Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська дипломна робота

на тему: «Розробка програмного забезпечення для керування бізнес-процесами кінотеатрів. Частина 1. Модуль сервера»

Виконав: студент IV курсу групи ЗПІ-20б
спеціальності 121 – Інженерії програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Михальнюк О.О.
(прізвище та ініціали)

Керівник: к.т.н., доцент кафедри ПЗ
Майданюк В. П.
(прізвище та ініціали)
«____»_____2024 р.

Рецензент: к.т.н., доцент кафедри ОТ
Кожем'яко А. В.
(прізвище та ініціали)
«____»_____2024 р.

Допущено до захисту
Зав. кафедри ПЗ _____
«_10_» ___червня___ 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О.Н Романюк.
«12» березня 2024 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Михальнюку Олександру Олександровичу

1. Тема роботи – «Розробка програмного забезпечення для керування бізнес-процесами кінотеатрів. Частина 1. Модуль сервера.»

Керівник роботи: Майданюк Володимир Павлович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 11 березня 2024 р. №80.

2. Строк подання студентом роботи 3 червня 2024 р.

3. Вихідні дані до роботи: середовище розробки Visual Studio Code, мова розробки JavaScript, платформа Node.js.

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану питання розробки; порівняльний аналіз аналогів; аналіз методів розв'язання задачі; постановка задачі; розробка структури модуля сервера, розробка моделі модуля сервера, розробка структури бази даних, розробка алгоритму відправлення запрошення на кіносеанс; тестування розробленого програмного застосунку; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження, новизна, практична цінність одержаних результатів; порівняльний аналіз аналогів, використані технології, алгоритм відправлення запрошення на кіносеанс, тестування, висновки, апробація результатів роботи і публікації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Майданюк В.П., к.т.н., доцент кафедри ПЗ	13.03.24	03.06.24

7. Дата видачі завдання 13 березня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану питання розробки модуля сервера	14.03.24 – 24.03.24	Виконано
2	Розробка структури модуля сервера	24.03.24 – 06.04.24	Виконано
3	Варіантний аналіз та вибір мови програмування	06.04.24 – 11.04.24	Виконано
4	Розробка серверної частини модуля керування фільмами та кінотеатрами	12.04.24 – 29.04.24	Виконано
5	Тестування модуля сервера	29.04.24 – 11.05.24	Виконано
6	Оформлення матеріалів до захисту БДР	12.05.24– 30.05.24	Виконано

Студент _____ (підпис) Михальнюк О.О. (прізвище та ініціали)

Керівник бакалаврської дипломної роботи _____ (підпис) Майданюк В.П. (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.42

Михальнюк О. О. Розробка програмного забезпечення для керування бізнес-процесами кінотеатрів. Частина 1. Модуль сервера: бакалаврська дипломна робота зі спеціальності 121 Інженерія програмного забезпечення, освітня програма – Іінженерія програмного забезпечення. Вінниця: ВНТУ, 2024. 92 с. На укр. мові. Бібліогр. : 20 назв ; рис. : 42; табл. 2.

У бакалаврській дипломній роботі проведено аналіз стану питання розробки модуля сервера . Було проведено аналіз аналогів, визначено їх переваги і недоліки та доведено доцільність розробки власного модуля.

Виконане обґрунтування вибору мови та середовища програмування, а також технологій, які були використані під час розробки модуля сервера.

Розроблено програмний продукт, який являє собою модуль сервера для для керування бізнес-процесами кінотеатрів.

Робота реалізована за допомогою мови програмування JavaScript. В якості середовища для розробки програмного забезпечення було обрано Visual Studio Code

Отримані в бакалаврській дипломній роботі результати можна використати для інтеграції з модулем клієнта для реалізації функціоналу, що дозволяє створювати кіносеанси перегляду фільмів, керувати кінотеатрами, бронювати місця в кінозалах для перегляду фільму, отримувати запрошення електронною поштою.

Ключові слова: веб-система, кінотеатр, програмне забезпечення.

ANNOTATION

UDC 004.42

Mykhalnyuk O. O. Development of software for managing business processes of cinemas. Part 1. Server module: Bachelor's thesis on the specialty 121 Software engineering, educational program - Software engineering. Vinnytsia: VNTU, 2024. 92 p. In Ukrainian speech Bibliogr. : 20 titles; Fig. : 42; table 2.

In the bachelor's thesis, an analysis of the state of the development of the server module was carried out. An analysis of analogues was carried out, their advantages and disadvantages were determined, and the expediency of developing one's own module was proven.

The justification of the choice of the programming language and environment, as well as the technologies that were used during the development of the server module, was carried out.

A software product has been developed, which is a server module for managing the business processes of cinemas.

The work is implemented using the JavaScript programming language. Visual Studio Code was chosen as the software development environment

The results obtained in the bachelor's thesis can be used for integration with the client module to implement functionality that allows you to create movie viewing sessions, manage movie theaters, reserve seats in movie theaters, receive invitations by e-mail.

Keywords: web system, cinema, software.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Веб-системи для управління роботи бізнес-процесів кінотеатрів є невід'ємною частиною сучасної індустрії розваг. Вони дозволяють автоматизувати та спрощувати різні операції, пов'язані з керуванням кінотеатром, від продажу квитків до управління розкладом фільмів [1].

Однією з ключових функцій таких систем є онлайн-продаж квитків. Це дозволяє глядачам придбати квитки на свій улюблений фільм в будь-який час і з будь-якого місця, без необхідності стояти в черзі в касі кінотеатру. Крім того, веб-системи зазвичай надають можливість обрати конкретні місця в залі, що робить перегляд фільму більш комфортним.

Іншою важливою функцією веб-систем для кінотеатрів є управління розкладом фільмів. За допомогою таких систем, адміністратори кінотеатрів можуть легко планувати покази фільмів, враховуючи часові рамки, доступність залів та інші фактори. Це допомагає уникнути конфліктів у розкладі та забезпечує ефективне використання ресурсів кінотеатру.

Крім того, веб-системи для управління роботи бізнес-процесів кінотеатрів часто включають в себе модулі для управління персоналом, бухгалтерського обліку, маркетингу та інші корисні інструменти. Це дозволяє керівництву кінотеатру ефективно контролювати всі аспекти роботи бізнесу, приймати обґрунтовані рішення та підвищувати ефективність роботи. Тому тема роботи є актуальною.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є полегшення керування бізнес-процесами кінотеатрів і оптимізація управління різноманітними аспектами кінотеатрального бізнесу.

Основними задачами дослідження є:

- розробити метод відправлення запрошення на кіносеанс;

- розробити серверну частину модуля управління кінотеатрами та фільмами;

- розробити функціонал для формування статистики перегляду фільмів;

- розробити серверну частину модуля покупки квитків та функціонал для генерації QR-коду на фільм;

- провести тестування розробленого функціоналу.

Об’єкт дослідження - процес розробки модуля сервера веб-системи для керування бізнес-процесами кінотеатрів.

Предмет дослідження - методи і програмні засоби реалізації модуля сервера веб-системи для керування бізнес-процесами кінотеатрів.

Методи дослідження. У процесі досліджень використовувались: теорія алгоритмів, теорія баз даних, методи теорії мереж для розробки моделей роботи серверної частини веб-системи; комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Подальшого розвитку отримав метод побудови веб-систем для керування бізнес процесами кінотеатрів, у якому, на відміну від існуючих, використано технологію Node.js для реалізації серверної частини веб-системи, що дозволило спростити розробку та підвищити продуктивність на 10%.
2. Подальшого розвитку отримав метод визначення статистики відвідування сайту веб-систем для керування бізнес процесами кінотеатрів, у якому, на відміну від існуючих, визначаються пристрої з яких виконується вхід в систему.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в бакалаврській дипломній роботі теоретичних положень запропоновано алгоритми та розроблено з використанням технології Node.js серверну частину веб-системи для керування бізнес процесами кінотеатрів.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській дипломній роботі, отримані автором особисто. У друкованій праці,

опублікованій у співавторстві, автору належать такі результати: розробка алгоритму роботи веб-системи [2].

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на Міжнародній науково-практична інтернет-конференція «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)».

Публікації. Основні результати досліджень опубліковано в 1 науковій праці у матеріалах конференції.

1 АНАЛІЗ СТАНУ РОЗВИТКУ ПРОГРАМ ДЛЯ КЕРУВАННЯ БІЗНЕС-ПРОЦЕСАМИ КІНОТЕАТРІВ

1.1 Аналіз стану питання розробки модуля сервера

Ринок кінотеатрів характеризується високою конкуренцією, як між великими мережами, так і незалежними закладами. Це змушує кінотеатри постійно покращувати клієнтський досвід та оптимізувати свої бізнес-процеси. Крім того, утримання кінотеатрів вимагає значних експлуатаційних витрат, ефективне управління якими є критичним для рентабельності бізнесу. Ринок також сильно залежить від трендів у кіноіндустрії, тому кінотеатри повинні вчасно реагувати на зміни в уподобаннях глядачів. Водночас, сучасні кінотеатри активно впроваджують цифрові технології, включно з програмними системами для управління квитками, CRM, розкладом сеансів та фінансами. Ці системи допомагають підвищувати ефективність бізнесу та якість обслуговування клієнтів.

Розробка модуля сервера для веб-системи програмного забезпечення, що керує бізнес-процесами кінотеатрів, є актуальним завданням з кількох причин. По-перше, кінотеатри мають складні бізнес-процеси, такі як бронювання квитків, управління розкладом сеансів та облік касових операцій. Модуль сервера дозволить автоматизувати ці процеси, підвищити їх ефективність та знизити ризик помилок. Особливо це важливо для великих мереж кінотеатрів, які потребують централізованого управління. Модуль сервера забезпечить синхронізацію даних, моніторинг та контроль над всіма закладами.

Крім того, веб-система, побудована на модулі сервера, дозволить значно покращити клієнтський сервіс. Користувачі зможуть самостійно бронювати квитки, переглядати розклад сеансів та отримувати персоналізовані рекомендації. Це сприятиме підвищенню лояльності клієнтів та їх задоволеності. Також, модуль сервера забезпечуватиме збір та аналіз детальної аналітики про продажі, відвідуваність та профілі глядачів. Це дасть можливість приймати обґрунтовані управлінські рішення на основі даних. Крім того, модульна архітектура системи забезпечить її масштабованість та гнучкість, що дозволить розширювати

функціональність в майбутньому.

Отже, розробка модуля сервера для веб-системи управління бізнес-процесами кінотеатрів є актуальним завданням, яке дозволить підвищити ефективність, клієнтоорієнтованість та аналітичні можливості таких закладів в умовах конкурентного та динамічного ринку.

1.2 Порівняльний аналіз аналогів

На ринку програмних застосунків для керування бізнес-процесами кінотеатрів існують деякі готові системи. Розглянемо їх і порівняємо із власною розробкою.

Cinema Management System від компанії Vista [3] - це комплексна система, що охоплює ключові процеси кінотеатрів - від бронювання квитків і управління розкладом до фінансового обліку та аналітики. Cinema Management System надає веб-портал для клієнтів, дозволяючи їм купувати квитки онлайн, а також мобільний додаток для персоналу. Система інтегрується з POS-терміналами, системами розпізнавання відвідувачів та іншими сторонніми сервісами. Перевагами CMS є її всебічність та широкі можливості кастомізації під потреби конкретного кінотеатру.

Система Cinema Management System продемонстрована на рисунку 1.1.

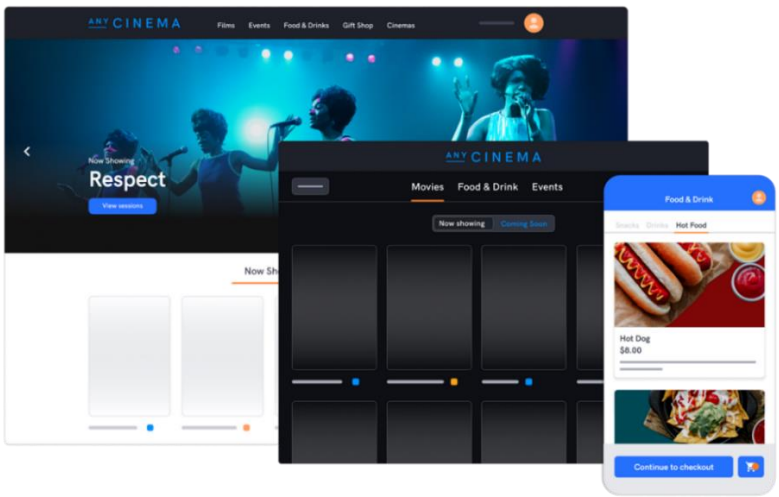

Рисунок 1.1 - Cinema Management System від компанії Vista

Cineverse [4] - це хмарне рішення, яке спеціалізується на управлінні маркетингом та взаємодією з клієнтами для кінотеатрів. Cineverse дозволяє створювати персоналізовані пропозиції, проводити таргетовані кампанії, аналізувати уподобання та поведінку відвідувачів. Перевагами Cineverse є можливість глибокого аналізу споживчої аналітики та ефективного таргетованого маркетингу.

Система Cineverse продемонстрована на рисунку 1.2.

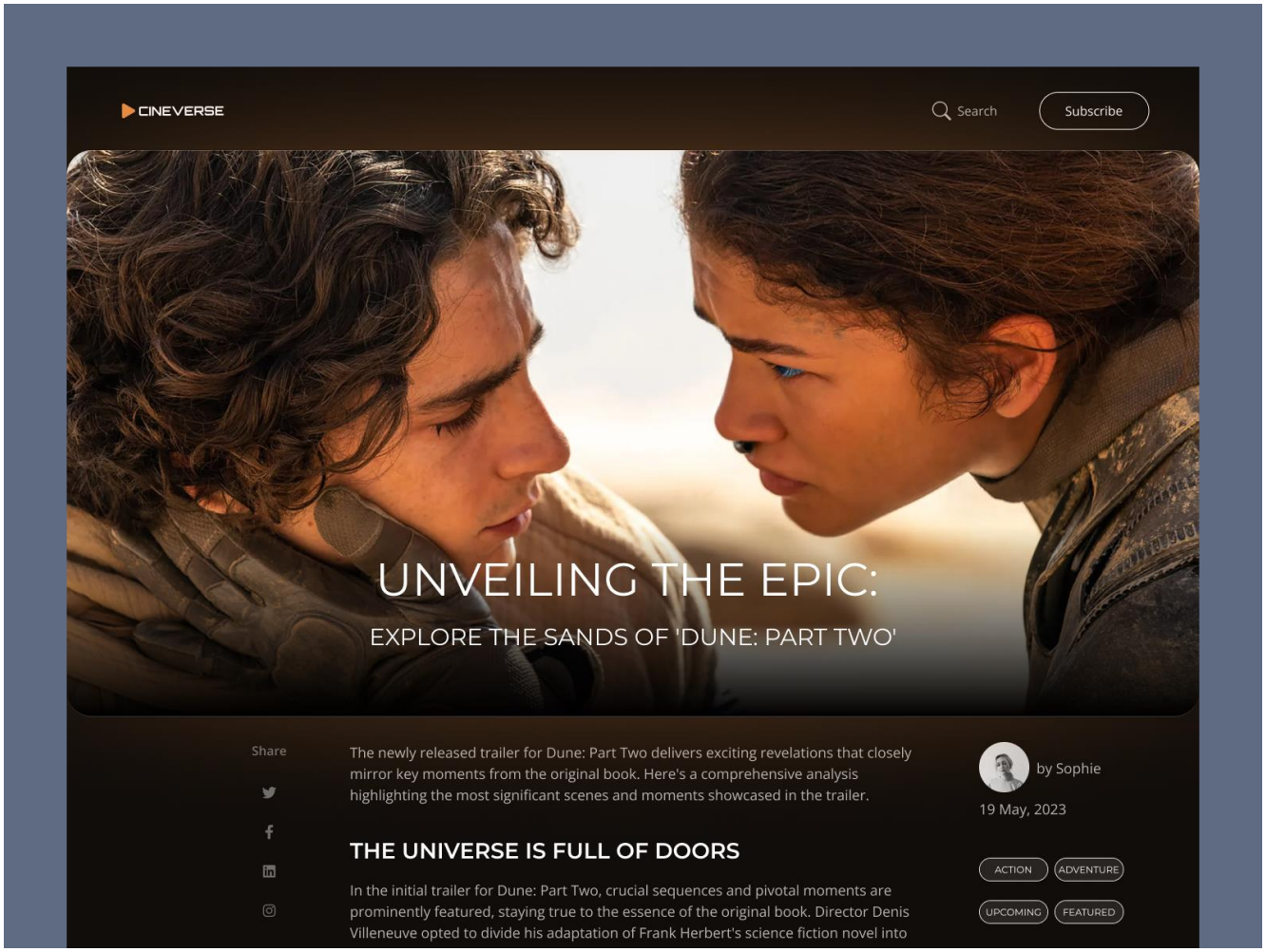

Рисунок 1.2 – Cineverse

Система Movio Cinema [5] - спрямована на комплексне управління всіма аспектами діяльності кінотеатрів - від продажу квитків і управління розкладом до фінансового обліку та звітності. Movio Cinema надає веб-портал для клієнтів, мобільний додаток для персоналу, а також потужні аналітичні інструменти. Ця

система дозволяє оптимізувати ключові показники, такі як відвідуваність, середній чек та лояльність клієнтів.

Система Movio Cinema продемонстрована на рисунку 1.3.

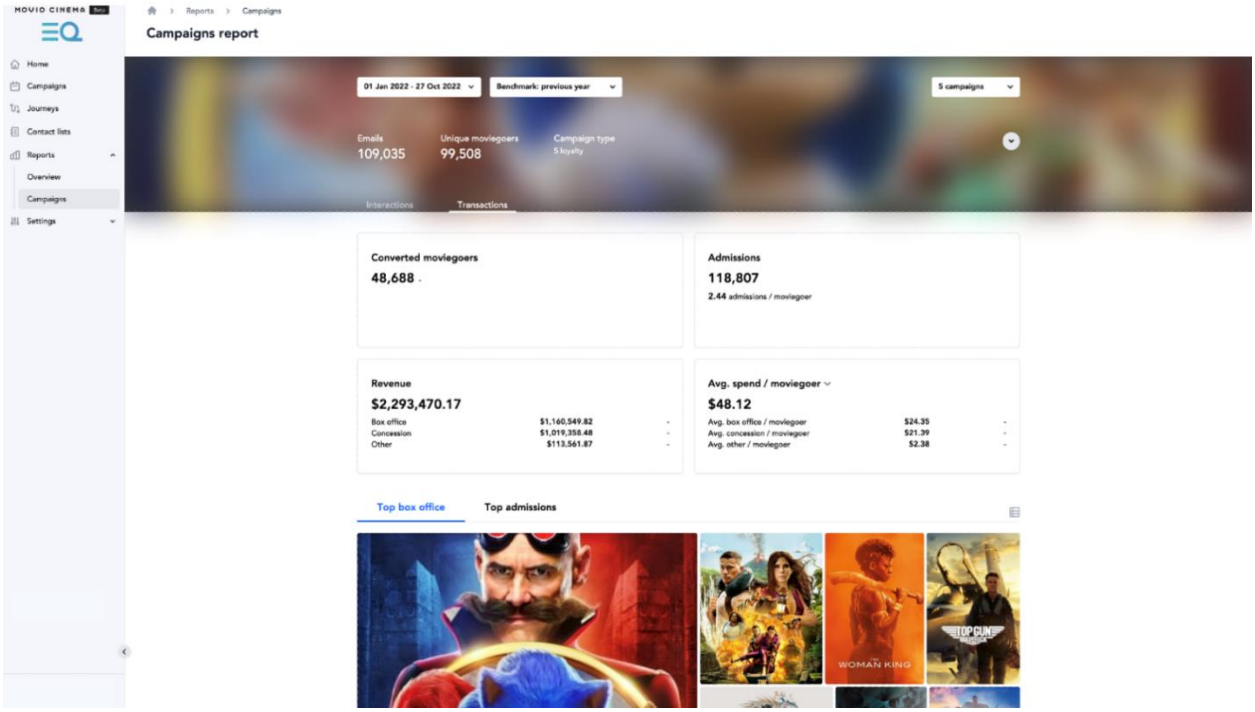

Рисунок 1.3 – Movio Cinema

Для демонстрації відмінностей між розглянутими системами та власною розробкою, було проведено порівняння їх характеристик та створено порівняльну таблицю (таблиця 1.1).

Таблиця 1.1 – Порівняння характеристик застосунків

Характеристика	Cinema Management System	Cineverse	Movio Cinema	Власна розробка
Безкоштовний доступ	0	0	0	1
Додавання інформації про кінозали	1	1	0	1

Продовження таблиці 1.1.

Формування звітів	1	1	1	1
Статистика користувачів	0	0	1	1
Створення кіносеансів	1	1	1	1
Сумарний бал	3	3	3	5

Таким чином, власна розробка за характеристиками перевершує аналоги на 40% (100% - 3/5*100% = 40%).

Таким чином, власна розробка для управління бізнес-процесами кінотеатрів є актуальною.

1.3 Аналіз методів розв’язання задачі

Для реалізації модуля сервера для веб-системи управління бізнес-процесами кінотеатрів можна застосувати кілька різних підходів та інструментів. Один з варіантів - використання традиційних веб-фреймворків, таких як Spring Boot, Django або Ruby on Rails.

Ці технології мають багату екосистему плагінів та бібліотек, чітко визначену архітектуру і перевірені шаблони проєктування, що забезпечує надійність та зрілість рішення. Однак, вони також вимагають більших зусиль для розгортання та масштабування, а також потребують самостійної розробки багатьох базових модулів.

Альтернативним варіантом є застосування серверних платформ «без сервера» (serverless), наприклад AWS Lambda, Google Cloud Functions (рисунок 1.4) або Azure Functions (рисунок 1.5). Такі рішення забезпечують високу масштабованість та відмовостійкість, при цьому звільняючи розробників від необхідності управляти інфраструктурою.

Проте, вони також мають певні обмеження на час виконання та розмір пам'яті,

а налагодження та тестування таких систем може бути складнішим. Крім того, вибір платформи прив'язує проєкт до конкретного хмарного провайдера.

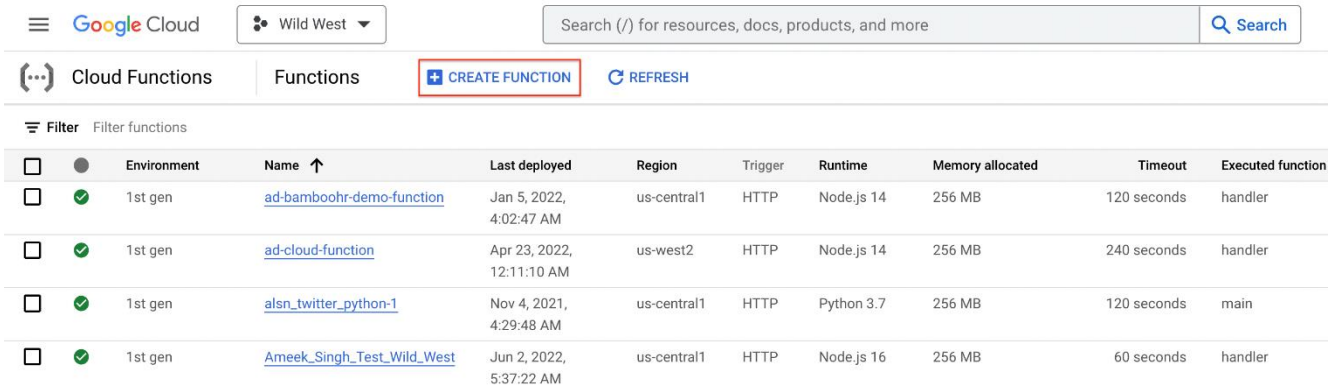

Рисунок 1.4 – Google Cloud Functions

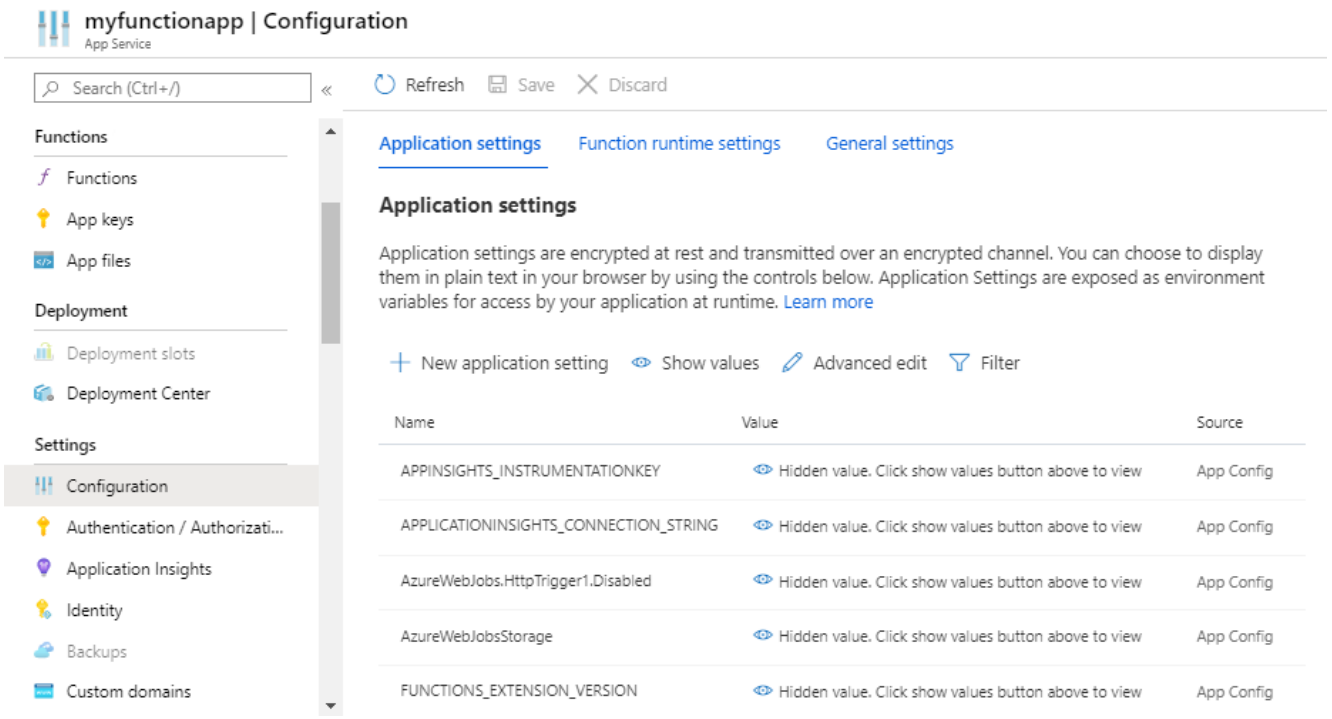

Рисунок 1.5 – Azure functions

Ще одним підходом є використання мікросервісної архітектури з інструментами на кшталт Docker (рисунок 1.6), Kubernetes та NGINX (рисунок 1.7). Така архітектура забезпечує гнучкість, масштабованість та ізольованість сервісів, дозволяючи незалежну розробку та розгортання. Втім, управління розподіленою системою може бути доволі складним завданням, що потребує додаткових засобів

оркестрації та моніторингу.

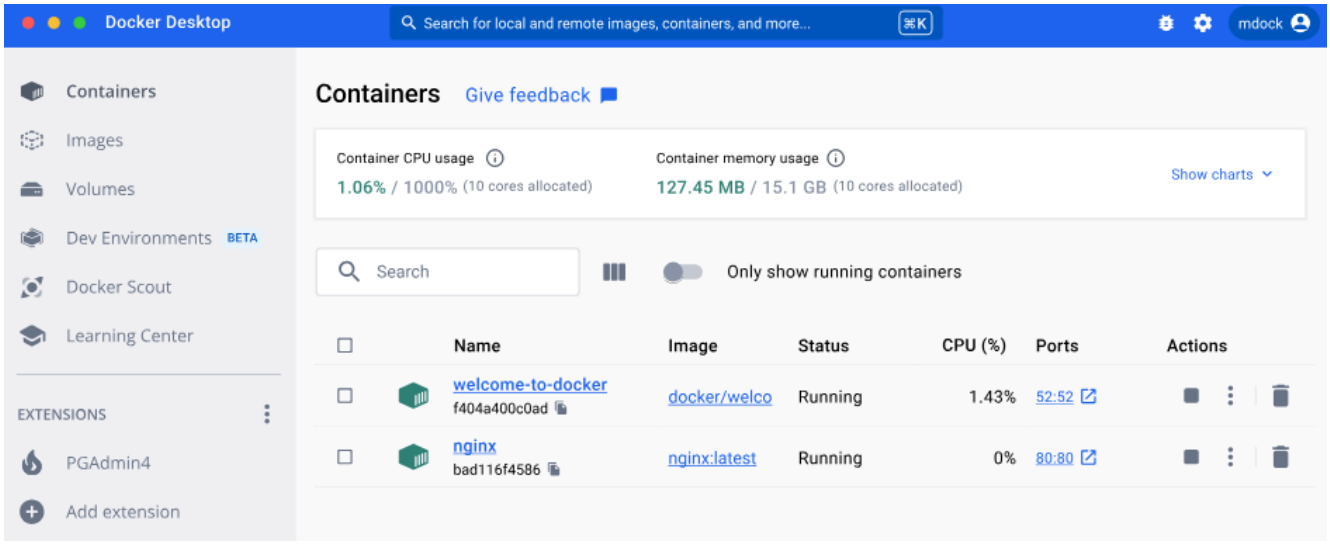

Рисунок 1.6 – Docker

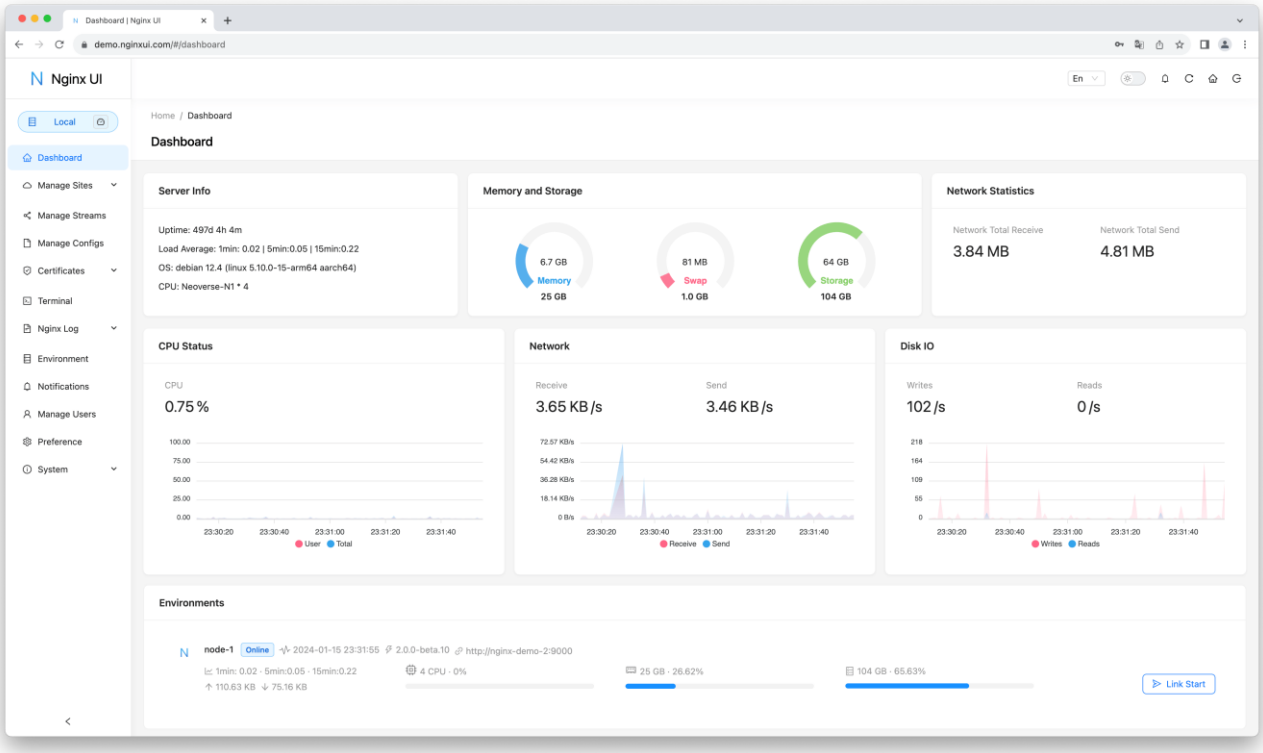

Рисунок 1.7 - NGINX

Також можна розглянути готові серверні рішення, такі як Node.js, ASP.NET або Laravel. Вони дозволяють пришвидшити початок розробки завдяки багатій екосистемі модулів та плагінів, а також великим спільнотам розробників. Проте,

такі рішення можуть мати меншу гнучкість у налаштуванні та кастомізації.

З огляду на потреби кінотеатрів у високій масштабованості, надійності та інтеграції з існуючими системами, найоптимальнішим варіантом буде розробка на основі Node.js з бібліотекою Express.

Архітектура Node.js наведена на рисунку 1.8.

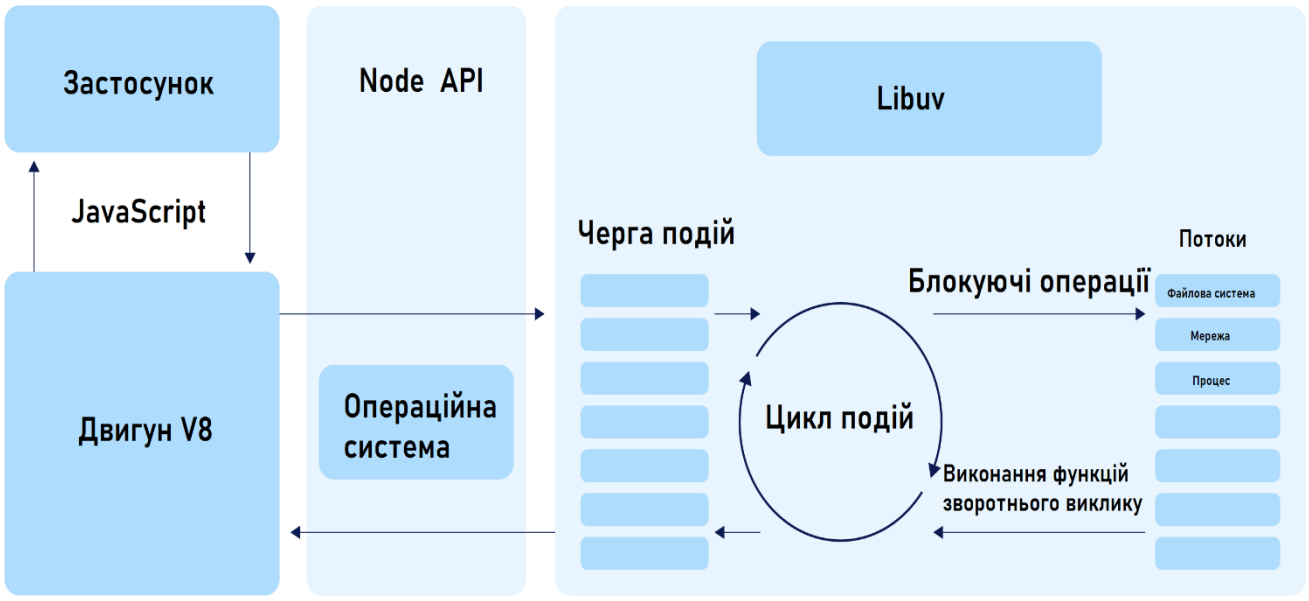

Рисунок 1.8 – Архітектура Node.js

Ця платформа дозволяє швидко створювати високопродуктивні, легко масштабовані сервери. Вона має відкриту екосистему, що включає безліч готових рішень для управління бізнес-процесами, роботи з базами даних, авторизації, моніторингу тощо.

1.4 Постановка задач дослідження

В результаті аналізу стану питання розробки модуля сервера, порівняльного аналізу аналогів та аналізу методів розв’язання задачі, було поставлено такі задачі дослідження:

- розробити метод відправлення запрошення на кіносеанс;
- розробити серверну частину модуля управління кінотеатрами та фільмами;
- розробити функціонал для формування статистики перегляду фільмів;

- розробити серверну частину модуля покупки квитків та функціонал для генерації QR-коду на фільм.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

В цьому розділі було проведено аналіз стану питання розробки серверної частини веб-системи для управління бізнес-процесами кінотеатрів. Було проведено порівняльний аналіз аналогів, в результаті якого було доведено актуальність власної розробки. Було проведено аналіз методів розв’язання задачі та прийнято рішення розробити серверну частину з використанням Node.js і Express. Було проведено постановку задач дослідження.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ РОБОТИ МОДУЛЯ СЕРВЕРА

2.1 Розробка структури модуля сервера

Для розробки модуля сервера було використано паттерн Model-View-Controller (MVC).

Патерн MVC (Model-View-Controller) [6] - це архітектурний шаблон, який використовується для розробки програмного забезпечення, передусім веб-додатків. Він забезпечує чітке розділення відповідальностей між різними частинами додатку, що спрощує розробку, тестування та подальше обслуговування.

Модель (Model) - це частина, яка відповідає за доступ до даних та їх маніпуляцію. Вона визначає структуру даних, правила валідації та логіку бізнес-процесів. Модель не має жодного уявлення про те, як дані будуть відображатися чи оброблятися.

Вид (View) - це компонент, який відповідає за відображення даних користувачеві. Він отримує дані від моделі та формує їх у зрозумілий для користувача інтерфейс, наприклад, HTML-сторінку або JSON-відповідь для AJAX-запитів.

Контролер (Controller) - це проміжна ланка між моделлю та видом. Він отримує вхідні дані від користувача, передає їх моделі для обробки, а потім вибирає відповідний вид для відображення результатів.

Одна з ключових переваг патерну MVC - це можливість повторного використання компонентів. Наприклад, ту саму модель можна використовувати з різними видами для різних цілей, або один і той самий вид може відображати дані з різних моделей.

Патерн MVC широко застосовується у веб-розробці, оскільки він спрощує розробку складних додатків, забезпечуючи модульність та гнучкість. Він також допомагає підтримувати чистий та організований код, що полегшує командну роботу над проектом.

Структуру модуля сервера наведено на рисунку 2.1.

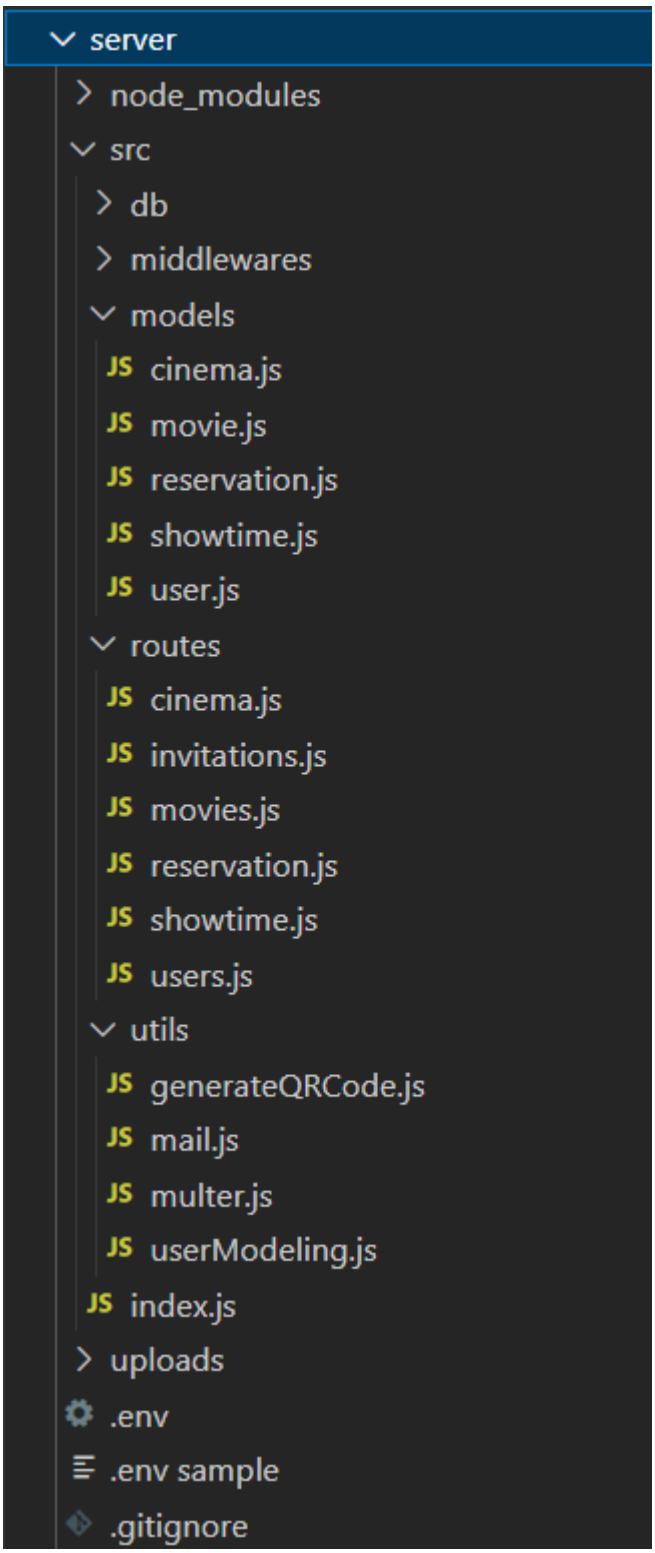

Рисунок 2.1 – Структура модуля сервера

Директорія "models" містить файли, що визначають моделі даних для сутностей, таких як Cinema, Movie, Reservation, Showtime та User. Ці файли, ймовірно, описують схеми даних, правила валідації та бізнес-логіку, пов'язану з відповідними сутностями. Це повністю відповідає ролі "Моделі" в патерні MVC.

Директорія "routes" містить файли, які, ймовірно, діють як контролери. Вони визначають маршрути або ендпоїнти додатку та оброблюють HTTP-запити. Кожен файл, такий як cinema.js, movies.js, reservation.js тощо, відповідає за певний набір маршрутів та обробку відповідних запитів. Ці файли виконують роль "Контролера" в патерні MVC, отримуючи вхідні дані, взаємодіючи з моделями та вибираючи відповідні відображення (вигляди) для надсилання відповіді.

У наданій структурі немає явної директорії для "Вигляду". Проте, залежно від того, яку платформу або фреймворк використано, відображення (наприклад, HTML-шаблони або JSON-відповіді) можуть генеруватися контролерами або окремими модулями для відображення.

Також, створені інші директорії:

«utils» - містить допоміжні функції та модулі, які можуть використовуватися в різних частинах додатку;

«middlewares» - містить проміжне програмне забезпечення для обробки запитів/відповідей;

«db» - містить конфігурацію та з'єднання з базою даних;

«src» - містить основний вихідний код додатку.

2.2 Розробка моделі модуля сервера

Для кращого розуміння архітектури серверної частини, було розроблено модель модуля сервера (рисунок 2.2).

Для клієнтської частини системи використовується фреймворк React. Серверна частина системи пов’язана із клієнтською з використанням Express.js.

Express.js [6] - це потужний і гнучкий фреймворк для Node.js, який значно спрощує розробку веб-додатків та API. Ось детальніший опис його ключових особливостей:

Маршрутизація є однією з головних функцій Express.js. Фреймворк надає просту й інтуїтивну систему для визначення маршрутів (routes) та відповідних обробників для різних HTTP-методів, таких як GET, POST, PUT, DELETE тощо. Це дозволяє легко структурувати логіку додатку та обробляти різні типи запитів від

клієнтів.

Отже, Express.js використовується для створення API, що поєднує клієнтську та серверну частини системи.

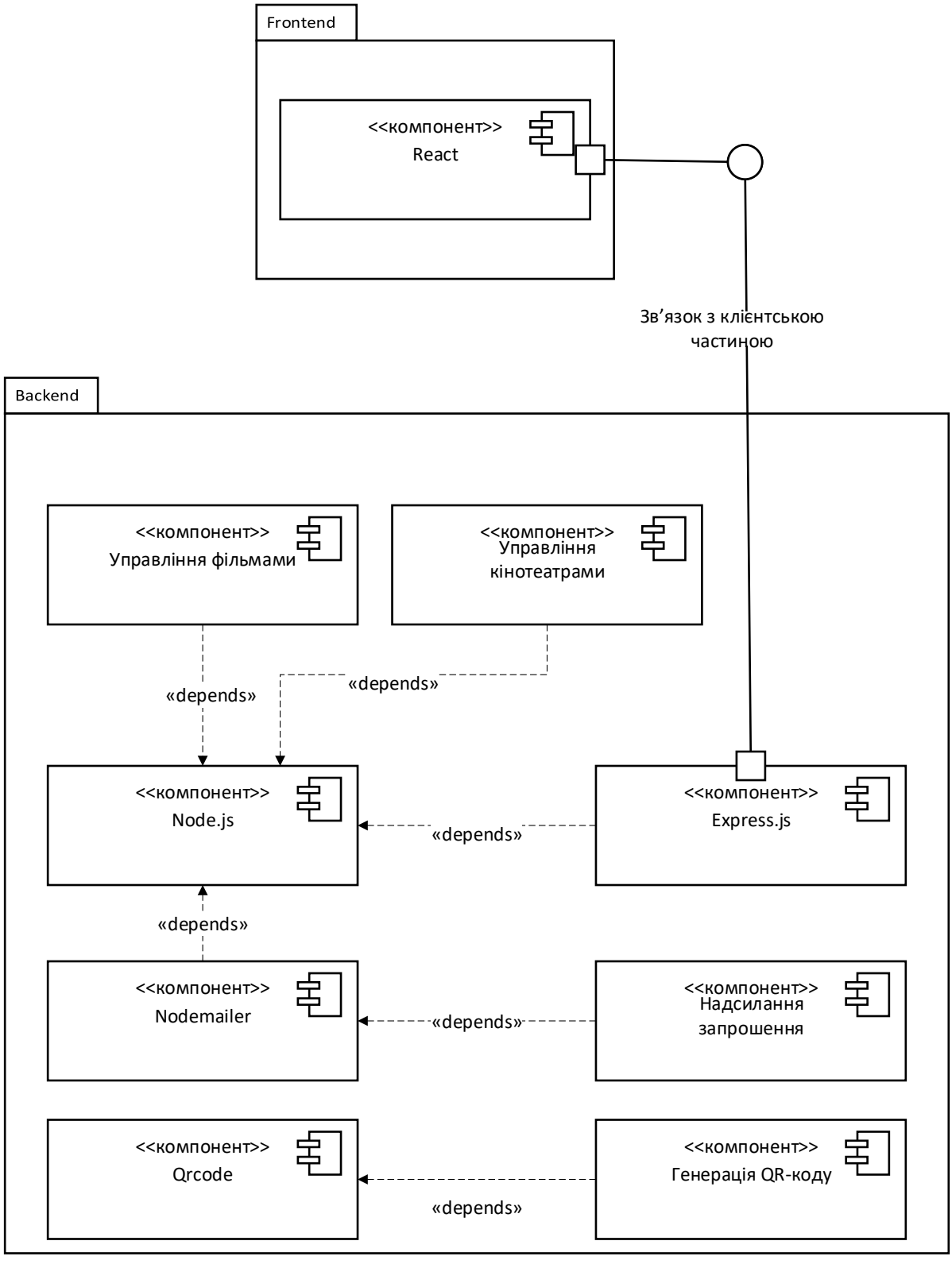

Рисунок 2.2 – Модель модуля сервера

Управління фільмами – компонент, для якого розроблена його серверна частина, який відповідає за керування даними про фільми, такими як назва, опис, жанр, акторський склад тощо.

Управління кінотеатрами - компонент, для якого розроблена його серверна частина та який відповідає за керування даними про кінотеатри, такими як місцезнаходження, розклад сеансів, ціни на квитки тощо.

Node.js [8] - середовище виконання JavaScript на сервері, яке дозволяє запускати серверні додатки, написані на JavaScript.

Node.js широко використовується для створення серверних додатків, таких як веб-сервери, API-сервери, проксі-сервери, а також для різноманітних інструментів для розробки, наприклад, інструментів для збірки коду, тестування та автоматизації завдань. Завдяки своїй легковажності та масштабованості, Node.js також широко застосовується в інтернеті речей (IoT) та мікросервісних архітектурах.

Спочатку Node.js був розроблений для оптимізації програм, які вимагають високої продуктивності і паралельної обробки великої кількості підключень, проте зараз він застосовується у найрізноманітніших сферах розробки JavaScript, як на стороні сервера, так і на стороні клієнта.

Nodemailer [9] - бібліотека для Node.js, яка надає функціональність для відправки електронних листів з серверної частини.

Qrcode [10] - компонент або бібліотека для генерації QR-кодів, які можуть використовуватися для швидкого доступу до інформації, наприклад, посилань на веб-ресурси або купівлі квитків.

Надсилання запрошення - компонент серверної частини системи, який відповідає за відправку запрошень на відвідування фільму користувачам на електронну пошту.

Генерація QR-коду - компонент, який використовує бібліотеку Qrcode для генерації QR-кодів, що виступають в ролі квитків на фільм.

2.3 Розробка схеми бази даних

Для веб-системи програмного забезпечення з керування бізнес-процесами

кінотеатрів необхідна надійна та масштабована база даних. Вона повинна забезпечувати зберігання великих обсягів інформації про фільми, сеанси, квитки, а також дані про клієнтів та працівників.

Було прийнято рішення використати нереляційну базу даних MongoDB. Вона забезпечує високу масштабованість та гнучкість структури даних, що може бути корисним для зберігання неструктурованих даних [11].

Крім того, важливо врахувати питання безпеки та конфіденційності даних. База даних повинна підтримувати шифрування даних, управління доступом та регулярне резервне копіювання для забезпечення цілісності та захисту інформації. Також необхідно передбачити можливість масштабування бази даних у майбутньому, оскільки система буде розвиватися та збільшуватиметься кількість користувачів.

Було розроблено схему бази даних для веб-системи керування бізнес-процесами кінотеатрів. Вона наведена на рисунку 2.3.

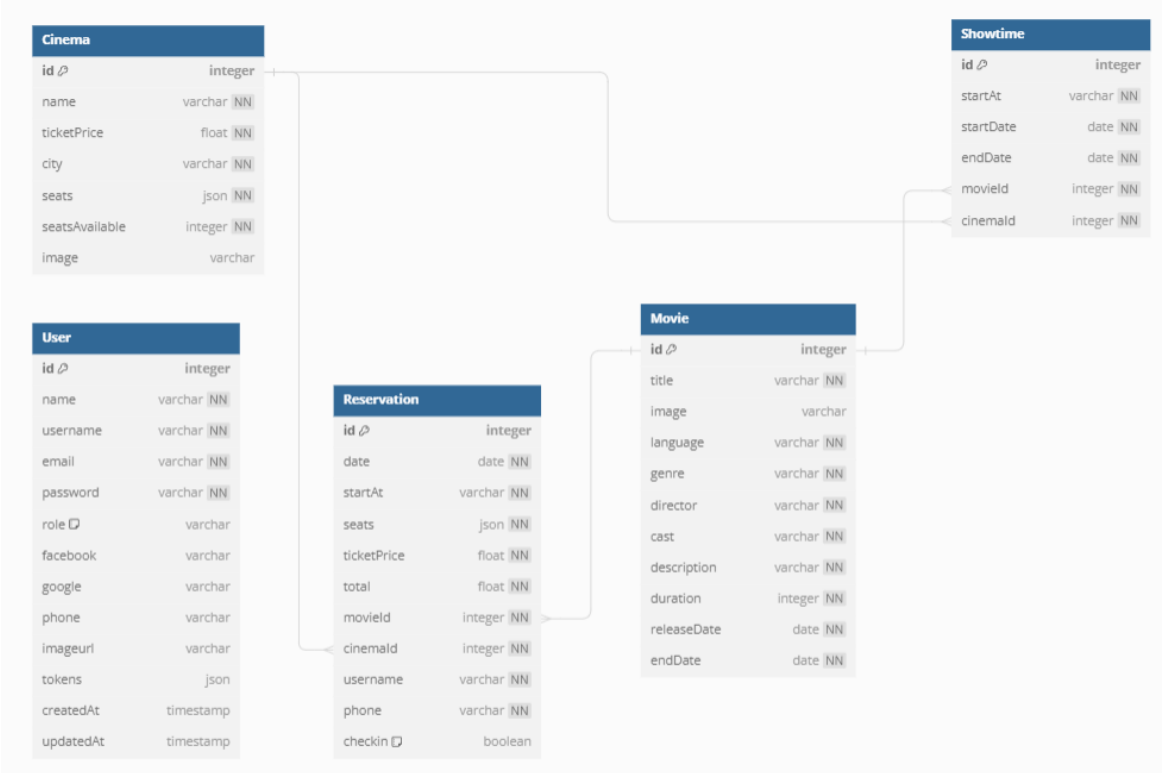

Рисунок 2.3 – Схема бази даних

Таблиця User містить дані про користувачів системи.

Опис таблиці User:

- id – ідентифікаційний номер користувача в системі;
- name – повне ім’я користувача;
- username – нікнейм користувача;
- email – електронна адреса користувача;
- password – пароль користувача;
- role – роль користувача в системі;
- facebook – аккаунт Facebook;
- google – аккаунт Google;
- phone – номер телефону;
- imageUrl – посилання на фото користувача;
- tokens – токени авторизації;
- createdAt – дата створення аккаунту;
- updatedAt – дата останнього оновлення аккаунту.

Таблиця Cinema містить дані про кінотеатри, що знаходяться в управлінні.

Опис таблиці Cinema:

- id – ідентифікаційний номер кінотеатру в системі;
- name – назва кінотеатру;
- ticketPrice – ціна квитка;
- city – місто;
- seats – кількість місць;
- seatsAvailable – кількість вільних місць;
- image – фото кінотеатру.

Таблиця Reservation містить дані про бронювання місць для перегляду кіносеансів відвідувачами.

Опис таблиці Reservation:

- id – ідентифікаційний номер в системі;
- date – дата бронювання;
- startAt – час початку;
- seats – кількість місць;

- ticketPrice – ціна квитка;

- total – загальна сума;

- movieId – ідентифікаційний номер фільму;

- cinemaId – ідентифікаційний номер кінотеатру;

- username – ім’я користувача;

- phone – номер телефону;

- checkIn – бронювання підтверджено.

Таблиця Movie містить інформацію про фільми, що доступні для перегляду у кінотеатрах.

Опис таблиці Movie:

- id – ідентифікаційний номер фільму у системі;

- title – назва;

- image – постер;

- language – мова показу;

- genre – жанр фільму;

- director – режисер;

- cast – акторський склад;

- description – опис фільму;

- duration – тривалість фільму;

- releaseDate – дата виходу фільму;

- endDate – дата завершення показу.

Таблиця ShowTime містить інформацію про дату та час показів фільмів.

Опис таблиці ShowTime:

- id – ідентифікаційний номер показу;

- startAt – час початку;

- startDate – дата початку;

- endDate – дата завершення показів;

- movieId – ідентифікаційний номер фільму;

- cinemaId – ідентифікаційний номер кінотеатру.

2.4 Розробка алгоритму відправлення запрошення на кіносеанс

Відправлення запрошення на кіносеанс електронною поштою відвідувачу має ряд переваг. Воно допомагає нагадати про дату, час і місце кіносеансу. Це дозволяє уникнути плутанини чи запам'ятовування деталей події.

В електронному запрошенні можна розмістити додаткову інформацію про майбутні прем'єри, акції чи пропозиції кінотеатру. Це допоможе залучити відвідувачів до інших заходів.

Надсилання персоналізованих запрошень демонструє турботу кінотеатру про своїх відвідувачів, створюючи враження цінного клієнта. Це може покращити імідж бренду та підвищити лояльність аудиторії.

Електронне нагадування зменшує ймовірність того, що відвідувач забуде про куплений квиток і не з'явиться на сеанс.

Відстеження переглядів електронних запрошень може надати цінну аналітичну інформацію про поведінку аудиторії та ефективність маркетингових кампаній.

Електронні запрошення замінюють паперові квитки, що сприяє захисту навколишнього середовища та зменшенню витрат на друк.

Отже, надсилання електронних запрошень на кіносеанси є корисною практикою, яка забезпечує зручність для глядачів, маркетингові можливості для кінотеатру та потенціал для підвищення лояльності аудиторії.

На рисунку 2.4 зображено алгоритм розсилки запрошень на фільм. Він включає в себе збір інформації про усі придбані квитки та відповідно заброньовані в кінотеатрі місця, після якого відбувається відбір необхідної для розсилки листів. Коли усе підготовлено до розсилання запрошень, відбувається почергове формування запрошень та їх відправка на електронні адреси, вказані користувачами при реєстрації. Існує 2 сценарії відправки повідомлення: або лист надіслано успішно, або ж відбулася помилка при відправці. В будь-якому разі, система виводить сповіщення щодо успішності відправки листа.

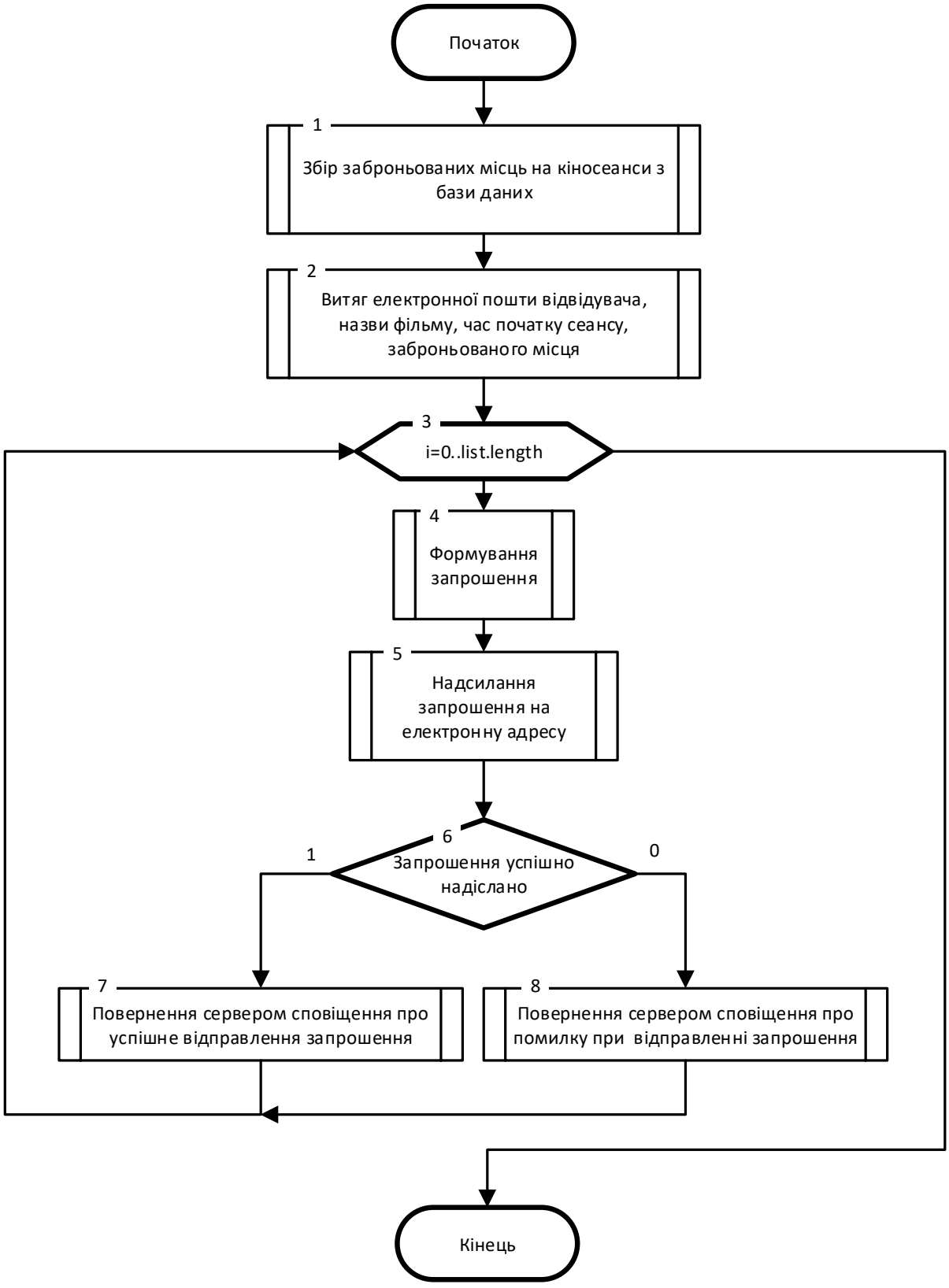

Рисунок 2.4 - Алгоритм відправлення запрошень

Структура листа-запрошення на фільм продемонстрована на рисунку 2.5.

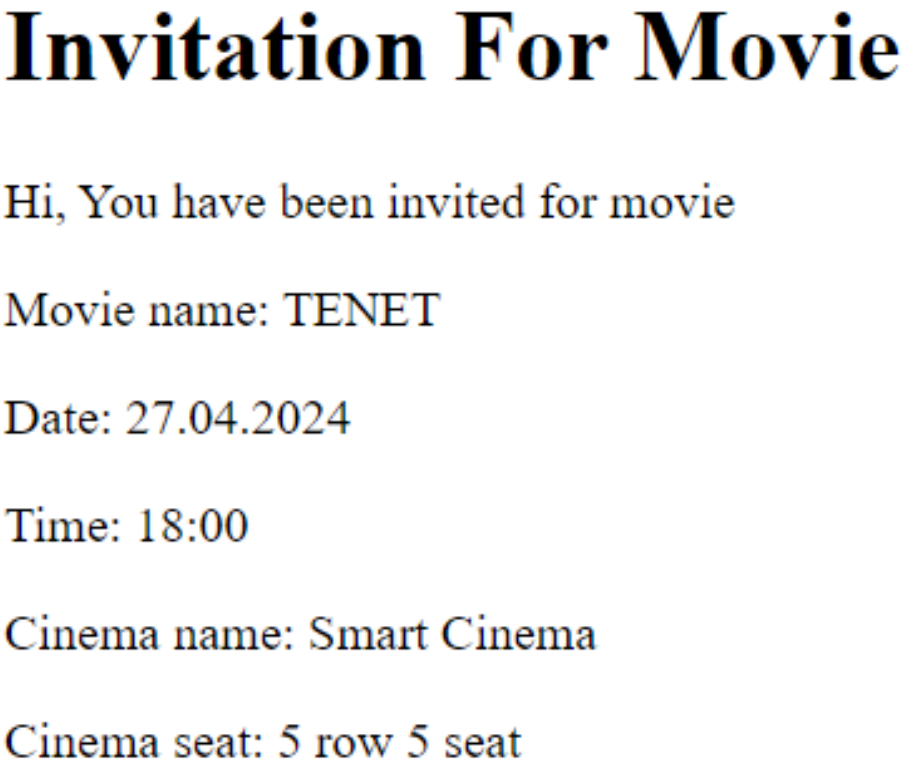

Рисунок 2.5 - Структура запрошення на фільм

2.5 Розробка рівнів доступу до системи

Працівників та відвідувачів кінотеатрів можна поділити на різні рівні залежно від їх повноважень та обов'язків роботи із забезпечення діяльності кінотеатрів.

Відвідувачі кінотеатру купують квитки та відвідують кіносеанси. Вони є тим, заради кого і завдяки кому працюють кінотеатри.

Так, наприклад, в кінотеатрах зазвичай існує операційний відділ, який відповідає за щоденне функціонування кінотеатру. Включає керівників залів, касирів та інший обслуговуючий персонал. Він займається проведенням кіносеансів та продажем квитків.

На вищому рівні знаходиться керівництво кінотеатру або мережі кінотеатрів. Воно відповідальне за діяльність кінотеатрів, приймає рішення стосовно того, які фільми показувати в кінотеатрах, займається формуванням цін на квитки.

З точки зору веб-системи для управління бізнес-процесами кінотеатрів, можна сформувати діаграму класів, яка демонструє ієрархію перечислених учасників діяльності роботи кінотеатрів (рисунок 2.6).

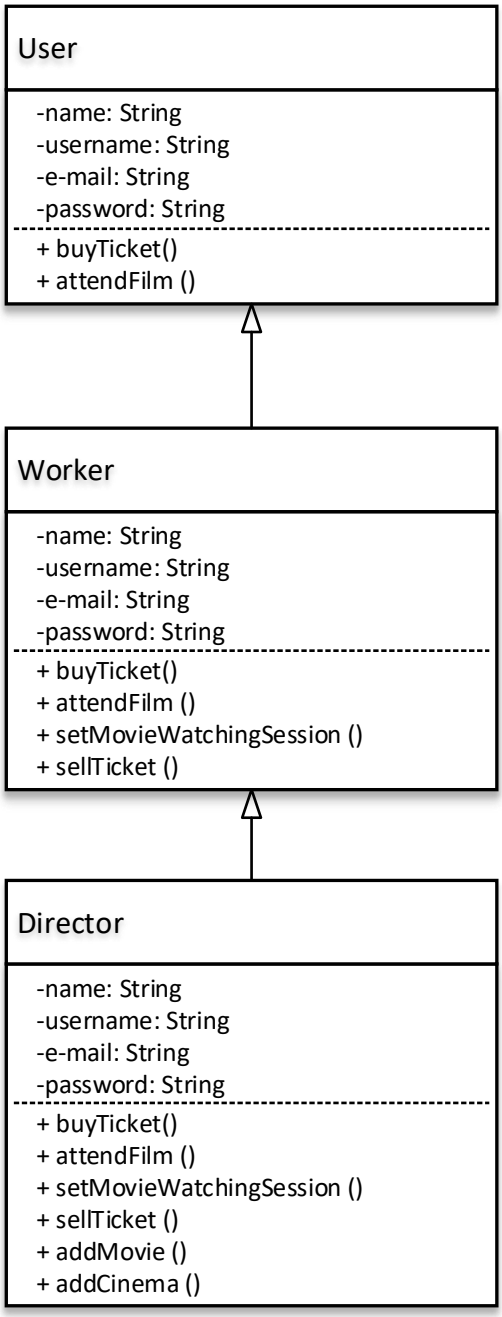

Рисунок 2.6 - Діаграма класів учасників процесу діяльності роботи кінотеатрів

З цієї діаграми класів ми можемо побачити, що User (відвідувач, користувач системи) має доступ до базового функціоналу системи, такого як відвідати фільм та придбати квиток.

Працівник (Worker) має той же функціонал, що і відвідувач, але, окрім того, може ще й продавати квитки та ставити сесії перегляду фільмів.

Директор же має функції Працівника та, окрім того, може додавати нові

фільми та кінотеатри для перегляду.

2.6 Висновки

В цьому розділі розроблено структуру модуля сервера, побудовано модель модуля сервера веб-системи для управління бізнес-процесами кінотеатрів. Розроблено схему бази даних, описано складові її таблиць. Розроблено метод відправлення запрошення на кіносеанс відвідувачам, описано його алгоритм. Розроблено рівень доступу користувачів до системи.

3 РОЗРОБКА МОДУЛЯ СЕРВЕРА ВЕБ-СИСТЕМИ ДЛЯ КЕРУВАННЯ БІЗНЕС-ПРОЦЕСАМИ КІНОТЕАТРІВ

3.1 Варіантний аналіз та вибір мови програмування

Java [12] – це об’єктно-орієнтована мова програмування високого рівня, розроблена компанією Sun Microsystems (зараз Oracle) у 1995 році. Вона була створена з метою забезпечення кросплатформеності, що дозволяє запускати Java-програми на різних операційних системах без необхідності повторної компіляції.

Однією з ключових особливостей Java є використання віртуальної машини (JVM). Коли Java-код компілюється, він перетворюється на байт-код, який потім інтерпретується та виконується JVM. Це забезпечує високу продуктивність та безпеку виконання.

Архітектура Java наведена на рисунку 3.1.

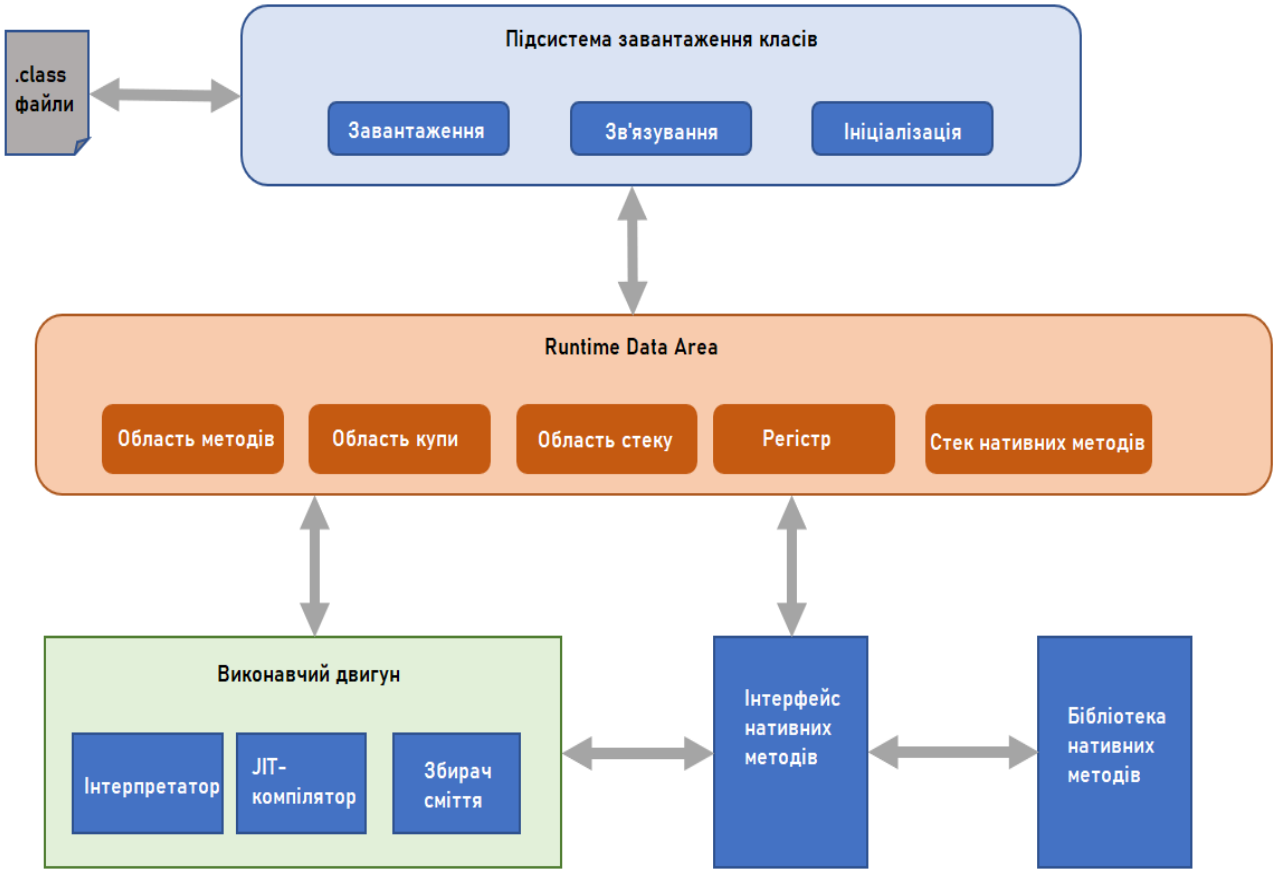

Рисунок 3.1 – Архітектура Java

Java широко використовується для розробки корпоративних застосунків, веб-сервісів, мобільних додатків та ігор. Популярні фреймворки для веб-розробки на Java включають Spring, Jakarta EE (колишня Java EE) та Play Framework. Вони надають багато функціональності та бібліотек для побудови масштабованих і надійних веб-додатків.

Перевагами Java є її кросплатформеність, багата екосистема бібліотек, потужні інструменти розробки та багаторічний досвід використання у виробничих середовищах. Однак деякі розробники критикують її за надмірну вербозність та повільність старту програм через використання JVM.

Незважаючи на поширення нових мов та технологій, Java залишається однією з найпопулярніших мов програмування, особливо в корпоративному середовищі. Її стабільність, безпека та надійність роблять її привабливим вибором для створення масштабованих та високонавантажених веб-додатків.

JavaScript [13] – це сценарна мова програмування, спочатку розроблена Netscape для додавання інтерактивності на веб-сторінки. Вона була випущена в 1995 році і швидко стала невід'ємною частиною веб-розробки на стороні клієнта.

JavaScript працює у веб-браузерах і дозволяє маніпулювати елементами HTML/CSS, обробляти події користувача, створювати динамічний вміст та взаємодіяти з веб-сервісами через AJAX. Це забезпечує багату взаємодію та покращений користувацький досвід на веб-сайтах.

З появою Node.js у 2009 році, JavaScript також став використовуватися для розробки серверних додатків. Node.js дозволяє виконувати JavaScript поза браузером, що відкрило нові можливості для створення повноцінних веб-додатків з використанням однієї мови на клієнті та сервері.

Для веб-розробки на JavaScript використовуються різноманітні фреймворки та бібліотеки, такі як React, Angular, Vue.js, Express.js та багато інших. Вони полегшують розробку, забезпечують модульність, повторне використання коду та покращують продуктивність.

JavaScript став однією з найпопулярніших мов програмування завдяки своїй гнучкості, кросплатформеності та зростаючій екосистемі інструментів та бібліотек.

Незважаючи на деякі проблеми з безпекою та сумісністю в минулому, JavaScript продовжує розвиватися та залишається невід'ємною частиною сучасної веб-розробки.

TypeScript [14] – це мова програмування, розроблена компанією Microsoft у 2012 році. Вона є надбудовою над JavaScript, яка додає статичну типізацію, класи та інші можливості об'єктно-орієнтованого програмування.

TypeScript був створений з метою покращення масштабованості, підтримки великих проектів та полегшення рефакторингу коду. Він дозволяє визначати типи змінних, функцій та об'єктів, що робить код більш зрозумілим, передбачуваним і легшим для виявлення помилок на етапі розробки.

Архітектура TypeScript наведена на рисунку 3.2.

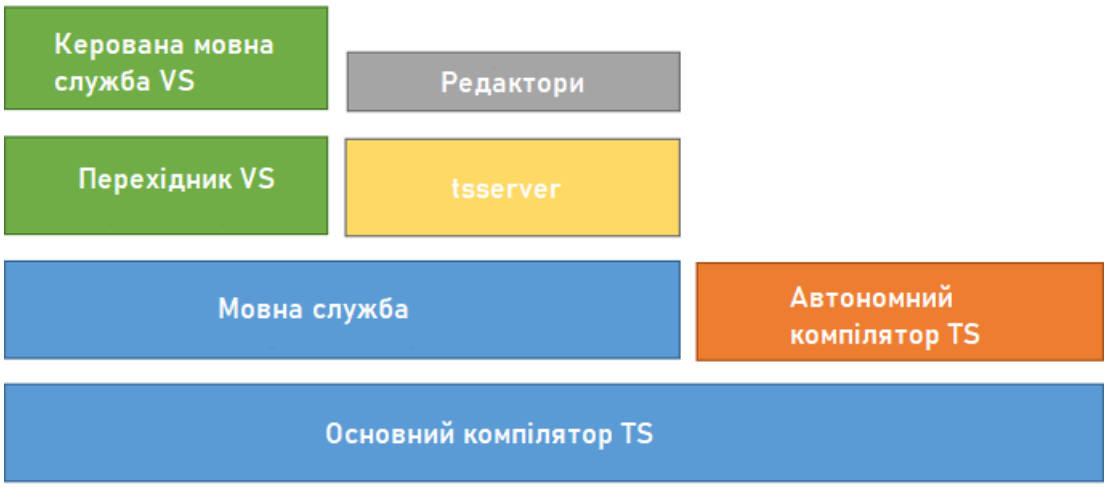

Рисунок 3.2 – Архітектура TypeScript

Код TypeScript транспілюється (перетворюється) в JavaScript, який потім може виконуватися в браузері або в Node.js. Це означає, що TypeScript є повністю сумісним з JavaScript і може використовувати його бібліотеки та фреймворки.

TypeScript широко використовується в проектах з Angular, React, Vue.js та Node.js. Його перевагами є кращий інструментальна підтримка, виявлення помилок під час компіляції, покращена читабельність коду та підвищена продуктивність розробки.

Хоча TypeScript не є обов'язковим для веб-розробки, він став популярним

вибором для великих проектів, де важливі масштабованість, підтримка та зрозумілість коду. Багато розробників вважають, що TypeScript допомагає писати більш надійний та легший для підтримки код, особливо в командних проектах.

Python [15] – це інтерпретована, об’єктно-орієнтована мова програмування високого рівня, розроблена Гвідо ван Россумом у 1991 році. Вона відрізняється простим і зрозумілим синтаксисом, що робить її легкою для вивчення та використання.

Python є мультипарадигмальною мовою, що підтримує процедурне, функціональне та об’єктно-орієнтоване програмування. Він пропонує динамічну типізацію, автоматичне керування пам’яттю та потужні стандартні бібліотеки для різноманітних задач.

Python широко використовується в різних галузях, таких як веб-розробка, аналіз даних, машинне навчання, автоматизація завдань, наукові обчислення та багато інших. Для веб-розробки популярними фреймворками Python є Django, Flask, Pyramid та FastAPI.

Архітектура Python наведена на рисунку 3.3.

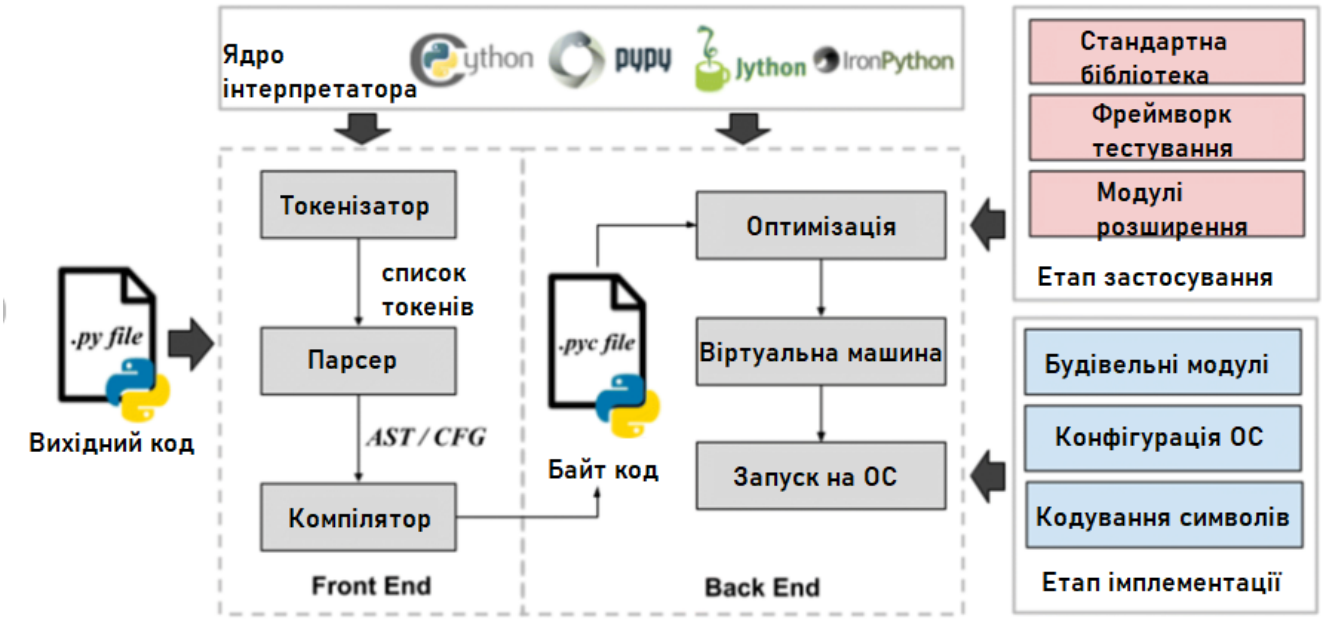

Рисунок 3.3 – Архітектура Python

Перевагами Python є його простота, читабельність, швидкість розробки,

величезна екосистема бібліотек та кросплатформеність. Однак у випадку обробки ресурсномістких задач, де необхідна висока продуктивність, Python може поступатися іншим мовам, таким як C++ або Java.

Python продовжує набирати популярність в усьому світі завдяки своїй універсальності, простоті та великій спільноті розробників [16]. Він широко використовується як у великих корпораціях, так і в стартапах, наукових установах та освітніх закладах. Завдяки своїй гнучкості та широкому спектру застосувань, Python залишається одним з найпопулярніших та затребуваних мов програмування.

Для визначення мови програмування, яка найкраще підходить для розробки серверної частини веб-системи для управління кінотеатрами, було сформовано таблицю порівняння характеристик (таблиця 3.1).

Таблиця 3.1 – Порівняння характеристик мов програмування

Характеристика	Java	JavaScript	TypeScript	Python
Кросплатформеність	+	+	+	+
Підтримка веб-фреймворків	+	+	+	+
Висока продуктивність	+/-	+	+/-	-
Об’єктно-орієнтованість	+	+/-	+	+
Зручність для швидкого прототипування	-	+	+/-	+

3.2 Розробка серверної частини модуля керування фільмами та кінотеатрами

Модуль керування фільмами та кінотеатрами включає в себе такий функціонал:

1. Додавання до системи нових кінотеатрів. Це включає в себе:

- додавання назви, фото кінотеатру та його опису;

- формування плану залів для перегляду (кількість рядів та кількість місць в кожному);

- встановлення ціни за квиток.

2. Додавання нових фільмів, показ яких заплановано в кінотеатрах. Це включає в себе: додавання детального опису фільму: назва фільму, опис, режисер, акторський склад, період показу, постер, мова показу, жанр фільму.

3. Створення сеансів показів фільмів. Це означає вибір, в якому кінотеатрі який фільм буде показуватися, а також дату і час показу.

4. Можливість купувати квитки та бронювати конкретні місця в кінозалі. Користувач обирає фільм, дату та час показу із можливих. Також, він повинен мати можливість обирати місце в залі, що йому сподобається. При цьому, мають відображатися вже зайняті місця.

5. Формування електронного листа із запрошенням на відвідування кіносеансу.

6. Формування у .pdf форматі QR-коду квитка на кіносеанс.

Для керування фільмами та кінотеатрами, користувач повинен мати статус «superadmin». Він проводить керування, використовуючи модуль клієнта, який з'єднується з модулем сервера. Для надсилання даних із сервера на модуль клієнта та навпаки, використовуються JSON-об'єкти.

Для додання нового фільму, користувач-адміністратор системи вводить необхідну про фільм інформацію, завантажує постер та підтверджує додавання даних. У разі, якщо всі дані введені правильно, тоді серверу надсилається JSON-об'єкт, як на рисунку 3.4.


```
{
    "_id": "662c0235d87a5f09e8d4df05",
    "title": "inglourious basterds",
    "language": "english",
    "duration": 153,
    "description": "here's a short description of the film \"inglourious basterds\":\ninglourious bas
    "director": "quentin tarantino",
    "cast": "brad pitt, christoph waltz, mélanie laurent",
    "releaseDate": "2024-04-26T19:31:49.989Z",
    "endDate": "2024-04-30T19:31:00.000Z",
    "genre": "historical",
    "__v": 0
}
```


Рисунок 3.4 – Заповнена форма додавання нового фільму

Він містить інформацію про назву фільму, мову, тривалість у хвилинах, опис, режисера фільму, акторський склад, жанр, дату виходу та дату завершення показу.

Блок-схему процесу додавання фільму наведено на рисунку 3.5. Цей метод addMovie() є асинхронною функцією, яка використовується для відправлення POST-запиту на сервер з метою додавання нового фільму.

Функція addMovie() приймає два аргументи: image (зображення) та newMovie (об’єкт з даними про новий фільм). Вона повертає асинхронну функцію, яка приймає dispatch в якості аргументу. Dispatch – це функція з Redux, яка використовується для відправлення дій (actions) до Redux store.

Всередині асинхронної функції виконується спроба виконати певні дії. Спочатку отримується JWT-токен з локального сховища (localStorage). Далі визначається URL-адреса /movies для відправлення запиту.

Використовується функція fetch() для відправлення POST-запиту на зазначену URL-адресу. В заголовках запиту передається JWT-токен для авторизації та вказується, що тіло запиту буде у форматі JSON. Тіло запиту містить об’єкт newMovie, який було передано як аргумент функції.

Після отримання відповіді від сервера, її тіло обробляється як JSON-об’єкт та зберігається у змінній movie. Якщо відповідь успішна (статус-код 200-299), то виконуються наступні дії:

1. Відправляється дія setAlert з Redux store для відображення повідомлення про успішне збереження фільму.
2. Якщо було передано зображення (image), викликається функція uploadMovieImage для завантаження зображення на сервер, передаючи їй ідентифікатор створеного фільму та саме зображення.
3. Відправляється дія getMovies для оновлення списку фільмів у Redux store.

Якщо під час виконання операції сталася помилка, то відправляється дія setAlert з Redux store для відображення повідомлення про помилку.

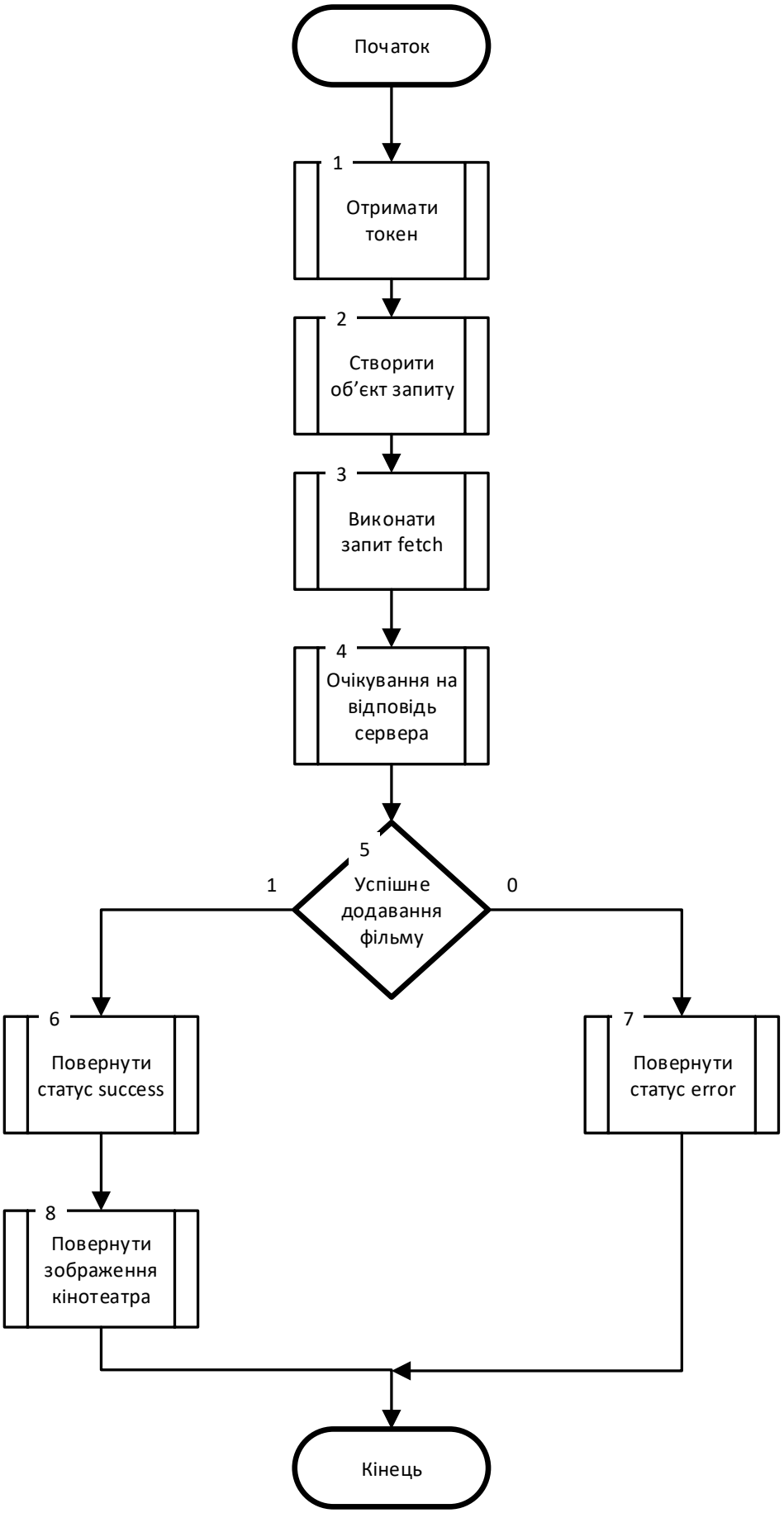

Рисунок 3.5 – Блок-схема процесу додавання фільму

Блок-схему процесу додавання кінотеатрів наведено на рисунку 3.6. Він дозволяє додавати до системи кінотеатри для управління ними і обробляє велику кількість можливих сценаріїв під час цього процесу.

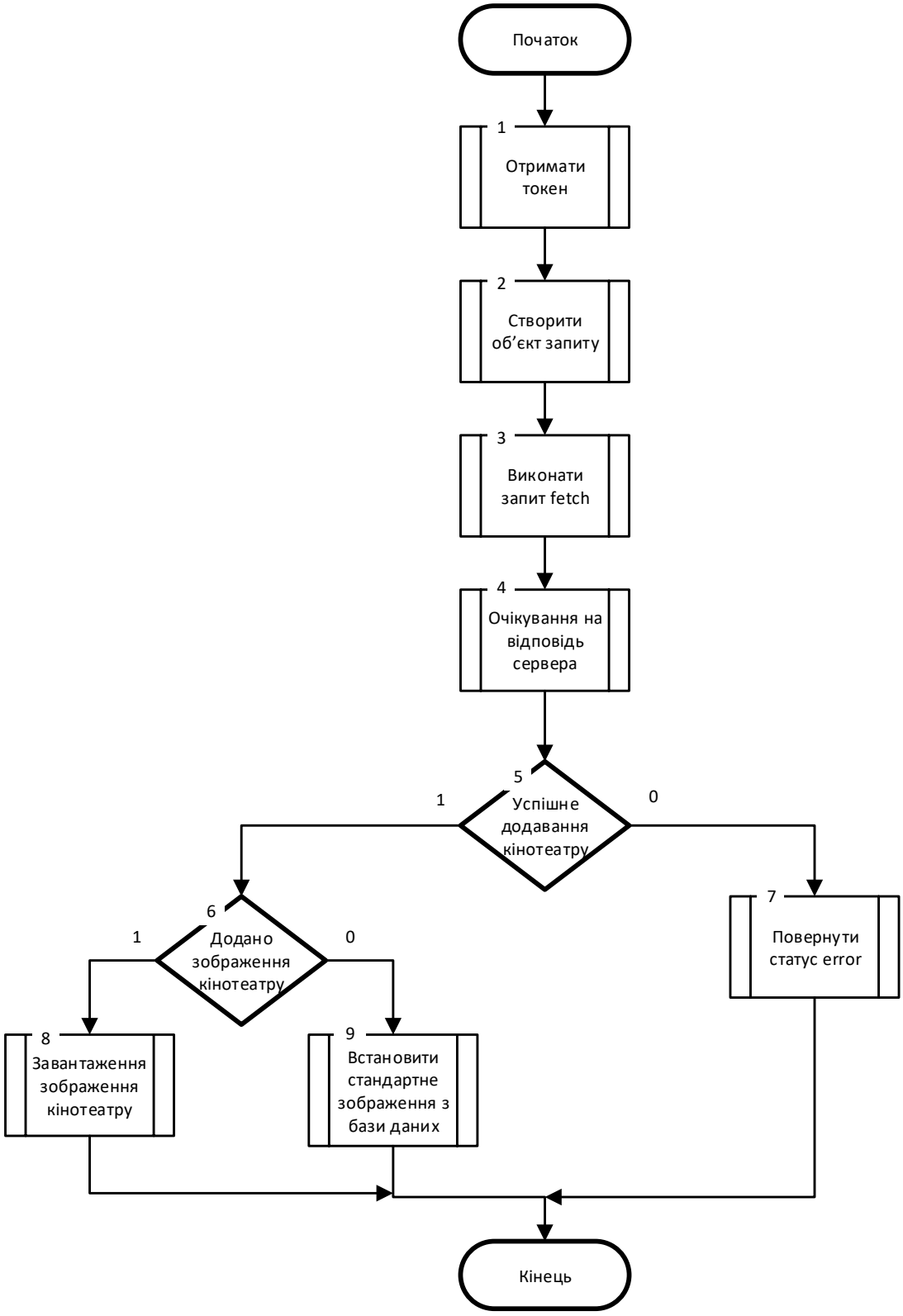

Рисунок 3.6 – Блок-схема процесу додавання кінотеатрів

Функція createCinemas приймає два аргументи: image (зображення) та newCinema (об’єкт з даними про новий кінотеатр). Вона повертає асинхронну функцію, яка приймає dispatch в якості аргументу для відправлення дій (actions) до Redux store.

Всередині асинхронної функції спочатку отримується JWT-токен з локального сховища (localStorage). Далі визначається URL-адреса /cinemas для відправлення запиту.

Використовується функція fetch для відправлення POST-запиту на зазначену URL-адресу. В заголовках запиту передається JWT-токен для авторизації та вказується, що тіло запиту буде у форматі JSON. Тіло запиту містить об’єкт newCinema, який було передано як аргумент функції.

Після отримання відповіді від сервера, її тіло парсується як JSON-об’єкт та зберігається у змінній cinema.

Якщо відповідь успішна (статус-код 200-299), виконуються наступні дії:

1. Відправляється дія setAlert з Redux store для відображення повідомлення про успішне створення кінотеатру.
2. Якщо було передано зображення (image), викликається функція uploadCinemaImage для завантаження зображення на сервер, передаючи їй ідентифікатор створеного кінотеатру та саме зображення.
3. Відправляється дія getCinemas для оновлення списку кінотеатрів у Redux store.
4. Повертається об’єкт з полями status: ‘success’ та message: ‘Cinema Created’.

Якщо під час виконання операції сталася помилка, виконуються наступні дії:

1. Відправляється дія setAlert з Redux store для відображення повідомлення про помилку.
2. Повертається об’єкт з полями status: ‘error’ та message: ‘Cinema have not been saved, try again.’.

Таким чином, ця функція дозволяє створювати новий кінотеатр на сервері, завантажувати зображення для нього (якщо воно надане) та оновлювати список кінотеатрів у клієнтському застосунку після успішної операції. У випадку помилки,

вона повертає відповідний об’єкт з повідомленням про помилку.

3.3 Розробка модуля статистики

Ведення статистики відвідування кінотеатрів у веб-системі для управління бізнес-процесами кінотеатрів є надзвичайно важливим аспектом для ефективного функціонування та розвитку бізнесу. Це дозволяє кінотеатрам вивчити поведінку відвідувачів, їхні переваги та смаки, що допомагає краще розуміти свою аудиторію та приймати більш обґрунтовані рішення щодо вибору фільмів, графіку їх показів та ціноутворення.

Крім того, ведення статистики дозволяє прогнозувати майбутні тенденції відвідуваності, що допомагає у плануванні робочого часу персоналу, управлінні квитковими касами та іншими ресурсами кінотеатру. Статистика відвідуваності також може бути використана для оцінки ефективності різних маркетингових кампаній, що допомагає визначити, які стратегії приносять найбільший результат, і оптимізувати витрати на маркетинг.

Аналізуючи статистику, кінотеатри можуть приймати рішення на основі даних, що допомагає керівництву приймати більш обґрунтовані та ефективні рішення. Крім того, статистика допомагає виявити проблемні області, такі як пікові навантаження, коли відвідувачі стикаються з тривалими чергами або нестачею місць, що допомагає покращити якість обслуговування та задоволеність клієнтів.

Важливо також зазначити, що ведення статистики необхідне для звітності перед інвесторами, партнерами та фіскальними органами. Це допомагає демонструвати прозорість бізнесу та підтримувати довіру до нього. Таким чином, ведення статистики відвідування кінотеатрів у веб-системі для управління бізнес-процесами кінотеатрів є ключовим фактором для ефективного управління бізнесом, підвищення його конкурентоспроможності та досягнення успіху на ринку.

Було розроблено статистику відвідування фільмів та статистику користувачів за пристроями. Кільцева діаграма представляє статистику користувачів системи управління бізнес-процесами кінотеатрів за типом пристроїв, з яких вони здійснюють доступ. Ця візуалізація дозволяє легко зрозуміти, які пристрої є

найпопулярнішими серед користувачів системи.

Структура кільцевої діаграми наведено на рисунку 3.7.

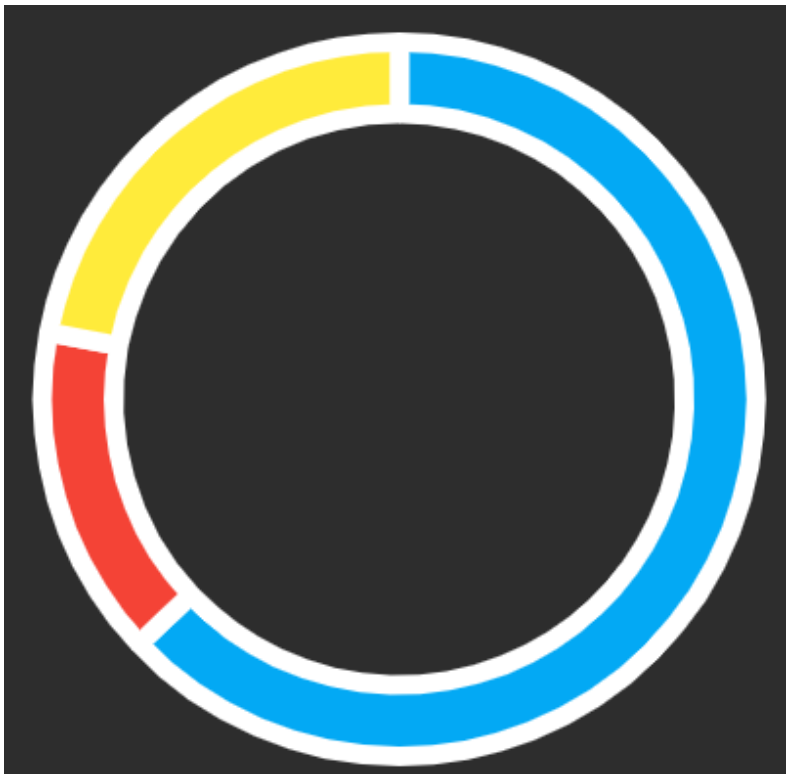

Рисунок 3.7 – Структура кільцевої діаграми

Стовпчаста діаграма відображає статистику переглядів фільмів у кінотеатрах. Кожен стовпчик представляє окремий фільм, а його висота відповідає кількості переглядів цього фільму.

Чим вищий стовпець і чим вище до вершини, тим популярніший фільм. Зі зниженням популярності фільму, стовпець стає нижчим. Популярність фільмів визначається кількістю придбаних на перегляд квитків.

Стовпчаста діаграма дає наочне уявлення про популярність різних фільмів у кінотеатрах та допомагає визначити тенденції в уподобаннях глядачів.

Структура стовпчастої діаграми наведена на рисунку 3.8.

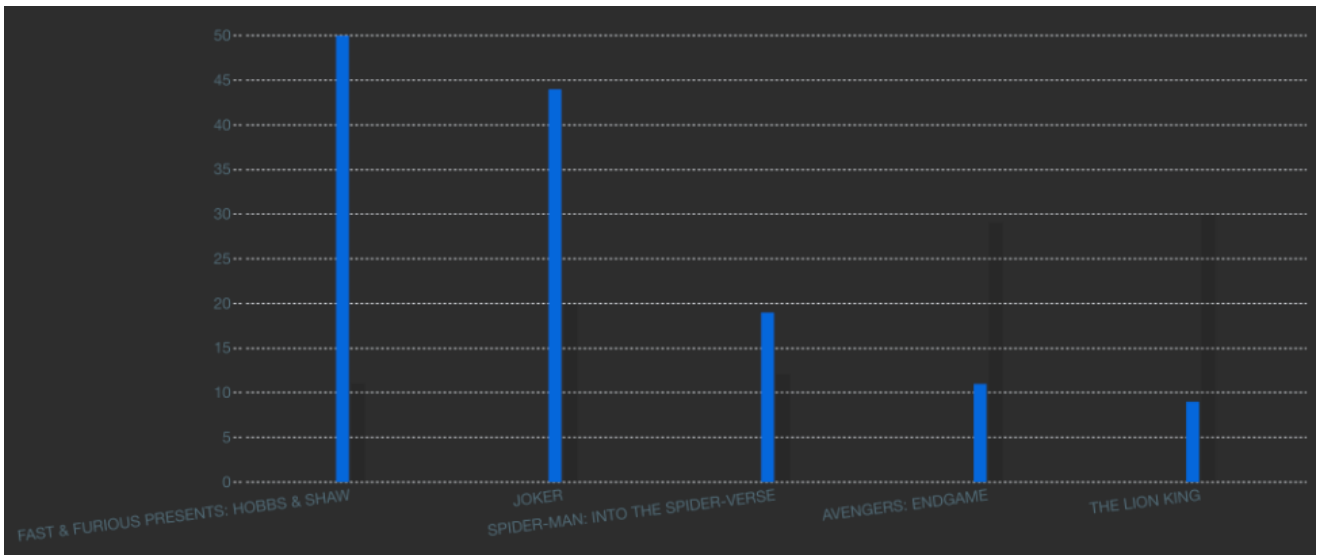

Рисунок 3.8 – Структура стовпчастої діаграми

На рисунку 3.9 наведено блок-схему процесу передачі даних для стовпчастої діаграми, що відображає популярність фільмів.

Цей метод називається `getBestMovies` і він приймає два аргументи: `reservations` (масив бронювань фільмів) та `movies` (масив об'єктів фільмів). Він також може прийняти третій необов'язковий аргумент `total`, який за замовчуванням дорівнює 5, якщо не зазначено інше.

Першим кроком є створення об'єкта `reservationCounter`, який відображає кожен ідентифікатор фільму на кількість бронювань для цього фільму. Це робиться за допомогою методів масиву `map` та `filter`.

Далі метод ініціалізує порожній масив `result` та порожній об'єкт `Map`. Потім він ітерується по об'єкту `reservationCounter`. Для кожного ідентифікатора фільму в `reservationCounter` він перевіряє, чи він вже є в `Map`. Якщо ні, він додає ідентифікатор фільму до `Map` з фіктивним значенням (`true`) і додає об'єкт, що містить ідентифікатор фільму та відповідну кількість, до масиву `result`.

Після обробки всіх ідентифікаторів фільмів масив `result` містить об'єкти з ідентифікаторами фільмів та їх відповідною кількістю. Цей масив сортується за спаданням на основі значення кількості.

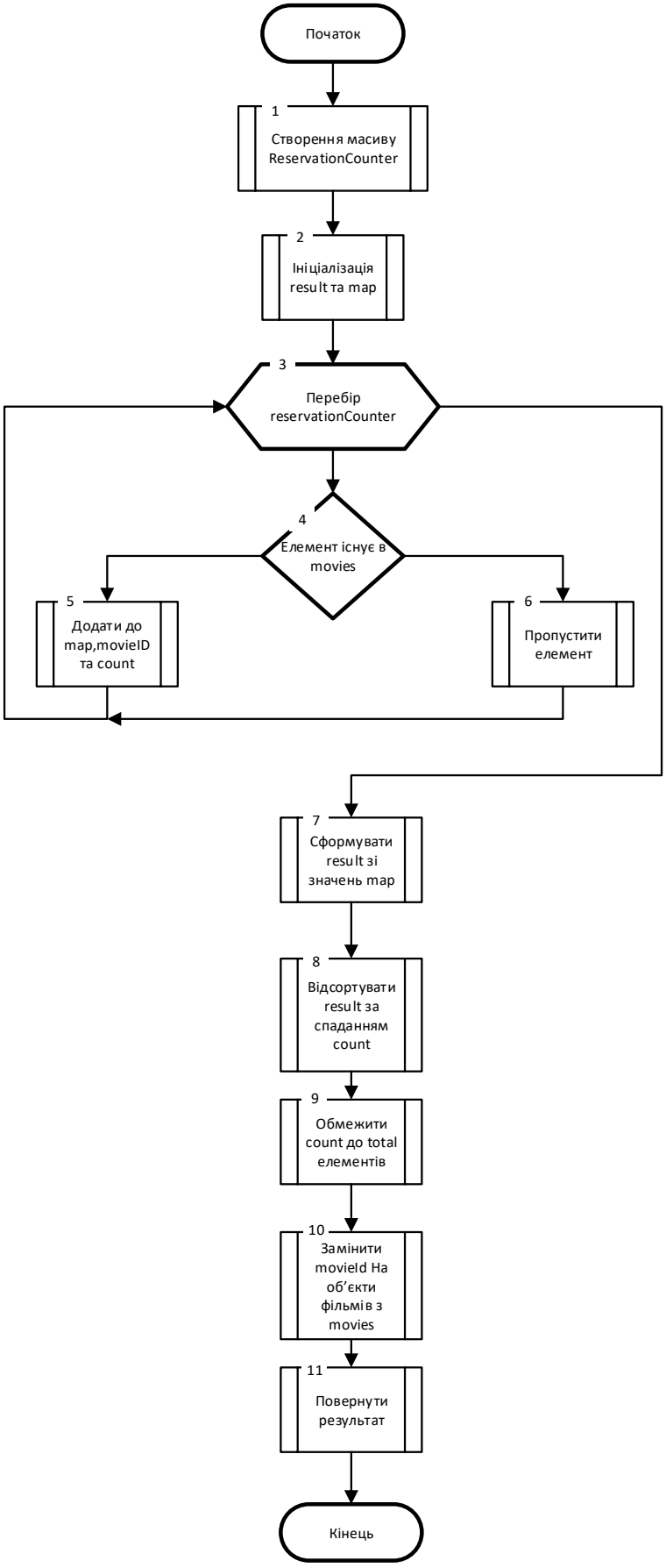

Рисунок 3.9 – Блок-схема процесу передачі даних для стовпчастої діаграми, що відображає популярність фільмів

Нарешті, метод бере перші `total` елементів з відсортованого масиву `result` за допомогою методу `slice`. Потім він відображає кожен елемент цього відрізаного масиву, замінюючи кожен ідентифікатор фільму на відповідний об’єкт фільму з масиву `movies` та повертаючи масив об’єктів з деталями фільму та кількістю.

У підсумку, цей метод знаходить топ `total` найбільш забронюваних фільмів, порахувавши бронювання для кожного фільму, відсортувавши фільми за кількістю за спаданням та повернувши масив об’єктів фільмів з відповідною кількістю бронювань.

3.4 Висновки

В цьому розділі було проведено варіатний аналіз та вибір мови програмування для розробки серверної частини веб-системи для управління бізнес-процесами кінотеатрів, розроблено серверну частину модуля керування фільмами та кінотеатрами, розроблено модуль статистики.

4 ТЕСТУВАННЯ СЕРВЕРНОЇ ЧАСТИНИ ВЕБ-СИСТЕМИ

4.1 Аналіз методів тестування

Тестування веб-систем є невід'ємною частиною розробки будь-якого веб-застосунку. Це процес, який дозволяє виявити та усунути помилки, неточності та невідповідності в роботі системи. Тестування допомагає поліпшити якість продукту, зробити його більш надійним, безпечним та зручним для користувачів [17].

Веб-системи можуть бути дуже різними: від простих сайтів-візиток до складних інтернет-магазинів та корпоративних порталів. Однак усі вони мають одну спільну рису: вони працюють в мережі Інтернет та доступні для користувачів за допомогою веб-браузера.

Види тестування наведено на рисунку 4.1.

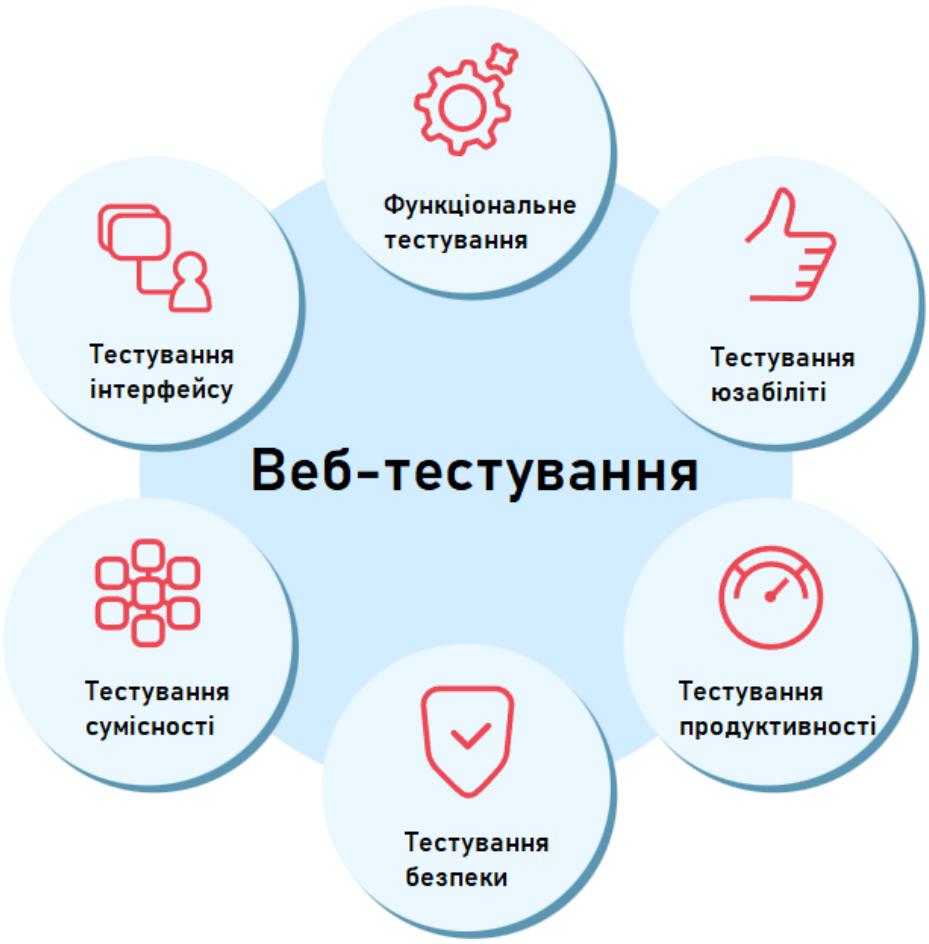

Рисунок 4.1 – Види веб-тестування

Тестування веб-систем включає в себе перевірку функціоналу, продуктивності, безпеки, сумісності, навантаження та стрес-тестування, коректності відображення на різних пристроях та в різних браузерах, а також багато інших аспектів (рисунок 4.1).

Однією з ключових складових веб-системи є її серверна частина. Це та частина системи, яка працює на сервері та відповідає за обробку даних, взаємодію з базою даних, логіку застосунку та інші серверні операції.

Тестування серверної частини веб-системи має свої особливості. На відміну від клієнтської частини, яка може бути протестована візуально, серверна частина вимагає більш глибокого розуміння внутрішньої логіки системи.

Тестування серверної частини включає в себе перевірку роботи серверних скриптів, взаємодію з базою даних, перевірку роботи API, тестування безпеки та авторизації, тестування на навантаження та стрес-тестування, а також перевірку роботи серверних служб та процесів.

Однією з ключових задач тестування серверної частини є виявлення та усунення помилок, які можуть призвести до порушення роботи системи або втрати даних. Це особливо важливо для бізнес-критичних систем, де помилка може призвести до значних фінансових або репутаційних втрат.

Тестування серверної частини також допомагає оптимізувати продуктивність системи. Під час тестування можна виявити вузькі місця та проблемні ділянки, які уповільнюють роботу системи, та вжити заходів для їх усунення.

Особливу увагу при тестуванні серверної частини потрібно приділяти безпеці. Серверна частина системи часто містить конфіденційну інформацію та відповідає за її обробку та зберігання. Тому важливо перевірити, чи є система захищеною від зовнішніх та внутрішніх загроз.

Тестування серверної частини веб-системи управління бізнес-процесами кінотеатрів має свої особливості. Ця система призначена для автоматизації роботи кінотеатрів, включаючи продаж квитків, управління розкладом сеансів, облік кінофільмів, управління персоналом та інші операції.

Веб-система управління бізнес-процесами кінотеатрів повинна працювати

швидко та безперебійно, забезпечувати точну та своєчасну обробку даних, бути зручною для користувачів та захищеною від зовнішніх та внутрішніх загроз.

Тестування серверної частини такої системи повинно включати в себе перевірку роботи серверних скриптів, взаємодію з базою даних, перевірку роботи API, тестування безпеки та авторизації, тестування на навантаження та стрес-тестування, а також перевірку роботи серверних служб та процесів.

Етапи стрес-тестування наведено на рисунку 4.2.

Рисунок 4.2 – Процес стрес-тестування

Однією з ключових задач тестування є перевірка роботи системи при великій кількості одночасних запитів. Кінотеатри часто мають пікові навантаження під час вихідних та свят, тому система повинна бути готова до обробки великої кількості даних.

Також важливо перевірити, чи працює система коректно при різних типах навантаження. Це може бути як рівномірне навантаження, так і навантаження з піками та спадами.

Важливо також перевірити, чи працює система коректно при різних типах збоїв та аварійних ситуаціях. Це може бути як збій в роботі сервера, так і збій в роботі мережі або бази даних.

Тестування серверної частини веб-системи управління бізнес-процесами кінотеатрів повинно проводитися в різних середовищах, включаючи тестове, проміжне та виробниче. Це допомагає виявити та усунути помилки на різних етапах розробки.

Для тестування серверної частини веб-системи можуть використовуватися різні інструменти та технології. Це може бути як спеціалізоване програмне забезпечення, так і ручне тестування.

Одним з популярних інструментів для тестування серверної частини веб-систем є Postman. Це програмне забезпечення дозволяє створювати та відправляти HTTP-запити, перевіряти відповіді сервера, тестувати API та виконувати інші операції.

Також для тестування серверної частини можуть використовуватися інструменти для автоматизації тестування, такі як Selenium, Jmeter, LoadRunner та інші. Ці інструменти дозволяють створювати автоматизовані тести, які можуть виконуватися в різних середовищах та з різними параметрами.

Однак автоматизовані тести не можуть повністю замінити ручне тестування. Ручне тестування дозволяє перевірити роботу системи в реальних умовах, виявити неточності та невідповідності, які не можуть бути виявлені автоматизованими тестами.

Для тестування серверної частини веб-системи управління бізнес-процесами

кінотеатрів можуть використовуватися різні підходи та методики. Це може бути як функціональне тестування, так і нефункціональне, тестування на проникнення, тестування на навантаження та стрес-тестування, тестування сумісності та інші види тестування.

Функціональне тестування дозволяє перевірити, чи виконує система свої функції коректно. Це може бути перевірка роботи серверних скриптів, взаємодії з базою даних, роботи API та інших функцій [18].

Нефункціональне тестування дозволяє перевірити нефункціональні характеристики системи, такі як продуктивність, надійність, безпека, зручність використання та інші.

Тестування на проникнення дозволяє перевірити, чи є система захищеною від зовнішніх та внутрішніх загроз. Під час тестування на проникнення можуть використовуватися різні методики, включаючи соціальну інженерію, сканування мережі, тестування веб-застосунків та інші.

Тестування на навантаження та стрес-тестування дозволяє перевірити, чи працює система коректно при великій кількості одночасних запитів та при різних типах навантаження. Це важливо для бізнес-критичних систем, де помилка може призвести до значних фінансових або репутаційних втрат.

Тестування сумісності дозволяє перевірити, чи працює система коректно в різних середовищах, з різними версіями програмного забезпечення та з різними типами пристроїв.

Тестування серверної частини веб-системи управління бізнес-процесами кінотеатрів має важливе значення для забезпечення надійності, безпеки та ефективності роботи системи. Тестування допомагає виявити та усунути помилки, неточності та невідповідності, поліпшити якість продукту та зробити його більш зручним для користувачів.

Також важливо мати чітко визначені критерії тестування, які дозволять оцінити результати тестування та зробити висновки про якість системи. Критерії тестування повинні враховувати особливості системи, її функціональні та нефункціональні характеристики, а також вимоги користувачів.

4.2 Тестування модуля сервера

Проведемо тестування серверної частини веб-системи для управління бізнес-процесами кінотеатрів з використанням Postman.

Postman [19] є популярним інструментом для тестування та розробки API. Це дозволяє користувачам надсилати HTTP-запити та переглядати відповіді, створювати та зберігати скрипти тестування, а також автоматизувати тести. Postman підтримує різні типи HTTP-запитів, такі як GET, POST, PUT, DELETE та інші, а також дозволяє користувачам додавати параметри запиту, заголовки та тіло запиту.

Однією з ключових переваг Postman є його простий та інтуїтивно зрозумілий інтерфейс. Користувачі можуть легко створювати та зберігати запити, а також організовувати їх у колекції. Також Postman дозволяє користувачам створювати та зберігати змінні, які можуть бути використані в запитах та скриптах тестування. Це дозволяє автоматизувати тести та зменшити час на їх написання.

Крім того, Postman має можливість роботи з різними середовищами, що дозволяє користувачам створювати та зберігати різні набори змінних для різних середовищ, таких як розробка, тестування та виробництво. Також Postman має можливість інтеграції з сервісами контролю версій, такими як Git, що дозволяє користувачам співпрацювати над колекціями та скриптами тестування.

Спершу, проведемо тестування, створивши тестовий запис для нового фільму. Введемо необхідні для цього дані через Postman: назва фільму, дата виходу, мова, жанр, дата завершення показу, тривалість, режисера, опис, акторський склад. Для цього, у Postman існують спеціальні поля, в яких вводяться дані у вигляді «ключ-значення», де ключем є назва поля в таблиці бази даних, а значення – дані, які ми передаємо для створення нового фільму. Для цього, виконаємо POST-метод movies за адресою http://localhost:3000/movies. Заповнення даних продемонстровано на рисунку 4.3.

	KEY	VALUE
☑	cast	zendaya, mike faist, josh o'connor
☑	description	Tashi, a former tennis prodigy turned coach, turned her h...
☑	director	luca guandagnino
☑	duration	131
☑	endDate	2024-05-02T17:39:52.471Z
☑	genre	drama
☑	language	english
☑	releaseDate	2024-04-30T17:39:52.471Z
☑	title	Challengers

Рисунок 4.3 – Заповнення даних для створення нового фільму

Результат виконання цього запиту зображено на рисунку 4.4.

Рисунок 4.4 – Результат виконання запиту на створення нового фільму

Було отримано помилку 401 Unauthorized. Це означає, що для додавання нового фільму потрібно бути авторизованим в застосунку.

Щоб авторизуватися в Postman, необхідно ввести логін та пароль у вкладці «Authorization» (рисунок 4.5).

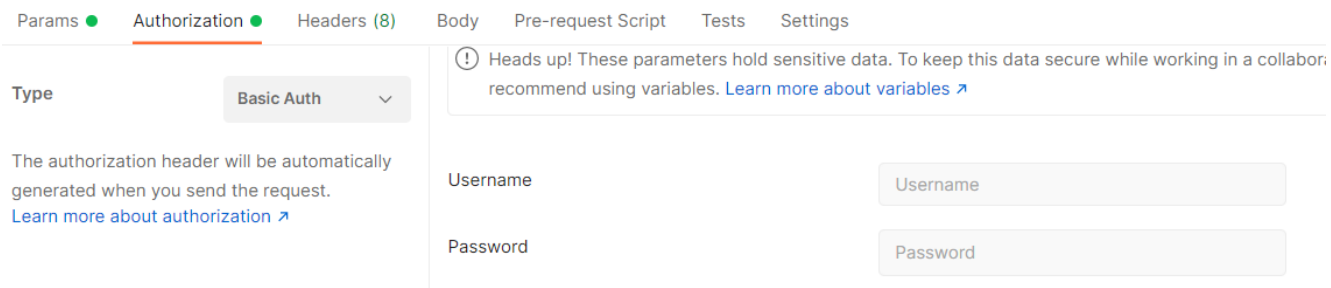

Рисунок 4.5 – Вкладка «Authorization»

Після введення логіну та пароля у відповідні поля, ще раз спробуємо додати

новий фільм з тими ж даними. Результат продемонстровано на рисунку 4.4.

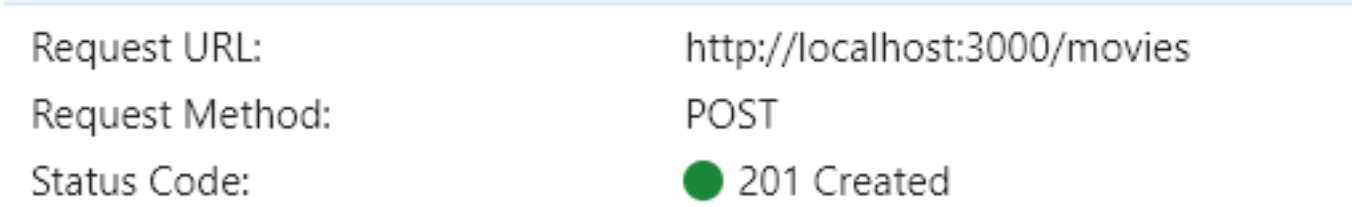

Рисунок 4.6 – Повідомлення про успішне додавання фільму

Доданий в базі даних запис про доданий фільм продемонстровано на рисунку 4.5.

Рисунок 4.6 – Доданий фільм у базі даних

Додамо до системи керування новий кінотеатр. Для цього, виконаємо POST-метод movies за адресою http://localhost:3000/movies. Заповнення даних продемонстровано на рисунку 4.7.

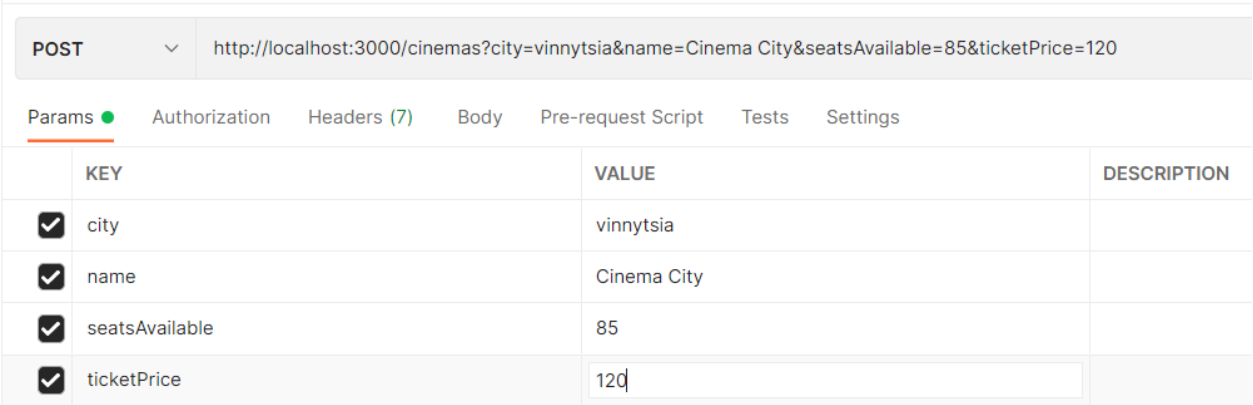

Рисунок 4.7 – Заповнення даних для додавання нового кінотеатру

Результат виконання цього запиту зображено на рисунку 4.8.

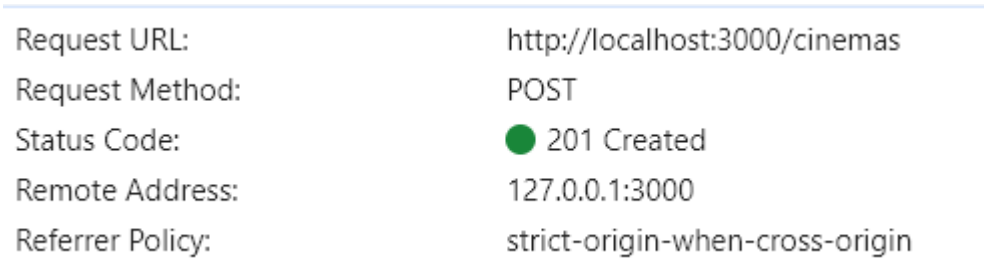

Рисунок 4.8 – Повідомлення про додавання кінотеатру

Доданий в базу даних запис про новий кінотеатр зображено на рисунку 4.9.

Рисунок 4.9 – Доданий в базу даних кінотеатр

В змінній seats міститься двовимірний масив, тобто, в усіх елементах масиву містяться масиви, що відповідають за місця в рядах. Приклад заповнення цих масивів подано на рисунку 4.10.

Рисунок 4.10 – Заповнення масиву, що відповідає за місця в кінотеатрі

Додамо новий сеанс перегляду фільму в кінотеатрі. Для цього, необхідно створити запис, як на рисунку 4.11.

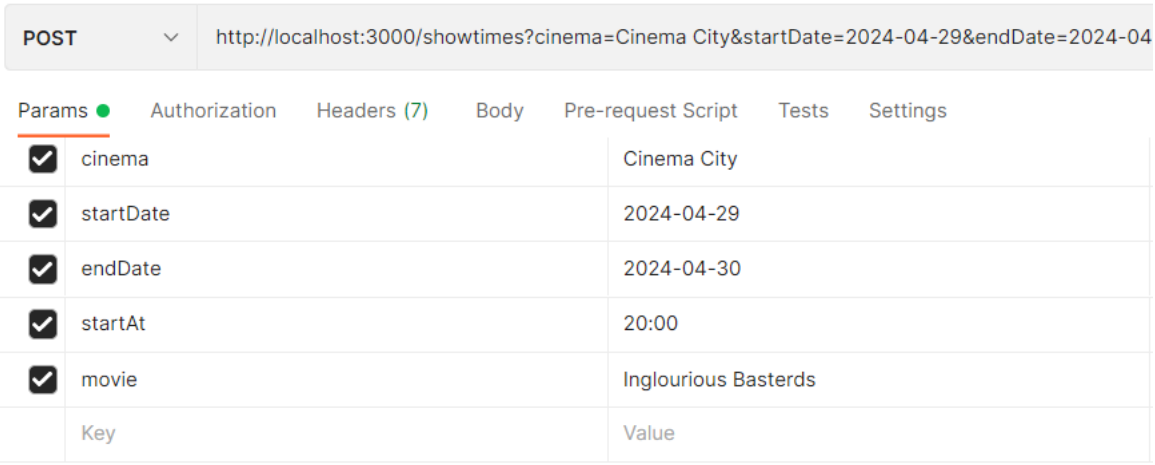

Рисунок 4.11 – Створення нового сеансу перегляду фільму

Як ми бачимо, необхідно обрати час показу, дату початку показів, дату завершення показів, фільм та кінотеатр. Результат про успішне виконання цього запиту на створення нового сеансу продемонстровано на рисунку 4.12.

Рисунок 4.12 – Успішне створення нового сеансу перегляду

Створений об’єкт в базі даних зображено на рисунку 4.13.

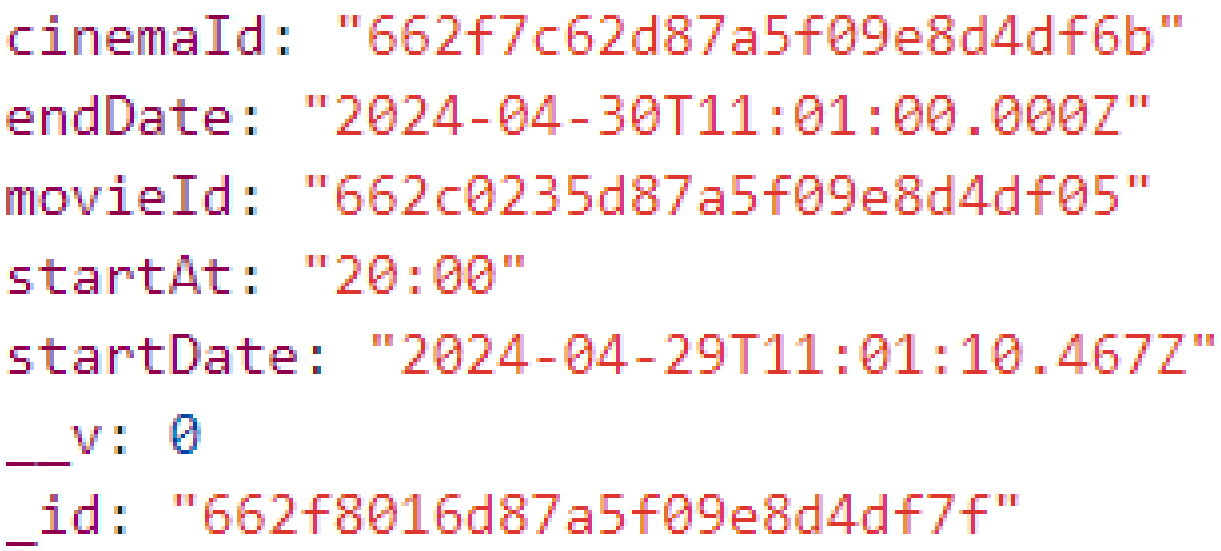

Рисунок 4.13 – Створений об’єкт сеансу перегляду фільму в базі даних

Створені сеанси перегляду фільмів зображено на рисунку 4.14.

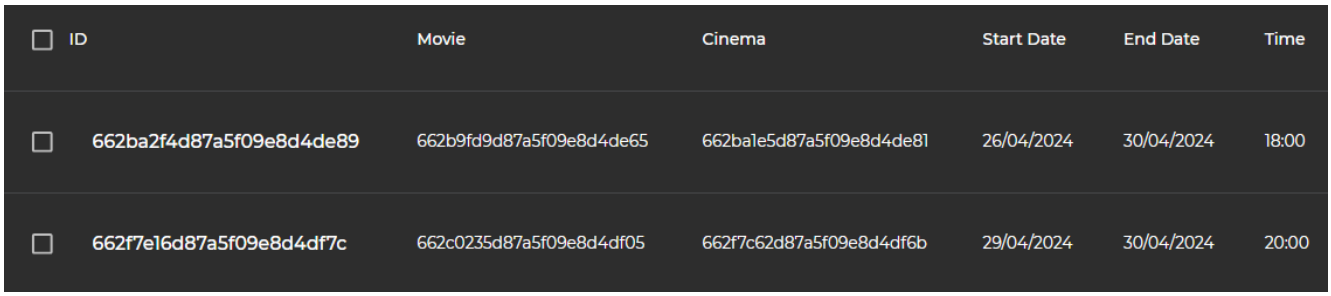

Рисунок 4.14 – Таблиця сеансів перегляду фільмів

Таблиця з придбаними квитками зображена на рисунку 4.15.

User	Phone	Start At	Movie	Cinema	Ticket Price	Total	Checkin
Oleksandr	0938748373	18:00	tenet	SmartCinema	150	150	no
Oleksandr	0938748373	18:00	tenet	SmartCinema	150	450	no

Рисунок 4.15 – Придбані квитки

Придбаємо новий квиток. Для цього, введемо дані, як на рисунку 4.16.

KEY	VALUE
movie	Inglourious Basterds
cinema	Cinema City
startDate	2024-04-29
startTime	20:00

Рисунок 4.16 – Придбання квитка на перегляд

Доданий в базу даних квиток зображено на рисунку 4.17.


```
▼ reservation: {seats: [[3, 3]], checkin: false, _id: "662f855cd87a5f09e8d4df9a", date: "2024-04-29T11:32:
    checkin: false
    cinemaId: "662f7c62d87a5f09e8d4df6b"
    date: "2024-04-29T11:32:00.000Z"
    movieId: "662c0235d87a5f09e8d4df05"
    phone: "0938748373"
  ▶ seats: [[3, 3]]
    startAt: "20:00"
    ticketPrice: 120
    total: 120
```


Рисунок 4.17 – Доданий в базу даних квиток

Зміни в таблиці з квитками зображено на рисунку 4.18.

User	Phone	Start At	Movie	Cinema	Ticket Price	Total	Checkin
Oleksandr	0938748373	20:00	inglourious basterds	Cinema City	120	120	no
Oleksandr	0938748373	18:00	tenet	SmartCinema	150	150	no
Oleksandr	0938748373	18:00	tenet	SmartCinema	150	450	no

Рисунок 4.18 – Оновлена таблиця з інформацією про продані квитки

Також, можна завантажити .pdf файл з QR-кодом, який являє собою квиток на фільм (рисунок 4.19).

Рисунок 4.19 – QR-код на фільм

4.3 Висновки

В цьому розділі проведено аналіз методів тестування загалом та зокрема серверних частин веб-систем. Обрано застосунок Postman для тестування запитів до серверної частини веб-системи. Проведено тестування системи, перевірено різні сценарії її використання, а саме: створення фільмів, створення кінотеатрів, створення сеансів перегляду фільмів, покупка квитків на перегляд фільму. Продемонстровано працездатність розробленої серверної частини та її придатність для використання.

ВИСНОВКИ

В результаті виконання бакалаврської дипломної роботи розроблено серверну частину веб-системи управління бізнес-процесами роботи кінотеатрів. Розроблена серверна частина дозволяє додавати до бази даних фільми та кінотеатри, створювати кіносеанси та продавати квитки.

Для розробки використано середовище розробки Visual Studio Code, мову програмування JavaScript, платформу Node.js. Робота оформлена з урахуванням рекомендацій [20].

У першому розділі проведено аналіз стану питання серверної частини веб-системи, порівняльний аналіз аналогів, аналіз методів розв’язання задачі. На їх основі було доведено доцільність та актуальність власної розробки, поставлено задачі дослідження.

У другому розділі розроблено структуру та модель модуля сервера, структуру бази даних, розроблено метод відправлення запрошення на відвідування фільму електронною поштою.

У третьому розділі проведено варіантний аналіз та обґрунтування вибору мови програмування розробки, розроблено серверну частину для створення фільмів та кінотеатрів у системі. Розроблено серверну частину модуля статистики

У четвертому розділі проведено аналіз методів тестування, проведено тестування розробленої серверної частини. Тестування довело працездатність та коректність роботи застосунку.

Таким чином можна зробити висновок, що поставлені задачі та цілі бакалаврської дипломної роботи виконано.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Vanessa Cristina Grabowski Aoki, Silvia Spagnol Simi dos Santos. Film analysis in management: a journey through the metaphors of the concept of leadership. Бінґлі: Emerald Publishing, 2020. 134с.

2. Майданюк В.П. Розробка програмного забезпечення для керування бізнес процесами кінотеатрів/ В.П. Майданюк, О.О. Михальнюк, А.Р. Шкляр // Матеріали конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)», Вінниця, 2024. [Електронний ресурс] – Режим доступу до ресурсу: https://conferences.vntu.edu.ua/index.php/mn/mn2024/paper/viewFile/21525/17813

3. Vista Co [Електронний ресурс] – Режим доступу до ресурсу: https://www.vista.co/

4. Cineverse [Електронний ресурс] – Режим доступу до ресурсу: https://www.cineverse.com/

5. Movio Cinema [Електронний ресурс] – Режим доступу до ресурсу: https://www.movio.co/eq

6. MVC [Електронний ресурс] – Режим доступу до ресурсу: https://brander.ua/technologies/mvc

7. Express.js [Електронний ресурс] – Режим доступу до ресурсу: https://expressjs.com/

8. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en

9. Nodemailer [Електронний ресурс] – Режим доступу до ресурсу: https://www.nodemailer.com/

10. Qrcode[Електронний ресурс] – Режим доступу до ресурсу: https://davidshimjs.github.io/qrcodejs/

11. MongoDB [Електронний ресурс] – Режим доступу до ресурсу: https://www.mongodb.com/

12. Java [Електронний ресурс] – Режим доступу до ресурсу: https://www.java.com/

13. What is JavaScript? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/javascript/

14. TypeScript [Електронний ресурс] – Режим доступу до ресурсу: https://www.typescriptlang.org/

15. Python [Електронний ресурс] – Режим доступу до ресурсу: https://www.python.org/

16. Python: The most popular language [Електронний ресурс] – Режим доступу до ресурсу: https://datascientest.com/en/python-the-most-popular-language

17. Types of Web application testing [Електронний ресурс] – Режим доступу до ресурсу: https://katalon.com/resources-center/blog/types-of-web-testing

18. Functional testing [Електронний ресурс] – Режим доступу до ресурсу: https://www.browserstack.com/guide/functional-testing

19. Postman [Електронний ресурс] – Режим доступу до ресурсу: https://www.postman.com/

20. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

59

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«_12_» _березня_ 2024 р.

Технічне завдання
на бакалаврську дипломну роботу «Розробка програмного забезпечення для керування бізнес-процесами кінотеатрів. Частина 1. Модуль сервера»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської дипломної роботи:
к.т.н., доц. В.П. Майданюк
«_12_» _березня_ 2024 року.

Виконав:
студент гр. ЗПІ-20б Михальнюк О.О.
«_12_» _березня_ 2024 року.

м. Вінниця – 2024 рік

1. Найменування та галузь застосування

Бакалаврська дипломна робота: «Розробка програмного забезпечення для керування бізнес-процесами кінотеатрів. Частина 1. Модуль сервера».

Галузь застосування – веб-технології.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №80 від 11 березня 2024 року ректора по ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою роботи є полегшення керування бізнес-процесами кінотеатрів і оптимізація управління різноманітними аспектами кінотеатрального бізнесу.

Основними задачами дослідження є:

- розробити метод відправлення запрошення на кіносеанс;
- розробити серверну частину модуля управління кінотеатрами та фільмами;
- розробити функціонал для формування статистики перегляду фільмів;
- розробити серверну частину модуля покупки квитків та функціонал для генерації QR-коду на фільм;
- провести тестування розробленого функціоналу.

4. Вихідні дані для проведення НДР

1. Vanessa Cristina Grabowski Aoki, Silvia Spagnol Simi dos Santos. Film analysis in management: a journey through the metaphors of the concept of leadership. Бінґлі: Emerald Publishing, 2020. 134с.

2. MVC [Електронний ресурс] – Режим доступу до ресурсу: https://brander.ua/technologies/mvc

3. Node.js [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/en

4. What is JavaScript? [Електронний ресурс] – Режим доступу до ресурсу: https://aws.amazon.com/what-is/javascript/

5. Технічні вимоги

З боку сервера: Комп'ютер з 64-розрядним (x64) процесором та тактовою частотою 2 ГГц. Об'єм оперативної пам'яті 8 ГБ, розмір жорсткого диску 256 ГБ. Операційна система Linux. З клієнтського боку: мобільний пристрій чи ПК з будь-якою операційною системою та будь-який браузер.

6. Конструктивні вимоги.

Мобільний застосунок та веб-клієнт повинні відповідати ергономічним та технічним вимогам. Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

- пояснювальна записка до бакалаврської дипломної роботи;
- технічне завдання;
- лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи
1	Аналіз стану питання розробки модуля сервера	14.03.24 – 24.03.24
2	Розробка структури модуля сервера	24.03.24 – 06.04.24

3	Варіантний аналіз та вибір мови програмування	06.04.24 – 11.04.24
4	Розробка серверної частини модуля керування фільмами та кінотеатрами	12.04.24 – 29.04.24
5	Тестування модуля сервера	29.04.24 – 11.05.24
6	Оформлення матеріалів до захисту БДР	12.05.24– 30.05.24

10. Порядок контролю та прийняття.

Всі етапи бакалаврської дипломної роботи контролюються науковим керівником згідно плану по виконанню роботи. Прийняття бакалаврської дипломної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком захисту.

63

Додаток Б. Протокол перевірки бакалаврської дипломної роботи на наявність текстових запозичень

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Розробка програмного забезпечення для керування бізнес процесами кінотеатрів. Частина 1. Модуль сервера

Тип роботи: БДР

Підрозділ : кафедра програмного забезпечення, ФІТКІ

Науковий керівник: к.т.н., доц. каф. ПЗ Майданюк В.П.

Оригінальність	94.0 %
Схожість	6.0 %

Аналіз звіту подібності

■ **Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.**

□ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г. О.

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Михальнюк О.О.

Керівник роботи _____ Майданюк В.П.

Додаток В. Лістинг програми

Текст коду модуля сервера програмного забезпечення для керування бізнес процесами кінотеатрів

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

const { Schema } = mongoose;

const cinemaSchema = new Schema({
  name: {
    type: String,
    required: true,
    trim: true,
  },
  ticketPrice: {
    type: Number,
    required: true,
  },
  city: {
    type: String,
    required: true,
    trim: true,
    lowercase: true,
  },
  seats: {
    type: [Schema.Types.Mixed],
    required: true,
  },
  seatsAvailable: {
```

```
    type: Number,
    required: true,
  },
  image: {
    type: String,
  },
});

const Cinema = mongoose.model('Cinema', cinemaSchema);

module.exports = Cinema;

const mongoose = require('mongoose');

const { Schema } = mongoose;
const movieSchema = new Schema({
  title: {
    type: String,
    required: true,
    trim: true,
    lowercase: true,
  },
  image: {
    type: String,
  },
  language: {
    type: String,
    required: true,
    trim: true,
    lowercase: true,
```

```
},
genre: {
  type: String,
  required: true,
  trim: true,
  lowercase: true,
},
director: {
  type: String,
  required: true,
  trim: true,
  lowercase: true,
},
cast: {
  type: String,
  required: true,
  trim: true,
  lowercase: true,
},
description: {
  type: String,
  required: true,
  trim: true,
  lowercase: true,
},
duration: {
  type: Number,
  required: true,
},
releaseDate: {
```

```
    type: Date,
    required: true,
  },
  endDate: {
    type: Date,
    required: true,
  },
});

const Movie = mongoose.model('Movie', movieSchema);

module.exports = Movie;

const express = require('express');
const auth = require('../middlewares/auth');
const upload = require('../utils/multer');
const Cinema = require('../models/cinema');
const userModeling = require('../utils/userModeling');

const router = new express.Router();

// Create a cinema
router.post('/cinemas', auth.enhance, async (req, res) => {
  const cinema = new Cinema(req.body);
  try {
    await cinema.save();
    res.status(201).send(cinema);
  } catch (e) {
    res.status(400).send(e);
  }
```

```
});

router.post('/cinemas/photo/:id', upload('cinemas').single('file'), async (req, res, next) => {
  const url = `${req.protocol}://${req.get('host')}`;
  const { file } = req;
  const movieId = req.params.id;
  try {
    if (!file) {
      const error = new Error('Please upload a file');
      error.httpStatusCode = 400;
      return next(error);
    }
    const cinema = await Cinema.findById(movieId);
    if (!cinema) return res.sendStatus(404);
    cinema.image = `${url}/${file.path}`;
    await cinema.save();
    res.send({ cinema, file });
  } catch (e) {
    console.log(e);
    res.sendStatus(400).send(e);
  }
});

// Get all cinemas
router.get('/cinemas', async (req, res) => {
  try {
    const cinemas = await Cinema.find({});
    res.send(cinemas);
  } catch (e) {
```

```
    res.status(400).send(e);
  }
});

// Get cinema by id
router.get('/cinemas/:id', async (req, res) => {
  const _id = req.params.id;
  try {
    const cinema = await Cinema.findById(_id);
    if (!cinema) return res.sendStatus(404);
    return res.send(cinema);
  } catch (e) {
    return res.status(400).send(e);
  }
});

// Update cinema by id
router.patch('/cinemas/:id', auth.enhance, async (req, res) => {
  const _id = req.params.id;
  const updates = Object.keys(req.body);
  const allowedUpdates = ['name', 'ticketPrice', 'city', 'seats', 'seatsAvailable'];
  const isValidOperation = updates.every((update) => allowedUpdates.includes(update));

  if (!isValidOperation) return res.status(400).send({ error: 'Invalid updates!' });

  try {
    const cinema = await Cinema.findById(_id);
    updates.forEach((update) => (cinema[update] = req.body[update]));
    await cinema.save();
```

```
    if (!cinema) return res.sendStatus(404);
    return res.send(cinema);
  } catch (e) {
    return res.status(400).send(e);
  }
});

// Delete cinema by id
router.delete('/cinemas/:id', auth.enhance, async (req, res) => {
  const _id = req.params.id;
  try {
    const cinema = await Cinema.findByIdAndDelete(_id);
    if (!cinema) return res.sendStatus(404);
    return res.send(cinema);
  } catch (e) {
    return res.sendStatus(400);
  }
});

// Cinema User modeling (GET ALL CINEMAS)
router.get('/cinemas/usermodeling/:username', async (req, res) => {
  const { username } = req.params;
  try {
    const cinemas = await Cinema.find({});
    const cinemasUserModeled = await
userModeling.cinemaUserModeling(cinemas, username);
    res.send(cinemasUserModeled);
  } catch (e) {
    res.status(400).send(e);
  }
```

```
});

module.exports = router;

import { GET_RESERVATIONS, GET_RESERVATION_SUGGESTED_SEATS } from '../types';
import { setAlert } from './alert';

export const getReservations = () => async dispatch => {
 try {
  const token = localStorage.getItem('jwtToken');
  const url = '/reservations';
  const response = await fetch(url, {
   method: 'GET',
   headers: {
    Authorization: `Bearer ${token}`
   }
  });
  const reservations = await response.json();
  if (response.ok) {
   dispatch({ type: GET_RESERVATIONS, payload: reservations });
  }
 } catch (error) {
  dispatch(setAlert(error.message, 'error', 5000));
 }
};

export const getSuggestedReservationSeats = username => async dispatch => {
 try {
  const token = localStorage.getItem('jwtToken');
```

```
    const url = '/reservations/usermodeling/' + username;
    const response = await fetch(url, {
      method: 'GET',
      headers: {
        Authorization: `Bearer ${token}`
      }
    });
    const reservationSeats = await response.json();
    if (response.ok) {
      dispatch({
        type: GET_RESERVATION_SUGGESTED_SEATS,
        payload: reservationSeats
      });
    }
  } catch (error) {
    dispatch(setAlert(error.message, 'error', 5000));
  }
};

export const addReservation = reservation => async dispatch => {
  try {
    const token = localStorage.getItem('jwtToken');
    const url = '/reservations';
    const response = await fetch(url, {
      method: 'POST',
      headers: {
        Authorization: `Bearer ${token}`,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(reservation)
```

```
    });
    if (response.ok) {
      const { reservation, QRCode } = await response.json();
      dispatch(setAlert('Reservation Created', 'success', 5000));
      return {
        status: 'success',
        message: 'Reservation Created',
        data: { reservation, QRCode }
      };
    }
  } catch (error) {
    dispatch(setAlert(error.message, 'error', 5000));
    return {
      status: 'error',
      message: ' Reservation have not been created, try again.'
    };
  }
};

export const updateReservation = (reservation, id) => async dispatch => {
  try {
    const token = localStorage.getItem('jwtToken');
    const url = '/reservations/' + id;
    const response = await fetch(url, {
      method: 'PATCH',
      headers: {
        Authorization: `Bearer ${token}`,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(reservation)
```

```
    });
    if (response.ok) {
      dispatch(setAlert('Reservation Updated', 'success', 5000));
      return { status: 'success', message: 'Reservation Updated' };
    }
  } catch (error) {
    dispatch(setAlert(error.message, 'error', 5000));
    return {
      status: 'error',
      message: ' Reservation have not been updated, try again.'
    };
  }
};

export const removeReservation = id => async dispatch => {
  try {
    const token = localStorage.getItem('jwtToken');
    const url = '/reservations/' + id;
    const response = await fetch(url, {
      method: 'DELETE',
      headers: {
        Authorization: `Bearer ${token}`,
        'Content-Type': 'application/json'
      }
    });
    if (response.ok) {
      dispatch(setAlert('Reservation Deleted', 'success', 5000));
      return { status: 'success', message: 'Reservation Removed' };
    }
  } catch (error) {
```

```
      dispatch(setAlert(error.message, 'error', 5000));
      return {
        status: 'error',
        message: ' Reservation have not been deleted, try again.'
      };
    }
  };

  import {
    TOGGLE_DIALOG,
    SELECT_SHOWTIMES,
    SELECT_ALL_SHOWTIMES,
    GET_SHOWTIMES,
    DELETE_SHOWTIME
  } from '../types';
  import { setAlert } from './alert';

  export const toggleDialog = () => ({ type: TOGGLE_DIALOG });

  export const selectShowtime = showtime => ({
    type: SELECT_SHOWTIMES,
    payload: showtime
  });

  export const selectAllShowtimes = () => ({ type: SELECT_ALL_SHOWTIMES });

  export const getShowtimes = () => async dispatch => {
    try {
      const token = localStorage.getItem('jwtToken');
```

```
    const url = '/showtimes';
    const response = await fetch(url, {
      method: 'GET',
      headers: {
        Authorization: `Bearer ${token}`
      }
    });
    const showtimes = await response.json();
    if (response.ok) {
      dispatch({ type: GET_SHOWTIMES, payload: showtimes });
    }
  } catch (error) {
    dispatch(setAlert(error.message, 'error', 5000));
  }
};

export const addShowtime = showtime => async dispatch => {
  try {
    const token = localStorage.getItem('jwtToken');
    const url = '/showtimes/';
    const response = await fetch(url, {
      method: 'POST',
      headers: {
        Authorization: `Bearer ${token}`,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(showtime)
    });
    if (response.ok) {
      dispatch(setAlert('Showtime Created', 'success', 5000));
```

```
      return { status: 'success', message: 'Showtime Created' };
    }
  } catch (error) {
    dispatch(setAlert(error.message, 'error', 5000));
    return {
      status: 'error',
      message: ' Cinema have not been saved, try again.'
    };
  }
};

export const updateShowtime = (showtime, id) => async dispatch => {
  try {
    const token = localStorage.getItem('jwtToken');
    const url = '/showtimes/' + id;
    const response = await fetch(url, {
      method: 'PATCH',
      headers: {
        Authorization: `Bearer ${token}`,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(showtime)
    });
    if (response.ok) {
      dispatch(setAlert('Showtime Created', 'success', 5000));
      return { status: 'success', message: 'Showtime Created' };
    }
  } catch (error) {
    dispatch(setAlert(error.message, 'error', 5000));
    return {
```

```
      status: 'error',
      message: ' Cinema have not been saved, try again.'
    };
  }
};

export const deleteShowtime = id => async dispatch => {
  try {
    const token = localStorage.getItem('jwtToken');
    const url = '/showtimes/' + id;
    const response = await fetch(url, {
      method: 'DELETE',
      headers: {
        Authorization: `Bearer ${token}`,
        'Content-Type': 'application/json'
      }
    });
    if (response.ok) {
      dispatch({ type: DELETE_SHOWTIME, payload: id });
      dispatch(setAlert('Showtime Deleted', 'success', 5000));
      dispatch(getShowtimes());
      return { status: 'success', message: 'Showtime Removed' };
    }
  } catch (error) {
    dispatch(setAlert(error.message, 'error', 5000));
    return {
      status: 'error',
      message: ' Showtime have not been deleted, try again.'
    };
  }
```

```
};

import {
  GET_USERS,
  ADD_USER,
  UPDATE_USER,
  DELETE_USER,
  TOGGLE_USER_DIALOG,
  SELECT_USER,
  SELECT_ALL_USERS
} from '../types';

import { setAlert } from './alert';

export const toggleUserDialog = () => ({ type: TOGGLE_USER_DIALOG });

export const selectUser = user => ({
  type: SELECT_USER,
  payload: user
});

export const selectAllUsers = () => ({ type: SELECT_ALL_USERS });

export const getUsers = () => async dispatch => {
  try {
    const token = localStorage.getItem('jwtToken');
    const url = '/users';
    const response = await fetch(url, {
      method: 'GET',
      headers: {
```

```
      Authorization: `Bearer ${token}`
    }
  });
  const users = await response.json();
  if (response.ok) {
    dispatch({ type: GET_USERS, payload: users });
  }
 } catch (error) {
  dispatch(setAlert(error.message, 'error', 5000));
 }
};

export const addUser = user => async dispatch => {
 try {
  const token = localStorage.getItem('jwtToken');
  const url = '/users/';
  const response = await fetch(url, {
    method: 'POST',
    headers: {
      Authorization: `Bearer ${token}`,
      'Content-Type': 'application/json'
    },
    body: JSON.stringify(user)
  });
  const data = await response.json();
  const newUser = data.user;
  if (response.ok) {
    dispatch(setAlert('User Created', 'success', 5000));
    dispatch({ type: ADD_USER, payload: newUser });
    return { status: 'success', message: 'User Created' };
```

```
    } else {
      throw new Error(data._message);
    }
  } catch (error) {
    dispatch(setAlert(error.message, 'error', 5000));
    return {
      status: 'error',
      message: ' User have not been saved, try again.'
    };
  }
};

export const updateUser = (user, id) => async dispatch => {
  try {
    const token = localStorage.getItem('jwtToken');
    const url = '/users/' + id;
    const response = await fetch(url, {
      method: 'PATCH',
      headers: {
        Authorization: `Bearer ${token}`,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(user)
    });
    const data = await response.json();
    const newUser = data.user;
    if (response.ok) {
      dispatch(setAlert('User Updated', 'success', 5000));
      dispatch({ type: UPDATE_USER, payload: newUser });
      return { status: 'success', message: 'User Updated' };
```

```
    } else {
      throw new Error(data._message);
    }
  } catch (error) {
    dispatch(setAlert(error.message, 'error', 5000));
    return {
      status: 'error',
      message: ' User have not been saved, try again.'
    };
  }
};

export const deleteUser = id => async dispatch => {
  try {
    const token = localStorage.getItem('jwtToken');
    const url = '/users/' + id;
    const response = await fetch(url, {
      method: 'DELETE',
      headers: {
        Authorization: `Bearer ${token}`,
        'Content-Type': 'application/json'
      }
    });
    const data = await response.json();
    if (response.ok) {
      dispatch(setAlert('User Deleted', 'success', 5000));
      dispatch({ type: DELETE_USER, payload: id });
      return { status: 'success', message: 'User Removed' };
    } else {
      throw new Error(data._message);
```

```
    }
  } catch (error) {
    dispatch(setAlert(error.message, 'error', 5000));
    return {
      status: 'error',
      message: ' User have not been deleted, try again.'
    };
  }
};
```

Додаток Г. Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ КЕРУВАННЯ БІЗНЕС ПРОЦЕСАМИ КІНОТЕАТРІВ. ЧАСТИНА 1. МОДУЛЬ СЕРВЕРА

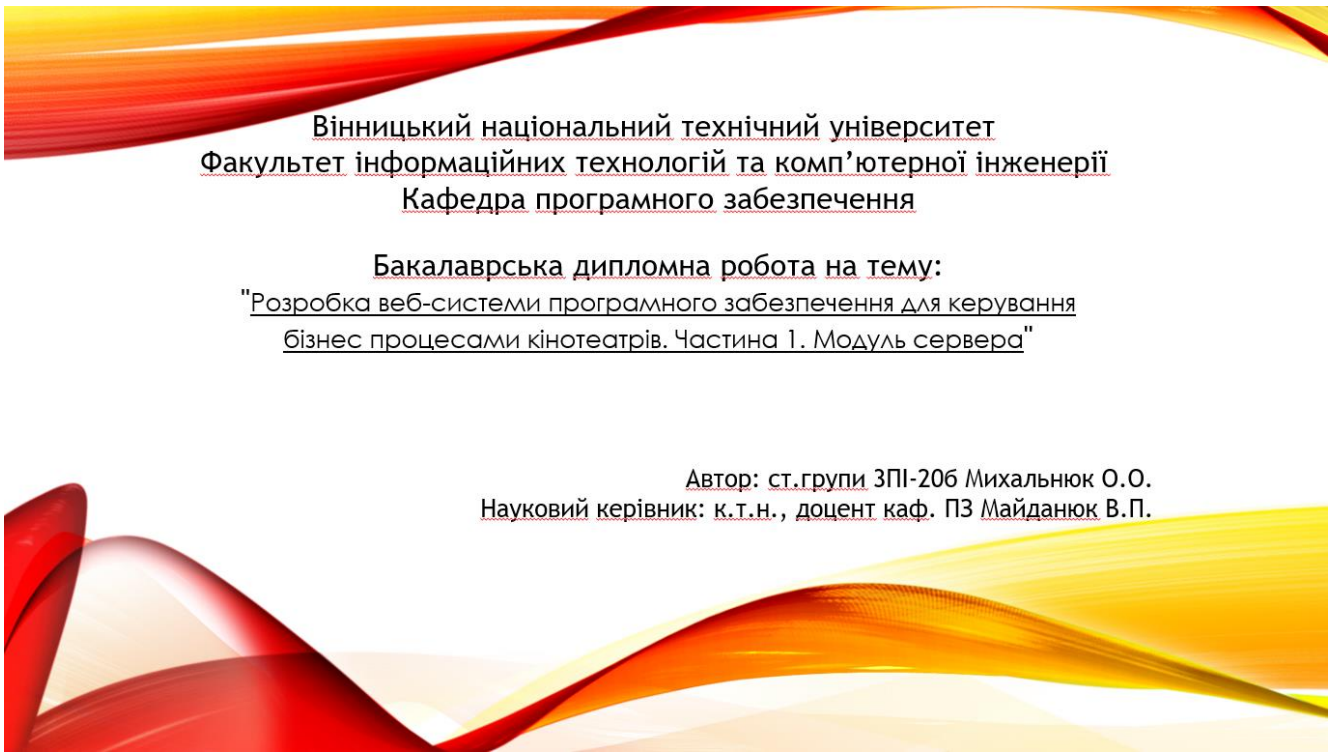

Рисунок Г.1 – Титульний слайд

- Веб-системи для управління роботи бізнес-процесів кінотеатрів є невід'ємною частиною сучасної індустрії розваг. Вони дозволяють автоматизувати та спрощувати різні операції, пов'язані з керуванням кінотеатром, від продажу квитків до управління розкладом фільмів.
- Однією з ключових функцій таких систем є онлайн-продаж квитків. Це дозволяє глядачам придбати квитки на свій улюблений фільм в будь-який час і з будь-якого місця, без необхідності стояти в черзі в касі кінотеатру. Крім того, веб-системи зазвичай надають можливість обрати конкретні місця в залі, що робить перегляд фільму більш комфортним.
- Іншою важливою функцією веб-систем для кінотеатрів є управління розкладом фільмів. За допомогою таких систем, адміністратори кінотеатрів можуть легко планувати покази фільмів, враховуючи часові рамки, доступність залів та інші фактори. Це допомагає уникнути конфліктів у розкладі та забезпечує ефективне використання ресурсів кінотеатру.
- Крім того, веб-системи для управління роботи бізнес-процесів кінотеатрів часто включають в себе модулі для управління персоналом, бухгалтерського обліку, маркетингу та інші корисні інструменти. Це дозволяє керівництву кінотеатру ефективно контролювати всі аспекти роботи бізнесу, приймати обґрунтовані рішення та підвищувати ефективність роботи.

Рисунок Г.2 – Актуальність теми

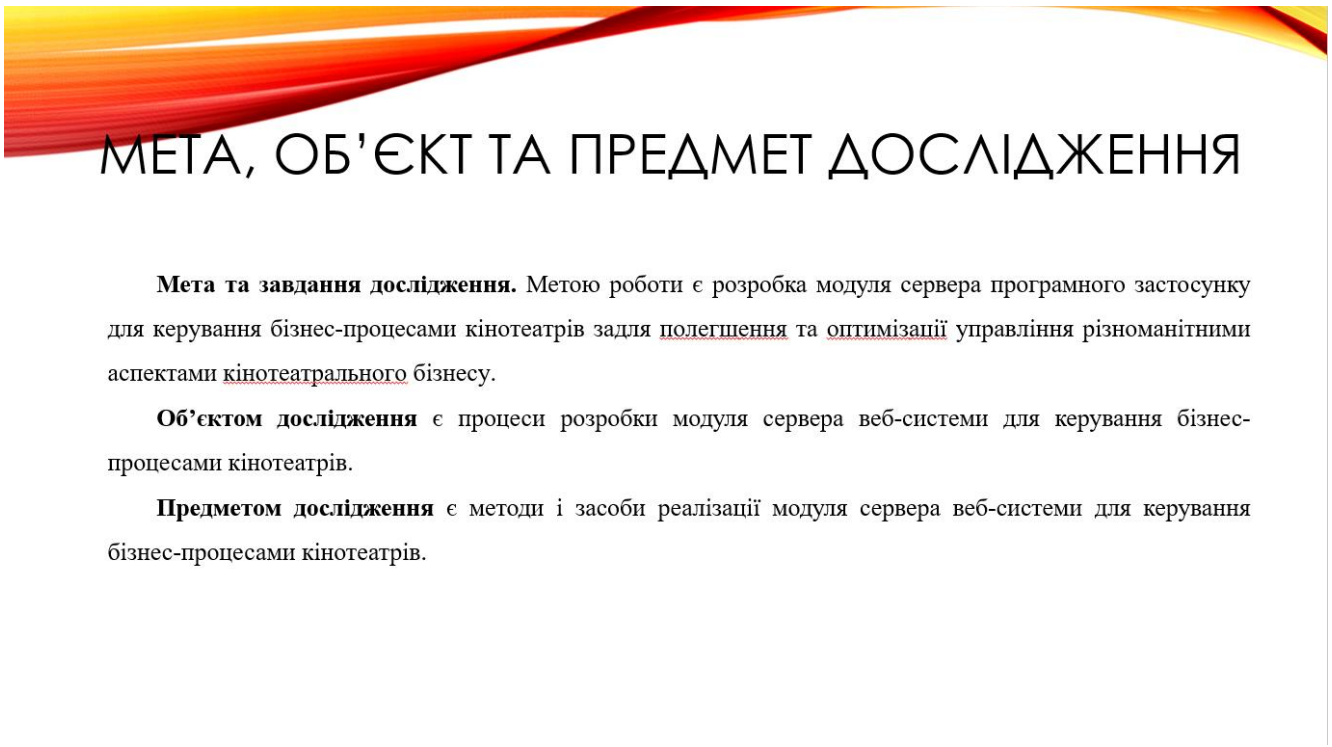

Рисунок Г.3 – Мета, об'єкт та предмет дослідження

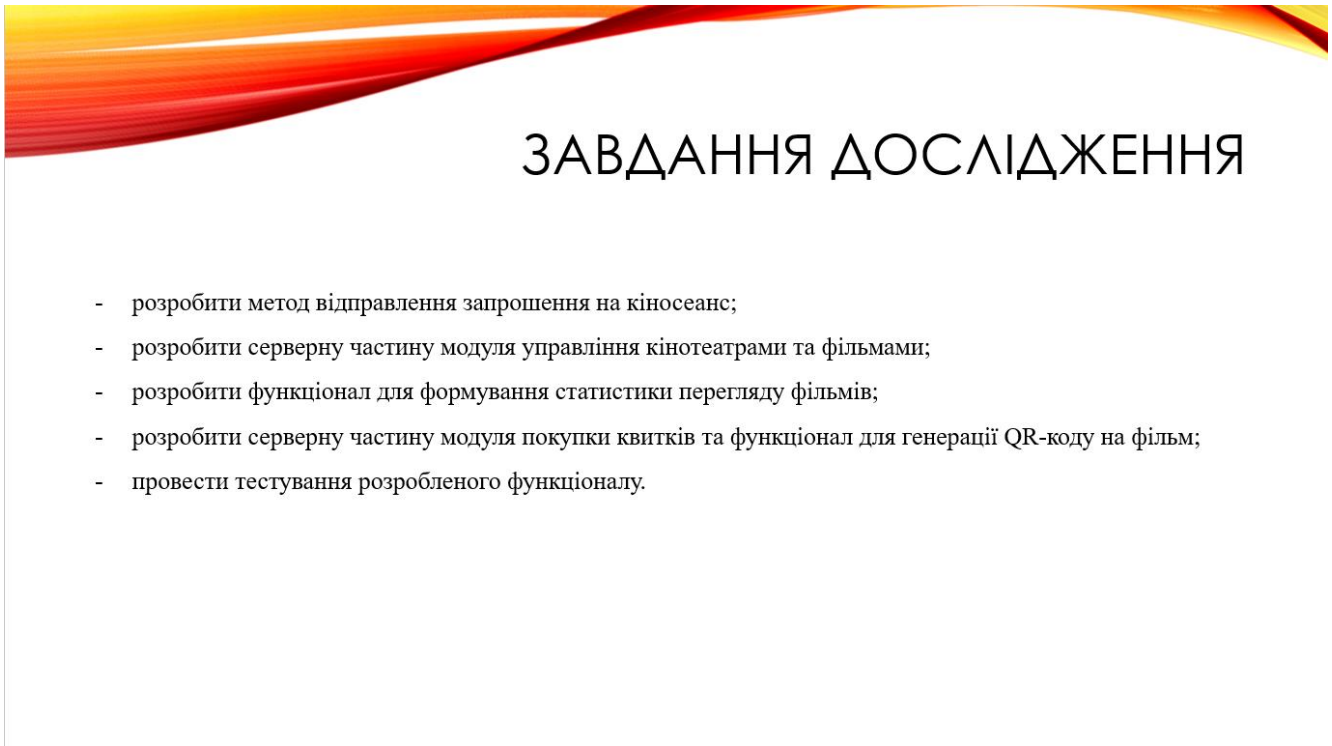

Рисунок Г.4 – Завдання дослідження

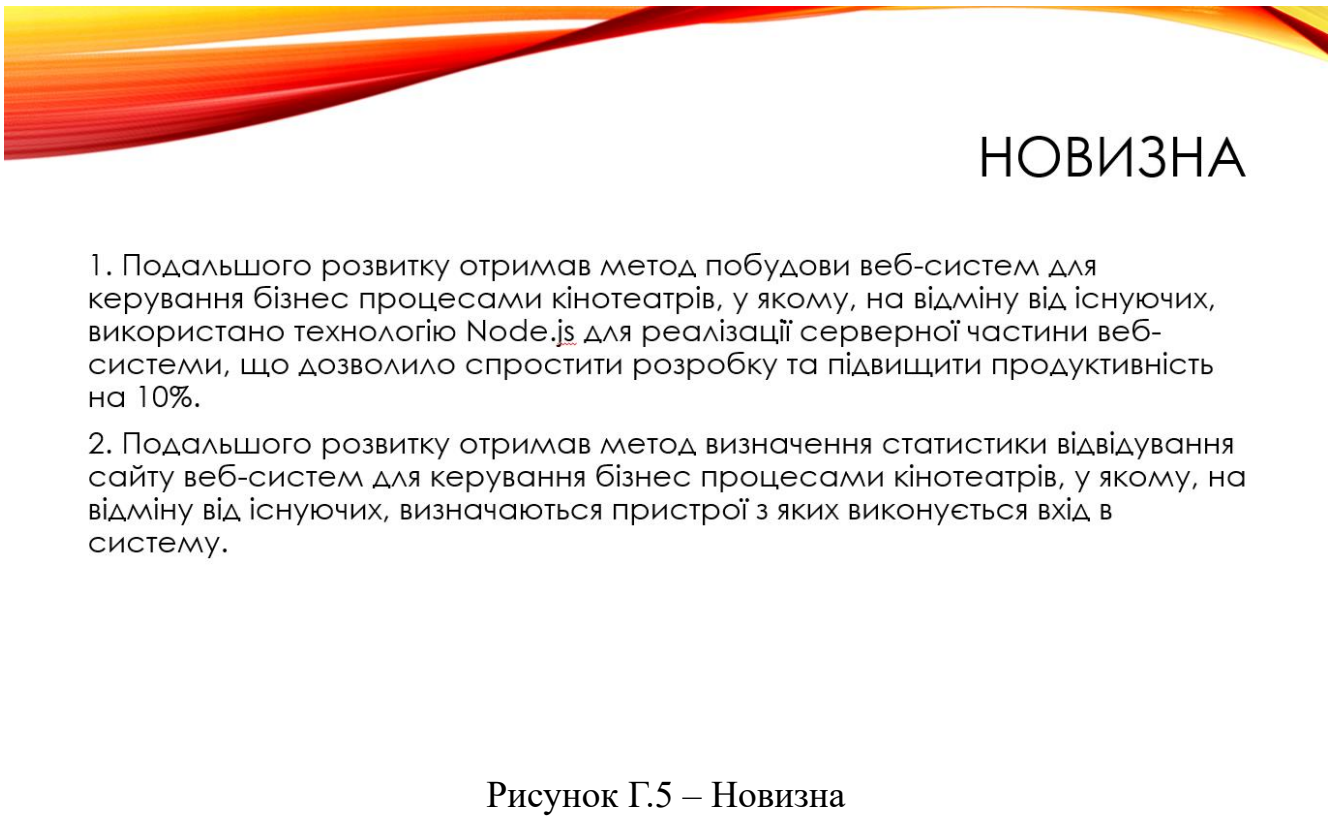

Рисунок Г.5 – Новизна

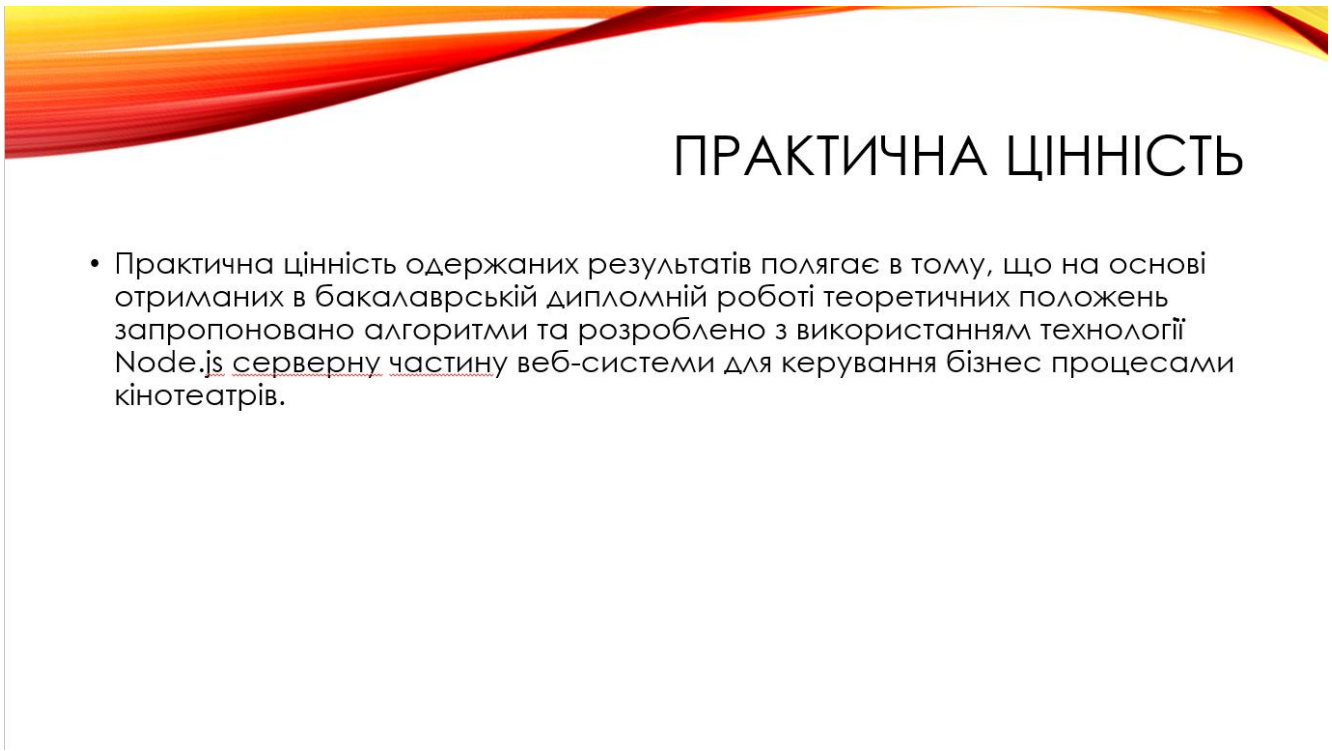

Рисунок Г.6 – Практична цінність

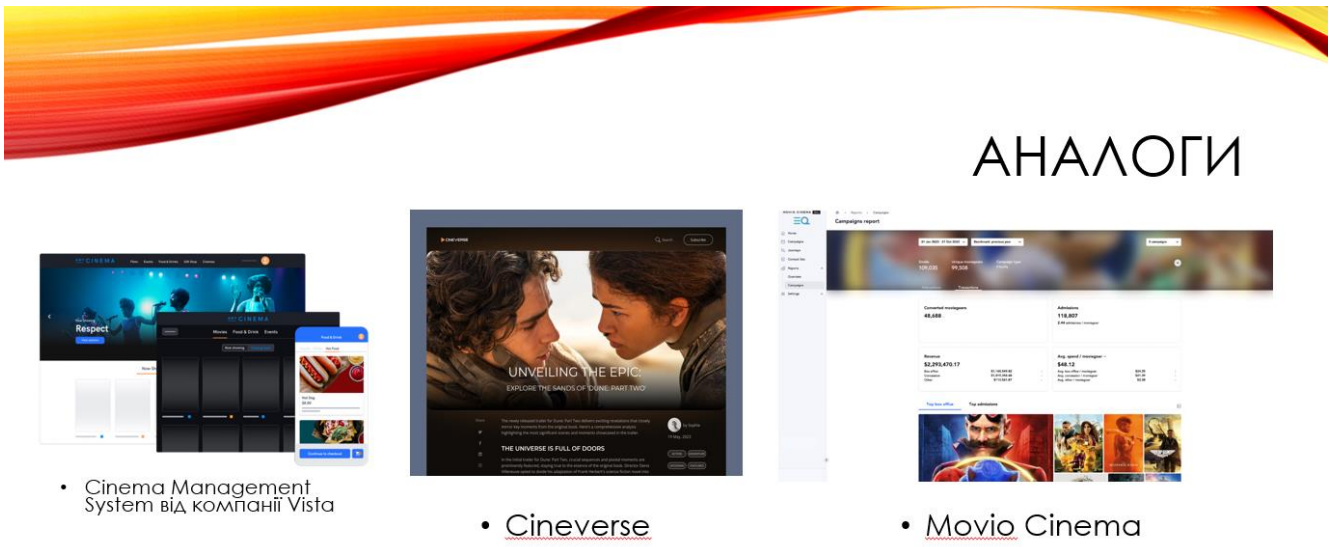

Рисунок Г.7 – Аналоги

Характеристика	Cinema Management System	Cineverse	Movio Cinema	Власна розробка
Безкоштовний доступ	0	0	0	1
Додавання інформації про кінозали	1	1	0	1
Формування звітів	1	1	1	1
Статистика користувачів	0	0	1	1
Створення кіносеансів	1	1	1	1
Сумарний бал	3	3	3	5

Рисунок Г.8 – Порівняльний аналіз аналогів

Рисунок Г.9 – Використані технології

Рисунок Г.10 – Розробка алгоритму відправлення запрошення на кіносеанс

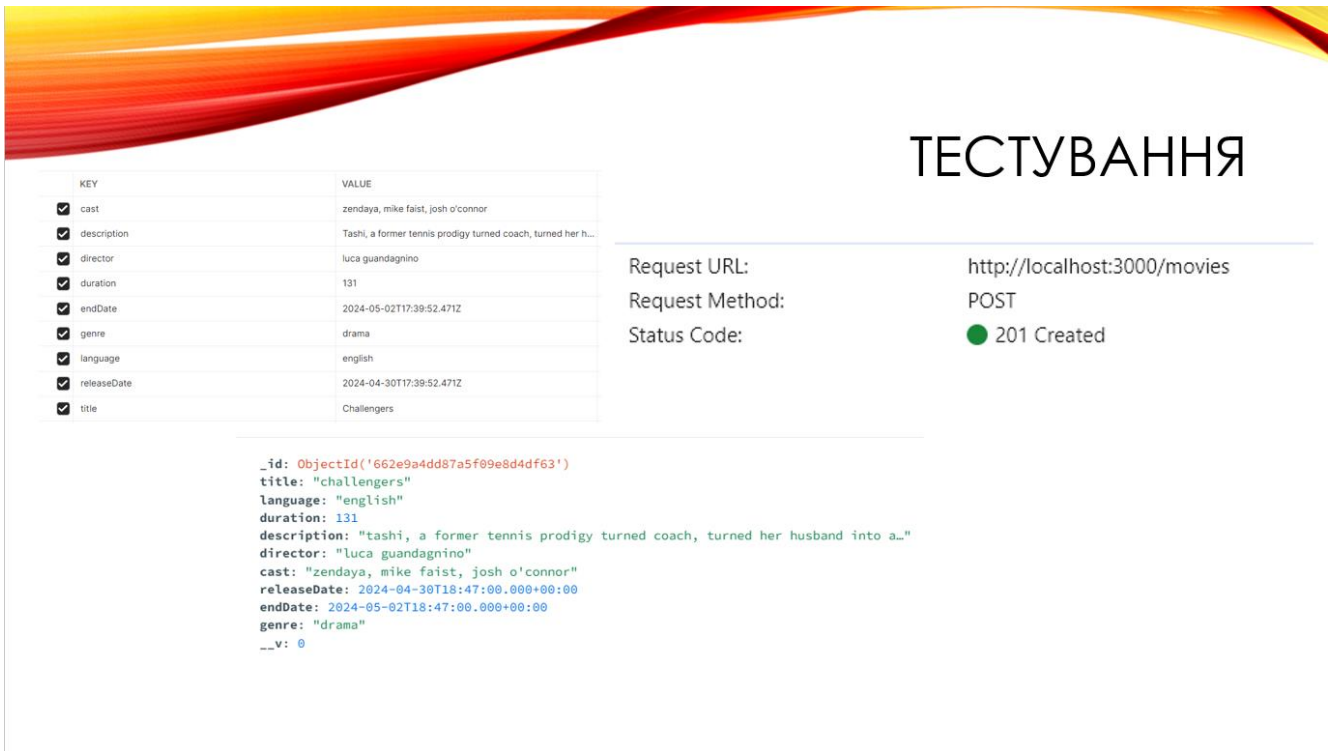

Рисунок Г.11 – Тестування функціоналу роботи з фільмами

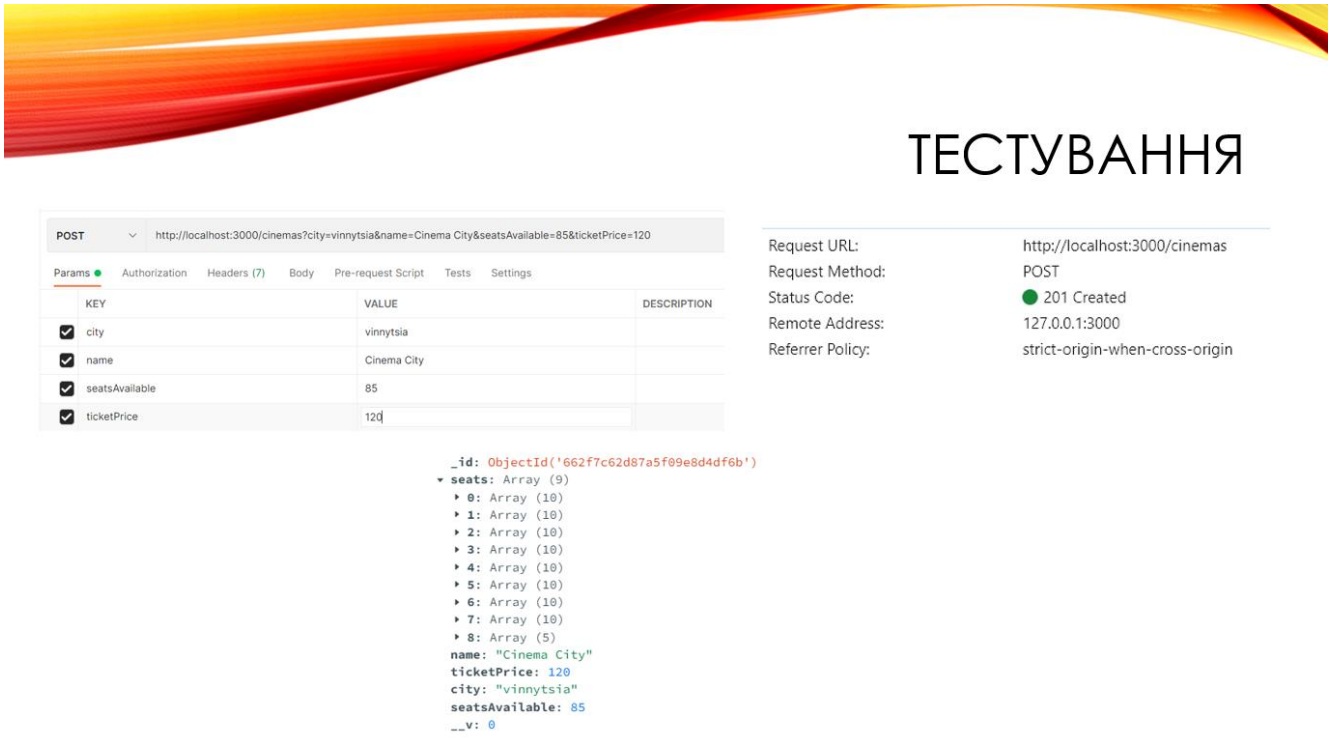

Рисунок Г.12 – Тестування функціоналу роботи з кінотеатрами

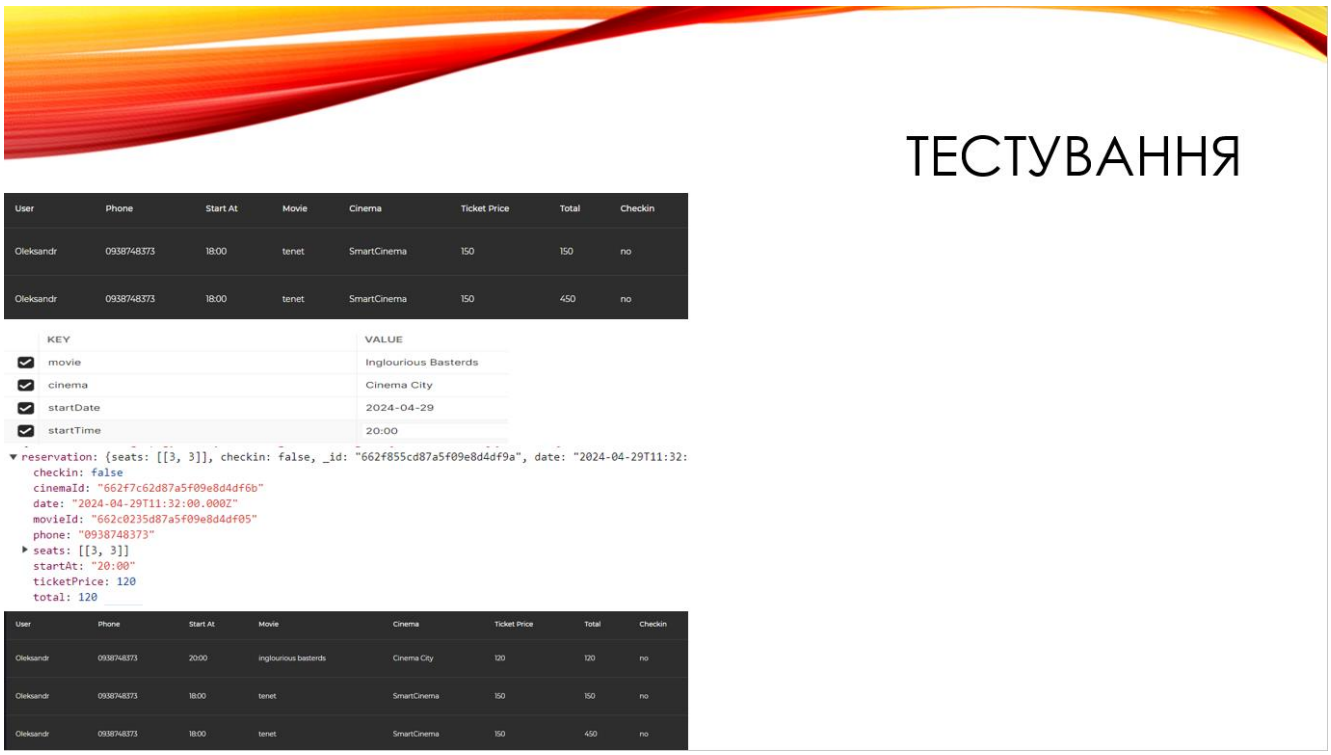

Рисунок Г.13 – Тестування функціоналу покупки квитків

В результаті виконання бакалаврської дипломної роботи, було розроблено серверну частину веб-системи управління бізнес-процесами роботи кінотеатрів. Розроблена серверна частина дозволяє додавати до бази даних фільми та кінотеатри, створювати кіносеанси та продавати квитки.

Для розробки було використано середовище розробки Visual Studio Code, мову програмування JavaScript, фреймоворк Node.js.

Рисунок Г.14 – Висновки (частина 1)

У першому розділі, було проведено аналіз стану питання серверної частини веб-системи, порівняльний аналіз аналогів, аналіз методів розв'язання задачі. На їх основі було доведено доцільність та актуальність власної розробки, поставлено задачі дослідження.

У другому розділі, було розроблено структуру та модель модуля сервера, структуру бази даних, розроблено метод відправлення запрошення на відвідування фільму електронною поштою.

У третьому розділі, було проведено варіантний аналіз та обгрунтування вибору мови програмування розробки, розроблено серверну частину для створення фільмів та кінотеатрів у системі. Розроблено серверну частину модуля статистики

У четвертому розділі, було проведено аналіз методів тестування, проведено тестування розробленої серверної частини. Тестування довело працездатність та коректність роботи застосунку.

Рисунок Г.15 – Висновки (частина 2)

Апробація матеріалів бакалаврської дипломної роботи. Основні положення бакалаврської дипломної роботи доповідалися та обговорювалися на Міжнародній науково-практична інтернет-конференція «Молодь в науці: дослідження, проблеми, перспективи (МН-2024)».

Публікації. Основні результати досліджень опубліковано в 1 науковій праці у матеріалах конференції.

Рисунок Г.16 – Апробація результатів роботи і публікації