response.data;
  } catch (error) {
    return error.response;
  }
}

// Видалити фільм
export const DeleteMovie = async (payload) => {
  try {
    const response = await axiosInstance.post("/api/movies/delete-movie", payload);
    return response.data;
  } catch (error) {
    return error.response;
  }
}

// Отримати фільм за ідентифікатором
export const GetMovieById = async (id) => {
  try {
    const response = await axiosInstance.get(`/api/movies/get-movie-by-id/${id}`);
    return response.data;
  } catch (error) {
    return error.response;
  }
}

```

index.js

```

import axios from 'axios';

export const axiosInstance = axios.create({
  headers : {
    'Content-Type': 'application/json',
    authorization : `Bearer ${localStorage.getItem('token')}`
  }
});

```

Головний компонент маршрутизації

```

function App() {
  const { loading } = useSelector((state) => state.loaders);
  return (
    <div>
      {loading && (
        <div className="loader-parent">
          <div className="loader"></div>
        </div>
      )}
      <BrowserRouter>
        <Routes>
          <Route
            path="/"
            element={
              <ProtectedRoute>
                <Home />
              </ProtectedRoute>
            }
          />
          <Route
            path="/movie/:id"
            element={
              <ProtectedRoute>
                <TheatresForMovie />
              </ProtectedRoute>
            }
          />
          <Route
            path="/book-show/:id"
            element={
              <ProtectedRoute>
                <BookShow />
              </ProtectedRoute>
            }
          />
          <Route
            path="/profile"
            element={
              <ProtectedRoute>
                <Profile />
              </ProtectedRoute>
            }
          />
          <Route
            path="/admin"
            element={
              <ProtectedRoute>

```

```

        <Admin />
      </ProtectedRoute>
    }
  />
  <Route path="/login" element={<Login />} />
  <Route path="/register" element={<Register />} />
</Routes>
</BrowserRouter>
</div>
);
}

export default App;

```