

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

## **ПОЯСНЮВАЛЬНА ЗАПИСКА**

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

на тему:

**«Розробка веб-сервісу для спільного перегляду фільмів та  
серіалів»**

Виконав студент групи ПЗ-20-4  
КОСТЮЧЕНКО Артем Валерійович

Керівник роботи:  
ЛОКТИКОВА Тамара Миколаївна

Рецензент:  
ЧИЖМОТРЯ Олена Геннадіївна

Житомир – 2024

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

**«ЗАТВЕРДЖУЮ»**

Зав. кафедри інженерії  
програмного забезпечення  
Тетяна Вакалюк  
«26» лютого 2024 р.

**ЗАВДАННЯ**  
**на кваліфікаційну роботу**

Здобувач вищої освіти: **КОСТЮЧЕНКО Артем Валерійович**  
Керівник роботи: **ЛОКТИКОВА Тамара Миколаївна**  
Тема роботи: **«Розробка веб-сервісу для спільного перегляду фільмів та серіалів»,**  
затверджена Наказом закладу вищої освіти від «23» лютого 2024 р., №74/с  
Вихідні дані для роботи: **об'єктом дослідження є інформаційні технології для реалізації обміну даними між додатками в мережі Інтернет, а предметом дослідження є використання сучасних методів та засобів розробки веб-додатків та організації їхньої взаємодії.**

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

| Розділ | Консультант    | Завдання видав | Завдання прийняв |
|--------|----------------|----------------|------------------|
| 1      | Локтікова Т.М. | 04.03.2024р.   | 04.03.2024р.     |
| 2      | Локтікова Т.М. | 04.04.2024р.   | 04.04.2024р.     |
| 3      | Локтікова Т.М. | 13.05.2024р.   | 13.05.2024р.     |

## РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра складається з веб-сервісу для спільного перегляду фільмів та серіалів та пояснювальної записки. Пояснювальна записка до випускної роботи містить текстову частину, викладену на 86 сторінок друкованого тексту, список використаних джерел містить 26 найменувань і займає 2 сторінки. В роботі наведено 52 рисунки та 31 таблиць. В роботі є 27 сторінок додатків. Загальна кількість сторінок пояснювальної записки становить 113.

Метою роботи є створення веб-сервісу з використанням сучасних технологій, який дозволить користувачам переглядати фільми та серіали одночасно разом з спільнотою чи друзями незалежно від їх географічного розташування.

В роботі визначено ключові завдання для розробки веб-сервісу, проведено аналіз існуючих веб-додатків, подібних до розроблювального, а також наведено детальний сценарій роботи програмного продукту. Представлено огляд загальної структури веб-додатку, описано алгоритми взаємодії між системами та компонентами, а також наведено опис використання та об'єктну структуру веб-додатку. Як архітектуру обрано клієнт-серверну, як інструментарій для серверної частини – Express.js, а для клієнтської – Vite React.js, як базу даних – MongoDB.

Було проведено тестування додатку, щоб оцінити ефективність його інтерфейсу, функціональність та загальний досвід використання.

Веб-додаток має зручний інтерфейс, який дозволяє користувачам швидко та легко переглядати різний контент, створювати кімнати та спілкуватися між собою.

**КЛЮЧОВІ СЛОВА:** ВЕБ-СЕРВІС, КЛІЄНТ-СЕРВЕР, СОКЕТ, JAVASCRIPT, MONGODB.

|                  |             |                 |               |            |                                                                                           |                                            |             |                |
|------------------|-------------|-----------------|---------------|------------|-------------------------------------------------------------------------------------------|--------------------------------------------|-------------|----------------|
|                  |             |                 |               |            | ІПЗ.КР.Б-121-24-ПЗ                                                                        |                                            |             |                |
| <i>Змн.</i>      | <i>Арк.</i> | <i>№ докум.</i> | <i>Підпис</i> | <i>Дат</i> | Розробка веб-сервісу для спільного перегляду фільмів та серіалів.<br>Пояснювальна записка | <i>Літ.</i>                                | <i>Арк.</i> | <i>Аркуців</i> |
| <i>Розроб.</i>   |             | Костюченко А.В. |               |            |                                                                                           |                                            |             |                |
| <i>Керівник.</i> |             | Локтікова Т.М.  |               |            |                                                                                           |                                            | 3           | 113            |
| <i>н. контр.</i> |             | Тичина О.П.     |               |            |                                                                                           | Житомирська політехніка,<br>група ІПЗ-20-4 |             |                |
| <i>Зав. каф.</i> |             | Вакалюк Т.А.    |               |            |                                                                                           |                                            |             |                |

## ABSTRACT

The bachelor's thesis consists of a web service for sharing movies and TV series and an explanatory note. The explanatory note to the graduation thesis contains a text part set out on 86 pages of printed text, the list of references contains 26 titles and occupies 2 pages. The work contains 52 figures and 31 tables. There are 27 pages appendices in the work. The total number of pages of the explanatory note is 113.

The aim of the work is to create a web service using modern technologies that will allow users to watch movies and TV shows simultaneously with their community or friends regardless of their geographical location.

The paper identifies the key tasks for developing a web service, analyzes existing web applications similar to the one under development, and provides a detailed scenario of the software product. An overview of the general structure of the web application is presented, the algorithms of interaction between systems and components are described, and the description of the use and object structure of the web application is given. The architecture chosen is client-server, the toolkit for the server side is Express.js, and for the client side is Vite React.js, and the database is MongoDB.

The application was tested to evaluate the effectiveness of its interface, functionality, and overall user experience.

The web application has a user-friendly interface that allows users to quickly and easily view various content, create rooms, and communicate with each other.

**KEYWORDS:** WEB SERVICE, CLIENT-SERVER, SOCKET, JAVASCRIPT, MONGODB.

## ЗМІСТ

|                                                                                                             |           |
|-------------------------------------------------------------------------------------------------------------|-----------|
| <b>РЕФЕРАТ</b> .....                                                                                        | <b>3</b>  |
| <b>ABSTRACT</b> .....                                                                                       | <b>4</b>  |
| <b>ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ</b> .....                                                                      | <b>6</b>  |
| <b>ВСТУП</b> .....                                                                                          | <b>7</b>  |
| <b>РОЗДІЛ 1 АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ<br/>ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-СЕРВІСУ</b> ..... | <b>8</b>  |
| 1.1 Постановка задачі.....                                                                                  | 8         |
| 1.2 Аналіз аналогів програмного продукту .....                                                              | 10        |
| 1.3 Вибір архітектури для веб-орієнтованої системи.....                                                     | 14        |
| 1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного<br>забезпечення .....             | 19        |
| Висновки до першого розділу.....                                                                            | 25        |
| <b>РОЗДІЛ 2 ПРОЕКТУВАННЯ ВЕБ-СЕРВІСУ ДЛЯ СПІЛЬНОГО<br/>ПЕРЕГЛЯДУ ФІЛЬМІВ ТА СЕРІАЛІВ</b> .....              | <b>26</b> |
| 2.1 Визначення варіантів використання та об'єктно-орієнтованої структур...<br>.....                         | 26        |
| 2.2 Розробка бази даних системи.....                                                                        | 43        |
| 2.3 Проектування та реалізація алгоритмів роботи системи .....                                              | 50        |
| 2.4 Реалізація веб-сервісу для спільного перегляду фільмів та серіалів .....                                | 52        |
| Висновки до другого розділу .....                                                                           | 59        |
| <b>РОЗДІЛ 3 ПРОЕКТУВАННЯ ВЕБ-СЕРВІСУ ДЛЯ СПІЛЬНОГО<br/>ПЕРЕГЛЯДУ ФІЛЬМІВ ТА СЕРІАЛІВ</b> .....              | <b>60</b> |
| 3.1 Порядок встановлення та налаштування параметрів веб-сервісу.....                                        | 60        |
| 3.2 Структура інтерфейсу веб-сервісу .....                                                                  | 61        |
| 3.3 Тестування веб-сервісу .....                                                                            | 76        |
| Висновки до третього розділу .....                                                                          | 81        |
| <b>ВИСНОВКИ</b> .....                                                                                       | <b>83</b> |
| <b>СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ</b> .....                                                                     | <b>84</b> |
| <b>ДОДАТКИ</b> .....                                                                                        | <b>86</b> |

## ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПЗ – програмне забезпечення

API – прикладний програмний інтерфейс

REST – передача репрезентативного стану

SOCKET – обмін даними у реальному часі

WEBRTC – організація голосового та відеозв'язку через інтернет у режимі реального часу

БД – база даних

CRUD – основні функції для управління даними (створення, читання, оновлення, видалення)

MVC – розподілення даних та керуючої логіки додатку на окремі компоненти

SPA – односторінковий додаток

## ВСТУП

**Актуальність** теми зумовлена тим, що останніми роками зростає популярність веб-сервісів для спільного перегляду фільмів та серіалів. Існує суттєвий попит на взаємодію з іншими користувачами, тому веб-сервіси для спільного перегляду стають критично важливими.

**Метою** випускної кваліфікаційної роботи бакалавра є розробка веб-сервісу для спільного перегляду фільмів та серіалів. Сервіс має забезпечувати функціонал для повної та безперервної роботи портала та взаємодіяти як з мобільними веб-додатками, так із комп'ютерними веб-додатками.

Поставлена мета обумовлює необхідність виконання наступних **завдань**:

1. Аналіз теоретичних та функціональних потреб проектування та реалізації веб-сервісу.
2. Аналіз існуючих веб-сервісів на обрану тематику, визначення унікальності.
3. Визначення технологій реалізації веб-сервісу.
4. Проектування бази даних, для збереження інформації сервісу та користувачів.
5. Виконання опису методів для взаємодії з веб-системою.
6. Визначення об'єктної структури веб-сервісу.

**Об'єктом дослідження** є інформаційні технології для реалізації обміну даними між додатками в мережі Інтернет.

**Предметом дослідження** є використання сучасних методів та засобів розробки веб-додатків та організації їхньої взаємодії.

За темою випускної кваліфікаційної роботи бакалавра були подані та прийняті до публікації тези: Костюченко А.В., Кушнір Н.О., Локтікова Т.М. Веб-сервіс для спільного перегляду фільмів та серіалів у режимі реального часу. Всеукраїнська науково-практична on-line конференція здобувачів вищої освіти і молодих учених, присвячена Дню науки (13-14 травня 2024 року).

|  |  |  |  |  |                     |   |
|--|--|--|--|--|---------------------|---|
|  |  |  |  |  | ІПЗ.КР.Б-121-24-ІПЗ |   |
|  |  |  |  |  |                     |   |
|  |  |  |  |  |                     | 7 |

# РОЗДІЛ 1 АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ ВЕБ-СЕРВІСУ

## 1.1 Постановка задачі

У сучасному світі, із шаленим темпом розвитку технологій та загальнодоступною мережею Інтернет, відкриваються безліч нових можливостей в розробці різних веб-сервісів, одним з яких є сервіс спільного перегляду фільмів та серіалів. Цей феномен переходить на зовсім новий рівень завдяки високому розвитку веб-сервісів, які створюють враження спільного перебування у віртуальному кінозалі разом із друзями або навіть сторонніми користувачами з усього світу.

Завдяки цим веб-сервісам, перегляд фільмів та серіалів перетворюється на справжню соціальну подію, де глядачі можуть обмінюватися враженнями, коментарями та емоціями прямо під час перегляду. Користувачі можуть ділитися своїми улюбленими моментами, висловлювати свої думки щодо сюжету та персонажів, обговорювати несподівані повороти сюжету та аналізувати різні аспекти фільмів. Це створює новий рівень співпереживання та взаєморозуміння між користувачами.

Більше того, ці веб-сервіси створюють можливість для формування власних віртуальних кіноклубів та спільнот, де люди зі схожими інтересами можуть знаходити одне одного, об'єднуючись навколо спільної любові до кіно. Кожен може відчувати себе частиною великої кінозалу, де захоплення фільмами та серіалами стає вагомим частиною власного життя. Він не лише розширює можливості для насолоди від перегляду аудіовізуальних шедеврів, а й створює нові зв'язки, дружби та спільні враження, що збагачують досвід кожного учасника. Це не лише розважальний відпочинок, але й джерело нових знайомств, дружби та культурного обміну. Такі веб-сервіси стають важливою складовою сучасної культурної інтеракції, яка зближує людей навіть на відстані тисяч кілометрів.

Метою розробки є створення інтерактивної та соціальної веб-платформи для спільного перегляду фільмів та серіалів для надання користувачам можливості спільно насолоджуватися контентом та взаємодіяти між собою у реальному часі.

Для реалізації поставленої мети необхідно виконати наступні завдання.

Спочатку потрібно спроектувати архітектуру системи.

Далі визначити структуру бази даних: створити таблиці для зберігання інформації про фільми, серіали та користувачів, кімнат і т.п., реалізувати зв'язки між таблицями для ефективного доступу до даних.

Наступним етапом стане реалізація функціональності веб-сервісу, а саме: додавання можливості створення фільмів та серіалів адміністратором, створення кімнат для спільного перегляду, інтеграція чату та голосового чату для обміну повідомленнями під час перегляду.

Важливим моментом розробки буде створення інтерфейсу користувача: зручного інтерфейсу для адміністратора для додавання та керування контентом, інтуїтивно зрозумілого інтерфейсу для користувачів для перегляду та спілкування.

Далі здійснюватиметься тестування функціональності розробленого сервісу та виправлення помилок, вдосконалення роботи сервісу.

Також важливе значення матимуть розгортання сервісу на сервері та запуск його в експлуатацію, постійна підтримка, вдосконалення та оновлення сервісу для забезпечення його стабільної роботи та відповідності потребам користувачів.

Результатом виконання цих задач буде створення веб-сервісу, який дозволить користувачам насолоджуватися переглядом фільмів та серіалів у віртуальних кімнатах.

## 1.2 Аналіз аналогів програмного продукту

У сфері веб-сервісів для спільного перегляду фільмів та серіалів вже існує значна кількість програм, що надають користувачам можливість спільного перегляду контенту в реальному часі.

Для аналізу кращого інструменту для вирішення задачі спільного перегляду аудіовізуального контенту було обрано наступні сервіси: Netflix Party, Scener, Kast, Takflix.

Netflix Party [1] – це розширення для браузера Google Chrome, яке дозволяє користувачам створювати віртуальні кімнати для спільного перегляду фільмів та серіалів на платформі Netflix. Учасники можуть спілкуватися у чаті та синхронізувати відтворення контенту. Інтерфейс Netflix Party показаний на рис. 1.1.

До переваг Netflix Party можна віднести легкість використання, доступний в онлайні з веб-браузера, легко використовувати, сервіс автоматично синхронізує відтворення контенту для всіх учасників кімнати, щоб усі одночасно переглядали фільм чи серіал.

До недоліків можна віднести обмежену підтримку браузерів, відсутність деяких функцій Netflix, розширення, що потребує додаткових дій користувача, щоб встановити його.

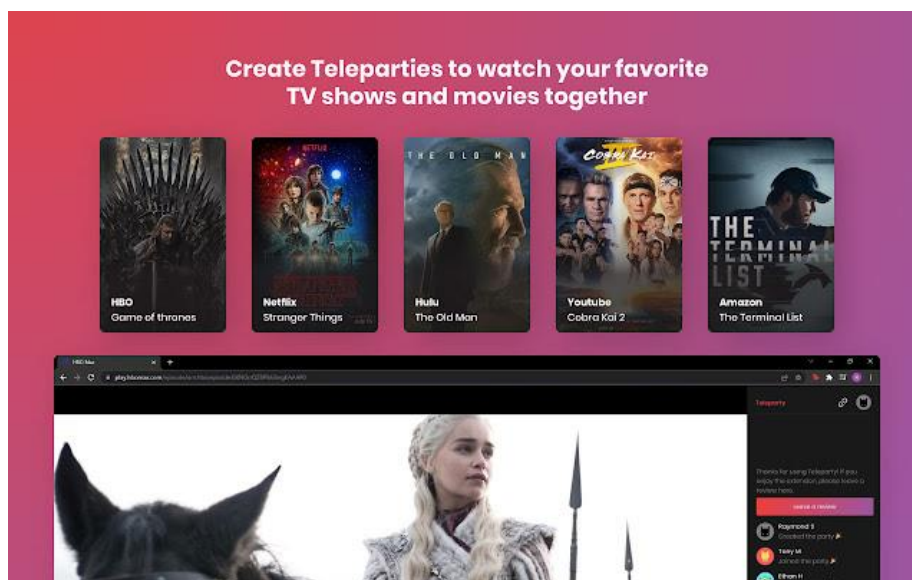


Рисунок 1.1 – Інтерфейс Netflix Party

Scener [2] – це платформа, яка підтримує спільний перегляд контенту з різних стрімінгових сервісів, таких як Netflix, HBO, Amazon Prime та інші. Вона дозволяє створювати віртуальні кімнати з чатом та голосовим спілкуванням. Інтерфейс Scener показаний на рис. 1.2.

До переваг Scener можна віднести інтеграцію голосового та текстового чату, можливість створювати приватні кімнати, підтримка різних платформ для перегляду, можливість перегляду контенту з інших платформ.

До недоліків можна віднести обмежену підтримку браузерів (тільки Google Chrome та Mozilla Firefox), що може обмежити доступність для користувачів, не всі функції доступні для всіх платформ, платна підписка.

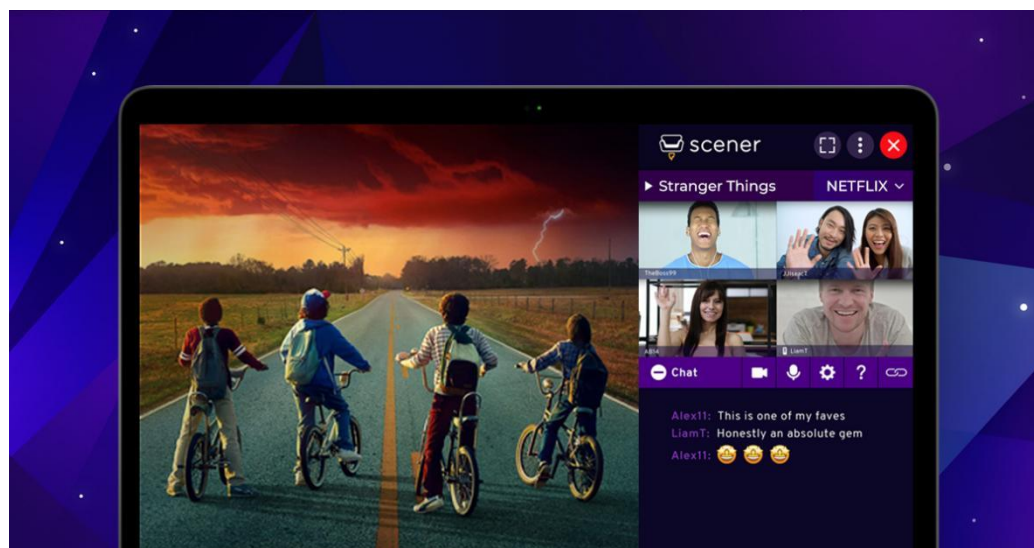


Рисунок 1.2 – Інтерфейс Scener

Kast [3] – платформа для спільного перегляду фільмів, серіалів та інших відео контентів разом з друзями чи іншими користувачами з усього світу. Схоже на інші сервіси, Kast дозволяє створювати віртуальні кімнати, де група користувачів може дивитися відео та одночасно обговорювати його у текстовому або голосовому чаті. Інтерфейс Kast показаний на рис. 1.3.

До переваг Kast можна віднести багатфункціональність пропонує більше, ніж просто спільний перегляд відео. Він також має функції, такі як голосовий та текстовий чат, підтримку веб-камери, можливість відтворення ігор та музики, широкий вибір контенту, можливість створення приватних кімнат, інтеграція з соціальними мережами.

До недоліків можна віднести обмежену підтримку платформ, можливі проблеми з безпекою.



Рисунок 1.3 – Інтерфейс Kast

Takflix [4] – це онлайн-платформа, де можна дивитися українські фільми та серіали на замовлення, справжня знахідка для любителів українського кінематографу, пропонуючи широкий спектр контенту, від класики до сучасних артхаусних стрічок. Інтерфейс Takflix показаний на рис. 1.4.

До переваг Takflix можна віднести зручний інтерфейс, платформа доступна на будь-якому пристрої з підключенням до інтернету, включаючи смартфони, планшети, комп'ютери та Smart TV, великий вибір українського контенту понад 1000 фільмів та серіалів українського виробництва, включаючи новинки та ексклюзивні покази, доступні ціни та підтримка української культури.

До недоліків Takflix можна віднести невеликий вибір зарубіжного контенту, технічні проблеми.

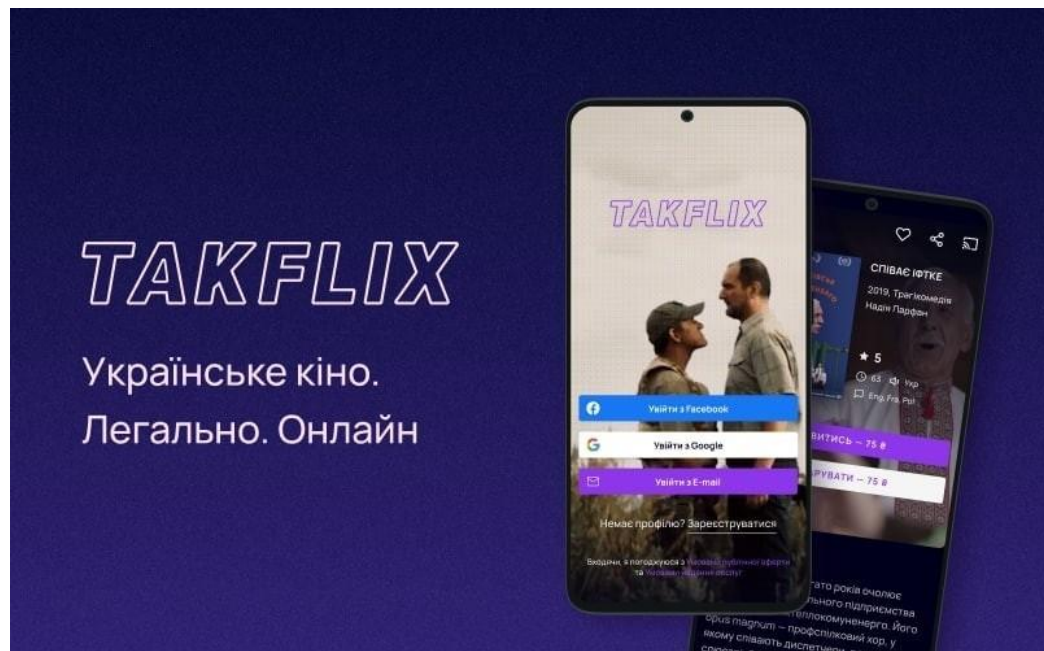


Рисунок 1.4 – Інтерфейс Takflix

Для наочності порівняння отримані результати подані в табл. 1.1.

Таблиця 1.1

Порівняльна характеристика аналогів програмного продукту

| Назва платформи | Зручність інтерфейсу | Інтеграція чату | Підтримка браузерів | Наповненість контентом | Підтримка платформи |
|-----------------|----------------------|-----------------|---------------------|------------------------|---------------------|
| Netflix Party   | +                    | +/-             | +/-                 | +                      | +                   |
| Scener          | +/-                  | +               | +                   | +                      | +                   |
| Kast            | +                    | +               | +/-                 | +                      | +                   |
| Takflix         | +/-                  | +/-             | +                   | +/-                    | +                   |

Після аналізу функціональних можливостей систем для спільного перегляду фільмів та серіалів, можна визначити вимоги до розроблювального сервісу: можливість створення кімнат для спільного перегляду контенту, функціонал для синхронізованого відтворення відео для усіх учасників кімнати, голосовий чат та чат для обміну повідомленнями під час перегляду, адаптивний дизайн для роботи на різних пристроях, акцент на користувацькому досвіді та зручності використання, можливість розширення функціоналу шляхом інтеграції з іншими платформами або додатками для покращення взаємодії та спільного досвіду перегляду.

|  |  |  |  |  |                     |    |
|--|--|--|--|--|---------------------|----|
|  |  |  |  |  | ІПЗ.КР.Б-121-24-ІПЗ |    |
|  |  |  |  |  |                     | 13 |

### 1.3 Вибір архітектури для веб-орієнтованої системи

Для забезпечення структурованості, організованості, легкості масштабування, продуктивності, безпеки для користувачів та підтримки змін у подальшій розробці веб-орієнтованої системи, критично вибрати правильну архітектуру програмного забезпечення.

Архітектура програмного забезпечення є фундаментальною структурою, що визначає організацію, взаємозв'язки та взаємодію різних компонентів системи. Це включає як концептуальне уявлення про систему, так і її фізичну реалізацію.

Архітектура включає в себе різні аспекти програмного забезпечення, такі як структура даних, системні компоненти, модулі програмного забезпечення, інтерфейси користувача, алгоритми та протоколи зв'язку. Вона також визначає принципи організації програмного коду та його розподілу між різними частинами системи.

Якщо правильно спроектувати архітектуру системи, то це дозволить легко розширювати та модифікувати систему, забезпечувати її стабільність, продуктивність та безпеку. Архітектура системи допомагає забезпечити зручний та ефективний розроблювальний процес.

Архітектура програмного забезпечення визначається у контексті конкретних вимог і обмежень, що стосуються проекту. Зокрема, веб-орієнтовані системи часто потребують особливої уваги до таких аспектів, як розподіленість, масштабованість, безпека та ефективність взаємодії з користувачем через веб-інтерфейс.

Одним з популярних підходів для веб-орієнтованих систем є модель архітектури клієнт-сервер. У цій моделі клієнтська частина відповідає за взаємодію з користувачем через веб-браузер або інші клієнтські програми, тоді як серверна частина обробляє запити користувачів, виконує бізнес-логіку і відповідає на запити клієнтів. Цей підхід дозволяє чітко розділити функціональні обов'язки між клієнтом і сервером, що сприяє підтримці і розширенню системи.

Ще однією важливою архітектурною концепцією для веб-систем є шарова архітектура. У цій моделі програмне забезпечення поділяється на рівні (або шари),

кожен з яких відповідає за певний аспект функціональності системи. Наприклад, можуть бути окремі шари для доступу до даних, бізнес-логіки та презентації. Це дозволяє зберігати систему модульною і спрощує розробку та підтримку.

Також важливо враховувати питання безпеки при виборі архітектури програмного забезпечення для веб-системи це означає, що потрібно застосовувати відповідні заходи безпеки на рівні мережі, додатку та даних, а також враховувати можливості для захисту від таких загроз, як атаки злову, витік інформації та інші типи кіберзагроз.

Для створення подібних веб-орієнтованих системи доцільно використовувати клієнтсерверну архітектуру програмного забезпечення.

Ще однією важливою архітектурною концепцією для веб-систем є патерни проектування.

Патерни проектування – це загальноприйняті рішення для типових проблем, що виникають під час розробки програмного забезпечення. Вони допомагають розробникам створювати структуровані, ефективні та легко змінювані системи. Патерни виникають на основі досвіду та кращих практик розробки програмного забезпечення і дозволяють уникати поширених помилок.

Патерни проектування можна класифікувати за їхнім призначенням і структурою. Серед найбільш відомих класифікацій можна виділити патерни структурного, поведінкового та породжувального типу. Кожен з цих типів має свої особливості і використовується для вирішення різних видів завдань під час проектування програмного забезпечення.

Патерни проектування є потужним інструментом для підвищення якості та ефективності розробки програмного забезпечення. Інтеграція патернів у процес розробки дозволяє створювати більш масштабовані, легко змінювані та довговічні програмні рішення.

Отже оглянемо патерни проектування, що можуть бути використані для створення даної веб-орієнтованої системи.

Серверна частина веб-орієнтовної відіграє ключову роль у забезпеченні функціональності, надійності та продуктивності системи. Вона відповідає за

обробку запитів від клієнтів, взаємодію з базою даних, виконання бізнес-логіки та надання необхідної інформації через мережу. Один із способів організації цієї частини є використання патернів проектування, таких як MVC та REST API.

Патерн MVC (Model-View-Controller) [5] – архітектурний шаблон проектування, що розділяє веб-додаток на три основні компоненти: модель (Model), представлення (View) та контролер (Controller). Модель відповідає за обробку даних, бізнес-логіку та взаємодію з базою даних, представлення - відображення інформації, контролер - обробка запитів від користувача, взаємодіє між моделлю та представленням. Діаграму взаємодії між компонентами MVC показано на рис. 1.5.

Мета патерну MVC полягає у забезпеченні ефективного розділення відповідальностей між компонентами додатку, що сприяє збереженню високої чистоти коду, полегшує розуміння та модифікацію програмного забезпечення, підвищує його масштабованість та гнучкість. Розділення логіки додатку на окремі компоненти дозволяє розробникам працювати над ним паралельно, сприяє покращенню швидкодії та зниженню ймовірності виникнення помилок під час розробки, а також полегшує впровадження нових функціональностей та рефакторинг коду у майбутньому.

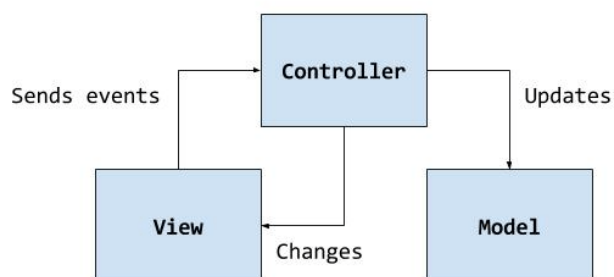


Рисунок 1.5 – Діаграма взаємодії між компонентами архітектурного шаблону

Також важливо розглянути ще один патерн серверної частини веб-орієнтованої системи для обробки запитів - REST API.

REST API (Representational State Transfer) [6] – це стиль архітектури веб-сервісів, що ґрунтується на використанні HTTP-протоколу для взаємодії між клієнтом і сервером. Основна мета RESTful API полягає в забезпеченні доступу до

ресурсів через стандартні HTTP-методи, такі як GET (для отримання даних), POST (для створення нових ресурсів), PUT/PATCH (для оновлення існуючих) та DELETE (для видалення). Діаграму взаємодії між клієнтом та сервером показано на рис. 1.6.

Ресурси – це об’єкти чи дані, до яких можна отримати доступ через API, кожен з яких має унікальний ідентифікатор та може мати різні стани. У відповідності з принципами REST, ресурси повинні бути представлені у відповідях сервера у вигляді структурованих даних, таких як JSON або XML. Цей підхід спрощує розробку та інтеграцію веб-сервісів, роблячи їх більш масштабованими та легкими для розуміння та використання.

Мета патерну REST API полягає в створенні веб-сервісів, які забезпечують простоту, масштабованість та надійність у взаємодії між клієнтом і сервером. Цей підхід передбачає використання стандартних протоколів та концепцій, що спрощує розробку, розуміння та підтримку веб-сервісів. REST API дозволяє покращити продуктивність розробки шляхом забезпечення стандартизованого способу взаємодії між різними компонентами системи, що сприяє підвищенню швидкості розгортання нових функцій, зменшенню кількості помилок та полегшує масштабування і підтримку системи у майбутньому. Крім того, використання стандартних протоколів HTTP дозволяє забезпечити сумісність із різноманітними клієнтськими та серверними платформами, що робить REST API відкритим та доступним для широкого кола розробників та користувачів.

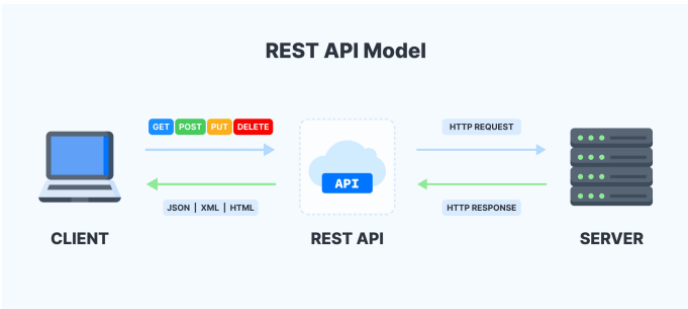


Рисунок 1.6 – Діаграма взаємодії між клієнтом та сервером архітектурного шаблону

Клієнтська частина веб-орієнтованої системи є важливим компонентом, що відповідає за інтерфейс та взаємодію з користувачем. Це та частина програми, яка

запускається на браузері або мобільному пристрої, і з якою користувач активно взаємодіє. Клієнтська частина додатка відображає дані, отримані від сервера, та забезпечує можливість користувачу взаємодіяти з цими даними шляхом введення інформації та виконання різних дій.

Патерн SPA (Single Page Application) [7] – архітектурний шаблон веб-додатків, де вся функціональність зосереджена на одній веб-сторінці.

Специфікації патерну включають асинхронне завантаження контенту для швидкості та плавності роботи додатка, маршрутизацію на клієнтському боці для зручності управління станом додатка та навігацією між різними відображеннями без перезавантаження сторінки, а також можливість кешування даних на клієнтському боці для зменшення часу завантаження та поліпшення загального досвіду користувача.

Головною метою патерну SPA є створення більш інтерактивного та зручного для користувача інтерфейсу веб-додатків. Цей підхід дозволяє знизити час завантаження, покращити продуктивність та реактивність додатка, а також спростити розробку та підтримку програмного забезпечення. Патерн SPA дозволяє створювати більш динамічні та ефективні веб-додатки, що забезпечує зручний та задоволений досвід користувача. Діаграму роботи SPA показано на рис. 1.7.

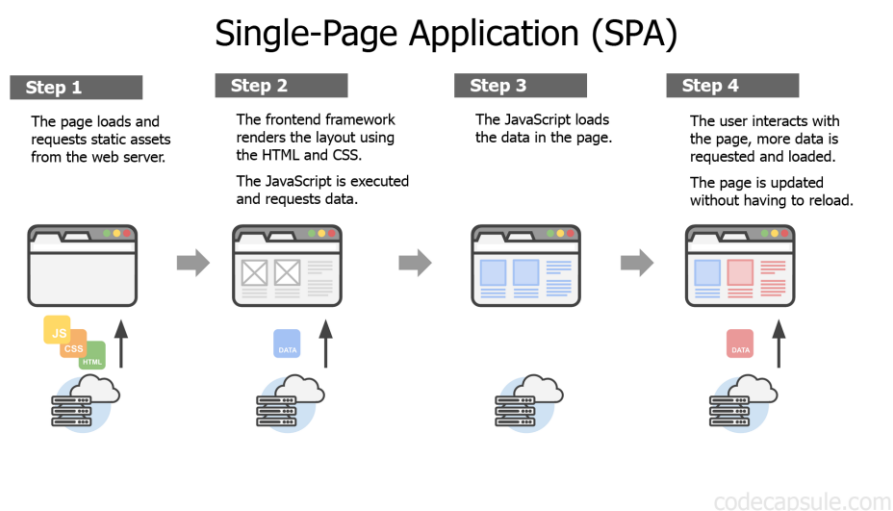


Рисунок 1.7 – Діаграма роботи архітектурного шаблону SPA

## 1.4 Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

Для виконання етапу кодування було обрано середовище розробки Visual Studio Code, мову програмування TypeScript та платформу Node.js.

Visual Studio Code (VS Code) [8] є легкий, потужний та безкоштовний текстовий редактор, розроблений компанією Microsoft для роботи з різними мовами програмування та технологіями. Однією з ключових переваг VS Code є його широкі можливості налаштування, розширення та інтеграції з іншими інструментами, що робить його одним з найпопулярніших редакторів серед розробників усіх рівнів.

TypeScript [9] – це мова програмування, яка є розширенням JavaScript, призначена для розробки великих та складних програмних продуктів. Використання TypeScript для розробки даного веб-сервісу надає кілька переваг, які можуть значно полегшити процес розробки та забезпечити більшу стабільність та безпеку програмного забезпечення.

Порівняно з JavaScript, TypeScript надає можливість статичної типізації, що дозволяє виявляти помилки ще до запуску програми. Це особливо важливо в розробці веб-сервісу, де велика кількість взаємодії між клієнтом та сервером може призвести до неточностей у передачі даних. Завдяки TypeScript можна визначити типи даних для об'єктів та функцій, що дозволяє уникнути багатьох типових помилок та зробити код більш надійним.

Для виконання коду TypeScript як на серверній, так і на клієнтській частині, було обрано Node.js як основне середовище.

Node.js [10] – це вільне, відкрите, крос-платформоване середовище виконання JavaScript, яка дозволяє виконувати JavaScript код на серверному боці. Однією з ключових особливостей Node.js є його асинхронний та подієво-орієнтований підхід до програмування, що дозволяє обробляти багато запитів одночасно без блокування виконання коду.

Ось деякі ключові аспекти Node.js:

1. Асинхронність та події (синхронна модель програмування, яка дозволяє ефективно обробляти багато запитів без чекання завершення кожного);

2. JavaScript на сервері (дозволяє використовувати JavaScript як мову програмування для серверного коду, що спрощує процес розробки, оскільки одна мова використовується як на клієнті, так і на сервері);

3. Платформонезалежність (підтримується на багатьох платформах, таких як Windows, macOS, Linux тощо, що робить його ідеальним вибором для розробки крос-платформових додатків);

4. Швидкодія (використання механізмів Just-In-Time (JIT) компіляції та оптимізації, що забезпечує високу швидкість виконання коду).

Серверна частина розроблювальної системи буде базуватися на фреймворку Express.js з використанням патернів MVC та REST API.

Express.js [11] – це фреймворк для Node.js, що вирізняється мінімалізмом та гнучкістю, що дозволяє створювати потужні серверні додатки та API з надійністю та ефективністю. З використанням патернів MVC та REST API, Express.js спрощує організацію коду та забезпечує зручний спосіб взаємодії з клієнтською частиною. Він має широкий вибір модулів та middleware, які дозволяють легко розширювати його функціональність, включаючи обробку запитів та відповідей. Швидкодія та активна спільнота роблять Express.js одним з найкращих виборів для розробки серверної частини, забезпечуючи ефективну та надійну роботу розроблювальної системи.

Для забезпечення можливості взаємодії користувачів у реальному часі під час перегляду відеоконтенту та спілкування в чаті, ідеальним рішенням є використання бібліотеки Socket.io.

Socket.io [12] – це бібліотека JavaScript для реалізації веб-сокетів, яка дозволяє побудувати потужні та ефективні додатки в режимі реального часу. Основна перевага Socket.io полягає в його здатності до двостороннього зв'язку між клієнтом та сервером, що дозволяє передавати дані безпосередньо та миттєво між усіма підключеними користувачами.

Використання Socket.io у даній розробці дозволяє забезпечити швидку та надійну передачу повідомлень, включаючи текстові повідомлення для чату. Бібліотека автоматично вибирає найефективніший механізм передачі даних для кожного клієнта, використовуючи WebSockets для сучасних браузерів та змінюючи протоколи в залежності від можливостей та обмежень підтримуваних клієнтів.

Для забезпечення можливості голосового чату під час перегляду відеоконтенту є використання інтернет-протоколу WebRTC.

WebRTC (Web Real-Time Communication) [13] – це технологія, яка дозволяє реалізувати додатки з аудіо, відео та обміном даними в режимі реального часу без необхідності встановлення додаткових плагінів або програмного забезпечення. Основна перевага WebRTC полягає в його здатності до прямої передачі медіа-даних між браузерами, що забезпечує низьку затримку та високу якість зв'язку.

Використання WebRTC у даній розробці дозволяє забезпечити швидку та надійну передачу аудіо та відеоданих для голосового чату. Технологія WebRTC забезпечує безпосередню передачу даних між клієнтами, що мінімізує затримки та покращує якість зв'язку. WebRTC підтримує шифрування даних, що забезпечує високий рівень безпеки та приватності для користувачів.

Клієнтська частина розроблювальної системи буде базуватися на Vite React.js, що є односторінковим додатком (SPA).

React.js [14] – це бібліотека JavaScript для розробки інтерфейсів користувача, що зосереджується на створенні динамічних та ефективних веб-додатків. Однією з ключових концепцій React є використання віртуального DOM, що дозволяє ефективно оновлювати тільки ті частини веб-сторінки, які змінилися, без перезавантаження всього документа. Це забезпечує високу продуктивність та швидкодію веб-додатків, що є важливими факторами для реалізації сучасних інтерактивних додатків для користувачів.

Використання React.js у даній розробці пропонує декларативний підхід до створення користувацького інтерфейсу, що дозволяє легко описати, як виглядає додаток у будь-який момент часу в залежності від даних, що робить код більш зрозумілим та підтримуваним, спрощуючи процес розробки. Також React.js

пропонує компонентний підхід до розробки, що дозволяє розбити інтерфейс на невеликі та самостійні компоненти, що полегшує управління кодом, сприяє повторному використанню компонентів та полегшує тестування.

Vite React [15] – це комбінація популярного інструменту для розробки веб-додатків Vite та бібліотеки React для створення користувацьких інтерфейсів. Завдяки швидкості та ефективності Vite, а також гнучкості та потужності React, розробники можуть створювати високоякісні додатки з покращеним досвідом розробки.

Ось деякі переваги Vite React:

1. Швидкий старт і продуктивність (Vite забезпечує миттєве завантаження додатків під час розробки, що значно скорочує час запуску проекту. Гаряча заміна модулів (HMR) дозволяє оновлювати компоненти React в реальному часі без перезавантаження сторінки, що покращує продуктивність розробки);

2. Модульний підхід і підтримка сучасних стандартів (Vite використовує ES модулі для швидкого завантаження та побудови додатків, що дозволяє розробникам використовувати новітні можливості JavaScript та React, забезпечуючи сумісність з сучасними браузерами);

3. Оптимізація збірки (Для продакшн-збірки Vite використовує Rollup, забезпечуючи ефективне мініфікацію, розщеплення коду та інші оптимізації, що дозволяє створювати швидкі та оптимізовані React-додатки для кінцевих користувачів);

4. Проста інтеграція з React (Vite має вбудовану підтримку для React, що дозволяє легко налаштувати проект. Розробники можуть швидко створювати нові додатки або інтегрувати Vite у існуючі проекти React).

Для зберігання даних було обрано нереляційну базу даних MongoDB.

MongoDB [16] – це документоорієнтована, нереляційна база даних, яка базується на моделі зберігання даних у вигляді JSON-подібних документів. Однією з основних особливостей MongoDB є гнучкість схеми даних, що дозволяє зберігати дані різної структури у тому ж сховищі.

MongoDB ідеально підходить для реалізації складних структур даних, таких як вкладеність документів, масиви, геопросторові дані тощо, що дозволяє легко моделювати дані відповідно до вимог проекту та робити їх доступними для швидкого та зручного отримання.

Використання MongoDB у даній розробці дозволяє зберігати та опрацьовувати великі обсяги даних з високою швидкістю завдяки своїй архітектурі, що базується на документах. Це особливо важливо для даної розробки, де потрібно ефективно зберігати та обробляти велику кількість медіа-даних та інформації про користувачів. Також MongoDB має гнучку та потужну запитову мову, яка дозволяє виконувати різноманітні операції з даними та ефективно взаємодіяти з базою даних, отримуючи необхідні дані швидко та ефективно.

В якості вибору платформи для розгортання даної розроблювальної веб-орієнтованої системи було обрано Vercel для розгортання серверної частини та Netlify для розгортання клієнтської частини.

Vercel [17] – це платформа для розробки та розгортання сучасних веб-додатків, яка надає безліч інструментів для розробників, щоб вони могли створювати, тестувати та розгортати свої проекти з максимальною ефективністю.

Обираючи Vercel для розгортання серверної частини, ми користуємося кількома перевагами, які ця платформа надає:

1. Швидке та просте розгортання додатків (Vercel забезпечує миттєве розгортання додатків завдяки інтеграції з Git-платформами (GitHub, GitLab, Bitbucket), що дозволяє зосередитися на розробці функціоналу додатку, не витрачаючи час на налаштування серверної інфраструктури);

2. Різноманітність інструментів для розробки додатків (платформа пропонує багатий набір інструментів, включаючи автоматичне масштабування, попередній перегляд додатків (Preview Deployments), та інтеграцію з популярними фреймворками (Express.js, Next.js, React, Vue.js, Svelte та інші), що робить процес розробки швидким і зручним);

3. Надійність та масштабованість (Vercel забезпечує високу доступність і надійність додатків завдяки автоматичному масштабуванню та глобальному

розподілу контенту через мережу CDN (Content Delivery Network), що дозволяє додаткам працювати без перебоїв навіть при великому навантаженні);

4. Простий та зрозумілий інтерфейс для управління додатками (інтерфейс Vercel дозволяє розробникам ефективно взаємодіяти з платформою, керувати розгортанням, переглядати журнали та аналітику, а також відстежувати продуктивність додатків в режимі реального часу).

Netlify [18] – це платформа для розробки та розгортання сучасних веб-додатків, яка спрощує процеси розгортання, управління та хостингу сайтів. Netlify пропонує безліч інструментів та функцій, що роблять розробку швидкою та ефективною.

Обираючи Netlify для розгортання клієнтської частини, користуємося кількома перевагами, які ця платформа надає:

1. Миттєве розгортання (завдяки простій інтеграції з Git-репозиторіями, Netlify дозволяє миттєво розгортати додатки та попередні перегляди);

2. Автоматичне масштабування (платформа автоматично масштабує додатки, забезпечуючи їх стабільну роботу під час пікових навантажень);

3. Глобальна доступність (завдяки мережі CDN, додатки на Netlify доступні користувачам по всьому світу з мінімальними затримками);

4. Простота використання (інтуїтивно зрозумілий інтерфейс дозволяє легко керувати проектами, відстежувати продуктивність та впроваджувати зміни);

5. Розширена функціональність (Netlify Functions дозволяють додавати серверні функції до додатків без необхідності налаштування серверів);

6. Інтеграція з популярними інструментами (підтримка інтеграцій з популярними фреймворками та інструментами робить Netlify зручним вибором для розробників).

Система повинна забезпечувати оптимальну функціональність на різних пристроях, включаючи комп'ютери, смартфони, планшети та інші пристрої, тому мінімальна комплектація для браузерів (Chrome, Opera, Edge, Mozilla Firefox) має такі вимоги:

1. 4024 Мб оперативної пам'яті (оптимальний рівень оперативної пам'яті дозволяє запускати та працювати без перешкод);

2. 300 Мб вільного дискового простору (наявність вільного дискового простору забезпечить безперебійну роботу додатків без необхідності постійного очищення кешу);

3. Процесор з тактовою частотою 1GHz;

4. Операційна система Windows 10, Mac OS X 10.6, GNU/Linux, Android 10, IOS 17.

### **Висновки до першого розділу**

Розглянуто особливості аналогічних веб-сервісів, їхні переваги та недоліки, після чого було визначено основні вимоги, які потрібно виконати, щоб створити веб-сервіс для спільного перегляду фільмів та серіалів у режимі реального часу.

Було обрано архітектуру веб-сервісу – клієнт-серверну, а також інструментарій для реалізації системи. Серверна частина буде розроблятися з використанням мови програмування JavaScript та бібліотеки Express.js, а клієнтська – бібліотеки React.js.

## РОЗДІЛ 2 ПРОЕКТУВАННЯ ВЕБ-СЕРВІСУ ДЛЯ СПІЛЬНОГО ПЕРЕГЛЯДУ ФІЛЬМІВ ТА СЕРІАЛІВ

### 2.1 Визначення варіантів використання та об'єктно-орієнтованої структури

Веб-сервіс для спільного перегляду фільмів та серіалів в режимі реального часу є веб-додатком, через який користувачі можуть переглядати різний відеоконтент (інформацію про нього і перегляд), створювати кімнати для спільного перегляду та взаємодіяти між собою (голосовий або текстовий чат). Функціональність системи залежить від активності користувачів, які поділяються на три основні ролі: адміністратори, користувачі та власники кімнати.

Адміністратори відповідають за управління контентом та забезпечення стабільної роботи платформи. Вони можуть додавати нові фільми, серіали і т.п, редагувати та видаляти існуючий контент.

Користувачі мають можливість переглядати, сортувати, шукати різний відеоконтент, переглядати акторів та режисерів, створювати кімнати для спільного перегляду. Для доступу для повного функціоналу сервісу, включаючи перегляд відеоконтенту та створення кімнат, користувачі повинні мати активну підписку. Кімнати дозволяють користувачам переглядати різний відеоконтент та спілкуватися між собою за допомогою голосового чи текстового чату.

Власники кімнати мають можливість керувати відтворенням та кімнатою, змінювати відеоконтент для перегляду та інформацію про кімнату.

Для наглядної взаємодії між елементами веб-додатку було створено діаграму варіантів використання (Use Case Diagram), яка відображає основні дії та взаємодії між адміністраторами та користувачами (рис. 2.1 - 2.3).

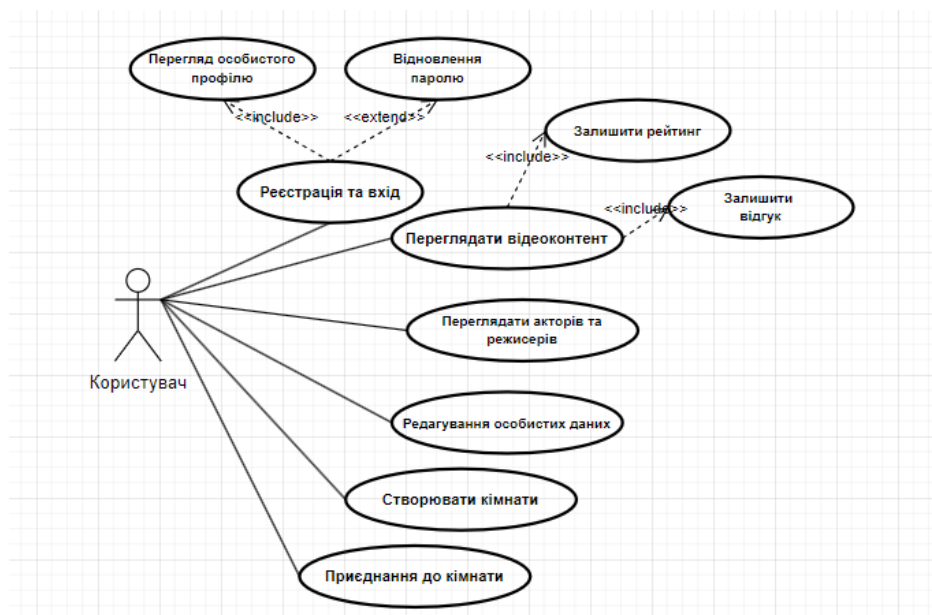


Рисунок 2.1 – Діаграма варіантів використання (Користувач)

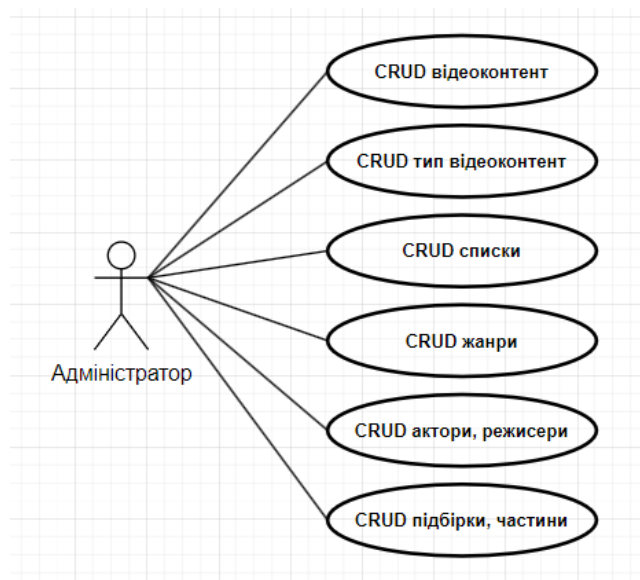


Рисунок 2.2 – Діаграма варіантів використання (Адміністратор)



Рисунок 2.3 – Діаграма варіантів використання (Власник кімнати)

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Опис діаграми варіантів використання наведено в табл. 2.1 – 2.15 у вигляді прецедентів.

Таблиця 2.1

### Прецедент реєстрації

|                                        |               |                                                                                                    |
|----------------------------------------|---------------|----------------------------------------------------------------------------------------------------|
| Ім'я варіанту використання: Реєстрація |               | Рівень важливості:<br>високий                                                                      |
| Основний актор: користувач             |               |                                                                                                    |
| Короткий опис:                         |               | Реєстрація з внесенням особистих даних користувача, перевірка існування такого ж облікового запису |
| Передумови:                            |               | Неавторизований користувач                                                                         |
| Пост-умови:                            |               |                                                                                                    |
| Відносини:                             |               |                                                                                                    |
|                                        | Асоціація:    |                                                                                                    |
|                                        | Включення:    |                                                                                                    |
|                                        | Розширення:   |                                                                                                    |
|                                        | Узагальнення: |                                                                                                    |
| Основний потік:                        |               | Користувачу надається доступ до веб-додатку та його дані зберігаються в базі даних                 |
| Альтернативні/виключні потоки:         |               | Якщо сталася помилка під час реєстрації, то потрібно проінформувати користувача                    |

Таблиця 2.2

### Прецедент входу

|                                  |  |                                                                                                   |
|----------------------------------|--|---------------------------------------------------------------------------------------------------|
| Ім'я варіанту використання: Вхід |  | Рівень важливості:<br>високий                                                                     |
| Основний актор: користувач       |  |                                                                                                   |
| Короткий опис:                   |  | Введення електронної пошти та паролю користувача, за потреби можливість входу за допомогою Google |
| Передумови:                      |  | Неавторизований користувач                                                                        |

|                                |               |                                                                            |
|--------------------------------|---------------|----------------------------------------------------------------------------|
| Пост-умови:                    |               |                                                                            |
| Відносини:                     |               |                                                                            |
|                                | Асоціація:    |                                                                            |
|                                | Включення:    |                                                                            |
|                                | Розширення:   |                                                                            |
|                                | Узагальнення: |                                                                            |
| Основний потік:                |               | Користувачу надається доступ до веб-додатку                                |
| Альтернативні/виключні потоки: |               | Якщо сталася помилка під час входу, то потрібно проінформувати користувача |

Таблиця 2.3

### Прецедент перегляду особистого профілю

|                                                         |               |                                                                                                                                                                                      |
|---------------------------------------------------------|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Ім'я варіанту використання: Перегляд особистого профілю |               | Рівень важливості:<br>високий                                                                                                                                                        |
| Основний актор: користувач                              |               |                                                                                                                                                                                      |
| Короткий опис:                                          |               | Користувач має можливість переглядати інформацію про себе, змінювати свої персональні дані, переглядати власні відгуки, збережений відеоконтент, зміна паролю та керування підпискою |
| Передумови:                                             |               | Авторизований користувач                                                                                                                                                             |
| Пост-умови:                                             |               |                                                                                                                                                                                      |
| Відносини:                                              |               |                                                                                                                                                                                      |
|                                                         | Асоціація:    |                                                                                                                                                                                      |
|                                                         | Включення:    |                                                                                                                                                                                      |
|                                                         | Розширення:   |                                                                                                                                                                                      |
|                                                         | Узагальнення: |                                                                                                                                                                                      |
| Основний потік:                                         |               | Користувачу надається доступ до веб-додатку                                                                                                                                          |
| Альтернативні/виключні потоки:                          |               | Якщо сталася помилка під час перегляду, то потрібно проінформувати користувача                                                                                                       |

Таблиця 2.4

## Прецендент перегляду відеоконтенту

|                                                    |               |                                                                                                                                                                                                                                          |
|----------------------------------------------------|---------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Ім'я варіанту використання: Перегляд відеоконтенту |               | Рівень важливості:<br>високий                                                                                                                                                                                                            |
| Основний актор: користувач                         |               |                                                                                                                                                                                                                                          |
| Короткий опис:                                     |               | Користувачу відображається список відеоконтенту на головній сторінці, він має можливість сортувати та фільтрувати відеоконтент, переглядати інформацію про відеоконтент, дивитися відеоконтент, залишати відгуки та оцінку відеоконтенту |
| Передумови:                                        |               | Авторизований користувач                                                                                                                                                                                                                 |
| Пост-умови:                                        |               |                                                                                                                                                                                                                                          |
| Відносини:                                         |               |                                                                                                                                                                                                                                          |
|                                                    | Асоціація:    |                                                                                                                                                                                                                                          |
|                                                    | Включення:    |                                                                                                                                                                                                                                          |
|                                                    | Розширення:   |                                                                                                                                                                                                                                          |
|                                                    | Узагальнення: |                                                                                                                                                                                                                                          |
| Основний потік:                                    |               | Користувачу надається доступ до веб-додатку                                                                                                                                                                                              |
| Альтернативні/виключні потоки:                     |               | Якщо сталася помилка під час перегляду, то потрібно проінформувати користувача                                                                                                                                                           |

Таблиця 2.5

## Прецендент перегляду акторів та режисерів

|                                                           |                                                                                                                                                                |                               |
|-----------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------|
| Ім'я варіанту використання: Перегляд акторів та режисерів |                                                                                                                                                                | Рівень важливості:<br>високий |
| Основний актор: користувач                                |                                                                                                                                                                |                               |
| Короткий опис:                                            | Користувачу відображається інформація про актора чи режисера та відеоконтент, в якому вони брали участь, також цей відеоконтент можна фільтрувати та сортувати |                               |
| Передумови:                                               | Авторизований користувач                                                                                                                                       |                               |

|                                |               |                                                                                |
|--------------------------------|---------------|--------------------------------------------------------------------------------|
| Пост-умови:                    |               |                                                                                |
| Відносини:                     |               |                                                                                |
|                                | Асоціація:    |                                                                                |
|                                | Включення:    |                                                                                |
|                                | Розширення:   |                                                                                |
|                                | Узагальнення: |                                                                                |
| Основний потік:                |               | Користувачу надається доступ до веб-додатку                                    |
| Альтернативні/виключні потоки: |               | Якщо сталася помилка під час перегляду, то потрібно проінформувати користувача |

Таблиця 2.6

### Прецедент створення кімнати

|                                               |               |                                                                                                                                                                |
|-----------------------------------------------|---------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Ім'я варіанту використання: Створення кімнати |               | Рівень важливості:<br>високий                                                                                                                                  |
| Основний актор: користувач                    |               |                                                                                                                                                                |
| Короткий опис:                                |               | Користувач обирає відеоконтент відповідно і заповнює форму для створення кімнати                                                                               |
| Передумови:                                   |               | Авторизований користувач                                                                                                                                       |
| Пост-умови:                                   |               |                                                                                                                                                                |
| Відносини:                                    |               |                                                                                                                                                                |
|                                               | Асоціація:    |                                                                                                                                                                |
|                                               | Включення:    |                                                                                                                                                                |
|                                               | Розширення:   |                                                                                                                                                                |
|                                               | Узагальнення: |                                                                                                                                                                |
| Основний потік:                               |               | Користувачу відображається форма створення кімнати, яку він повинен заповнити, щоб створити її, після чого виводиться повідомлення та переадресація на кімнату |

|                                |                                                                                |
|--------------------------------|--------------------------------------------------------------------------------|
| Альтернативні/виключні потоки: | Якщо сталася помилка під час створення, то потрібно проінформувати користувача |
|--------------------------------|--------------------------------------------------------------------------------|

Таблиця 2.7

### Прецендент під'єднання до кімнати

|                                                    |               |                                                                                                                                               |
|----------------------------------------------------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------|
| Ім'я варіанту використання: Під'єднання до кімнати |               | Рівень важливості:<br>високий                                                                                                                 |
| Основний актор: користувач                         |               |                                                                                                                                               |
| Короткий опис:                                     |               | Користувач переходить по запрошувальному посилання та під'єднується до кімнати                                                                |
| Передумови:                                        |               | Авторизований користувач                                                                                                                      |
| Пост-умови:                                        |               |                                                                                                                                               |
| Відносини:                                         |               |                                                                                                                                               |
|                                                    | Асоціація:    |                                                                                                                                               |
|                                                    | Включення:    |                                                                                                                                               |
|                                                    | Розширення:   |                                                                                                                                               |
|                                                    | Узагальнення: |                                                                                                                                               |
| Основний потік:                                    |               | Користувачу відображається інформація про кімнату (кількість користувачів і т.д), він повинен ввести пароль чи просто під'єднатися до кімнати |
| Альтернативні/виключні потоки:                     |               | Якщо сталася помилка під час приєднування до кімнати, то потрібно проінформувати користувача                                                  |

Таблиця 2.8

### Прецендент CRUD відеоконтенту

|                                               |  |                               |
|-----------------------------------------------|--|-------------------------------|
| Ім'я варіанту використання: CRUD відеоконтент |  | Рівень важливості:<br>високий |
| Основний актор: адміністратор                 |  |                               |

|                                |                                                                                             |
|--------------------------------|---------------------------------------------------------------------------------------------|
| Короткий опис:                 | Адміністратор керує відеоконтентом, може переглядати, створювати, редагувати та видаляти їх |
| Передумови:                    | Авторизований користувач                                                                    |
| Пост-умови:                    |                                                                                             |
| Відносини:                     |                                                                                             |
|                                | Асоціація:                                                                                  |
|                                | Включення:                                                                                  |
|                                | Розширення:                                                                                 |
|                                | Узагальнення:                                                                               |
| Основний потік:                | Адміністратор отримує результат без будь-яких помилок                                       |
| Альтернативні/виключні потоки: | Якщо сталася помилка, то потрібно проінформувати адміністратора                             |

Таблиця 2.9

### Прецедент CRUD типу відеоконтенту

|                                                   |               |                                                                                                   |
|---------------------------------------------------|---------------|---------------------------------------------------------------------------------------------------|
| Ім'я варіанту використання: CRUD тип відеоконтент |               | Рівень важливості:<br>високий                                                                     |
| Основний актор: адміністратор                     |               |                                                                                                   |
| Короткий опис:                                    |               | Адміністратор керує типами відеоконтенту, може переглядати, створювати, редагувати та видаляти їх |
| Передумови:                                       |               | Авторизований користувач                                                                          |
| Пост-умови:                                       |               |                                                                                                   |
| Відносини:                                        |               |                                                                                                   |
|                                                   | Асоціація:    |                                                                                                   |
|                                                   | Включення:    |                                                                                                   |
|                                                   | Розширення:   |                                                                                                   |
|                                                   | Узагальнення: |                                                                                                   |

|                                |                                                                 |
|--------------------------------|-----------------------------------------------------------------|
| Основний потік:                | Адміністратор отримує результат без будь-яких помилок           |
| Альтернативні/виключні потоки: | Якщо сталася помилка, то потрібно проінформувати адміністратора |

Таблиця 2.10

## Прецедент CRUD списків

|                                         |                                                                                       |                               |
|-----------------------------------------|---------------------------------------------------------------------------------------|-------------------------------|
| Ім'я варіанту використання: CRUD списки |                                                                                       | Рівень важливості:<br>високий |
| Основний актор: адміністратор           |                                                                                       |                               |
| Короткий опис:                          | Адміністратор керує списками, може переглядати, створювати, редагувати та видаляти їх |                               |
| Передумови:                             | Авторизований користувач                                                              |                               |
| Пост-умови:                             |                                                                                       |                               |
| Відносини:                              |                                                                                       |                               |
|                                         | Асоціація:                                                                            |                               |
|                                         | Включення:                                                                            |                               |
|                                         | Розширення:                                                                           |                               |
|                                         | Узагальнення:                                                                         |                               |
| Основний потік:                         | Адміністратор отримує результат без будь-яких помилок                                 |                               |
| Альтернативні/виключні потоки:          | Якщо сталася помилка, то потрібно проінформувати адміністратора                       |                               |

Таблиця 2.11

## Прецедент CRUD жанрів

|                                        |  |                               |
|----------------------------------------|--|-------------------------------|
| Ім'я варіанту використання: CRUD жанри |  | Рівень важливості:<br>високий |
| Основний актор: адміністратор          |  |                               |

|                                |                                                                                      |
|--------------------------------|--------------------------------------------------------------------------------------|
| Короткий опис:                 | Адміністратор керує жанрами, може переглядати, створювати, редагувати та видаляти їх |
| Передумови:                    | Авторизований користувач                                                             |
| Пост-умови:                    |                                                                                      |
| Відносини:                     |                                                                                      |
|                                | Асоціація:                                                                           |
|                                | Включення:                                                                           |
|                                | Розширення:                                                                          |
|                                | Узагальнення:                                                                        |
| Основний потік:                | Адміністратор отримує результат без будь-яких помилок                                |
| Альтернативні/виключні потоки: | Якщо сталася помилка, то потрібно проінформувати адміністратора                      |

Таблиця 2.12

### Прецедент CRUD акторів, режисерів

|                                                   |                                                                                                     |  |  |                               |
|---------------------------------------------------|-----------------------------------------------------------------------------------------------------|--|--|-------------------------------|
| Ім'я варіанту використання: CRUD актори, режисери |                                                                                                     |  |  | Рівень важливості:<br>високий |
| Основний актор: адміністратор                     |                                                                                                     |  |  |                               |
| Короткий опис:                                    | Адміністратор керує акторами та режисерами, може переглядати, створювати, редагувати та видаляти їх |  |  |                               |
| Передумови:                                       | Авторизований користувач                                                                            |  |  |                               |
| Пост-умови:                                       |                                                                                                     |  |  |                               |
| Відносини:                                        |                                                                                                     |  |  |                               |
|                                                   | Асоціація:                                                                                          |  |  |                               |
|                                                   | Включення:                                                                                          |  |  |                               |
|                                                   | Розширення:                                                                                         |  |  |                               |
|                                                   | Узагальнення:                                                                                       |  |  |                               |

|                                |                                                                 |
|--------------------------------|-----------------------------------------------------------------|
| Основний потік:                | Адміністратор отримує результат без будь-яких помилок           |
| Альтернативні/виключні потоки: | Якщо сталася помилка, то потрібно проінформувати адміністратора |

Таблиця 2.13

### Прецедент CRUD підбірок, частин

|                                                    |                                                                                                      |                               |
|----------------------------------------------------|------------------------------------------------------------------------------------------------------|-------------------------------|
| Ім'я варіанту використання: CRUD підбірки, частини |                                                                                                      | Рівень важливості:<br>високий |
| Основний актор: адміністратор                      |                                                                                                      |                               |
| Короткий опис:                                     | Адміністратор керує підбірками та частинами, може переглядати, створювати, редагувати та видаляти їх |                               |
| Передумови:                                        | Авторизований користувач                                                                             |                               |
| Пост-умови:                                        |                                                                                                      |                               |
| Відносини:                                         |                                                                                                      |                               |
|                                                    | Асоціація:                                                                                           |                               |
|                                                    | Включення:                                                                                           |                               |
|                                                    | Розширення:                                                                                          |                               |
|                                                    | Узагальнення:                                                                                        |                               |
| Основний потік:                                    | Адміністратор отримує результат без будь-яких помилок                                                |                               |
| Альтернативні/виключні потоки:                     | Якщо сталася помилка, то потрібно проінформувати адміністратора                                      |                               |

Таблиця 2.14

### Прецедент керування відтворенням

|                                                    |                                                                  |                               |
|----------------------------------------------------|------------------------------------------------------------------|-------------------------------|
| Ім'я варіанту використання: Керування відтворенням |                                                                  | Рівень важливості:<br>високий |
| Основний актор: власник кімнати                    |                                                                  |                               |
| Короткий опис:                                     | Власник кімнати керує відтворенням (перемотування та його зміна) |                               |

|                                |                                                                 |
|--------------------------------|-----------------------------------------------------------------|
| Передумови:                    | Авторизований користувач                                        |
| Пост-умови:                    |                                                                 |
| Відносини:                     |                                                                 |
|                                | Асоціація:                                                      |
|                                | Включення:                                                      |
|                                | Розширення:                                                     |
|                                | Узагальнення:                                                   |
| Основний потік:                | Власник кімнати керує відтворенням без будь-яких помилок        |
| Альтернативні/виключні потоки: | Якщо сталася помилка, то потрібно проінформувати адміністратора |

Таблиця 2.15

## Прецедент керування кімнатою

|                                                |               |                                                                           |
|------------------------------------------------|---------------|---------------------------------------------------------------------------|
| Ім'я варіанту використання: Керування кімнатою |               | Рівень важливості:<br>високий                                             |
| Основний актор: власник кімнати                |               |                                                                           |
| Короткий опис:                                 |               | Власник кімнати керує кімнатою (редагування та завершення сеансу кімнати) |
| Передумови:                                    |               | Авторизований користувач                                                  |
| Пост-умови:                                    |               |                                                                           |
| Відносини:                                     |               |                                                                           |
|                                                | Асоціація:    |                                                                           |
|                                                | Включення:    |                                                                           |
|                                                | Розширення:   |                                                                           |
|                                                | Узагальнення: |                                                                           |
| Основний потік:                                |               | Власник кімнати керує кімнатою без будь-яких помилок                      |
| Альтернативні/виключні потоки:                 |               | Якщо сталася помилка, то потрібно проінформувати адміністратора           |

Основна ціль розробки веб-сервісу для спільного перегляду фільмів та серіалів в режимі реального часу полягає в створенні інтерактивного онлайн-середовища, яке дозволяє користувачам разом насолоджуватися відеоконтентом незалежно від їхнього географічного місця знаходження. Платформа спрямована на забезпечення ефективної комунікації та взаємодії між користувачами, створення зручних умов для перегляду та обговорення фільмів і серіалів.

Завдяки цьому, основною метою є створення умов для соціальної інтеракції та спільного перегляду відеоконтенту, що може покращити досвід користувачів, об'єднати їх у спільноти та сприяти формуванню нових знайомств та дружби. Платформа також надає можливість користувачам ділитися своїми враженнями та відкриттями у світі кіно та серіалів, що робить процес перегляду більш інтерактивним та захоплюючим.

Вимоги користувача:

1. Створювати та редагувати кімнати для спільного перегляду;
2. Можливість переглядати та редагувати особисті дані;
3. Можливість переглядати фільми, серіали без створення кімнати;
4. Можливість переглядати акторів та режисерів;
5. Можливість залишати відгуки під різними фільмами, серіалами.

Вимоги адміністратора:

1. Можливість створення, редагування та видалення відеоконтенту;
2. Можливість створення, редагування та видалення списків;
3. Можливість створення, редагування та видалення акторів та режисерів;
4. Можливість створення, редагування та видалення підбіроків та частин.

Функціональні вимоги:

1. Авторизація у системі: в системі повинна бути представлена можливість реєстрації користувача;

2. Можливість збереження інформації: система повинна зберігати інформацію і надавати можливість користувачу керувати нею.

Нефункціональні вимоги:

1. Сприйняття

час, необхідний для навчання звичайних користувачів – 10-15 хвилин, а для навчання просунутих користувачів – 2-3 хвилини;

час відгуку для типових задач – не більше 15 секунд, для складних завдань – не більше 30 секунд;

інтерфейс має бути зрозумілим та з відповідними підказками.

## 2. Надійність

доступність – час, що витрачається на обслуговування системи не повинно перевищувати 5% від загального часу роботи;

середній час безвідмовної роботи – 15 робочих днів;

максимальна норма помилок та дефектів в роботі системи – 1 помилка на десять тисяч рядків коду.

## 3. Продуктивність

система повинна підтримувати мінімум 100 одночасно працюючих користувачів, пов'язаних із загальною базою даних.

## 4. Можливість експлуатації

масштабування – система повинна мати можливість збільшувати потужності (продуктивність), зі збільшенням користувачів таким чином, щоб це не вплинуло негативно на її роботу.

При аналізі функціональних вимог веб-сервісу було виділено сутності, які дозволили забезпечити програмну реалізацію веб-додатку. На рис. 2.4 - 2.5 наведено діаграму класів (Class Diagram), яка використовується для роботи веб-сервісу.

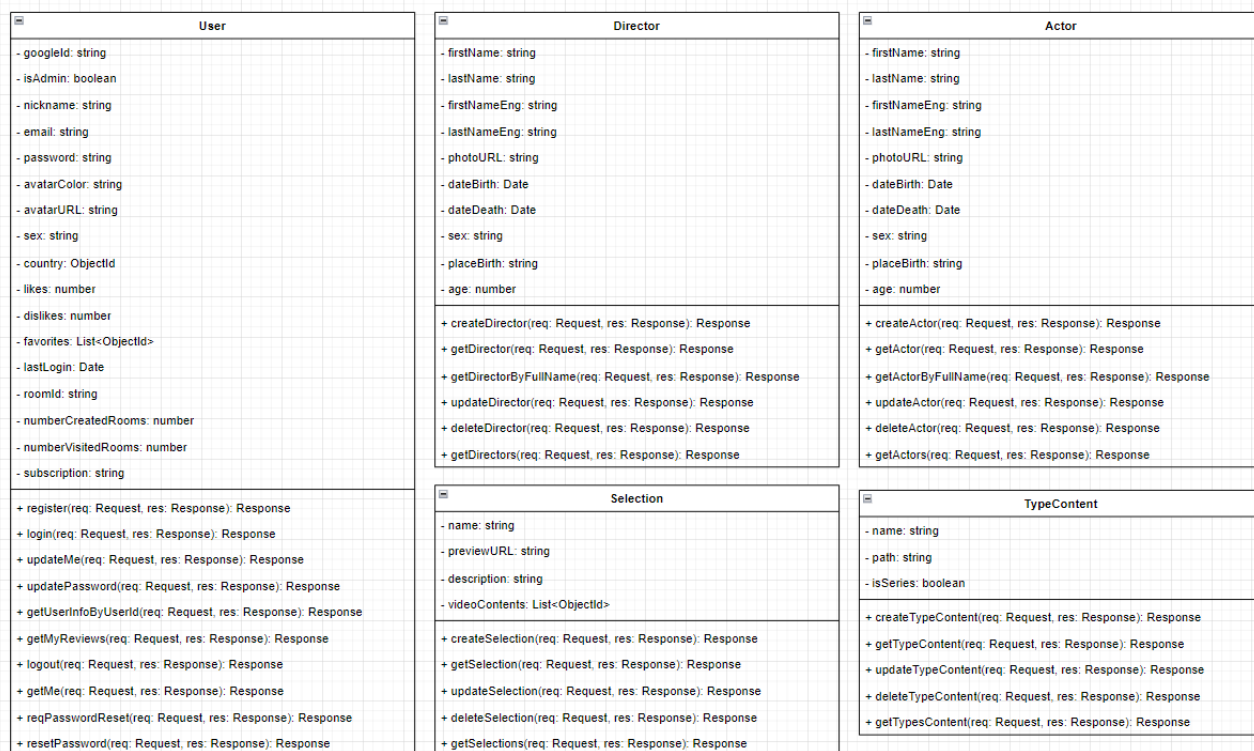


Рисунок 2.4 – Діаграма класів

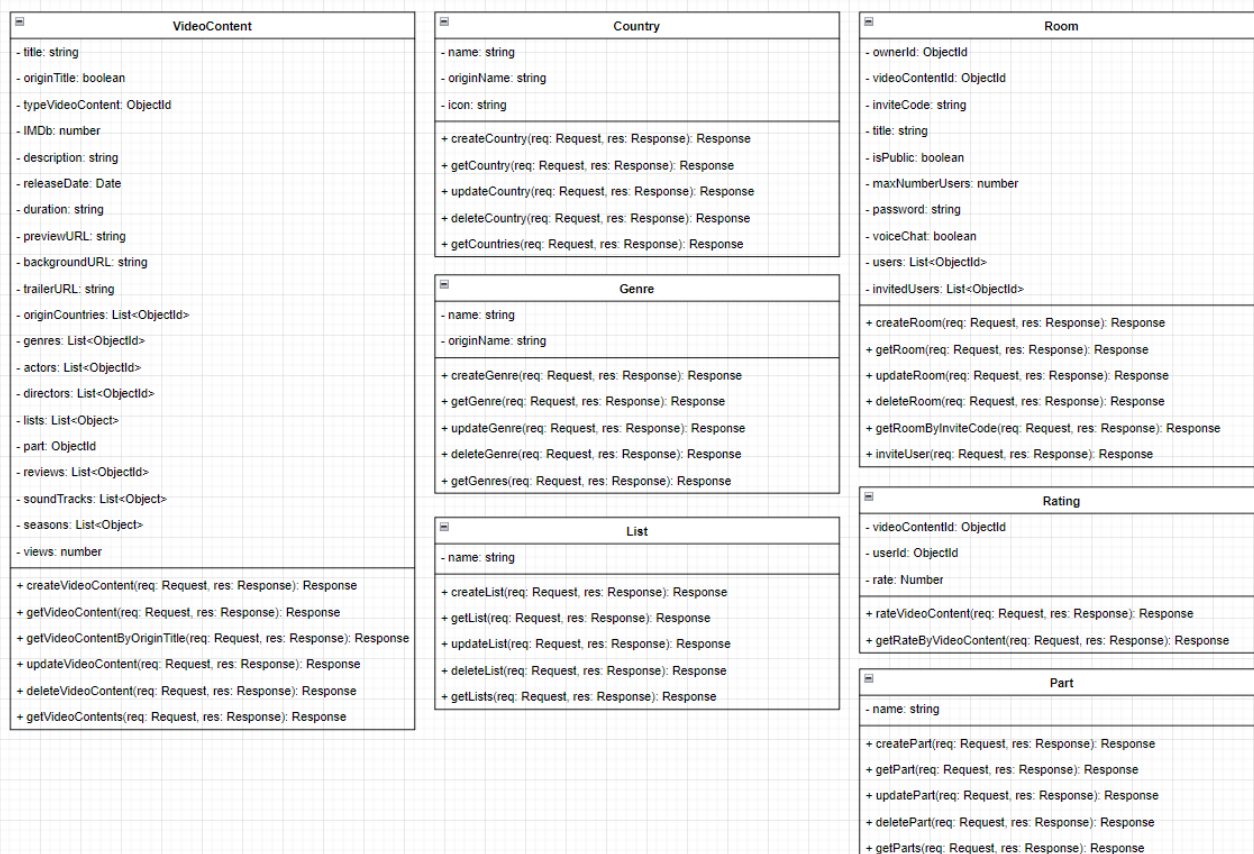


Рисунок 2.5 – Діаграма класів (Продовження)

Опишемо діаграму класів.

1. User – клас відповідає за користувача:

register() відповідає за реєстрацію користувача;

login() відповідає за вхід користувача;

updateMe() відповідає за оновлення користувача;

updatePassword() відповідає за оновлення паролю користувача;

getUserInfoByUserId() відповідає за отримання інформації про користувача по ідентифікатору;

getMyReviews() відповідає за отримання відгуків користувача;

logout() відповідає за вихід користувача з веб-додатку;

getMe() відповідає за отримання інформації про користувача;

reqPasswordReset() відповідає за запит скидання пароля користувача;

resetPassword() відповідає за скидання пароля користувача.

2. Director – клас відповідає за режисера:

createDirector() відповідає за створення режисера;

getDirector() відповідає за отримання режисера по ідентифікатору;

getDirectorByFullName() відповідає за отримання режисера по повному імені;

updateDirector() відповідає за оновлення режисера;

deleteDirector() відповідає за видалення режисера;

getDirectors() відповідає за отримання режисерів.

3. Selection – клас відповідає за підбірки:

createSelection() відповідає за створення підбірки;

getSelection() відповідає за отримання підбірки;

updateSelection() відповідає за оновлення підбірки;

deleteSelection() відповідає за видалення підбірки;

getSelections() відповідає за отримання підбірок.

4. Actor – клас відповідає за актора:

createActor() відповідає за створення актора;

getActor() відповідає за отримання актора по ідентифікатору;

getActorByFullName() відповідає за отримання актора по повному імені;

updateActor() відповідає за оновлення актора;

deleteActor() відповідає за видалення актора;

getActors() відповідає за отримання акторів.

5. TypeContent - клас відповідає за тип відеоконтенту:

createTypeContent() відповідає за створення типу відеоконтенту;

getTypeContent() відповідає за отримання типу відеоконтенту по ідентифікатору;

updateTypeContent() відповідає за оновлення типу відеоконтенту;

deleteTypeContent() відповідає за видалення типу відеоконтенту;

getTypesContent() відповідає за отримання типів відеоконтенту.

6. VideoContent – клас відповідає за відеоконтент:

createVideoContent() відповідає за створення відеоконтенту;

getVideoContent() відповідає за отримання відеоконтенту по ідентифікатору;

getVideoContentByOriginTitle() відповідає за отримання відеоконтенту по оригінальній назві;

updateVideoContent() відповідає за оновлення відеоконтенту;

deleteVideoContent() відповідає за видалення відеоконтенту;

getVideoContents() відповідає за отримання відеоконтенту.

7. Country – клас відповідає за країни:

createCountry() відповідає за створення країни;

getCountry() відповідає за отримання країни по ідентифікатору;

updateCountry() відповідає за оновлення країни;

deleteCountry() відповідає за видалення країни;

getCountries() відповідає за отримання країн.

8. Genre – клас відповідає за жанри:

createGenre() відповідає за створення жанру;

getGenre() відповідає за отримання жанру по ідентифікатору;

updateGenre() відповідає за оновлення жанру;

deleteGenre() відповідає за видалення жанру;

getGenres() відповідає за отримання жанрів.

9. List – клас відповідає за списки:

createList() відповідає за створення списку;

getList() відповідає за отримання списку по ідентифікатору;

updateList() відповідає за оновлення списку;

deleteList() відповідає за видалення списку;

getLists() відповідає за отримання списків.

10. Room – клас відповідає за кімнати:

createRoom() відповідає за створення кімнати;

getRoom() відповідає за отримання кімнати по ідентифікатору;

updateRoom() відповідає за оновлення кімнати;

deleteRoom() відповідає за видалення кімнати;

getRoomByInviteCode() відповідає за отримання кімнати по запрошувальному коду;

inviteUser() відповідає за додавання користувача, як запрошеного.

11. Rating – клас відповідає за рейтинг відеоконтенту:

rateVideoContent() відповідає за додавання рейтингу до відеоконтенту;

getRateByVideoContent() відповідає за отримання рейтингу відеоконтенту.

12. Part – клас відповідає за частину

createPart() відповідає за створення частини;

getPart() відповідає за отримання частини по ідентифікатору;

updatePart() відповідає за оновлення частини;

deletePart() відповідає за видалення частини;

getParts() відповідає за отримання частин.

## 2.2 Розробка бази даних системи

Для збереження файлів та інформації у веб-сервісі для спільного перегляду фільмів та серіалів в режимі реального часу використовується MongoDB, нереляційна система управління базами даних, яка базується на моделі документів. MongoDB надає гнучкі можливості для збереження структурованих і

неструктурованих даних, що дозволяє ефективно керувати великими обсягами інформації.

База даних системи складається з 15 колекцій:

1. Actor – колекція, що зберігає інформацію про акторів;
2. VideoContent – колекція, що зберігає інформацію про відеоконтент;
3. Country – колекція, що зберігає інформацію про країни;
4. Director – колекція, що зберігає інформацію про режисерів;
5. Genre – колекція, що зберігає інформацію про жанри;
6. List – колекція, що зберігає інформацію про списки;
7. Message – колекція, що зберігає інформацію про повідомлення кімнат;
8. Part – колекція, що зберігає інформацію про частини;
9. Rating – колекція, що зберігає інформацію про рейтинг;
10. Review – колекція, що зберігає інформацію про відгуки користувачів;
11. Room – колекція, що зберігає інформацію про кімнати;
12. Selection – колекція, що зберігає інформацію про підбірки;
13. Token – колекція, що зберігає інформацію про токени скидання паролю;
14. TypeContent – колекція, що зберігає інформацію про типи відеоконтенту;
15. User – колекція, що зберігає інформацію про користувачів.

Опис кожної з колекцій бази даних наведений у табл. 2.16 - 2.30.

Таблиця 2.16

Опис колекції Actors

| Поле         | Тип      | Призначення             |
|--------------|----------|-------------------------|
| _id          | ObjectId | Ідентифікатор           |
| firstName    | string   | Ім'я актора             |
| lastName     | string   | Прізвище актора         |
| firstNameEng | string   | Ім'я актора (англ.)     |
| lastNameEng  | string   | Прізвище актора (англ.) |
| photoURL     | string   | Фотографія актора       |
| dateBirth    | Date     | Дата народження актора  |

|            |        |                         |
|------------|--------|-------------------------|
| dateDeath  | Date   | Дата смерті актора      |
| sex        | string | Стать актора            |
| placeBirth | string | Місце народження актора |
| age        | number | Вік актора              |

Таблиця 2.17

## Опис колекції VideoContents

| Поле             | Тип        | Призначення                 |
|------------------|------------|-----------------------------|
| _id              | ObjectId   | Ідентифікатор               |
| title            | string     | Назва відеоконтенту         |
| originTitle      | string     | Назва відеоконтенту (англ.) |
| typeVideoContent | ObjectId   | Тип відеоконтенту           |
| IMDb             | number     | Рейтинг IMDb відеоконтенту  |
| description      | string     | Опис відеоконтенту          |
| releaseDate      | Date       | Дата релізу відеоконтенту   |
| duration         | string     | Тривалість відеоконтенту    |
| previewURL       | string     | Обкладинка відеоконтенту    |
| backgroundURL    | string     | Фото відеоконтенту          |
| trailerURL       | string     | Трейлер відеоконтенту       |
| originCountries  | ObjectId[] | Країни відеоконтенту        |
| genres           | ObjectId[] | Жанри відеоконтенту         |
| actors           | ObjectId[] | Актори відеоконтенту        |
| directors        | ObjectId[] | Режисери відеоконтенту      |
| lists            | Object[]   | Списки відеоконтенту        |
| part             | ObjectId   | Частина відеоконтенту       |
| reviews          | ObjectId[] | Відгуки відеоконтенту       |
| soundTracks      | Object[]   | Доріжки відеоконтенту       |
| seasons          | Object[]   | Сезони відеоконтенту        |
| views            | number     | Перегляди відеоконтенту     |

Таблиця 2.18

## Опис колекції Countries

| Поле       | Тип      | Призначення          |
|------------|----------|----------------------|
| _id        | ObjectId | Ідентифікатор        |
| name       | string   | Назва країни         |
| originName | string   | Назва країни (англ.) |
| icon       | string   | Іконка країни        |

Таблиця 2.19

## Опис колекції Directors

| Поле         | Тип      | Призначення               |
|--------------|----------|---------------------------|
| _id          | ObjectId | Ідентифікатор             |
| firstName    | string   | Ім'я режисера             |
| lastName     | string   | Прізвище режисера         |
| firstNameEng | string   | Ім'я режисера (англ.)     |
| lastNameEng  | string   | Прізвище режисера (англ.) |
| photoURL     | string   | Фотографія режисера       |
| dateBirth    | Date     | Дата народження режисера  |
| dateDeath    | Date     | Дата смерті режисера      |
| sex          | string   | Стать режисера            |
| placeBirth   | string   | Місце народження режисера |
| age          | number   | Вік режисера              |

Таблиця 2.20

## Опис колекції Genres

| Поле       | Тип      | Призначення         |
|------------|----------|---------------------|
| _id        | ObjectId | Ідентифікатор       |
| name       | string   | Назва жанру         |
| originName | string   | Назва жанру (англ.) |

Таблиця 2.21

## Опис колекції Lists

| Поле | Тип      | Призначення   |
|------|----------|---------------|
| _id  | ObjectId | Ідентифікатор |
| name | string   | Назва списку  |

Таблиця 2.22

## Опис колекції Messages

| Поле    | Тип      | Призначення               |
|---------|----------|---------------------------|
| _id     | ObjectId | Ідентифікатор             |
| roomId  | ObjectId | Ідентифікатор кімнати     |
| userId  | ObjectId | Ідентифікатор користувача |
| message | string   | Повідомлення користувача  |

Таблиця 2.23

## Опис колекції Parts

| Поле | Тип      | Призначення   |
|------|----------|---------------|
| _id  | ObjectId | Ідентифікатор |
| name | string   | Назва частини |

Таблиця 2.24

## Опис колекції Ratings

| Поле           | Тип      | Призначення                 |
|----------------|----------|-----------------------------|
| _id            | ObjectId | Ідентифікатор               |
| videoContentId | ObjectId | Ідентифікатор відеоконтенту |
| userId         | ObjectId | Ідентифікатор користувача   |
| rate           | number   | Оцінка користувача          |

Таблиця 2.25

## Опис колекції Reviews

| Поле      | Тип      | Призначення                 |
|-----------|----------|-----------------------------|
| _id       | ObjectId | Ідентифікатор               |
| userId    | ObjectId | Ідентифікатор користувача   |
| contentId | ObjectId | Ідентифікатор відеоконтенту |
| message   | string   | Відгук користувача          |
| likes     | number   | Вподобань відгука           |
| dislikes  | number   | Негативність відгука        |

Таблиця 2.26

## Опис колекції Rooms

| Поле           | Тип        | Призначення                  |
|----------------|------------|------------------------------|
| _id            | ObjectId   | Ідентифікатор                |
| ownerId        | ObjectId   | Ідентифікатор користувача    |
| videoContentId | ObjectId   | Ідентифікатор відеоконтенту  |
| inviteCode     | string     | Запрошувальний код           |
| title          | string     | Назва кімнати                |
| isPublic       | boolean    | Публічність кімнати          |
| maxNumberUsers | number     | Макс. кількість користувачів |
| password       | string     | Пароль кімнати               |
| voiceChat      | boolean    | Голосовий чат                |
| users          | ObjectId[] | Підключені користувачі       |
| invitedUsers   | ObjectId[] | Запрошені користувачі        |

Таблиця 2.27

## Опис колекції Selections

| Поле | Тип      | Призначення   |
|------|----------|---------------|
| _id  | ObjectId | Ідентифікатор |

|               |            |                       |
|---------------|------------|-----------------------|
| name          | string     | Назва підбірки        |
| previewURL    | ObjectId   | Обкладинка підбірки   |
| description   | string     | Опис підбірки         |
| videoContents | ObjectId[] | Відеоконтент підбірки |

Таблиця 2.28

## Опис колекції Tokens

| Поле      | Тип      | Призначення               |
|-----------|----------|---------------------------|
| _id       | ObjectId | Ідентифікатор             |
| userId    | ObjectId | Ідентифікатор користувача |
| token     | ObjectId | Токен скидання паролю     |
| createdAt | Date     | Термін дати токену        |

Таблиця 2.29

## Опис колекції TypeContents

| Поле     | Тип      | Призначення              |
|----------|----------|--------------------------|
| _id      | ObjectId | Ідентифікатор            |
| name     | string   | Назва типу відеоконтенту |
| path     | string   | Шлях типу відеоконтенту  |
| isSeries | boolean  | Серійний тип             |

Таблиця 2.30

## Опис колекції Users

| Поле     | Тип      | Призначення          |
|----------|----------|----------------------|
| _id      | ObjectId | Ідентифікатор        |
| googleId | GoogleId | Ідентифікатор Google |
| isAdmin  | boolean  | Користувач Адмін     |
| nickname | string   | Нікнейм користувача  |
| email    | string   | Пошта користувача    |

|                    |            |                              |
|--------------------|------------|------------------------------|
| password           | string     | Пароль користувача           |
| avatarColor        | string     | Колір аватару користувача    |
| avatarURL          | string     | Аватар користувача           |
| sex                | string     | Стать користувача            |
| country            | ObjectId   | Країна користувача           |
| likes              | number     | Позитивна оцінка користувача |
| dislikes           | number     | Негативна оцінка користувача |
| favorites          | ObjectId[] | Збережений відеоконтент      |
| lastLogin          | number     | Останній вхід користувача    |
| roomId             | string     | Створена кімната користувача |
| numberCreatedRooms | number     | Кількість створених кімнат   |
| numberVisitedRooms | number     | Кількість відвіданих кімнат  |
| subscription       | string     | Підписка користувача         |

### 2.3 Проектування та реалізація алгоритмів роботи системи

Як вже було зазначено, веб-додаток є зручним і простим у використанні, тому не потребує складних алгоритмів для реалізації. Це веб-додаток, який дозволяє користувачам легко взаємодіяти з веб-сервісом. Діаграма активності роботи веб-додатку зображена на рис. 2.6, а нижче описано основні дії користувачів веб-сервісу.

Якщо користувач неавторизований:

1. Користувач може переглянути інформацію про даний веб-сервіс;
2. Користувач може зареєструватися чи увійти у веб-сервіс;
3. Користувач може увійти за допомогою Google;
4. Користувач може відновити пароль.

Якщо користувач авторизований:

1. Користувач може взаємодіяти з слайдером відеоконтенту на головній сторінці;
2. Користувач може переглядати весь відеоконтент і сортувати його;

3. Користувач може переглядати конкретний відеоконтент і інформацію про нього;

4. Користувач може переглядати інформацію про акторів та режисерів;

5. Користувач може переглядати списки, до яких входить обраний ним відеоконтент;

6. Користувач може переглядати жанри, до яких входить обраний ним відеоконтент;

7. Користувач може переглядати частини, до яких входить інший відеоконтент;

8. Користувач може зберегти відеоконтент в збережене;

9. Користувач може переглянути свій профіль;

10. Користувач може редагувати особисті дані;

11. Користувач може переглядати збережений відеоконтент;

12. Користувач може переглядати власні відгуки;

13. Користувач може змінити пароль;

14. Користувач може керувати підпискою;

15. Користувач може приєднуватися до кімнати без підписки;

16. Користувач може спілкуватися в чаті кімнати.

Якщо користувач авторизований і має підписку:

1. Користувач може дивитися відеоконтент;

2. Користувач може створювати кімнати.

Якщо користувач власник кімнати:

1. Користувач може керувати відтворенням (тільки йому доступне керування);

2. Користувач може редагувати кімнату;

3. Користувач може завершити сеанс кімнати.

Якщо користувач адміністратор:

1. Можливість керувати типом відеоконтенту;

2. Можливість керувати відеоконтентом;

3. Можливість керувати країнами;

4. Можливість керувати жанрами, списками, підбірками та списками;
5. Можливість керувати акторами та режисери.

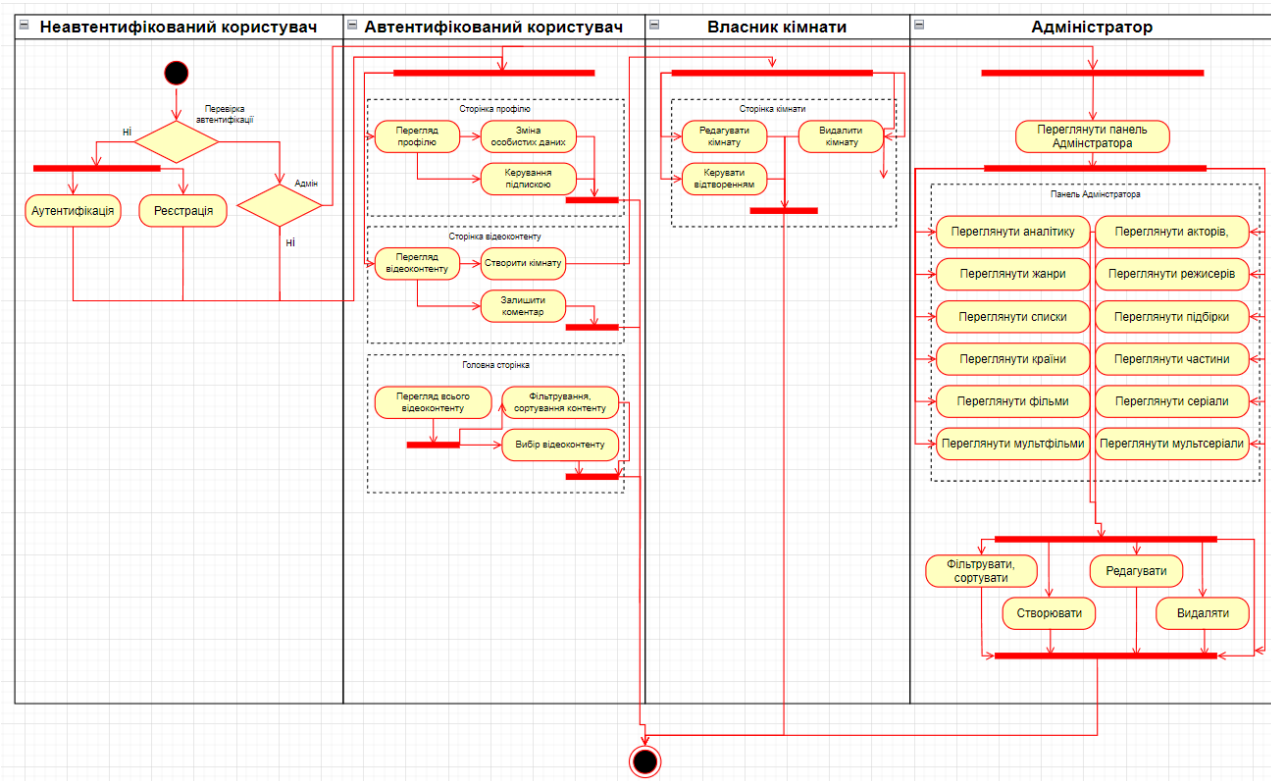


Рисунок 2.6 – Діаграма активності користувачів

## 2.4 Реалізація веб-сервісу для спільного перегляду фільмів та серіалів

Розглянемо реалізацію підключення та взаємодії користувачів кімнати. Коли користувач відкриває сторінку кімнати, відбувається перевірка автентифікації користувача. Відповідний код перевірки автентифікації наведений нижче.

```

io.use(async (socket, next) => {
  try {
    const userId = socket.handshake.auth.user._id;
    const user = await authenticatedUser(userId);
    socket.user = user;
    next();
  } catch (e) {
    console.log(e);
    return next(new Error("Authentication invalid"));
  }
});

```

Після того як авторизація пройшла успішно, користувач під'єднується до кімнати за допомогою події `join_room`. Код під'єднання до кімнати наведений нижче.

```
socket.on("join", async (roomId) => {
  const existUser = await RoomRepository.existUserInRoom(
    roomId,
    socket.user?._id
  );

  if (existUser) {
    const { rooms: joinedRooms } = socket;

    if (Array.from(joinedRooms).includes(roomId)) {
      return console.warn(
        `${socket.user?._id} is already joined to ${roomId}`
      );
    }

    const roomClients = Array.from(
      io.sockets.adapter.rooms.get(roomId) || []
    );

    roomClients.forEach((clientId) => {
      io.to(clientId).emit("web_rtc_peer", {
        peerId: socket.id,
        createOffer: false,
      });

      socket.emit("web_rtc_peer", {
        peerId: clientId,
        createOffer: true,
      });
    });

    await RoomRepository.connectUser(roomId, socket.user?._id);

    socket.join(roomId);

    socket.roomId = roomId;

    let room = userRooms.find((room) => room.roomId === roomId);
```

```

    if (!room) {
      userRooms.push({
        roomId,
        isPlaying: false,
        time: 0,
        episode: 0,
        season: 0,
      });

      room = userRooms.find((room) => room.roomId === roomId);
    }

    console.log(`User ${socket.user?.nickname} connected to room`);

    socket.to(roomId).emit("update_live");
    socket.emit("update_live");
    socket.emit("play", room?.isPlaying || false);
    socket.emit("time", room?.time || 0);
    socket.emit("update_series", room);

    const messages = await MessageRepository.getLastMessagesByRoomId(
      roomId
    );

    socket.to(roomId).emit("receive_messages", messages);
    socket.emit("receive_messages", messages);
  } else {
    socket.emit("connection_error");
  }
});

```

У цьому коді перевіряється чи користувач є запрошеним до кімнати, тільки тоді відбувається підєднання до кімнати, після чого створюється з'єднання між іншим користувачами через протокол WebRTC та відправлення інформації про стан відеовідтворення користувачу, який приєднався до кімнати.

Повідомлення, які приходять прослуховуються в події `send_message`, де воно зберігається в базі даних, після чого отримуються всі останні повідомлення і відправляються всім користувачам кімнати включаючи відправника. Відповідний

код отримання повідомлення і відправлення іншим користувачам наведений нижче.

```
socket.on("send_message", async (roomId, data) => {
  if (roomId) {
    await MessageRepository.addMessage({
      roomId: new Types.ObjectId(roomId),
      userId: socket.user?._id,
      message: data.message,
    });

    const messages = await MessageRepository.getLastMessagesByRoomId(
      roomId
    );

    socket.to(roomId).emit("receive_messages", messages);
    socket.emit("receive_messages", messages);
  }
});
```

Відправлення даних користувачів для спілкування через інтернет-протокол WebRTC відбувається через події `web_rtc_leave`, `web_rtc_relay_sdp` та `web_rtc_relay_ice`. Подія `web_rtc_leave` обробляє вихід користувача з голосового чату. Подія `web_rtc_relay_sdp` передає опис сеансу WebRTC (SDP) іншим користувачам. Подія `web_rtc_relay_ice` передає ICE-кандидатів для встановлення WebRTC-з'єднання між користувачами. Код прослуховування цих подій та обробки передачі даних наведений нижче.

```
socket.on("web_rtc_leave", leaveRoom);

socket.on("web_rtc_relay_sdp", ({ peerId, sessionDescription }) => {
  io.to(peerId).emit("web_rtc_session_description", {
    peerId: socket.id,
    sessionDescription,
  });
});

socket.on("web_rtc_relay_ice", ({ peerId, iceCandidate }) => {
  io.to(peerId).emit("web_rtc_ice_candidate", {
    peerId: socket.id,
```

```

        iceCandidate,
    });
});

```

Коли власник кімнати завершує сеанс, відбувається подія `finish_session`. Подія `finish_session` знаходить кімнату по ідентифікатору і видаляє її з масиву кімнат, після чого відбувається відправлення усім користувачам події `finish_session`, щоб користувачі відключилися від кімнати. Відповідний код завершення сеансу наведений нижче.

```

socket.on("finish_session", (roomId) => {
    const room = userRooms.find((room) => room.roomId === roomId);

    if (roomId && room) {
        userRooms = userRooms.filter((room) => room.roomId !== roomId);
        socket.to(roomId).emit("finish_session");
    }
});

```

Більш детальний код взаємодії користувачів у режимі реального часу через WebSockets наведено в додатку А.

Оплата підписки користувача відбувається через бібліотеку Stripe.

Stripe – це технологічна компанія, яка надає платіжну інфраструктуру для онлайн-бізнесу. Основні послуги Stripe включають обробку платежів для інтернет-магазинів, підписок, SaaS-платформ та інших комерційних сервісів. За допомогою Stripe бізнеси можуть приймати кредитні та дебетові картки, а також інші форми електронних платежів. Платформа також пропонує інструменти для управління фінансами, виставлення рахунків, запобігання шахрайству та аналітики.

Функція `createSubscription` відповідає за створення сесії оплати підписки для користувача. Вона приймає `userId` та `email` користувача, створює сесію Stripe Checkout для оформлення підписки, а потім повертає створену сесію, після чого користувача переадресовує на сторінку оплати, якщо користувач оплачує підписку, його переадресовує на `success_url`, якщо скасування оплати підписки, то `cancel_url`. Код функції наведений нижче.

```

async createSubscription(userId: string, email: string) {
    const session = await stripe.checkout.sessions.create({

```

|  |  |  |  |  |                     |    |
|--|--|--|--|--|---------------------|----|
|  |  |  |  |  | ІПЗ.КР.Б-121-24-ІПЗ |    |
|  |  |  |  |  |                     |    |
|  |  |  |  |  |                     | 56 |

```

mode: "subscription",
customer_email: email,
line_items: [
    {
        price: "price_1PQ4i5F7ep0vJnbvMkgGXdiG",
        quantity: 1,
    },
],
success_url:
`http://localhost:5000/api/v1/subscription/successPayment?sessionId={CHECKOUT_SESSION_ID}&userId=${userId}`,
cancel_url:
`http://localhost:5000/api/v1/subscription/cancelPayment`,
});

return session;
}

```

Щоб перевірити чи користувач має підписку використовується функція `getSubscriptionDetailsBySubscription`. Ця функція приймає ідентифікатор підписки після чого перевіряє чи вона існує, якщо існує то повертає інформацію про підписку, якщо ні то `null`. Відповідний код функції наведений нижче.

```

async getSubscriptionDetailsBySubscription(subscription: string) {
    try {
        if (!subscription) {
            return null;
        }
        return await stripe.subscriptions.retrieve(subscription);
    } catch (e) {
        return null;
    }
}

```

Щоб скасувати підписку використовується функція `cancelSubscription`. Ця функція приймає ідентифікатор підписки та ідентифікатор користувач після чого перевіряє чи існує користувач і чи він має підписку, якщо існує то підписка скасовується. Код функції наведений нижче.

```

async cancelSubscription(subscription: string, userId: string) {
    try {

```

|  |  |  |  |  |                     |    |
|--|--|--|--|--|---------------------|----|
|  |  |  |  |  | ІПЗ.КР.Б-121-24-ІПЗ |    |
|  |  |  |  |  |                     |    |
|  |  |  |  |  |                     | 57 |

```

const user = await UserModel.findById(userId);
if (!user) {
  throw new BadRequestError("Subscription not found");
}
if (!user.subscription) {
  throw new BadRequestError("Subscription not found");
}
const delSubscription = await
stripe.subscriptions.cancel(subscription);
} catch (e) {
  throw new BadRequestError("Subscription not found");
}
}

```

На рис. 2.7 наведено діаграму компонентів (Component Diagram) системи, що виконують важливу роль у забезпеченні безперебійної роботи веб-сервісу.

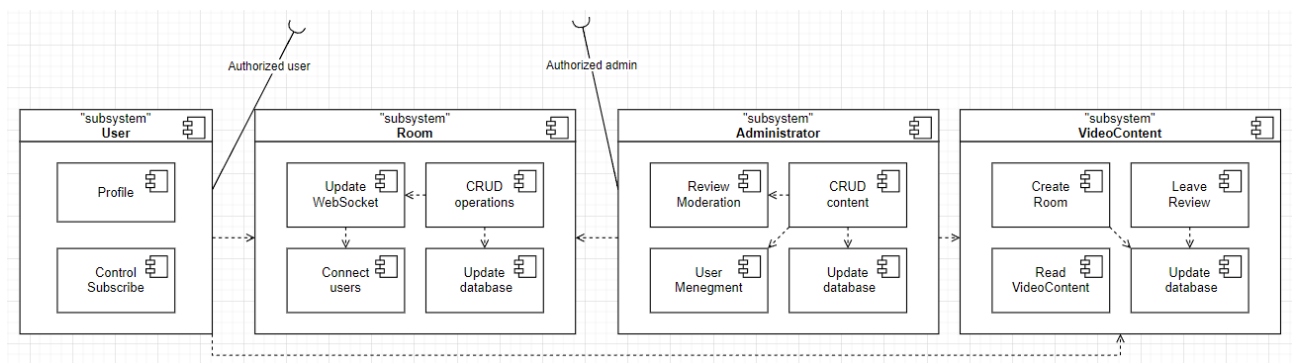


Рисунок 2.7 – Діаграма компонентів

Опишемо діаграму компонентів.

1. Для роботи з основними компонентами користувач повинен бути авторизованим;
2. Систему було поділено на чотири підкомпоненти;
3. Компонент User представляє собою керування даними користувачів, керує підпискою;
4. Компонент Room відповідає за операції над кімнатою та її керування;
5. Компонент Administrator представляє собою керування даними веб-сервісу адміністратором;
6. Компонент VideoContent представляє собою керування та взаємодію з відеоконтентом.

## Висновки до другого розділу

Під час проектування веб-сервісу для спільного перегляду фільмів та серіалів було проаналізовані варіанти використання даного продукту з вимогами програмного забезпечення. На основі проаналізованих варіантів використання було спроектовано структуру системи.

Також було спроектовано базу даних для зберігання основної інформації користувачів та контенту веб-сервісу в цілому, яка містить 15 колекцій.

Було описано основний алгоритм роботи з додатком, а саме основні дії користувачів варіантів використання веб-сервісу.

Також було описано основний функціонал веб-додатку, а саме взаємодія користувачів у режимі реального часу, їх підключення та взаємодія між собою, голосове спілкування через WebRTC, оформлення та скасування підписки користувача.

## РОЗДІЛ 3 ПРОЕКТУВАННЯ ВЕБ-СЕРВІСУ ДЛЯ СПІЛЬНОГО ПЕРЕГЛЯДУ ФІЛЬМІВ ТА СЕРІАЛІВ

### 3.1 Порядок встановлення та налаштування параметрів веб-сервісу

Порядок встановлення та налаштування веб-сервісу є критичним етапом для забезпечення його безперебійної роботи та зручного користування. Цей процес включає в себе розгортання компонентів, а саме вибір клієнтської та серверної платформи, де потрібно розгорнути компоненти додатку.

На рис. 3.1 наведено діаграму розгортання (Deployment Diagram), що описує архітектуру веб-сервісу.

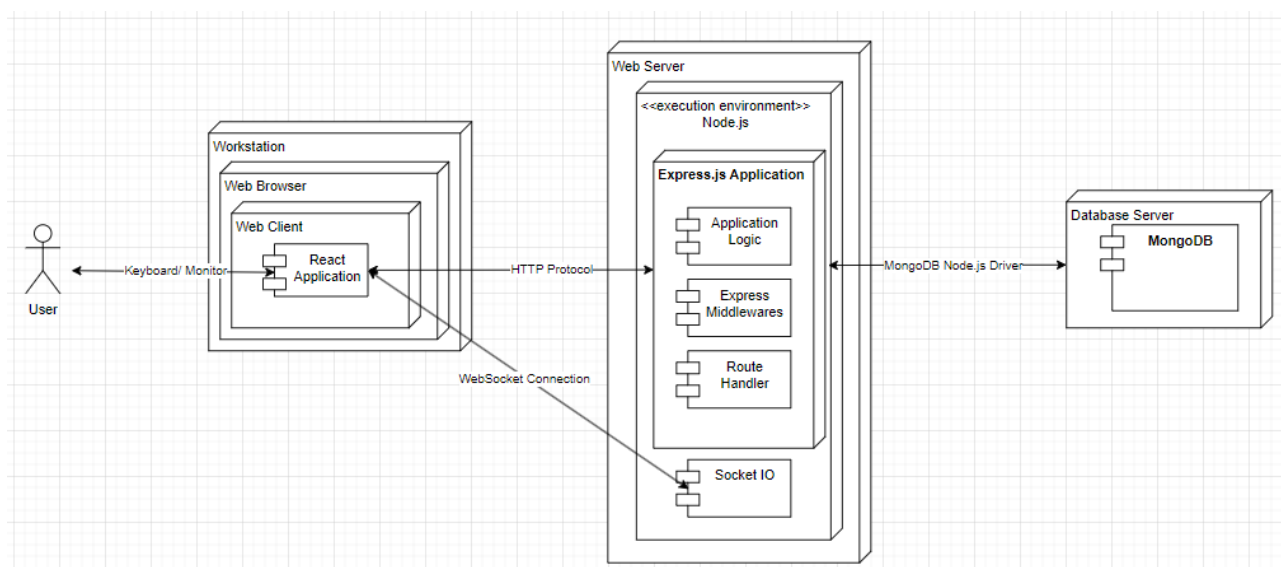


Рисунок 3.1 – Діаграма розгортання веб-сервісу

Опишемо архітектуру діаграми розгортання.

1. Workstation – пристрій, за допомогою якого, користувач буде відкривати браузер.
2. Web Browser – веб-браузер користувача.
3. Web Client – розгортання клієнтської частини веб-додатку, як React-додатку, яка взаємодія з веб-сервером через HTTP протокол, обмін між користувачами в реальному часі відбувається через WebSocket Connection.
4. Web Server – розгортання серверної частини веб-додатку, як середовище виконання Node.js, яка проводить CRUD операції з даними та зберігає їх у базі

даних через MongoDB Node.js Driver, виконує обмін повідомленнями між користувачами веб-додатку.

5. Express.js Application – додаток, який відповідає за маршрутизацію, основну логіку отримання та відправлення даних до база даних та авторизацію користувачів.

6. Database Server – зберігає всі дані, які приходять з веб-серверу та зберігає їх в MongoDB.

### 3.2 Структура інтерфейсу веб-сервісу

Вся взаємодію користувачів відбувається через веб-додаток. На головній сторінці неавторизований користувач може переглянути особливості та переваги веб-сервісу. На рис. 3.2 наведено головну сторінку неавторизовано користувача.

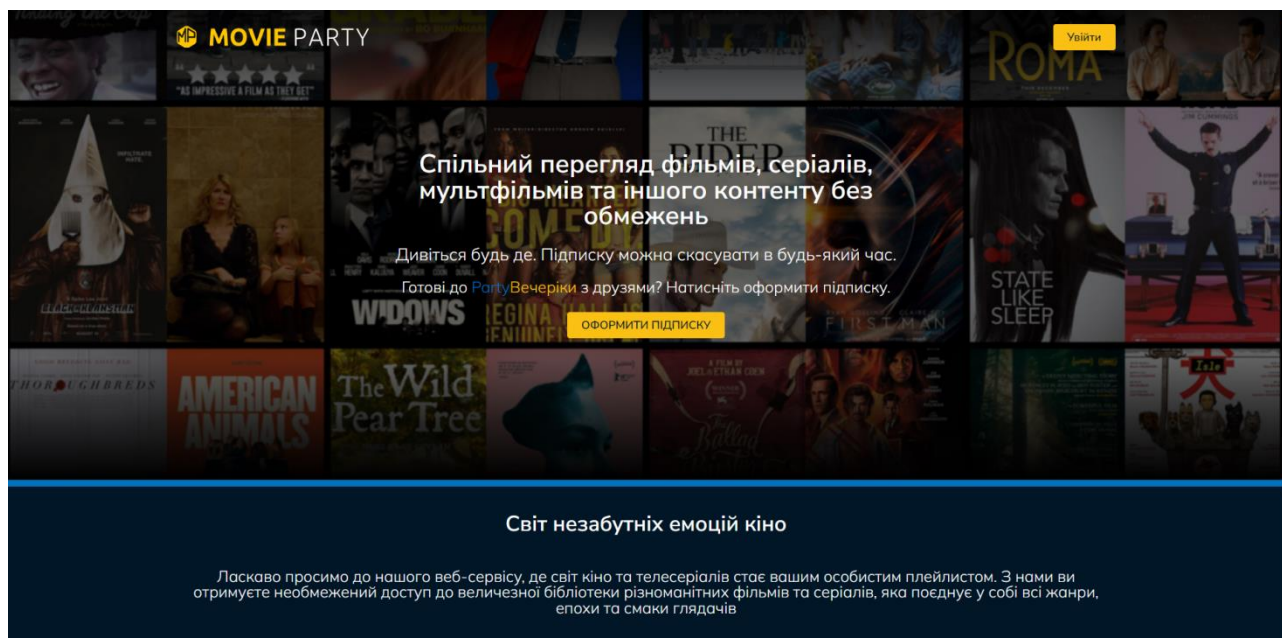


Рисунок 3.2 – Головна сторінка неавторизованого користувача

Після того як користувач натиснув кнопку Увійти, є можливість увійти через електронну пошту та пароль або через Google. Якщо користувач не зареєстрований він може зареєструватися, якщо натисне посилання Зареєструватися. Якщо користувач забув пароль він може його відновити натисканням на посилання Забули пароль. На рис. 3.3 наведено сторінку входу у веб-сервіс.

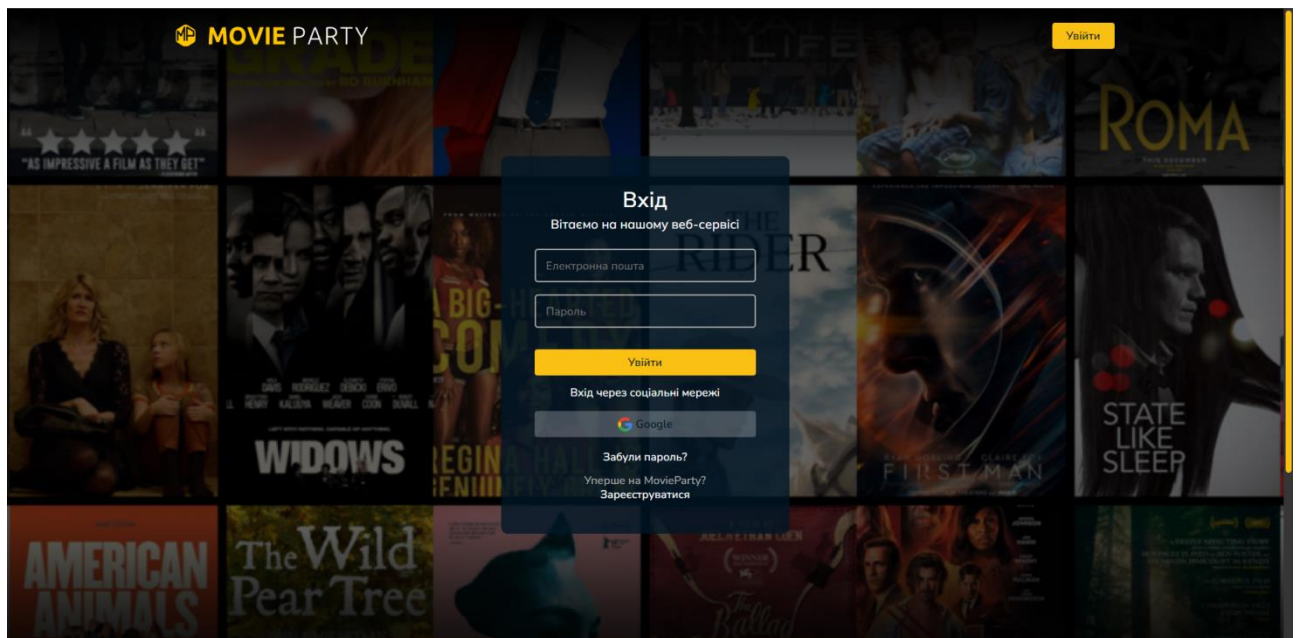


Рисунок 3.3 – Сторінка входу у веб-сервіс

Якщо користувач увійшов або зареєструвався, перед ним з'явиться оновлена головна сторінка. Користувач може побачити слайдер відеоконтенту, з яким він може взаємодіяти, і відеоконтент, який він може фільтрувати та сортувати по своїм власним критеріям. На рис. 3.4 наведено головну сторінку авторизованого користувача.

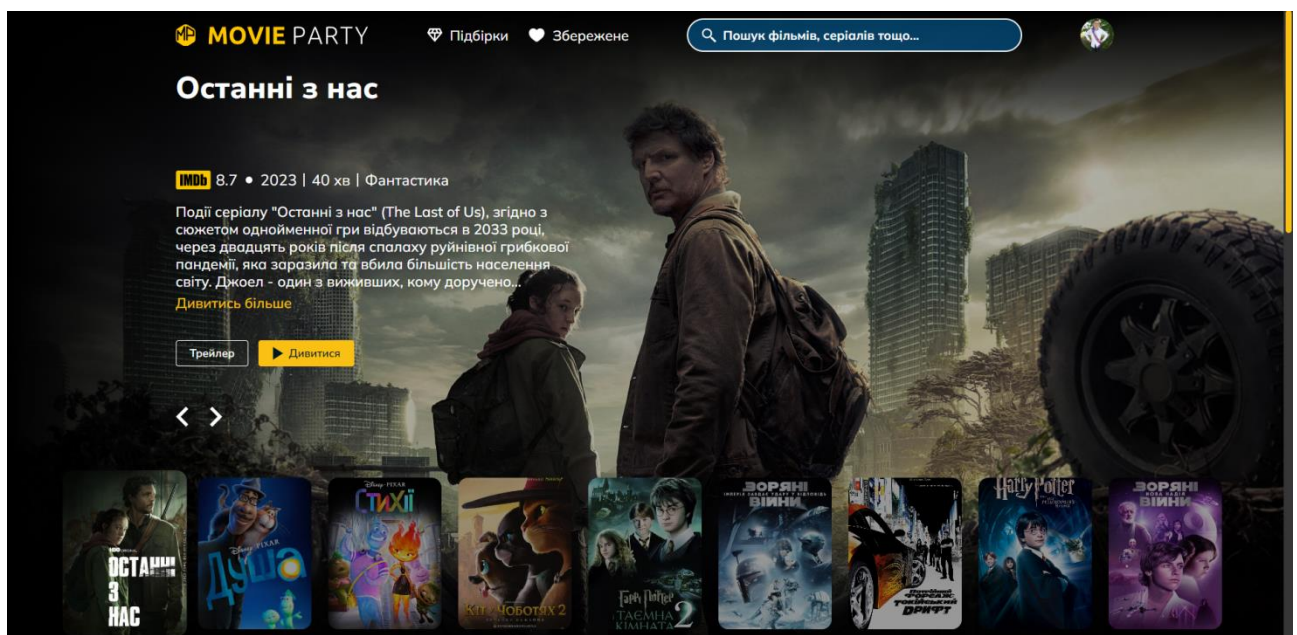


Рисунок 3.4 – Головна сторінка авторизованого користувача

Також користувач має можливість шукати відеоконтент по назві, якщо він натисне на поле пошуку у верхньому меню веб-сайту і введе пошуковий запит,

після чого з'явиться відеоконтент, що відповідає запиту користувача. На рис. 3.5 наведено можливість пошуку відеоконтенту.

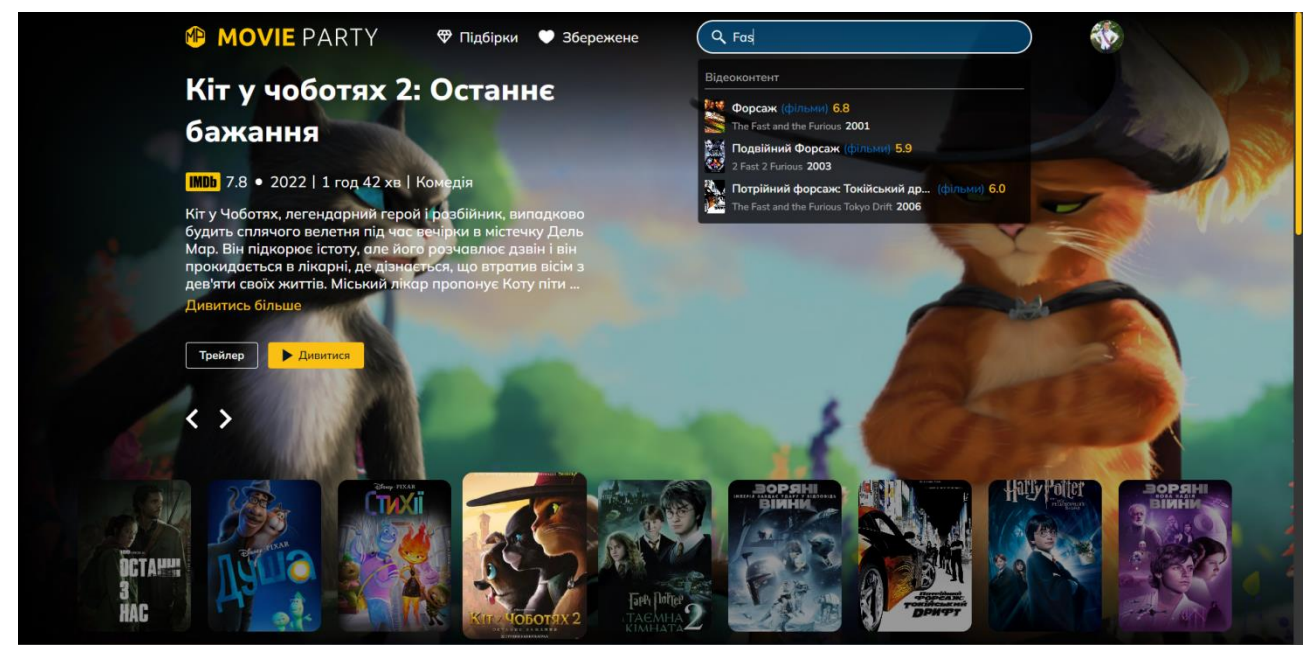


Рисунок 3.5 – Пошук відеоконтенту

Якщо користувач хоче переглянути інформацію про себе, то він повинен натиснути на свій аватар у верхньому меню веб-сайту, після чого він буде перенаправлений на сторінку профілю. На рис. 3.6 наведено профіль користувача.

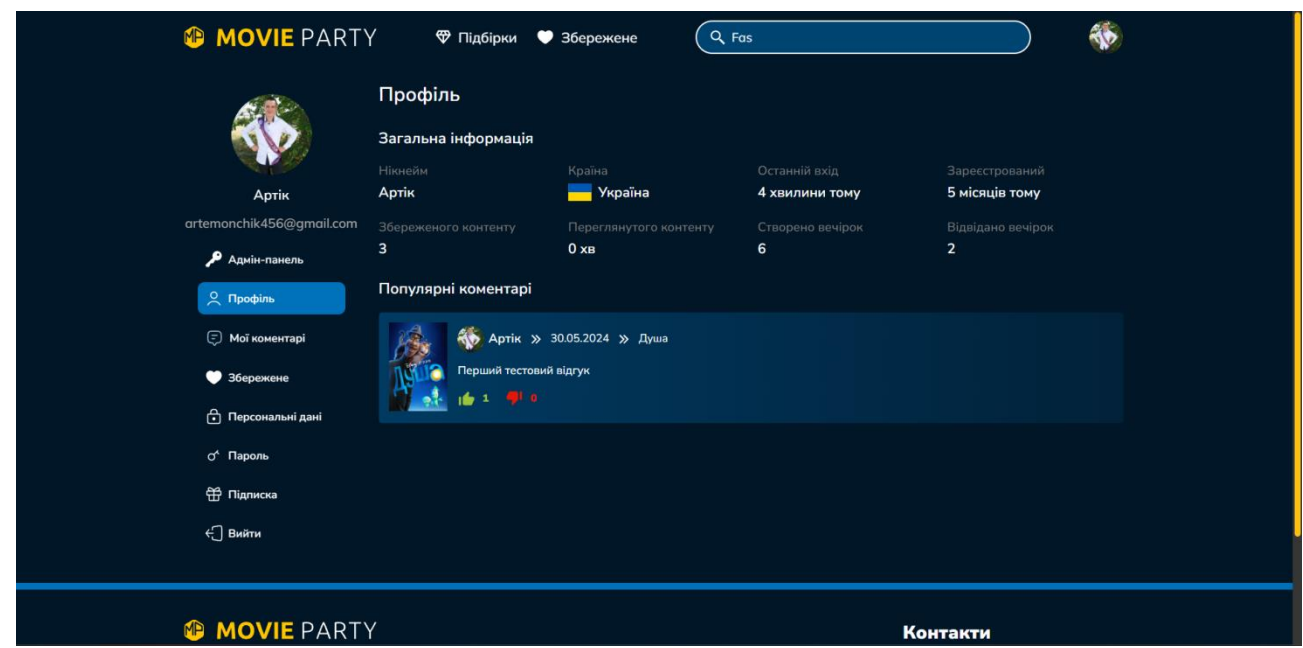


Рисунок 3.6 – Профіль користувача

В профілі користувач може побачити всю інформацію про себе, та додаткову, яку він отримав під час користування сайтом, наприклад кількість створених та відвіданих вечірок. Якщо користувач хоче відредагувати свої персональні дані, він

може це зробити кладнувши на Персональні дані в лівій частині меню профілю. На рис. 3.7 наведено сторінку редагування персональних даних.

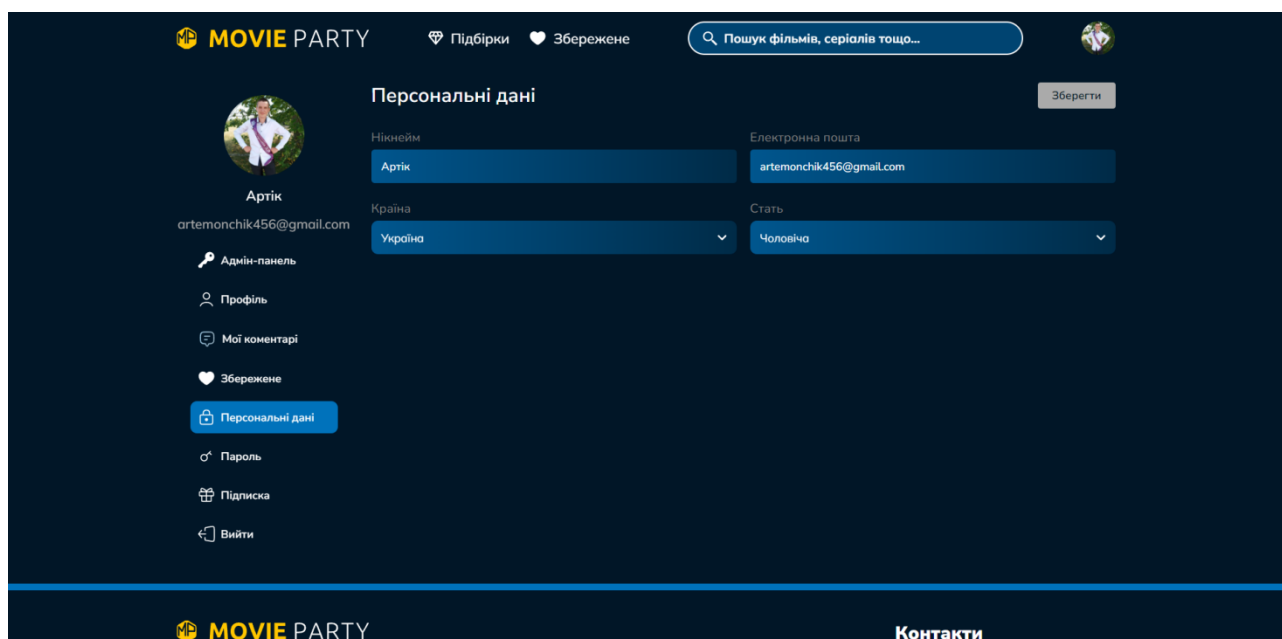


Рисунок 3.7 – Сторінка редагування персональних даних

Перейдемо безпосередньо до сторінки відеоконтенту. Після того, як користувач обрав відеоконтент, який він хоче переглянути, він може просто натиснути на картку відеоконтенту, після чого відкриється інформація про обраний ним відеоконтент. Тут він має змогу переглянути інформацію про відеоконтент, зберегти його в улюблене, перейти до перегляду (якщо є підписка), створити кімнату (якщо є підписка), переглянути трейлер і багато іншого. На рис. 3.8 наведено сторінку відеоконтенту.

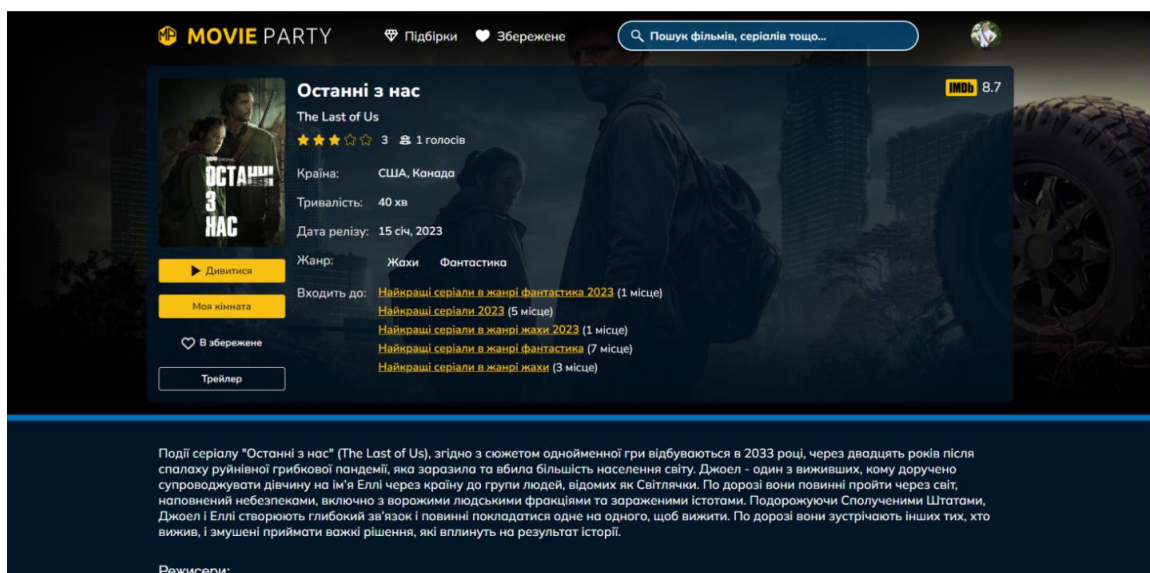


Рисунок 3.8 – Сторінка відеоконтенту

Якщо користувач, натиснув на кнопку В збережене, він може переглянути збережений відеоконтент через верхнє меню веб-сайту натиснувши на Збережене, або перейти в профіль і там натиснути Збережене в лівій частині меню профілю. На рис. 3.9 - 3.10 наведено приклад додавання та перегляд доданого відеоконтенту.

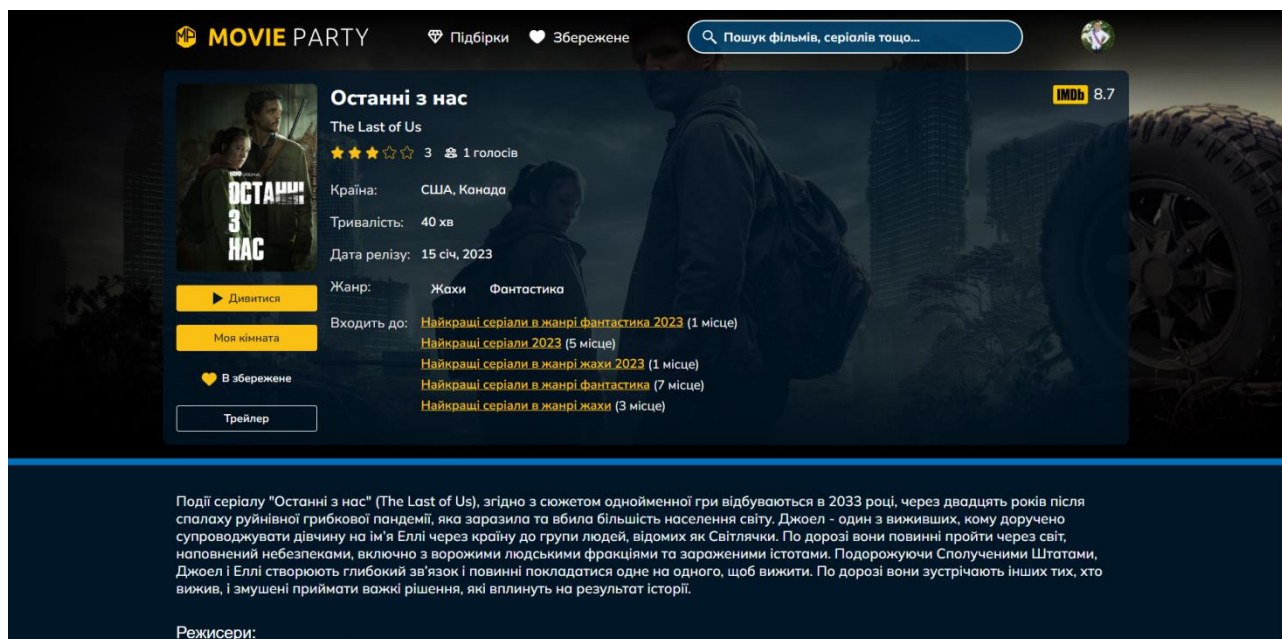


Рисунок 3.9 – Додано до збереженого

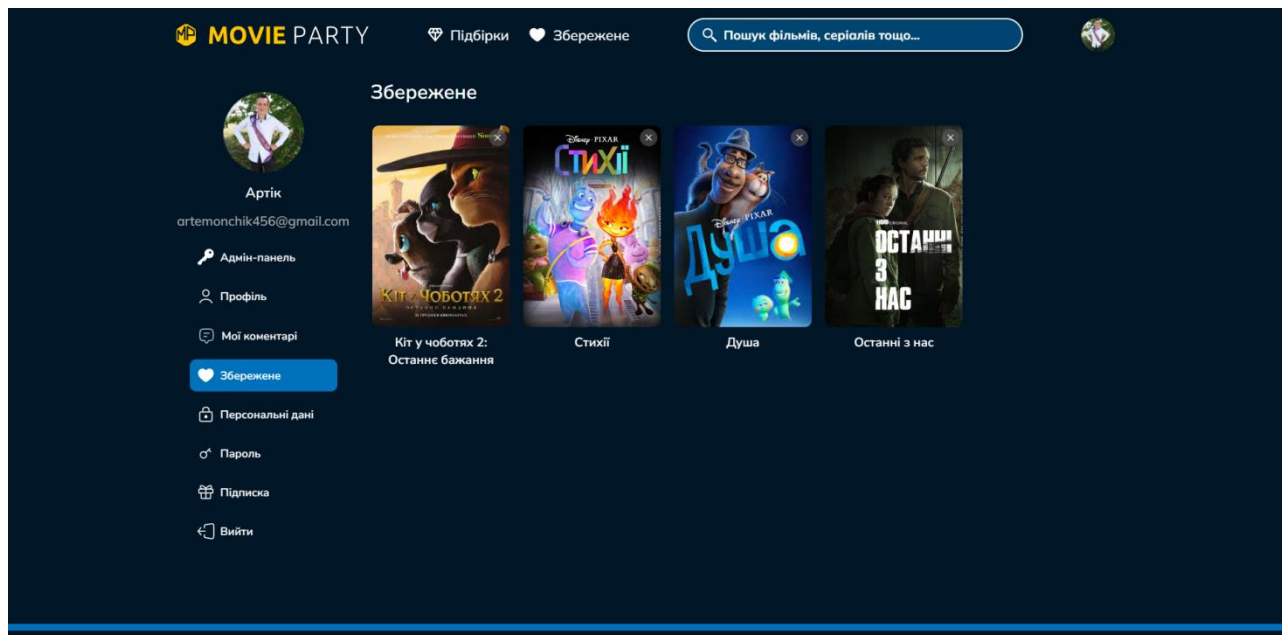


Рисунок 3.10 – Збережений відеоконтент

Користувач має можливість шукати інший відеоконтент, що відповідає цьому типу по жанрам або спискам. На рис. 3.11 наведено відеоконтент, яким відповідає жанр фантастика.

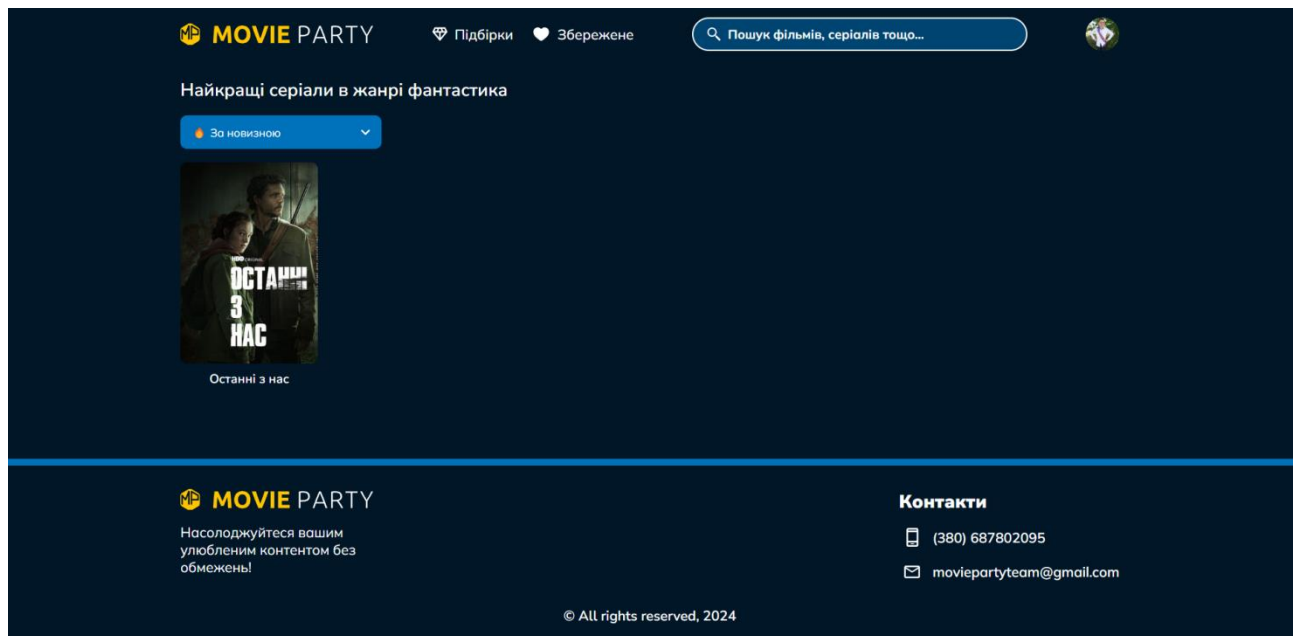


Рисунок 3.11 – Відеоконтент, яким відповідає жанр фантастика

Користувач має можливість переглядати інформацію про акторів або режисерів, які відповідають обраному їм відеоконтенту, натиснувши на картку актора або режисера. На рис. 3.12 наведено сторінку актора.

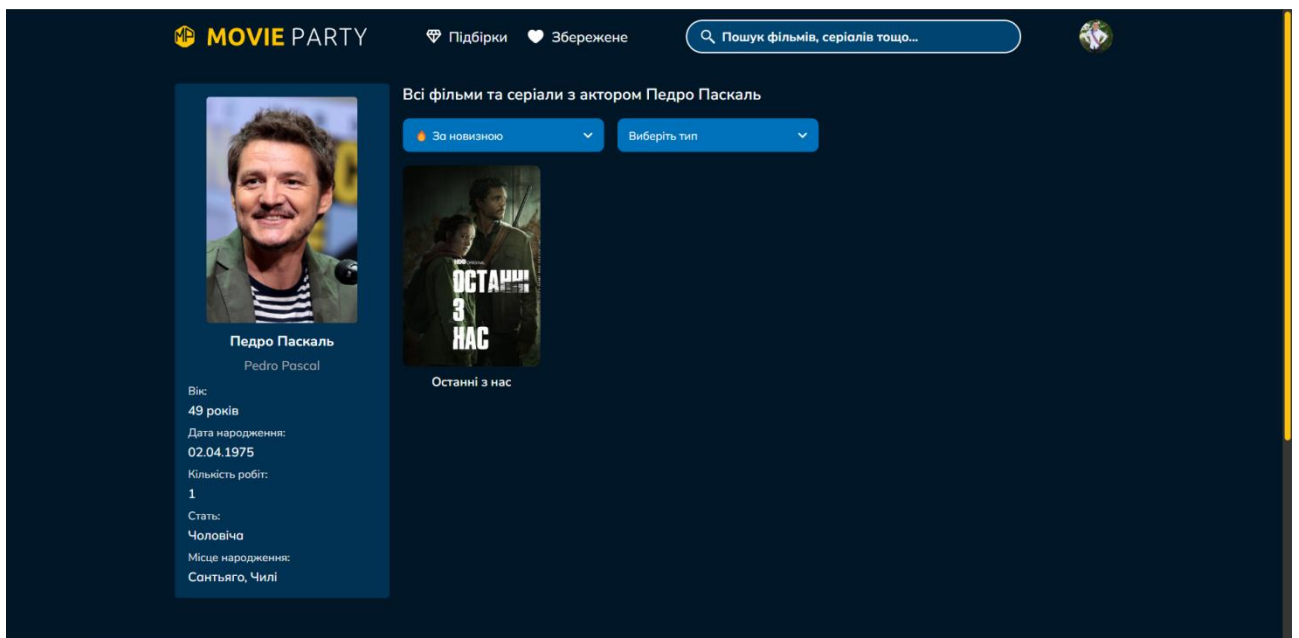


Рисунок 3.12 – Сторінка актора

На сторінці актора він може переглянути додаткову інформацію про актора та весь відеоконтент в якому він брав участь, можливість сортувати та фільтрувати відеоконтент.

Користувач має можливість залишати відгуки під кожним відеоконтентом, переглядати або видаляти їх на сторінці профілю. На рис. 3.13 - 3.14 наведено відгуки користувачів та персональні відгуки користувача.

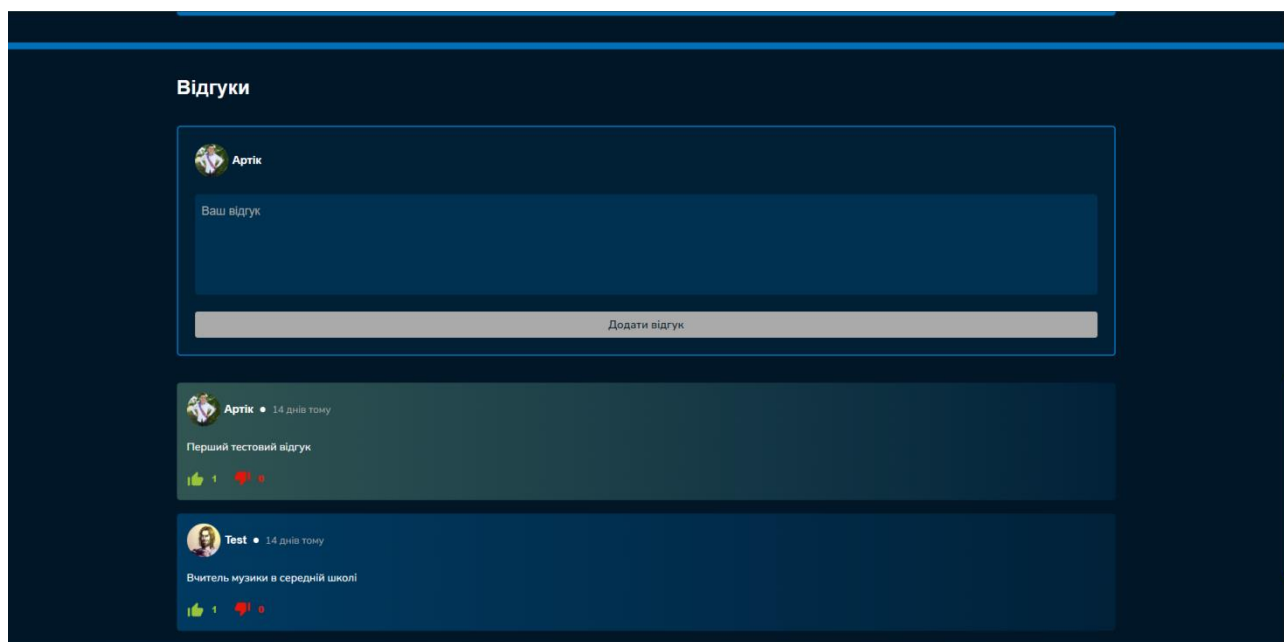


Рисунок 3.13 – Відгуки користувачів

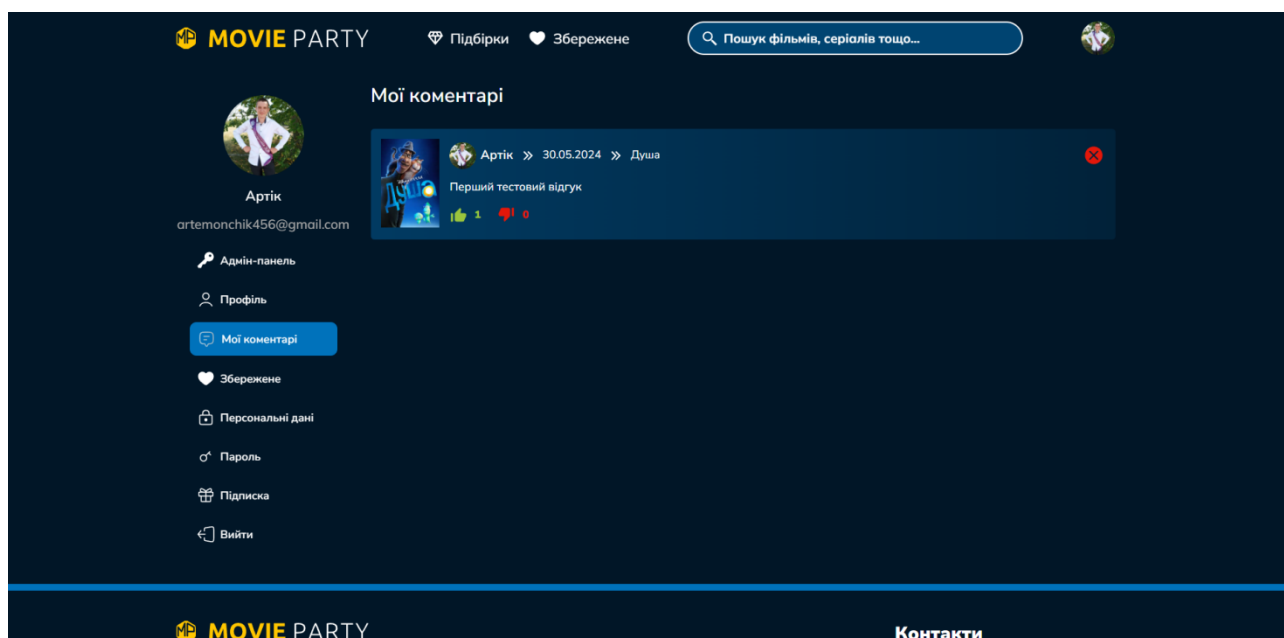


Рисунок 3.14 – Перегляд персональних відгуків користувача

Щоб дивитися відеоконтент користувач повинен натиснути на кнопку Дивитися на сторінці відеоконтенту або на слайдері відеоконтенту на головній сторінці, після чого він потрапить на сторінку перегляду, якщо у нього є підписка. На рис. 3.15 наведено сторінку перегляду відеоконтенту.

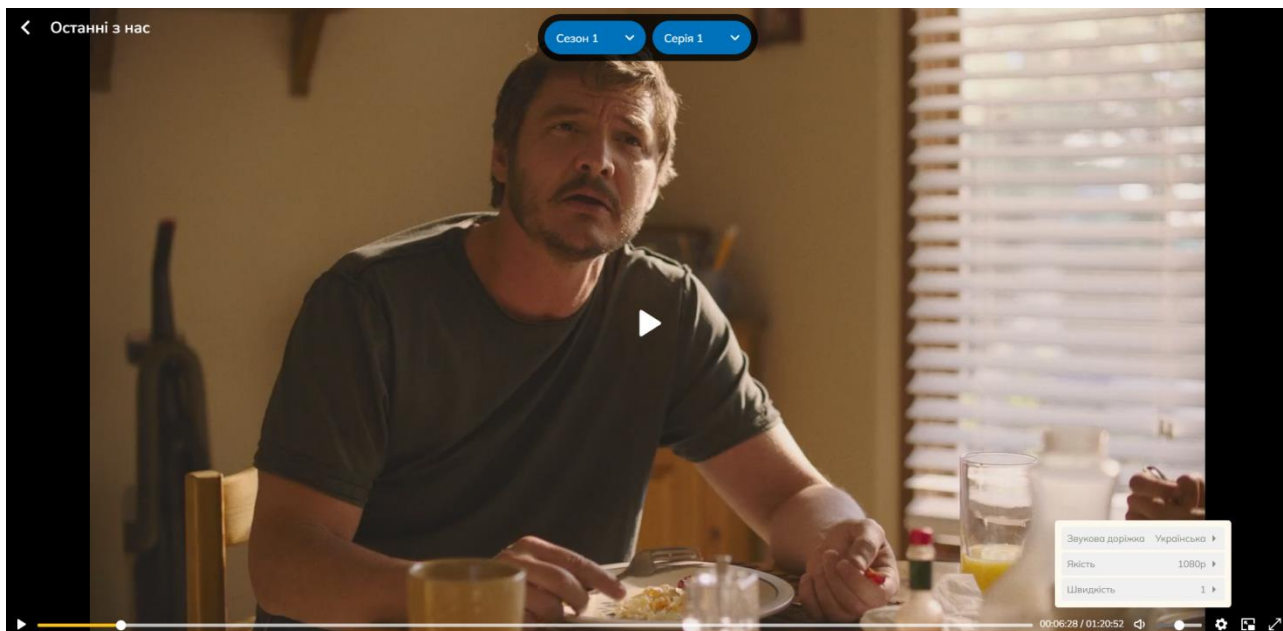


Рисунок 3.15 – Сторінка перегляду відеоконтенту

Тут користувач має можливість перемикатися між сезонами та серіями, якщо це серіал, керувати якістю та швидкістю відтворення і т.п. Користувач має можливість обирати звукову доріжку, щоб переглядати відеоконтент на іншій мові або в іншому озвученні.

Щоб створити кімнату, користувач повинен мати платну підписку. Користувачі, що приєднуються до кімнати можуть і не мати її. Керувати підпискою можна в профілі в пункті Підписка в лівій частині меню профілю. На рис. 3.16 наведено сторінку підписки користувача.

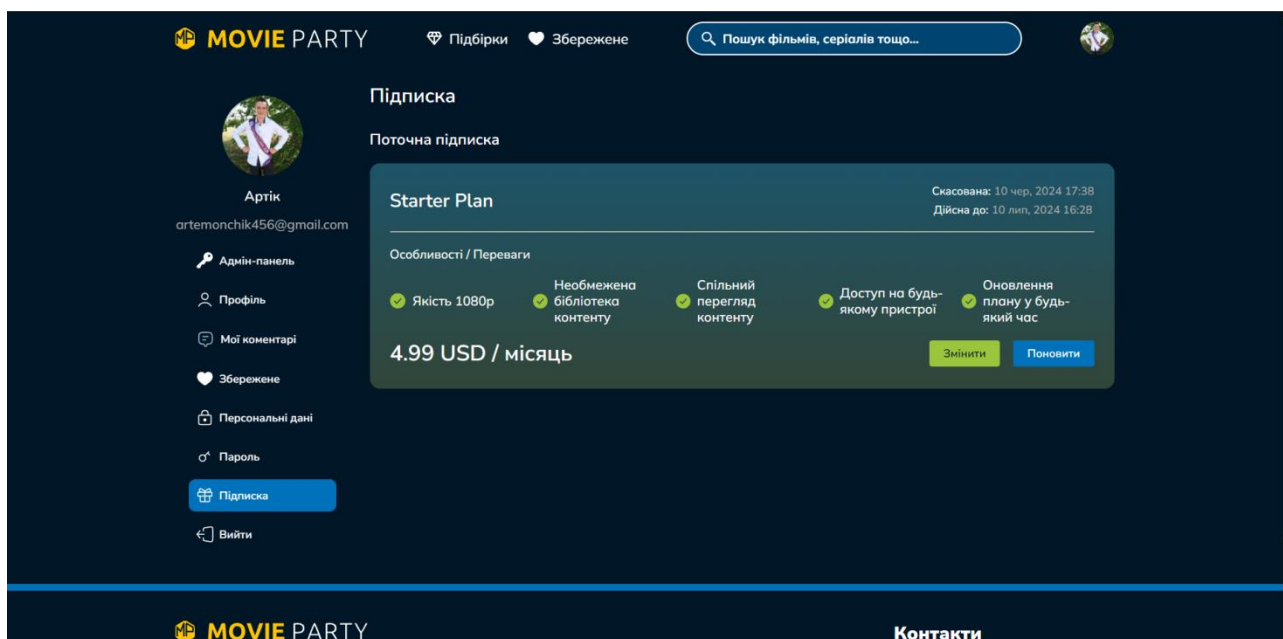


Рисунок 3.16 – Сторінка підписки користувача

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Тут користувач має можливість змінити, скасувати або поновити підписку, якщо вона у нього є, в іншому випадку буде відображено текст про відсутність підписки.

Перейдемо до створення кімнати, користувач повинен заповнити всі поля, щоб створити кімнату, після чого його буде перенаправлено на сторінку створеної кімнати. На рис. 3.17 – 3.18 наведено приклад створення кімнати та власне створену кімнату.

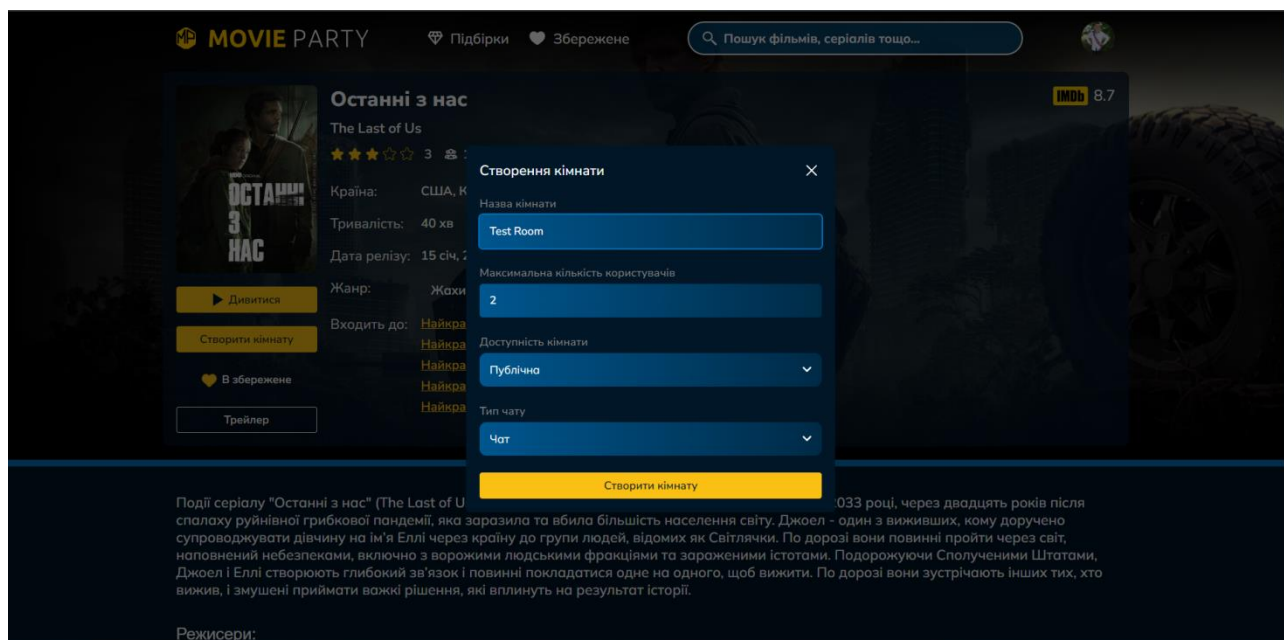


Рисунок 3.17 – Створення кімнати

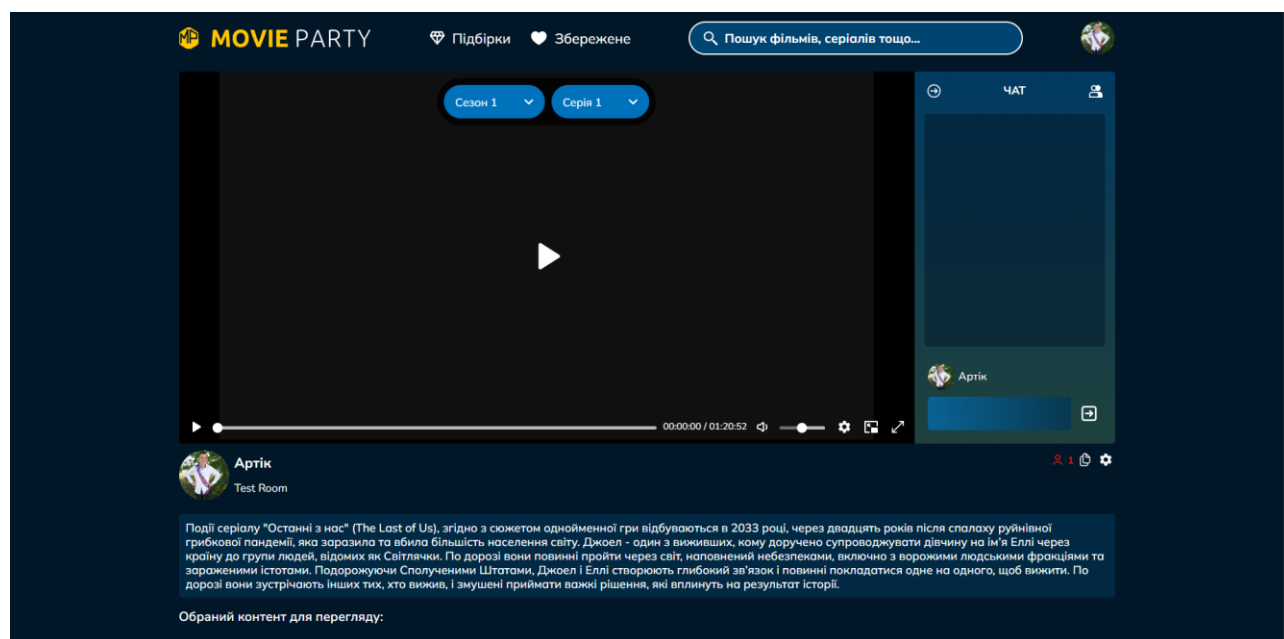


Рисунок 3.18 – Створена кімната

Користувач, який створив кімнату, має можливість змінити дані кімнати та завершити сеанс, якщо це необхідно. Щоб в інших користувачів була можливість приєднатися до кімнати їм потрібне запрошувальне посилання. Для цього власник кімнати натискає на іконку копіювання, після чого запрошувальне посилання буде скопійоване у буфер обміну. На рис. 3.19 наведено приклад під'єднання користувача до кімнати.

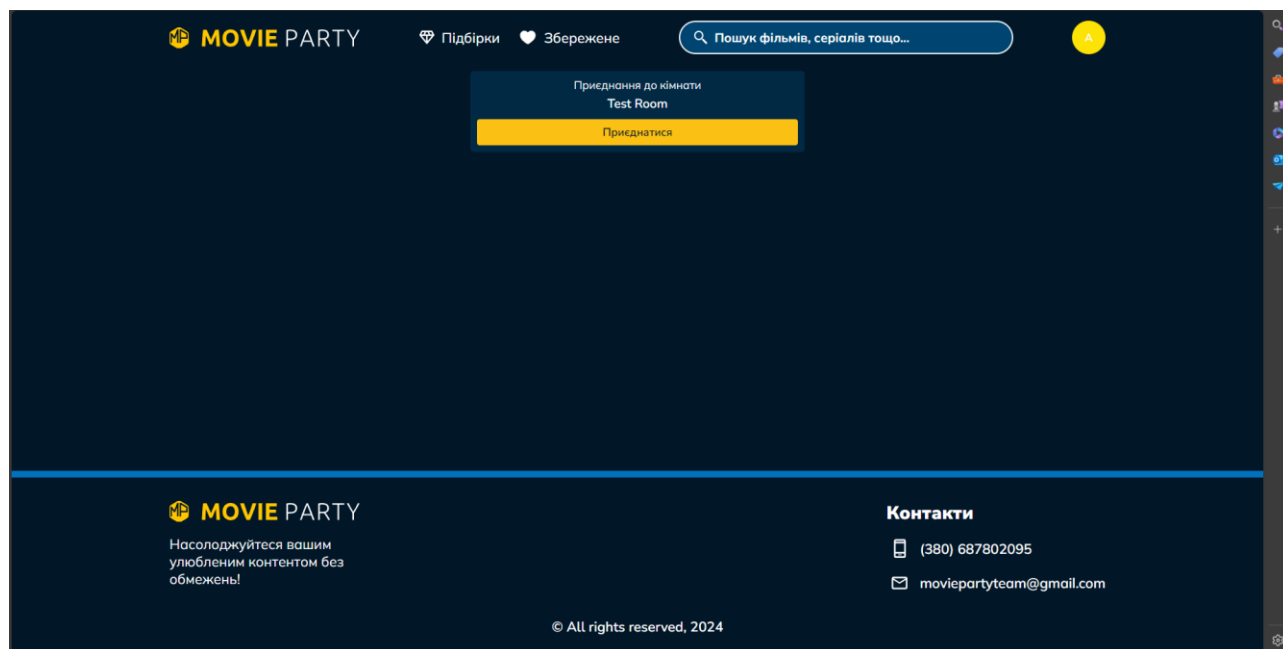


Рисунок 3.19 – Під'єднання до кімнати іншим користувачем

Якщо кімната є публічна, користувачу не потрібно вводити пароль кімнати, йому достатньо натиснути на кнопки Приєднатися, після чого він приєднається до кімнати. На рис. 3.20 наведено приклад приєднаного користувача.

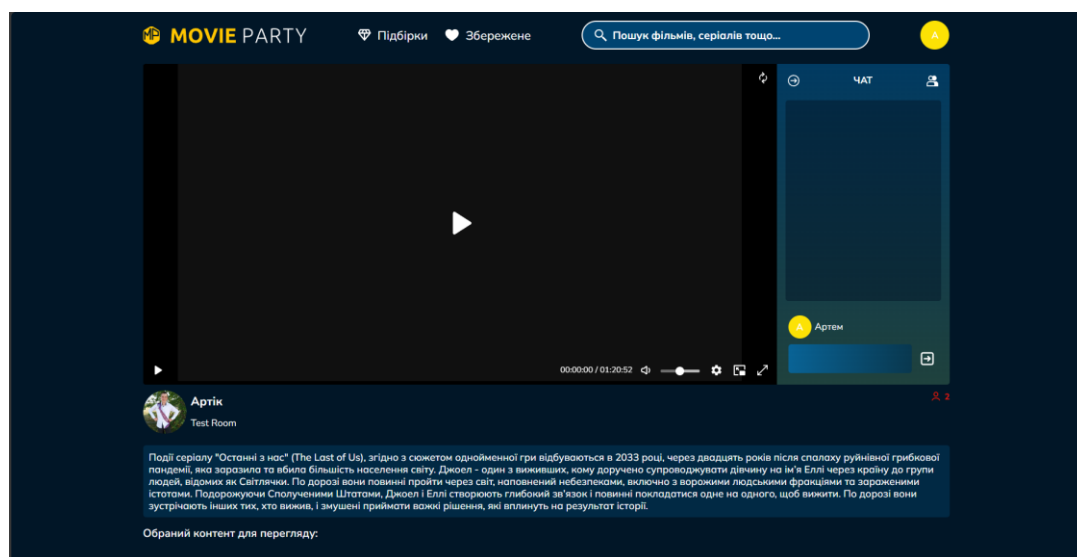


Рисунок 3.20 – Під'єднаний користувач

Користувачі, що під'єдналися до кімнати мають можливість писати в чат або спілкуватися, переглядати підключених користувачів. Користувачі не можуть керувати відтворенням окрім власника кімнати. На рис. 3.21 наведено список підключених користувачів до кімнати.



Рисунок 3.21 – Підключені користувачі

Перейдемо до відтворення відео. Після того як власник кімнати натиснув на програвання відео, в усіх користувачів починається відтворення. На рис. 3.22 наведено синхронізація відео з відеопотоком власника кімнати.

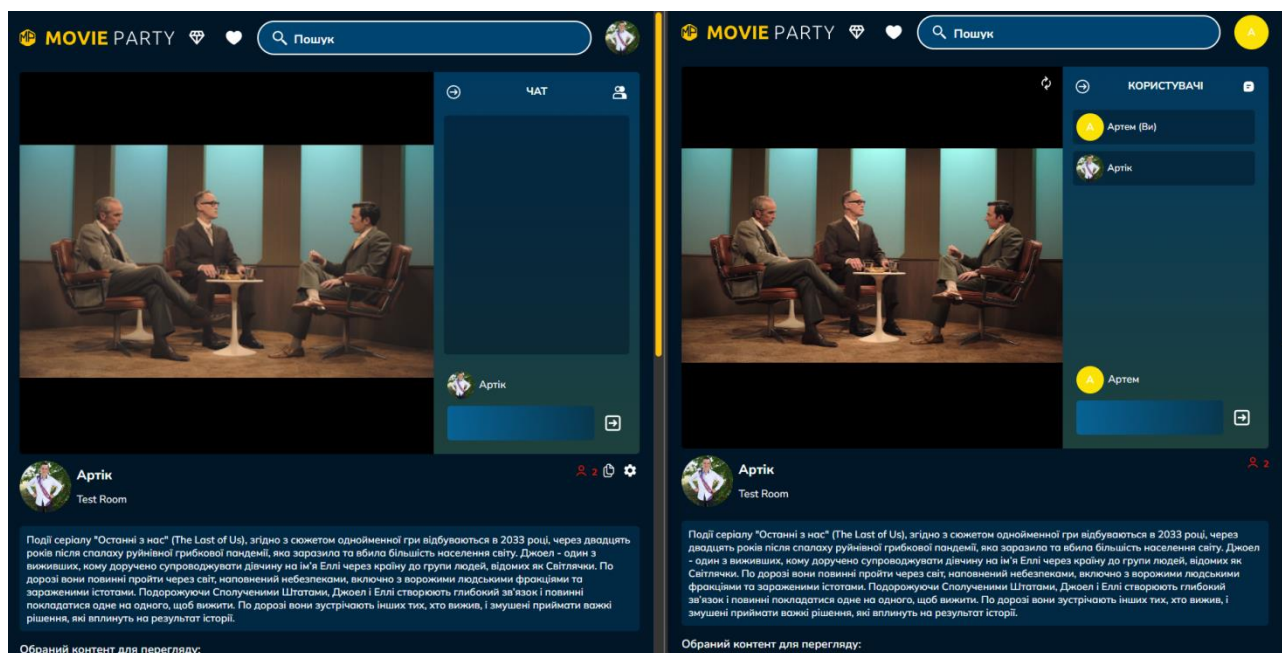


Рисунок 3.22 – Синхронізоване відтворення

Якщо власник кімнати перемотує відео, в інших користувачів воно також перемотується. На рис. 3.23 наведено синхронізація перемотування з відеопотоком власника кімнати.

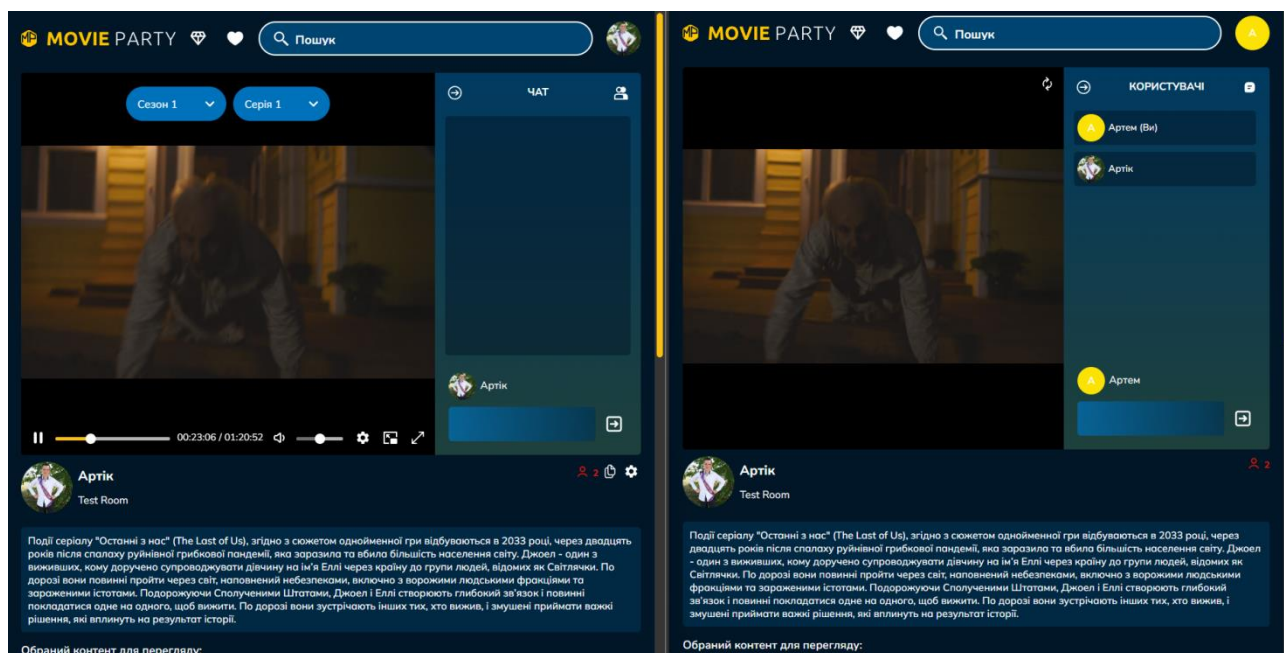


Рисунок 3.23 – Синхронізоване перемотування

Якщо користувачі хочуть синхронізувати відео з відтворенням власника кімнати, потрібно натиснути на кнопку синхронізації в верхньому правому куті відеоплеєру, після чого відео синхронізується. На рис. 3.24 наведено синхронізація відтворення за допомогою кнопки синхронізації.

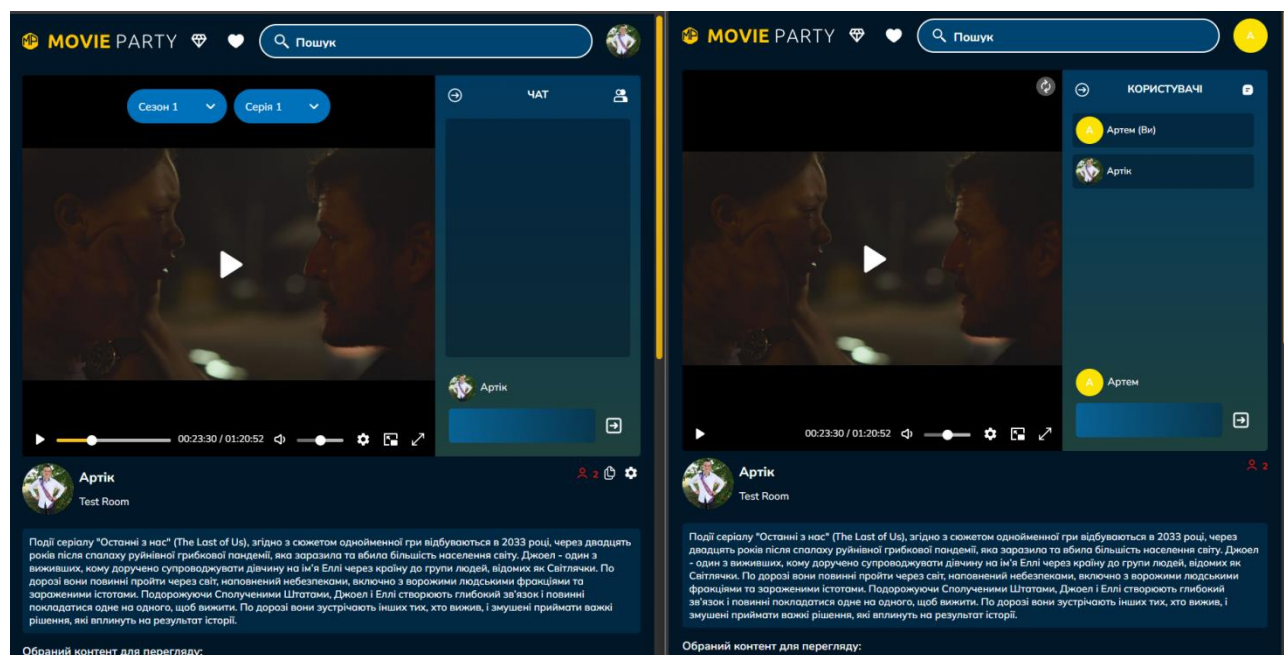


Рисунок 3.24 – Синхронізація відтворення через кнопку

На рис. 3.25 показано повідомлення відправлені користувачами кімнати.

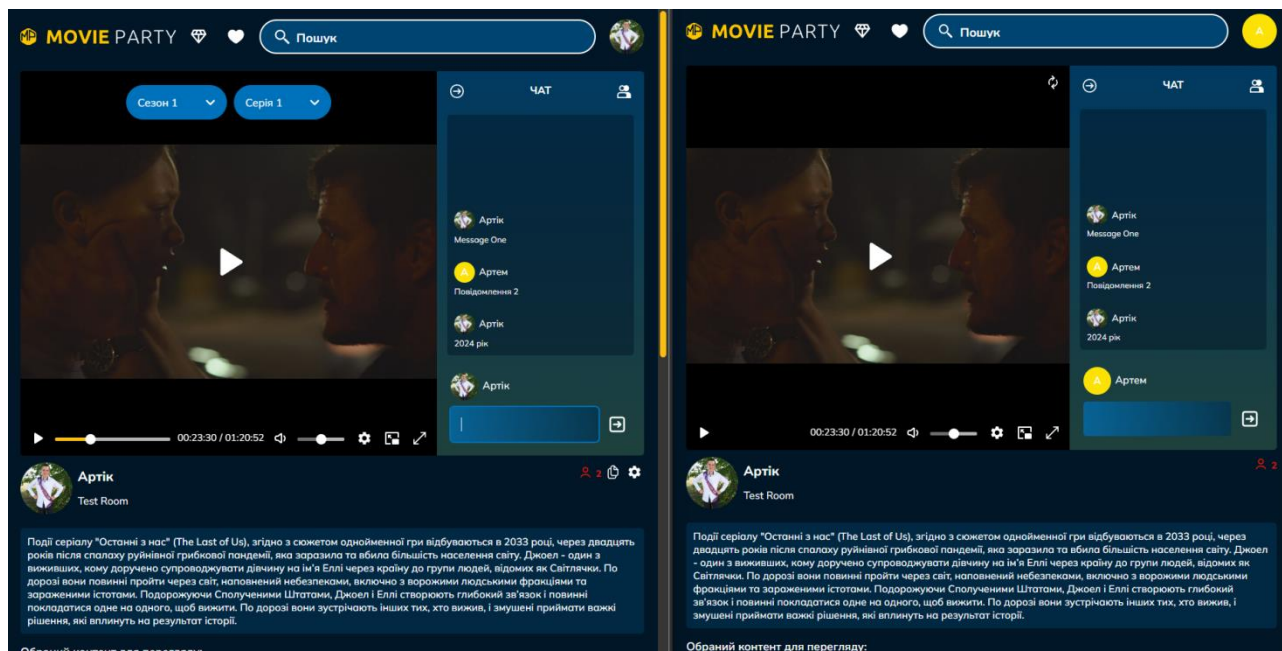


Рисунок 3.25 – Повідомлення кімнати

Щоб завершити сеанс кімнати, власник кімнати повинен натиснути на кнопку налаштувань і після чого натиснути на кнопку Завершити сеанс. Далі всіх підключених користувачів переадресує на головну сторінку веб-сайту, а власнику кімнати виведеться повідомлення про завершений сеанс. На рис. 3.26 - 3.27 наведено редагування та завершення сеансу кімнати.

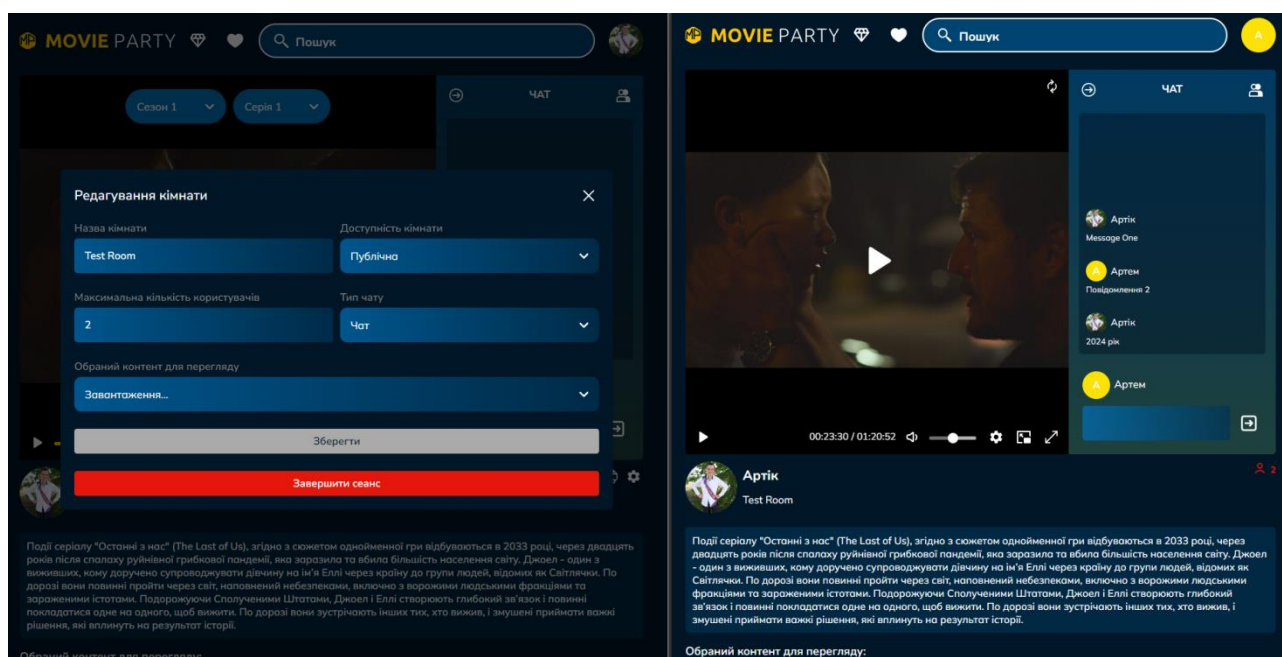


Рисунок 3.26 – Редагування кімнати

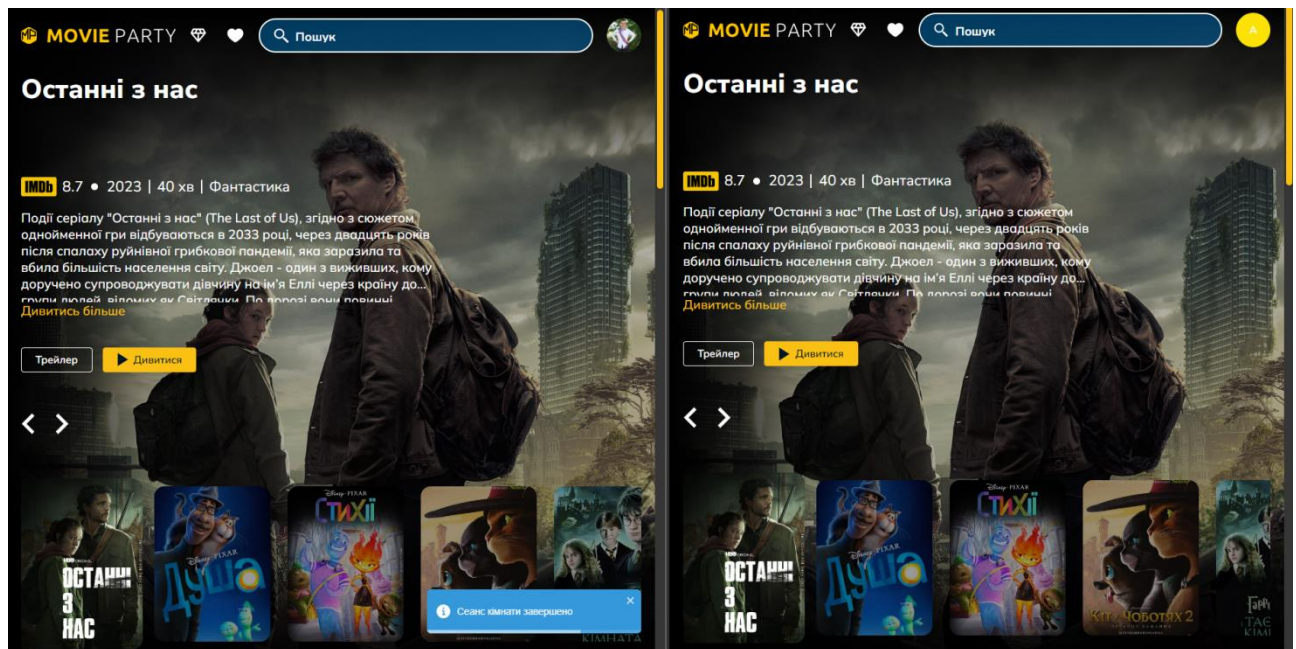


Рисунок 3.27 – Завершений сеанс кімнати

Перейдемо до управління контентом веб-сторінки, для цього користувач повинен бути адміністратором. В адміністраторі відображається схоже меню як і у меню профілю користувачів. На рис. 3.28 наведено сторінку адміністратора.

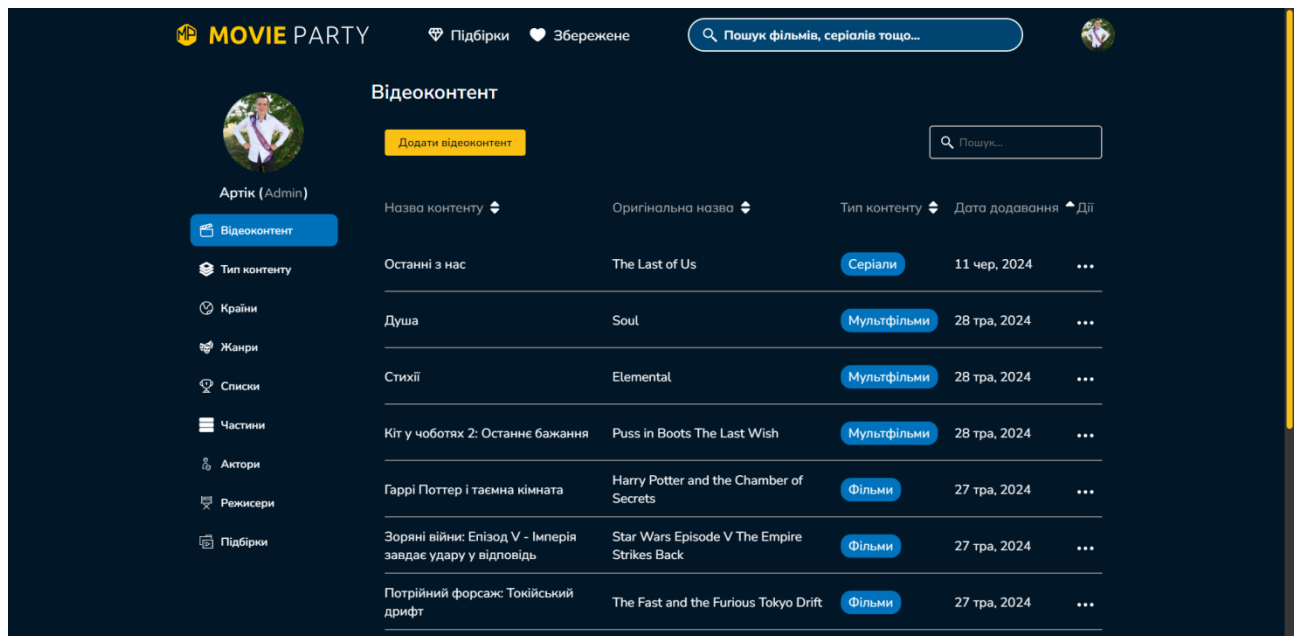


Рисунок 3.28 – Сторінка адміністратора

Адміністратор може обрати будь-який пункт меню і перейти до нього. Після чого відобразиться список створених ним даних. Ці дані він може сортувати та шукати по назвам. Щоб видалити або відредагувати якісь дані адміністратор

натискає на кнопку дії, після чого йому відображається меню дій, які він може виконати. На рис. 3.29 наведено меню дій над даними.

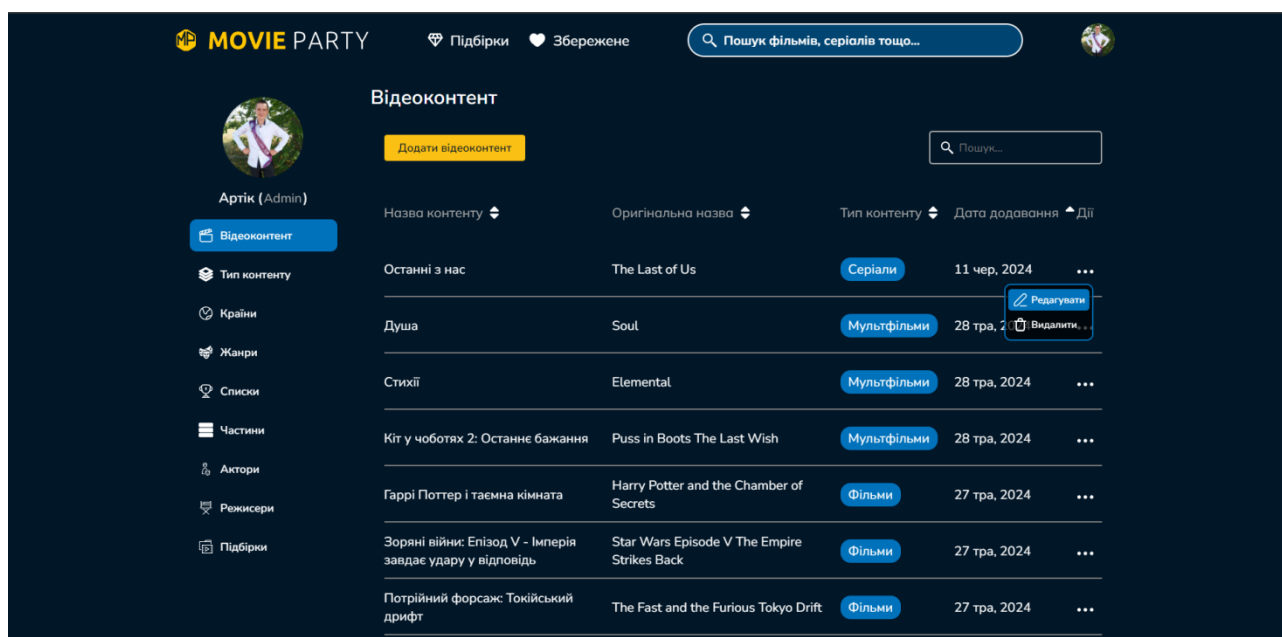


Рисунок 3.29 – Меню дій

Щоб додати якісь дані, адміністратор натискає на кнопку, що розташована в лівій частині від пошуку, після чого відобразиться сторінка додавання даних чи модальне вікно. На рис. 3.30 - 3.31 наведено сторінку додавання відеоконтенту та країни.

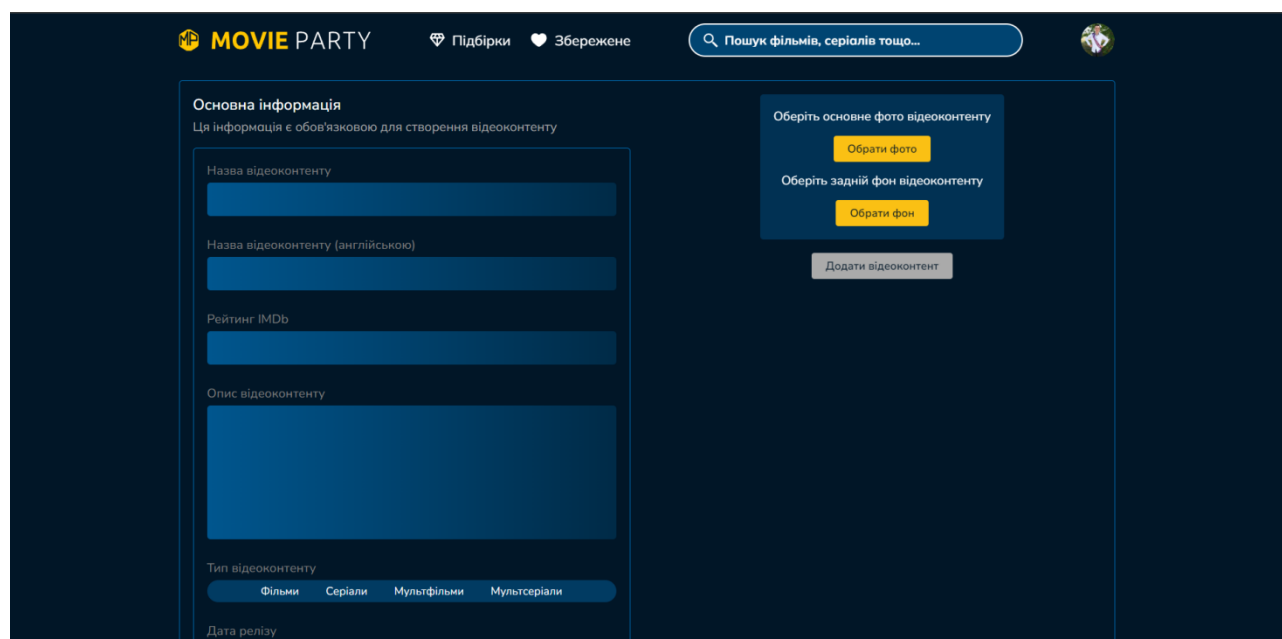


Рисунок 3.30 – Сторінка додавання відеоконтенту

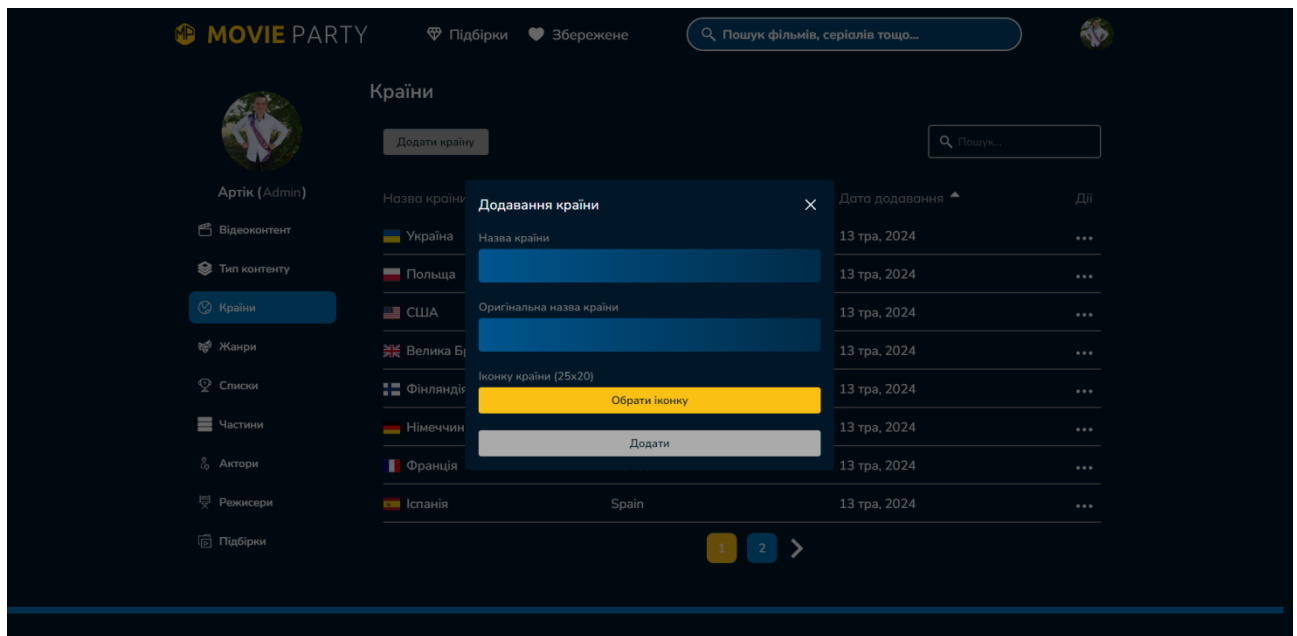


Рисунок 3.31 – Сторінка додавання країни (модальне вікно)

### 3.3 Тестування веб-сервісу

Тестування веб-сервісу є критично важливим етапом розробки програмного забезпечення, який забезпечує стабільність, надійність і відповідність веб-додатку заданим вимогам.

Оскільки це веб-додаток, для його тестування було обрано два онлайн-сервіси для тестування веб-додатків.

PageSpeed Insights (PSI) – це інструмент від Google, який надає аналіз швидкості і відповідності мобільних і десктопних веб-сторінок.

Він оцінює рівень оптимізації вашого веб-сайту і надає рекомендації щодо поліпшення швидкості завантаження сторінок. Основні можливості PageSpeed Insights включають аналіз швидкості з використанням метрик, таких як Largest Contentful Paint (LCP), Cumulative Layout Shift (CLS) і First Input Delay (FID). На рис. 3.32 наведено тестування веб-додатку за допомогою PSI щодо його ефективності.

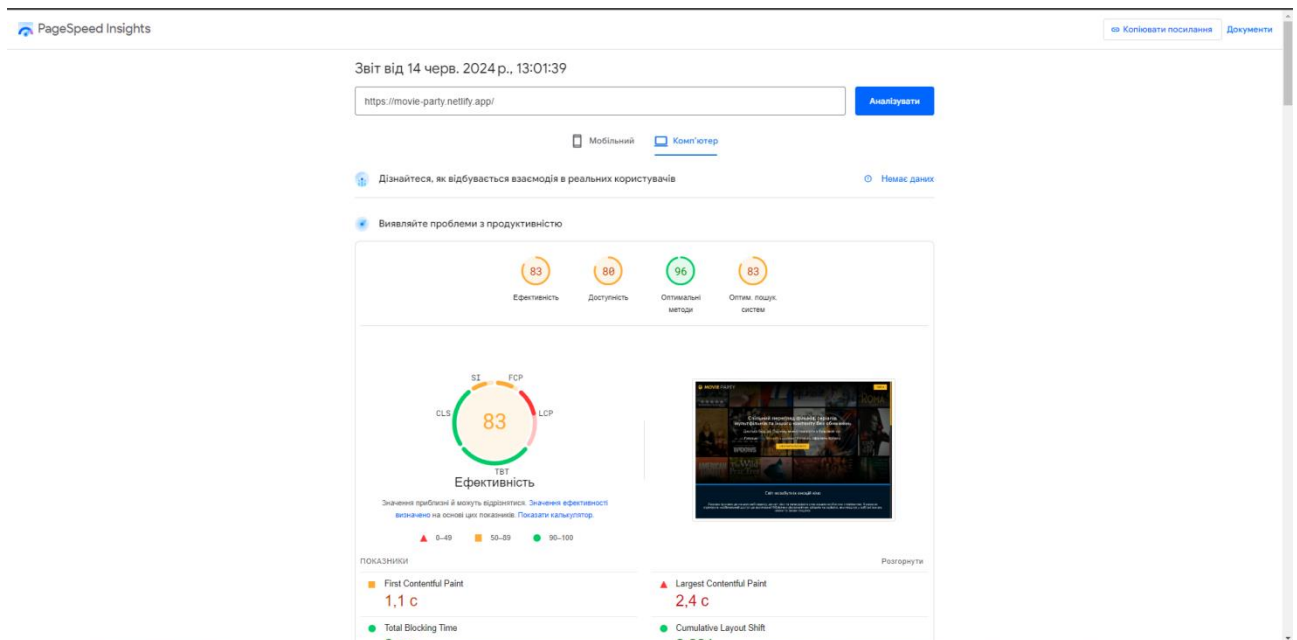


Рисунок 3.32 – Тестування веб-додатку за допомогою PSI (Ефективність)

Інструмент також надає рекомендації щодо оптимізації, які включають мінімізацію та стиснення ресурсів (зображення, CSS, JavaScript), налаштування кешування та виправлення проблем, що впливають на рендеринг сторінки. На рис. 3.33 наведено тестування веб-додатку за допомогою PSI з порадами щодо оптимізації.

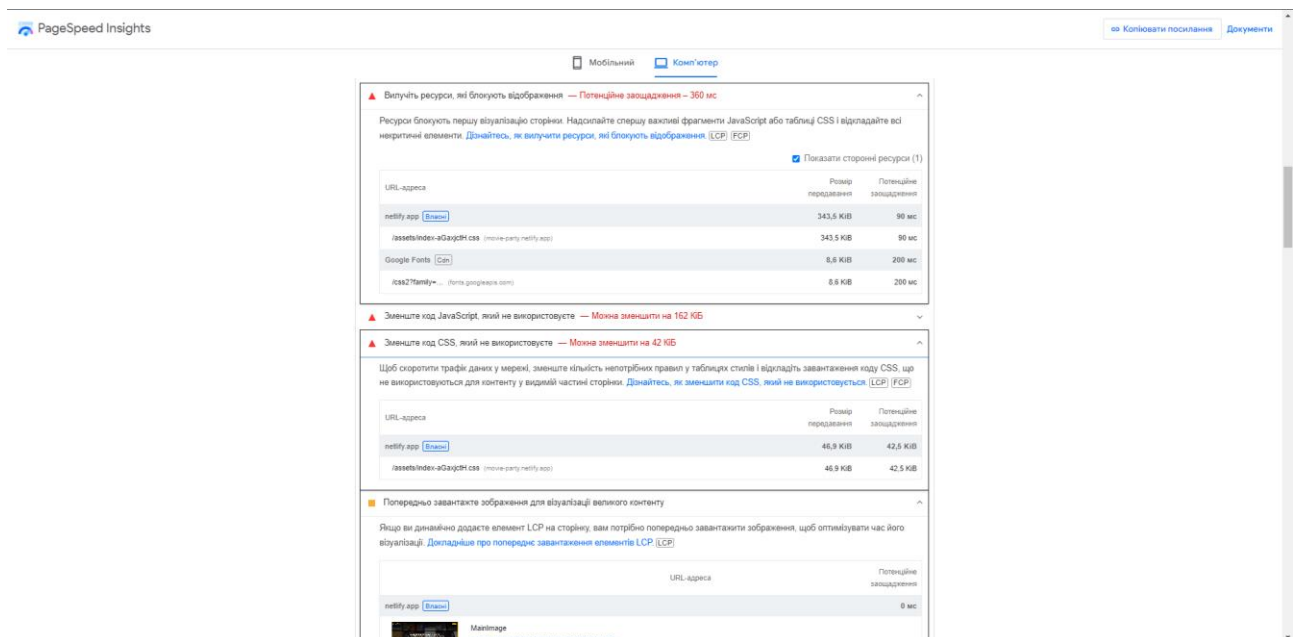


Рисунок 3.33 – Тестування веб-додатку за допомогою PSI (Поради щодо оптимізації)

PageSpeed Insights дозволяє порівнювати швидкість вашого сайту з аналогічними сайтами і з секторними стандартами, що допомагає зрозуміти, як поліпшити його продуктивність.

WebPageTest (WPT) – інструмент, який дозволяє вам перевіряти швидкість завантаження веб-сторінок з різних точок світу. Ви можете вибрати тип пристрою (мобільний або десктопний) і мережу (наприклад, 3G, 4G, WiFi) для тестування. На рис. 3.34 наведено вибір пристрою для тестування веб-додатку за допомогою WPT.

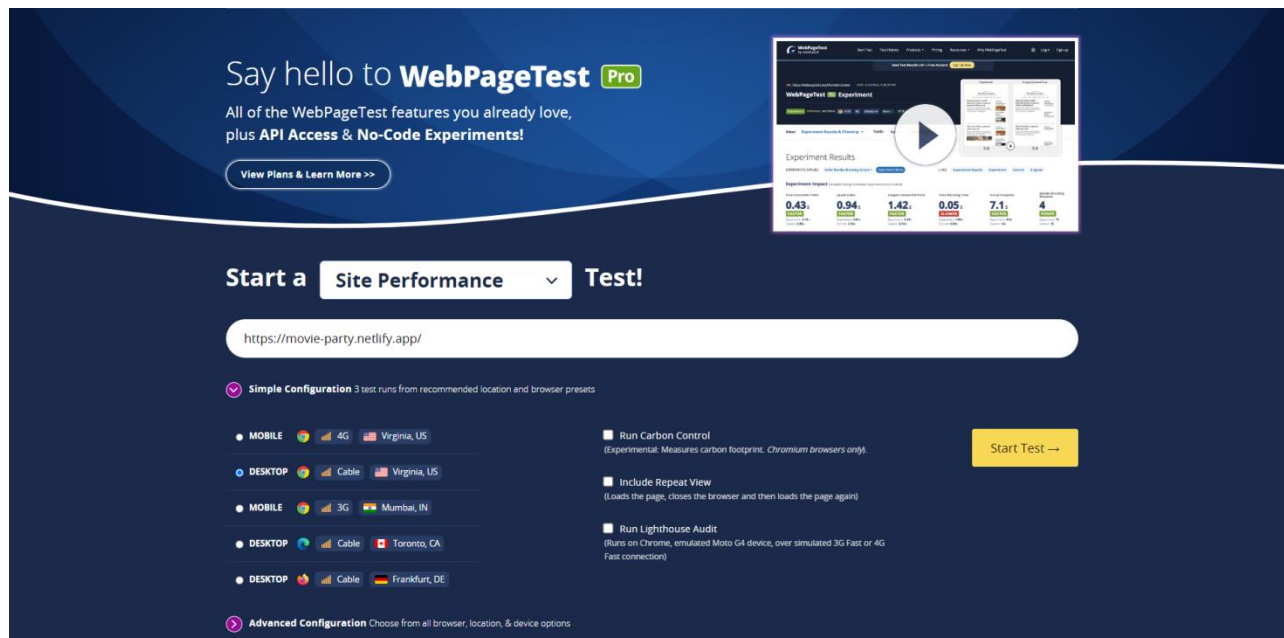


Рисунок 3.34 – Тестування веб-додатку за допомогою WPT (Вибір пристрою)

Інструмент вимірює час, який потрібен для завантаження сторінки і надає детальні метрики, такі як час до першого байту, Speed Index, Time to Interactive, Total Blocking Time та інші, що допомагають зрозуміти, як швидкість завантаження впливає на користувацький досвід. На рис. 3.35 наведено інформацію про тестування веб-додатку за допомогою WPT.

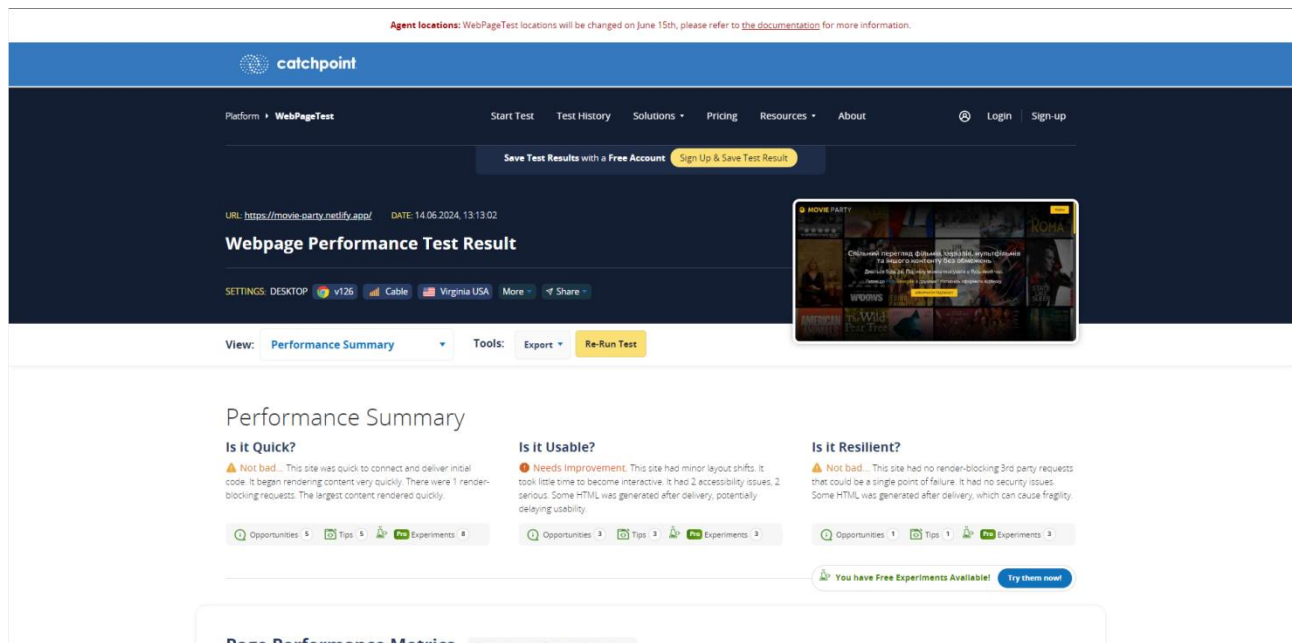


Рисунок 3.35 – Тестування веб-додатку за допомогою WPT (Інформація про тест)

Також доступна можливість перегляду відео завантаження сторінки і аналізу водоспаду ресурсів, що дозволяє ідентифікувати потенційні проблеми з продуктивністю. На рис. 3.36 наведено водоспад веб-додатку за допомогою WPT.

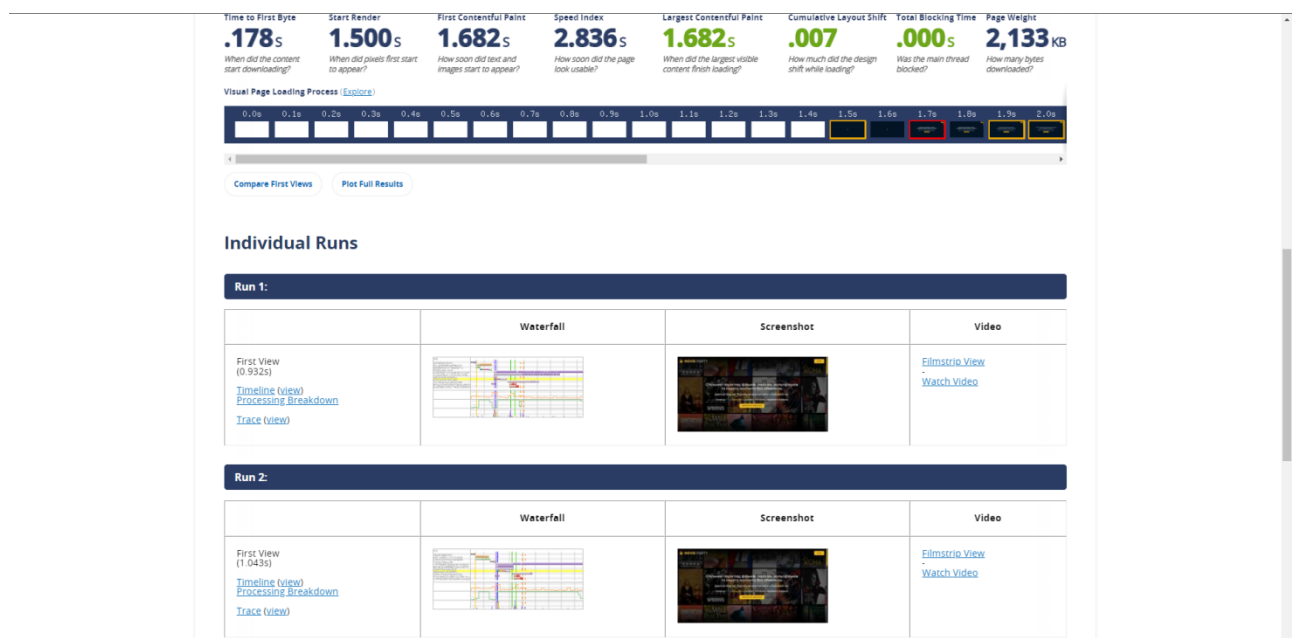


Рисунок 3.36 – Тестування веб-додатку за допомогою WPT (Водоспад сторінки)

Для тестування навантаження веб-додатку використаємо JMeter.

JMeter – це інструмент для тестування продуктивності веб-додатків, розроблений Apache Software Foundation. Він дозволяє створювати навантажувальні тести, які імітують активність тисяч користувачів одночасно.

JMeter підтримує різні протоколи, включаючи HTTP, HTTPS, SOAP, REST, FTP, та

дозволяє вимірювати різні метрики продуктивності, такі як час відповіді сервера, пропускна здатність і завантаження CPU. Інструмент має графічний інтерфейс для зручного створення тестів і може бути легко інтегрований у процеси автоматизації тестування через підтримку скриптів на мові програмування Groovy або Java. JMeter є відкритим програмним забезпеченням і широко використовується для забезпечення надійності та продуктивності веб-додатків. На рис. 3.37 наведено параметри навантаження для тестування веб-сторінки.

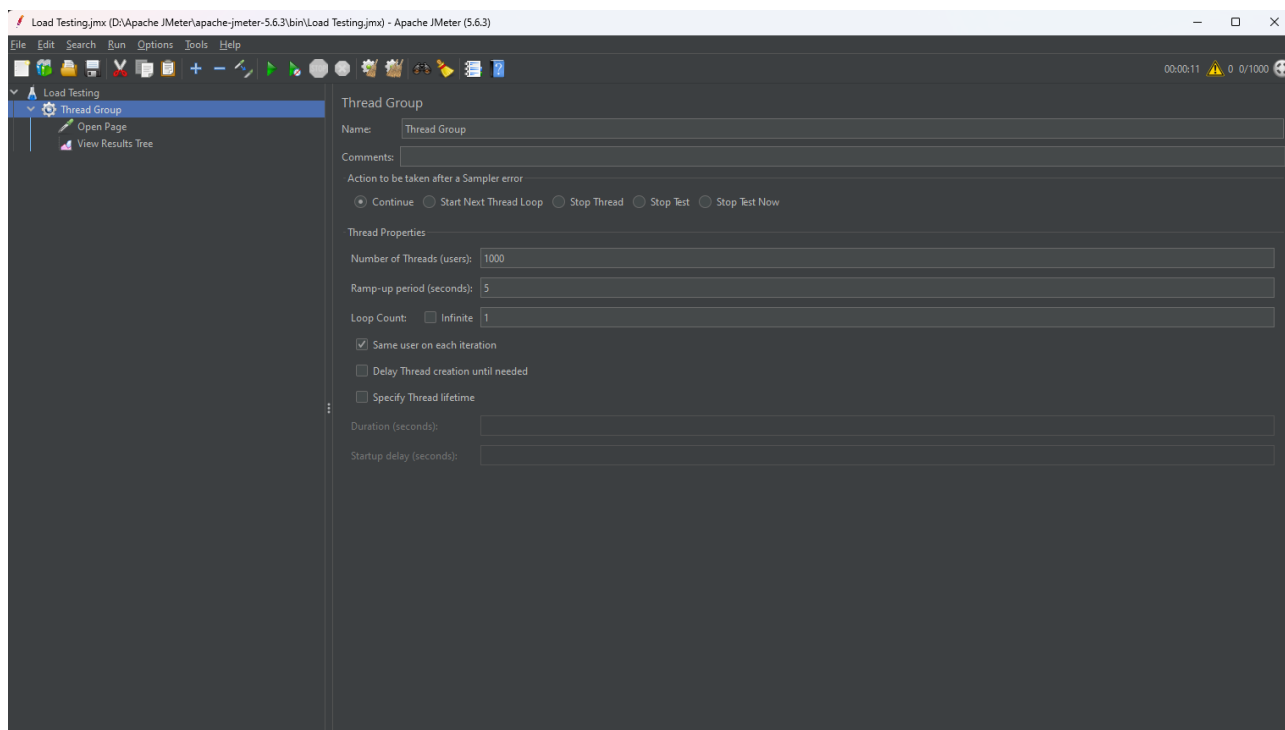


Рисунок 3.37 – Навантажувальне тестування веб-додатку (Параметри навантаження)

Встановимо кількість користувачів (потоків) рівною 1000. Це буде відповідати кількості одночасних користувачів, які будуть навантажувати веб-додаток протягом 5 секунд. На рис. 3.38 наведено результати навантажувального тестування.

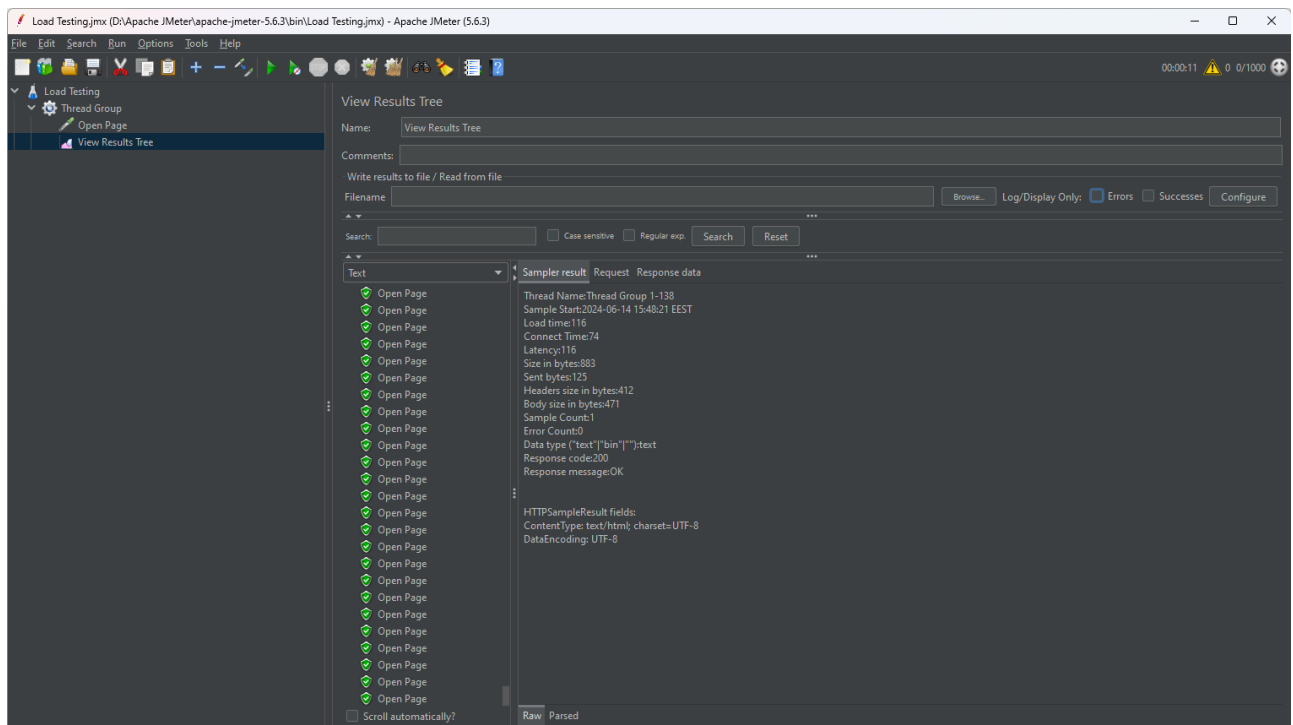


Рисунок 3.38 – Навантажувальне тестування веб-додатку (Результат)

Навантажувальне тестування веб-додатку з 1000 користувачами протягом 5 секунд завершилося успішно. Всі запити позначені зеленими іконками, що вказує на їх успішне виконання. Це свідчить про те, що сервер зміг обробити всі запити без помилок. Тестова конфігурація показує, що кожна сторінка була відкрита правильно і відповідь від сервера була отримана без будь-яких проблем. Відсутність червоних або жовтих іконок у дереві результатів означає, що не було зафіксовано жодних помилок або збоїв протягом тестування. Це є позитивним показником стійкості та ефективності веб-додатку під значним навантаженням.

## Висновки до третього розділу

Детально розглянуто процес розгортання веб-додатку, починаючи з вибору відповідних клієнтської та серверної платформ для розміщення різних компонентів додатку. Було проаналізовано різні варіанти платформ та інфраструктур для оптимального розгортання додатку, враховуючи такі аспекти, як масштабованість, продуктивність, безпека та зручність адміністрування. На основі проведеного аналізу вибрано найкращі рішення для кожного компонента додатку. Описано

деталі взаємодії між клієнтською та серверною частинами веб-сервісу, включаючи механізми передачі та збереження даних, автентифікації користувачів.

Також було детально описано інтерфейс розробленого програмного продукту. Розглянуто всі функціональні можливості додатку, які залежать від ролі користувача. Кожен користувач має свої унікальні можливості, що дозволяють ефективно виконувати необхідні завдання. Інтерфейс додатку розроблено з урахуванням принципів зручності та інтуїтивності, що дозволяє легко зорієнтуватися у функціоналі навіть новим користувачам. Також забезпечено високу ступінь доступності інтерфейсу, включаючи можливості для користувачів з обмеженими можливостями. Це досягається шляхом впровадження адаптивного дизайну, який автоматично підлаштовується під різні розміри екранів та типи пристроїв, а також використанням зручних шрифтів і контрастних кольорів.

Було проведено тестування ефективності веб-додатку двома онлайн сервісами PageSpeed Insights та WebPageTest, проведено навантажувальне тестування, яке показало, що веб-додаток зміг обробити всі запити без помилок.

## ВИСНОВКИ

Під час виконання кваліфікаційної роботи бакалавра було спроектовано та розроблено веб-сервіс для спільного перегляду фільмів та серіалів. Проведено детальний аналіз предметної області, включаючи вивчення потреб користувачів та функціональних вимог, а також глибокий аналіз існуючих аналогів, що допомогло визначити їхні сильні та слабкі сторони та виявити можливості для вдосконалення. На основі аналізів було обрано оптимальну архітектуру, інструменти та технології для реалізації сервісу з акцентом на продуктивність, масштабованість, безпеку та зручність розробки й підтримки – мову програмування JavaScript, бібліотеку Express.js, бібліотеку React.js. Це дозволило створити інноваційний і надійний продукт, що відповідає сучасним стандартам якості та забезпечує унікальний користувацький досвід.

Проектування веб-сервісу включало всебічний аналіз варіантів використання програмного продукту та детальне проектування його структури. Зокрема, було змодельовано взаємодію між усіма складовими частинами системи, що дозволило забезпечити їх узгоджену роботу. Було також спроектовано базу даних для ефективного отримання, зберігання та управління інформацією через серверну частину додатку, яка містить 15 колекцій.

В результаті було створено зручний та комфортний веб-сервіс для спільного перегляду фільмів та серіалів, що дозволяє користувачам швидко та легко створювати віртуальні кімнати для спільного перегляду, де вони можуть насолоджуватися різним відеоконтентом разом, незалежно від їхнього географічного розташування.

Інтерфейс сервісу спроектовано з урахуванням принципів інтуїтивності та зручності використання, що дозволяє легко зорієнтуватися навіть новачкам. Надійна інфраструктура та використання сучасних технологій забезпечують високу продуктивність та стабільність роботи додатку.

## СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Netflix Party [Електронний ресурс] – Режим доступу до ресурсу: <https://www.netflixparty.com.co/uk>.
2. Scener [Електронний ресурс] – Режим доступу до ресурсу: <https://www.scener.com/>.
3. Kast [Електронний ресурс] – Режим доступу до ресурсу: <https://kastapp.co/>.
4. Takflix [Електронний ресурс] – Режим доступу до ресурсу: <https://takflix.com/uk>.
5. MVC Design Pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://www.geeksforgeeks.org/mvc-design-pattern/>.
6. REST API Pattern [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.stoplight.io/api-design-patterns-for-rest-web-services>.
7. Single-Page Application [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Glossary/SPA>.
8. VS Code documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://code.visualstudio.com/docs>.
9. TypeScript documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://www.typescriptlang.org/docs/>.
10. Node.js documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://nodejs.org/en/docs>.
11. Express.js documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://expressjs.com/>.
12. Socket.io documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://socket.io/docs/v4/>.
13. WebRTC documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://webrtc.org/>.
14. React.js documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://legacy.reactjs.org/docs/getting-started.html>.
15. Vite documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://vitejs.dev/>.

16. MongoDB documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://www.mongodb.com/docs/>.

17. Vercel documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://vercel.com/docs>.

18. Netlify documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.netlify.com/>.

19. React Router [Электронный ресурс] – Режим доступа до ресурсу: <https://reactrouter.com/en/main>.

20. Stripe API documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://docs.stripe.com/api>.

21. Web Development with Node and Express: Leveraging the JavaScript Stack 2nd Edition, 2019. – 340 с. – (O'Reilly).

22. Cherny B. Programming TypeScript: Making Your JavaScript Applications Scale 1st Edition / Boris Cherny., 2019. – 342 с.

23. Lombardi A. WebSocket: Lightweight Client-Server Communications / Andrew Lombardi., 2015. – 141 с.

24. Manson R. Getting Started with WebRTC / Rob Manson., 2013. – 114 с.

25. Bradshaw S. MongoDB: The Definitive Guide / S. Bradshaw, E. Brazil, C. Kristina., 2020. – 512 с.

26. Gordon Z. React Explained: Your Step-By-Step Guide to React / Zac Gordon., 2019. – 168 с.

# ДОДАТКИ

|  |  |  |  |  |                     |    |
|--|--|--|--|--|---------------------|----|
|  |  |  |  |  | ІПЗ.КР.Б-121-24-ІПЗ |    |
|  |  |  |  |  |                     |    |
|  |  |  |  |  |                     | 86 |

```

import dotenv from "dotenv";
import "express-async-errors";
import express from "express";
import { createServer } from "http";
import { Server } from "socket.io";
import cors from "cors";
import { json } from "body-parser";
import connectToDB from "../db/connection";
import authRouter from "../routes/auth.route";
import genreRouter from "../routes/genre.route";
import listRouter from "../routes/list.route";
import countryRouter from "../routes/country.route";
import actorRouter from "../routes/actor.route";
import directorRouter from "../routes/director.route";
import partRouter from "../routes/part.route";
import selectionRouter from "../routes/selection.route";
import reviewRouter from "../routes/review.route";
import videoContentRouter from "../routes/content.route";
import roomRouter from "../routes/room.route";
import typeContentRouter from "../routes/type-content.route";
import subscriptionRouter from "../routes/subscription.route";
import errorMiddleware from "../middlewares/error.middleware";
import RoomRepository from "../repositories/room.repository";
import MessageRepository from "../repositories/message.repository";
import { Types } from "mongoose";
import passport from "passport";
import "../config/passportGoogleAuth";
import "../config/passportJWT";
import cookieSession from "cookie-session";
import cookieParser from "cookie-parser";
import { authenticatedUser } from "../middlewares/auth.middleware";
import { Room } from "../utils/interfaces";

dotenv.config();

const app = express();

// static

```

```

app.use("/api/v1/static/files", express.static(__dirname + "/files"));

app.use(json());
app.use(
  cors({
    origin: [
      "http://localhost:3000",
      "http://192.168.1.59:3000",
      process.env.CLIENT_URL!,
    ],
    credentials: true,
    methods: ["GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS"],
  })
);

app.use(cookieParser());

app.use(
  cookieSession({
    name: "google-auth-session",
    keys: [process.env.COOKIE_SECRET as string],
    maxAge: 24 * 60 * 60 * 1000 * 31,
  })
);

app.use(passport.initialize());
app.use(passport.session());

// routers
app.use("/api/v1/auth", authRouter);
app.use("/api/v1/genres", genreRouter);
app.use("/api/v1/lists", listRouter);
app.use("/api/v1/countries", countryRouter);
app.use("/api/v1/actors", actorRouter);
app.use("/api/v1/directors", directorRouter);
app.use("/api/v1/parts", partRouter);
app.use("/api/v1/selections", selectionRouter);
app.use("/api/v1/reviews", reviewRouter);
app.use("/api/v1/content/v/", videoContentRouter);
app.use("/api/v1/rooms", roomRouter);
app.use("/api/v1/content/type", typeContentRouter);

```

|  |  |  |  |  |                     |    |
|--|--|--|--|--|---------------------|----|
|  |  |  |  |  | ИПЗ.КР.Б-121-24-ИПЗ |    |
|  |  |  |  |  |                     |    |
|  |  |  |  |  |                     | 88 |

```

app.use("/api/v1/subscription", subscriptionRouter);

// middleware
app.use(errorMiddleware);

const port = process.env.PORT || 5000;

let userRooms: Room[] = [];

const validRoom = (roomId: string) => {
  const objectIdRegex = /^[0-9a-fA-F]{24}$/;
  return objectIdRegex.test(roomId);
};

const start = async () => {
  try {
    await connectToDB(process.env.MONGO_URI!);

    const server = createServer(app);

    const io = new Server(server, {
      cors: {
        origin: [
          "http://localhost:3000",
          "http://192.168.1.59:3000",
          process.env.CLIENT_URL!,
        ],
        credentials: true,
      },
    });

    const getRooms = () => {
      const { rooms } = io.sockets.adapter;

      return Array.from(rooms.keys()).filter((roomId) =>
validRoom(roomId));
    };

    io.use(async (socket, next) => {
      try {
        const userId = socket.handshake.auth.user._id;
        const user = await authenticatedUser(userId);

```

```

        socket.user = user;
        next();
    } catch (e) {
        console.log(e);
        return next(new Error("Authentication invalid"));
    }
});

io.on("connection", (socket) => {
    const leaveRoom = async () => {
        const { rooms } = socket;

        Array.from(rooms).forEach((roomId) => {
            const roomClients = Array.from(
                io.sockets.adapter.rooms.get(roomId) || []
            );

            roomClients.forEach((clientId) => {
                io.to(clientId).emit("web_rtc_un_pair", {
                    peerId: socket.id,
                });

                socket.emit("web_rtc_un_pair", {
                    peerId: clientId,
                });
            });

            socket.leave(roomId);
        });
    };

    socket.join(socket.user?._id);

    socket.on("join", async (roomId) => {
        const existUser = await RoomRepository.existUserInRoom(
            roomId,
            socket.user?._id
        );

        if (existUser) {
            const { rooms: joinedRooms } = socket;

```

```

if (Array.from(joinedRooms).includes(roomId)) {
  return console.warn(
    `${socket.user?._id} is already joined to ${roomId}`
  );
}

const roomClients = Array.from(
  io.sockets.adapter.rooms.get(roomId) || []
);

roomClients.forEach((clientId) => {
  io.to(clientId).emit("web_rtc_peer", {
    peerId: socket.id,
    createOffer: false,
  });

  socket.emit("web_rtc_peer", {
    peerId: clientId,
    createOffer: true,
  });
});

await RoomRepository.connectUser(roomId, socket.user?._id);

socket.join(roomId);

socket.roomId = roomId;

let room = userRooms.find((room) => room.roomId === roomId);

if (!room) {
  userRooms.push({
    roomId,
    isPlaying: false,
    time: 0,
    episode: 0,
    season: 0,
  });

  room = userRooms.find((room) => room.roomId === roomId);
}

```

```

        console.log(`User ${socket.user?.nickname} connected to room`);

        socket.to(roomId).emit("update_live");
        socket.emit("update_live");
        socket.emit("play", room?.isPlaying || false);
        socket.emit("time", room?.time || 0);
        socket.emit("update_series", room);

        const messages = await MessageRepository.getLastMessagesByRoomId(
            roomId
        );

        socket.to(roomId).emit("receive_messages", messages);
        socket.emit("receive_messages", messages);
    } else {
        socket.emit("connection_error");
    }
});

socket.on("send_message", async (roomId, data) => {
    if (roomId) {
        await MessageRepository.addMessage({
            roomId: new Types.ObjectId(roomId),
            userId: socket.user?._id,
            message: data.message,
        });

        const messages = await MessageRepository.getLastMessagesByRoomId(
            roomId
        );

        socket.to(roomId).emit("receive_messages", messages);
        socket.emit("receive_messages", messages);
    }
});

socket.on("seek", (roomId, data) => {
    const room = userRooms.find((room) => room.roomId === roomId);
    if (roomId && room) {
        room.time = data;
        socket.to(roomId).emit("seek", data);
    }
});

```

```

});

socket.on("play", (roomId, data) => {
    const room = userRooms.find((room) => room.roomId === roomId);

    if (roomId && room) {
        room.isPlaying = Boolean(data);
        socket.to(roomId).emit("play", data);
    }
});

socket.on("time", (roomId, time) => {
    const room = userRooms.find((room) => room.roomId === roomId);

    if (roomId && room) {
        if (time !== 0) {
            room.time = time;
        }
    }
});

socket.on("sync", (roomId) => {
    const room = userRooms.find((room) => room.roomId === roomId);
    if (roomId && room) {
        socket.emit("time", room?.time || 0);
    }
});

socket.on("update_live", (roomId) => {
    const room = userRooms.find((room) => room.roomId === roomId);

    if (roomId && room) {
        room.isPlaying = false;
        room.time = 0;
        socket.to(roomId).emit("update_live");
        socket.to(roomId).emit("play", false);
    }
});

socket.on("update_series", (roomId, data) => {
    const room = userRooms.find((room) => room.roomId === roomId);

```

```

    if (roomId && room) {
        room.isPlaying = false;
        room.time = 0;
        room.season = data.season;
        room.episode = data.episode;
        socket.to(roomId).emit("play", false);
        socket.to(roomId).emit("seek", 0);
        socket.to(roomId).emit("update_series", room);
        socket.to(roomId).emit("seek", 0);
    }
});

socket.on("finish_session", (roomId) => {
    const room = userRooms.find((room) => room.roomId === roomId);

    if (roomId && room) {
        userRooms = userRooms.filter((room) => room.roomId !== roomId);
        socket.to(roomId).emit("finish_session");
    }
});

socket.on("web_rtc_leave", leaveRoom);

socket.on("web_rtc_relay_sdp", ({ peerId, sessionDescription }) => {
    io.to(peerId).emit("web_rtc_session_description", {
        peerId: socket.id,
        sessionDescription,
    });
});

socket.on("web_rtc_relay_ice", ({ peerId, iceCandidate }) => {
    io.to(peerId).emit("web_rtc_ice_candidate", {
        peerId: socket.id,
        iceCandidate,
    });
});

socket.on("disconnect", async () => {
    const roomId: string = socket.roomId!;

    if (roomId) {
        await RoomRepository.disconnectUser(roomId, socket.user?._id);
    }
});

```

```

        socket.to(roomId).emit("update_live");
    }

    console.log(`User ${socket.user?.nickname} disconnected room`);
  });
});

server.listen(port, () => {
  console.log(`Server is listening on ${port}`);
});
} catch (err) {
  console.error(err);
}
};

start();

```

```

import React, { useEffect, useRef, useState } from "react";
import { CopyToClipboard } from "react-copy-to-clipboard";
import { toast } from "react-toastify";
import "../style/page.scss";
import { useForm } from "react-hook-form";
import { yupResolver } from "@hookform/resolvers/yup";
import { MessageSchema } from "../../features/validations";
import { Link, useNavigate, useParams } from "react-router-dom";
import { useGetRoomByIdQuery } from
"../../features/services/rooms/roomsService";
import useFill from "../../hooks/useFill";
import {
  Avatar,
  Loader,
  MessageItems,
  NotFound,
  VideoPlayer,
} from "../../components";
import Rating from "../../components/Rating/Rating";
import Favorite from "../../components/Favorites/Favorite";
import { formatDate } from "../../features/utils/functions";
import useUser from "../../hooks/useUser";
import useRoom from "../../hooks/useRoom";
import useSocket from "../../hooks/useSocket";
import useVideoPlayer from "../../hooks/useVideoPlayer";
import usePopUp from "../../hooks/usePopUp";
import RoomEditPopup from "../RoomEditPopup";
import useWebRTC from "../../hooks/useWebRTC";

const RoomPage = () => {
  const { id: roomId } = useParams();
  const {} = useFill();
  const { user } = useUser();
  const { autoPlay, setAutoPlay } = useState(false);
  const {
    isPlaying,
    time,
    handlePlay,
    handlePause,
  }

```

```

    season,
    episode,
    setSeason,
    setEpisode,
  } = useVideoPlayer();
  const {
    isChatOpen,
    isUsersOpen,
    isMicOn,
    toggleChat,
    toggleUsers,
    toggleMicrophone,
  } = useRoom();
  const { socket, connect, disconnect } = useSocket();
  const { roomClients, provideAudioRef, muteMicrophone, unmuteMicrophone }
=
    useWebRTC();
  const { editId, handleEditPopUp } = usePopUp();
  const [messages, setMessages] = useState([]);
  const [seek, setSeek] = useState(0);
  const [roomOwner, setRoomOwner] = useState(null);
  const isMounted = useRef(false);
  const navigate = useNavigate();

  const {
    register,
    handleSubmit,
    reset,
    formState: { errors, isDirty, isValid },
  } = useForm({
    resolver: yupResolver(MessageSchema),
  });

  const { data, isLoading, refetch } = useGetRoomByIdQuery(roomId);

  useEffect(() => {
    if (!isLoading && data) {
      socket.auth.user._id = user._id;
      connect();
      socket.emit("join", roomId);
    }
  })

```

```

    }, [data, isLoading]);

    useEffect(() => {
        if (!isMounted.current) {
            return;
        }

        if (!roomOwner) {
            return;
        }

        if (socket) {
            socket.on("update_live", () => {
                refetch();
            });
            socket.on("receive_messages", (data) => {
                setMessages(data);
            });

            socket.on("seek", (seek) => {
                setSeek(seek);
            });

            socket.on("play", (play) => {
                if (play) {
                    handlePlay();
                } else {
                    handlePause();
                }
            });

            socket.on("update_series", ({ season, episode }) => {
                if (user._id === roomOwner._id) return;
                setSeason(season);
                setEpisode(episode);
            });

            socket.on("time", (time) => {
                setSeek(time);
            });

            socket.on("finish_session", () => {

```

```

        navigate("/");
    });
}

return () => {
    socket.disconnect();
};
}, [socket, roomOwner]));

useEffect(() => {
    if (!isMounted.current) {
        return;
    }

    if (!roomOwner) {
        return;
    }

    if (user._id !== roomOwner._id) return;

    socket.emit("play", roomId, isPlaying);
    socket.emit("time", roomId, time);
}, [isPlaying, roomOwner]);

useEffect(() => {
    if (!isMounted.current) {
        return;
    }

    if (!roomOwner) {
        return;
    }

    if (seasons.length == 0) {
        return;
    }

    if (user._id !== roomOwner._id) return;

    handlePause();

    socket.emit("update_series", roomId, {

```

```

        season,
        episode,
    ));
}, [season, episode]);

useEffect(() => {
    if (!isMounted.current) {
        return;
    }

    if (!roomOwner) {
        return;
    }

    if (user._id !== roomOwner._id) return;

    socket.emit("time", roomId, time);
}, [time, roomOwner]);

useEffect(() => {
    if (data?.ownerUser) {
        setRoomOwner(data.ownerUser);
    }
}, [data]);

useEffect(() => {
    if (!isMounted.current) {
        isMounted.current = true;
        return;
    }

    return () => {
        disconnect();
    };
}, []);

useEffect(() => {
    if (!isMounted.current) {
        return;
    }

    if (editId) {

```

```

        if (isMicOn) {
            handleMicToggle();
        }
        handlePause();
    }
}, [editId]);

useEffect(() => {
    const handleBeforeUnload = () => {
        if (roomOwner._id === user._id) socket.emit("play", roomId, false);
    };

    window.addEventListener("beforeunload", handleBeforeUnload);

    return () => window.removeEventListener("beforeunload",
handleBeforeUnload);
}, [roomOwner]);

const handleMicToggle = async () => {
    if (!isMicOn) {
        unMuteMicrophone();
    } else {
        muteMicrophone();
    }
    toggleMicrophone();
};

const onSubmitMessage = async (data) => {
    socket.emit("send_message", roomId, { message: data.message });
    reset();
};

if (isLoading) {
    return <Loader />;
}

if (!data) {
    return (
        <NotFound
            title="Кімнату не знайдено"
            image="room"
            description="Кімнати не існує чи сеанс кімнати було завершено"

```

```

        />
    );
}

const {
    title: roomTitle,
    videoContent,
    ownerUser,
    users,
    inviteCode,
    voiceChat,
} = data;

const {
    _id: videoContentId,
    title,
    originTitle,
    previewURL,
    IMDb,
    rating,
    description,
    originCountries,
    duration,
    releaseDate,
    genres,
    typeVideoContent,
    lists,
    soundTracks,
    seasons,
} = videoContent;

const contentPreviewURL = `${
    import.meta.env.VITE_BACK_HOST
}/static/files/images/content/${previewURL}`;

const inviteLink = `${
    import.meta.env.VITE_FRONT_HOST
}/room/invite?code=${inviteCode}`;

return (
    <div className="container cnt-mn">

```

```

<div className="container">
  <div className="wrapper">
    <div className="room">
      <div className="room__grid">
        <div className={`room__player${!isChatOpen ? " full" : ""}`}>
          <VideoPlayer
            controls={{
              seek: ownerUser._id === user._id,
              play: ownerUser._id === user._id,
              pInp: ownerUser._id === user._id,
              settings: {
                speed: ownerUser._id === user._id,
              },
            }}
            autoplay={autoplay}
            seek={seek}
            soundTracks={soundTracks}
            seasons={seasons}
            handleSeekChange={(seek) => socket.emit("seek", roomId,
seek)}}
          />
        <div className="room__player-buttons">
          {ownerUser._id !== user._id && (
            <button
              className="room__player-button-sync"
              onClick={() => socket.emit("sync", roomId)}
            >
              <div className="icon sync" />
            </button>
          )}
          {!isChatOpen && (
            <button
              className="room__player-button-chat"
              onClick={toggleChat}
            >
              <div className="icon c-arrow" />
            </button>
          )}
        </div>
      </div>
    </div>
  </div>
</div>

```

```

{isChatOpen && (
  <div className="room__chat">
    <div className="chat">
      <div className="chat__bar">
        <button className="chat__button"
onClick={toggleChat}>
          <div className="icon c-arrow" />
        </button>
        <div className="chat__title">
          {!isUsersOpen ? "Чат" : "Користувачі"}
        </div>
        <button className="chat__button"
onClick={toggleUsers}>
          <div
            className={`icon${
              !isUsersOpen ? " c-users" : " c-icon"
            }`}
          />
        </button>
      </div>
      {isUsersOpen && (
        <div className="chat__users">
          <div className="chat__user">
            <div className="chat__user-info">
              <div className="chat__user-avatar">
                <Avatar
                  photoURL={user.avatarURL}
                  nickname={user.nickname}
                  avatarColor={user.avatarColor}
                  width={40}
                  height={40}
                />
              </div>
            </div>
            <div className="chat__user-nickname">
              {user.nickname} (Ви)
            </div>
          </div>
          {voiceChat && (
            <div className="chat__user-actions">
              <div

```

```

        className="chat__button"
        onClick={handleMicToggle}
      >
        <div
          className={`icon microphone${
            isMicOn ? " on" : " off"
          }`}
        />
      </div>
    </div>
  )}
</div>
{users.map((u) => {
  if (u._id !== user._id) {
    return (
      <div className="chat__user">
        <div className="chat__user-info">
          <div className="chat__user-avatar">
            <Avatar
              photoURL={u.avatarURL}
              nickname={u.nickname}
              avatarColor={u.avatarColor}
              width={40}
              height={40}
            />
          </div>
          <div className="chat__user-nickname">
            {u.nickname}{" "}
            {u._id === user._id ? "(Вы)" : ""}
          </div>
        </div>
      </div>
    );
  }
})}
</div>
)}
{!isUsersOpen && <MessageItems messages={messages} />}
<div className="chat__send-form">
  <div className="chat__container-f-s">

```

```

<div
  className={`chat__sender${
    isMicOn ? " m-on speak" : ""
  }}
>
  <div className="chat__sender-avatar">
    <Avatar
      photoURL={user.avatarURL}
      nickname={user.nickname}
      avatarColor={user.avatarColor}
      width={40}
      height={40}
    />
  </div>
  <div className="chat__sender-nickname">
    {user.nickname}
  </div>
</div>
{voiceChat && (
  <button
    className="chat__button"
    onClick={handleMicToggle}
  >
    <div
      className={`icon microphone${
        isMicOn ? " on" : " off"
      }}
    />
  </button>
)}
</div>
<form
  className="chat__form"
  onSubmit={handleSubmit(onSubmitMessage)}
>
  <input
    type="text"
    {...register("message")}
    className="form__input linear"
  />

```

```

        <button
            className="chat__button"
            type="submit"
            disabled={!isValid || !isDirty}
        >
            <div className="icon send" />
        </button>
    </form>
</div>
</div>
</div>
)}
<div className="room__info-bar">
    <div className="room__owner">
        <div className="room__owner-avatar">
            <Avatar
                photoURL={ownerUser.avatarURL}
                nickname={ownerUser.nickname}
                avatarColor={ownerUser.avatarColor}
                width={75}
                height={75}
            />
        </div>
        <div className="room__owner-info">
            <div className="room__owner-nickname">
                {ownerUser.nickname}
            </div>
            <div className="room__title">{roomTitle}</div>
        </div>
    </div>
    <div className="room__details">
        <div className="room__details-live">
            <div className="icon live" />
            <div className="room__details-
users">{users.length}</div>
        </div>
        {ownerUser._id === user._id && (
            <>
                <CopyToClipboard
                    onCopy={() =>

```

```

        toast.success("Запрошувальне посилання
скопійовано")

    }
    text={inviteLink}
  >
    <button className="room__details-action">
      <div className="icon copy" />
    </button>
  </CopyToClipboard>
  <button
    className="room__details-action"
    onClick={() => handleEditPopUp(roomId)}
  >
    <div className="icon settings" />
  </button>
</>
  )}
</div>
</div>
<div className="room__info">
  <div className="room__content">
    <div className="room__content-
description">{description}</div>
    <div className="room__content-info">
      <div className="room__content-title">
        Обраний контент для перегляду:
      </div>
      <div className="video-content__content ng">
        <div className="video-content__container wrapper">
          <div className="video-content__preview">
            <div className="video-content__preview-image">
              <img src={contentPreviewURL} alt={title} />
            </div>
            <div className="video-content__preview-actions">
              <Favorite videoContentId={videoContentId} />
            </div>
          </div>
          <div className="video-content__details">
            <div className="video-content__title-container">
              <div className="video-content__titles">

```

```

        <div className="video-content__title">
            {title}
        </div>
        <div className="video-content__original-
title">
            {originTitle}
        </div>
    </div>
    <div className="IMDb">
        <div className="icon IMDb" />
        <p
className="IMDb__rating">{IMDb.toFixed(1)}</p>
    </div>
</div>
<div className="video-content__rating">
    <div className="rating">
        <Rating
            rating={rating.averageRating || 0}
            maxRating={5}
            disabled={true}
        />
        <div className="rating__point">
            {rating.averageRating || 0}
        </div>
        <div className="rating__votes">
            <div className="icon group-users" />
            <div className="rating__votes-amount">
                {rating.voteCount} голосів
            </div>
        </div>
    </div>
</div>
<div className="video-content__information">
    <div className="video-content__information-
item">
        <div className="video-content__information-
title">
            Країна:
        </div>
    </div>
</div>

```

```

<div className="video-content__information-
content flex r g5 center-h">
    {originCountries.map((country, index) => {
        return (
            <div
                className="video-
content__information-text"
                key={country._id}
            >
                {country.name}
                {index !== originCountries.length - 1
                    ? ", "
                    : ""}
            </div>
        );
    })}
</div>
<div className="video-content__information-
item">
    <div className="video-content__information-
title">
        Тривалість:
    </div>
    <div className="video-content__information-
content">
        <div className="video-content__information-
text">
            {duration}
        </div>
    </div>
</div>
<div className="video-content__information-
item">
    <div className="video-content__information-
title">
        Дата релізу:
    </div>
    <div className="video-content__information-
content">

```

```

        <div className="video-content__information-
text">
            {formatDate(releaseDate)}
        </div>
    </div>
</div>
<div className="video-content__information-item
start genres">
    <div className="video-content__information-
title">
        Жанр:
    </div>
    <div className="video-content__information-
content">
        <div className="link__items">
            {genres.map((genre) => {
                const linkGenre = `${
                    typeVideoContent.path
                }/genre/${genre.originName.toLowerCase(
))}`;
                return (
                    <Link
                        to={linkGenre}
                        className="link outlined"
                        key={genre._id}
                    >
                        {genre.name}
                    </Link>
                );
            })}
        </div>
    </div>
</div>
{Boolean(lists.length) && (
    <div className="video-content__information-
item start v-lists">
        <div className="video-content__information-
title">
            Входить до:
        </div>

```

content">

```
<div className="video-content information-
```

```
<div className="link items col">
```

```
{lists.map((item) => {
  const {
    list: { _id, name
    placeInList,
  } = item;
```

```
const linkBest =
```

```
`/${typeVideoContent.path}/best-list/${name}`;
```

```
return (
```

<Link

```
to={linkBest}
```

className="link"

```
key={ id}
```

>

<div className="lists">

```
<div className="lists__name">
```

```
{name}
```

&lt;/div&gt;

```
<div className="lists__place">
```

```

({placeInList} micce)

```

&lt;/div&gt;

&lt;/div&gt;

&lt;/Link&gt;

);

} ) }

&lt;/div&gt;

&lt;/div&gt;

&lt;/div&gt;

 $\left. \vphantom{\int} \right\}$ 

&lt;/div&gt;

&lt;/div&gt;

&lt;/div&gt;

&lt;/div&gt;

&lt;/div&gt;

&lt;/div&gt;

&lt;/div&gt;

&lt;/div&gt;

```

</div>
<div className="room__audio">
  {roomClients.map((roomId) => {
    return (
      <audio
        key={roomId}
        ref={(instance) => {
          provideAudioRef(roomId, instance);
        }}
        autoPlay
        muted={roomId === "user_local_audio"}
      />
    );
  })}
</div>
{editId && (
  <RoomEditPopup
    handleUpdate={() => socket.emit("update_live", roomId)}
    handleFinish={() => socket.emit("finish_session", roomId)}
  />
)}
</div>
</div>
<div className="splitter"></div>
</div>
);
};

export default RoomPage;

```