Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

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

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему ІНЖЕНЕРІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ВЕБПОРТАЛУ ДЛЯ ПЕРЕГЛЯДУ ВІДЕО З ОЦІНЮВАННЯМ ТА АДМІНІСТРУВАННЯМ КОРИСТУВАЦЬКОЇ ВЗАЄМОДІЇ
SOFTWARE ENGINEERING OF A VIDEO CONTENT WEB PORTAL WITH USER INTERACTION MANAGEMENT AND RATING SYSTEM

Виконавла: студентка 4 курсу, групи КНТ-122
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)
забезпечення

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

ГОНЧАРЕНКО К.С.
(ПРІЗВИЩЕ та ініціали)
Керівник ОЛІЙНИК А.О.
(ПРІЗВИЩЕ та ініціали)
Рецензент ШИТІКОВА О.В.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

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

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти бакалавр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації)

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
_____Сергій СУББОТІН
"____" _____2026 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ РОБОТУ СТУДЕНТКИ

ГОНЧАРЕНКО Ксенії Сергіївни
(ПРІЗВИЩЕ, ім'я, по батькові)

1. Тема роботи Інженерія програмного забезпечення вебпорталу для перегляду відео з оцінюванням та адмініструванням користувацької взаємодії.
Software Engineering of a Video Content Web Portal with User Interaction Management and Rating Systems.
керівник роботи д.т.н., професор, ОЛІЙНИК Андрій Олександрович,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)
затверджені наказом закладу вищої освіти від "07" квітня 2026 року № 139
2. Строк подання студентом проєкту роботи 12 червня 2026 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Розробка програми 4. Експлуатація та тестування програми.

5. Перелік графічного матеріалу (з з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ОЛІЙНИК А.О., професор		
Нормоконтроль	КАЛІНІНА М.В., асистент		

7. Дата видачі завдання «14» квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного роботи	Строк виконання етапів роботи	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
	розробки.		
4	Розробка архітектури програми.	2 тиждень	Розділ 3
5	Розробка програми.	3-4 тижні	Розділ 3, 4
6	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
7	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
8	Нормоконтроль та рецензування.	7 тиждень	
9	Захист роботи.	8 тиждень	

Студентка _____ Ксенія ГОНЧАРЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник роботи _____ Андрій ОЛІЙНИК
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 88 с., 4 табл., 27 рис., 3 дод., 16 джерел.

HTTP, FRONTEND, NODE.JS, JAVASCRIPT, MONGODB, REST API.

Об’єкт дослідження – процес розробки та підтримки програмного забезпечення для розповсюдження та управління мультимедійним контентом.

Предмет дослідження – розробка методів, алгоритмів і програмних засобів для створення інтерактивного вебпорталу, на якому можна дивитися кіноконтент, а також забезпечити автоматизоване адміністрування.

Мета роботи – зменшення часу на оновлення медіаресурсів та підвищення швидкості доступу користувачів до цільового кіноконтенту шляхом розроблення спеціалізованого програмного забезпечення вебпорталу з оптимізованою структурою даних, системою групування за жанрами та автоматизованою панеллю адміністрування.

Матеріали, методи та технічні засоби: структурне та об’єктно-орієнтоване програмування, мови програмування JavaScript та TypeScript, мови розмітки та стилізації HTML, CSS (препроцесор SASS), середовище виконання Node.js, фреймворк Express для побудови REST API, нереляційна база даних MongoDB, використання зовнішніх програмних інтерфейсів, засоби контролю версій та управління проєктом, персональний комп’ютер з процесором Intel Core i5 під управлінням операційної системи Microsoft Windows 11.

Результати. Створено програмне забезпечення клієнт-серверної архітектури інтерактивного вебпорталу для розповсюдження кіноконтенту, що включає користувацький інтерфейс та автоматизований модуль адміністрування з інтеграцією зовнішнього API.

Висновки. Розроблено програмне забезпечення вебпорталу, впровадження якого дозволило підвищити швидкість доступу користувачів до цільового контенту шляхом оптимізації структури бази даних MongoDB, впровадження системи групування за жанрами та інтеграції відкритого API бази фільмів для швидкого та автоматизованого отримання актуальної інформації.

Галузь використання – індустрія розваг та медіабізнес (онлайн-кінотеатри, інформаційні кінопортали, стримінгові платформи).

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 88 pages, 4 tables, 27 figures, 3 appendixes, 16 sources.

HTTP, FRONTEND, NODE.JS, JAVASCRIPT, MONGODB, REST API.

Object of the research is the process of developing and supporting software for distributing and managing multimedia content.

Subject of the research is the development of methods, algorithms and software tools for creating an interactive web portal where you can watch film content, as well as provide automated administration.

The purpose of the work is to reduce the time for updating media resources and increase the speed of user access to the target film content by developing specialized web portal software with an optimized data structure, a genre grouping system and an automated administration panel.

Materials, methods and technical means: structured and object-oriented programming, JavaScript and TypeScript programming languages, HTML markup and styling languages, CSS (SASS preprocessor), Node.js runtime environment, Express framework for building REST API, MongoDB non-relational database, use of external program interfaces, version control and project management tools, personal computer with Intel Core i5 processor running Microsoft Windows 11 operating system.

Results. Software for client-server architecture of an interactive web portal for distributing film content was created, which includes a user interface and an automated administration module with external API integration.

Conclusions. Web portal software was developed, the implementation of which allowed to increase the speed of user access to target content by optimizing the structure of the MongoDB database, implementing a genre grouping system and

integrating the open API of the film database for quick and automated receipt of up-to-date information.

The field of use is the entertainment industry and media business (online cinemas, informational film portals, streaming platforms).

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

BSON – Binary JSON;

CMS – Content Management System;

CRUD – Create, Read, Update, Delete;

DOM – Document Object Model;

IDE – Integrated Development Environment;

JSON – JavaScript Object Notation;

ORM – Object-Relational Mapping;

REST – Representational State Transfer;

VOD – Video on Demand;

VS Code – Visual Studio Code;

СУБД – система управління базами даних.

ВСТУП

У сучасних умовах обсяги відеоконтенту постійно зростають, тому виникає потреба у вебпорталах, які забезпечують високу швидкість роботи та зручне управління. Більшість існуючих CMS є універсальними, але через це часто перевантажені зайвим функціоналом, що ускладнює роботу адміністраторів і знижує продуктивність системи. Тому актуальним завданням є розробка спеціалізованого кінопорталу з оптимізованою базою даних і зручною адмін-панеллю.

Метою даної дипломної роботи є створення такого вебпорталу. Для її досягнення було виконано ряд завдань: проведено аналіз сучасних технологій, спроєктовано клієнт-серверну архітектуру та нереляційну базу даних, розроблено серверну частину і інтерактивний інтерфейс користувача, реалізовано модуль адміністрування та проведено тестування системи.

У процесі розробки застосовувалися методи об’єктно-орієнтованого, структурного та асинхронного програмування, а також принципи проєктування баз даних і тестування програмного забезпечення.

Практична цінність роботи полягає у створенні готового рішення для онлайн-кінотеатрів. Використання розробленого порталу дозволяє зменшити час на його підтримку та наповнення, а також забезпечує швидкий і зручний доступ користувачів до контенту.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження предметної області та актуальність проблеми управління медіаресурсами

У сфері дистрибуції мультимедійного контенту, зокрема онлайн-кінотеатрів і відеопорталів, постійно зростає потреба в ефективних програмних рішеннях. Такі системи повинні обробляти великі обсяги даних, включаючи як користувацький контент, так і розширені метадані. Без правильно спроєктованої бази даних і зручної системи управління неможливо забезпечити швидкий пошук інформації та стабільний доступ до медіаресурсів. Разом з тим використання універсальних CMS для створення подібних платформ має ряд недоліків. Вони часто перевантажені зайвим функціоналом, що ускладнює роботу адміністратора. Також структура даних у таких системах не завжди оптимальна для швидкої фільтрації та пошуку, що негативно впливає на продуктивність.

Отже, актуальним є створення спеціалізованого програмного рішення з оптимізованою базою даних, зручною адмін-панеллю та можливістю автоматичного отримання даних. Це дозволить підвищити швидкодію системи та зменшити витрати часу на її обслуговування.

1.2 Огляд та порівняльний аналіз аналогів

1.2.1 Аналог UASerials

Популярний український медіапортал, орієнтований на стримінг кінофільмів та серіалів із великою базою активних користувачів.

Переваги:

– широкий асортимент контенту з розгалуженою системою категорій та підкатегорій;

– висока залученість аудиторії завдяки постійно діючій системі коментарів під відеоматеріалами.

Недоліки:

– неоптимізована клієнтська частина, через велику кількість сторонніх скриптів та важку об'єктну модель документа (DOM) сторінки завантажуються з помітними затримками, що негативно впливає на швидкість доступу до відео.

Демонстрація роботи аналогу наведена на рисунку 1.1 [1].

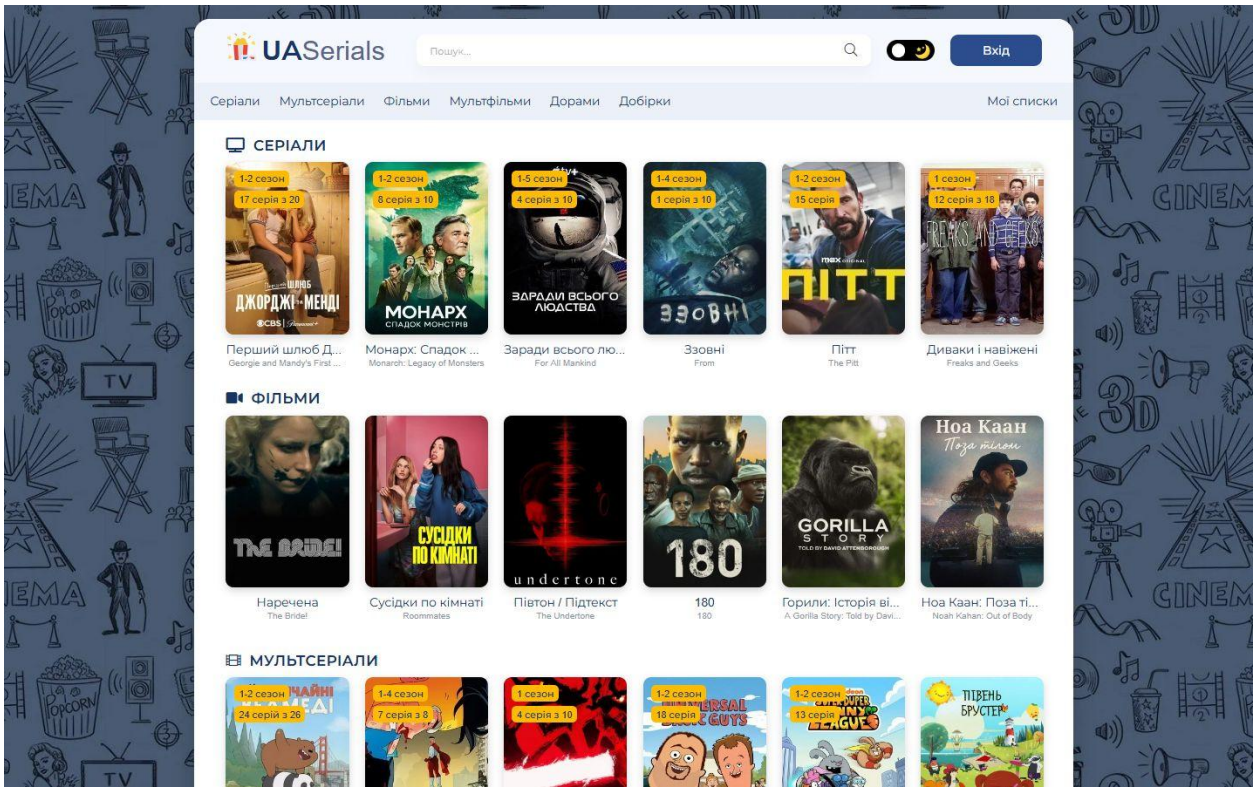

Рисунок 1.1 – Демонстрація роботи UASerials [1]

1.2.2 Аналог Kinoukr.tv

Популярний український онлайн-кінотеатр, що спеціалізується на трансляції відеоконтенту та має широку базу фільмів і серіалів.

Переваги:

– наявність великої кількості локалізованого контенту та чітко визначеної категорії;

– простий, інтуїтивно зрозумілий підхід до навігації сторінками відеоматеріалів.

Недоліки:

– використання монолітної архітектури на базі стандартних «коробкових» систем управління контентом, що передбачає роботу з реляційними базами даних, які повільніше обробляють запити на вибірку складних ієрархічних даних через необхідність об'єднання багатьох таблиць;

– залежність від універсальної архітектури бекенду, оскільки платформа побудована на базі готового рішення, її система управління є універсальною, а не розробленою спеціально під потреби кінопорталу, що означає наявність надлишкового базового функціоналу і уповільнює роботу контент-менеджерів порівняно із кастомними панелями адміністрування.

Демонстрація роботи аналогу наведена на рисунку 1.2 [2].

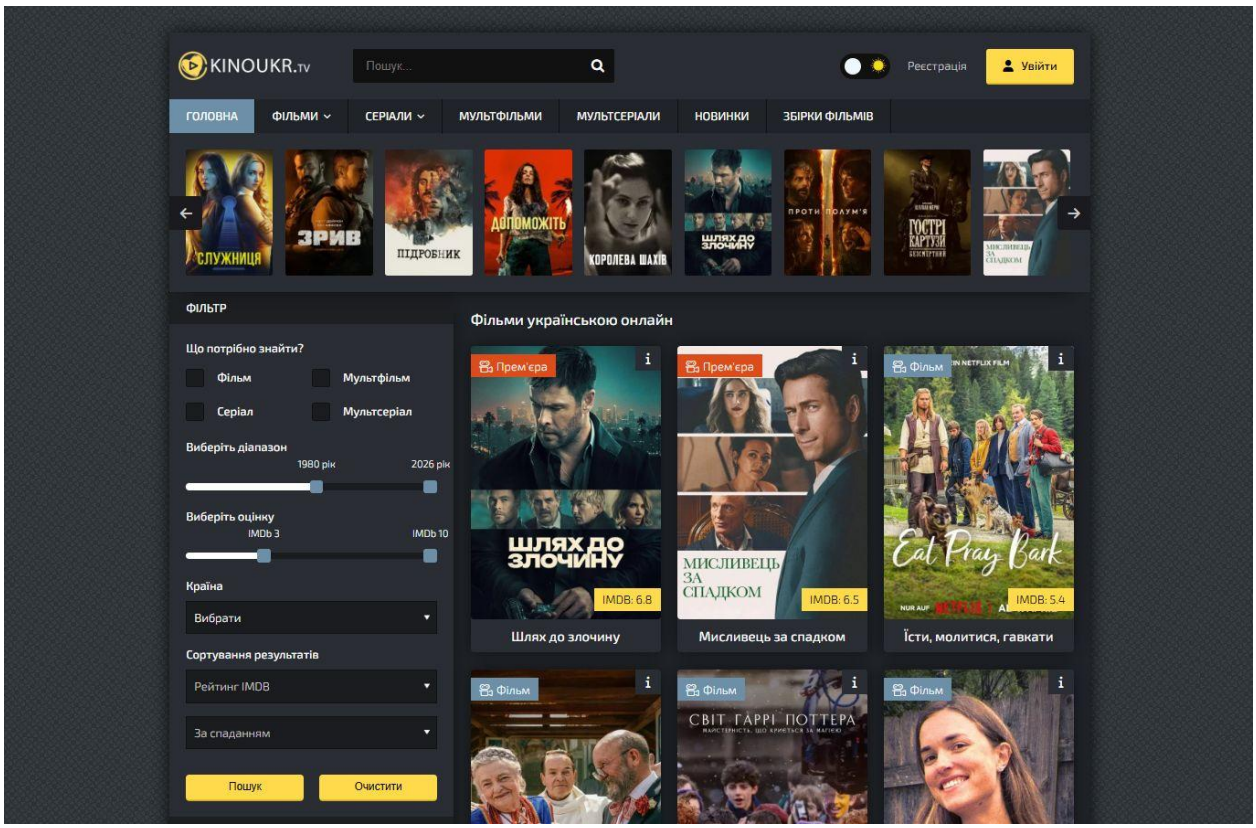

Рисунок 1.2 – Демонстрація роботи Kinoukr.tv [2]

1.2.3 Аналог Megogo

Один із найбільших легальних VOD-сервісів із професійним підходом до структурування медіа.

Переваги:

– швидка передача відеоконтенту;

– якісна та швидка структуризація каталогів за жанром.

Недоліки:

– користувачі не мають можливості вільного коментування, обговорення фільмів користувачами та створення спільноти навколо порталу, а його функції зосереджені на споживанні контенту.

Демонстрація роботи аналогу наведена на рисунку 1.3 [3].

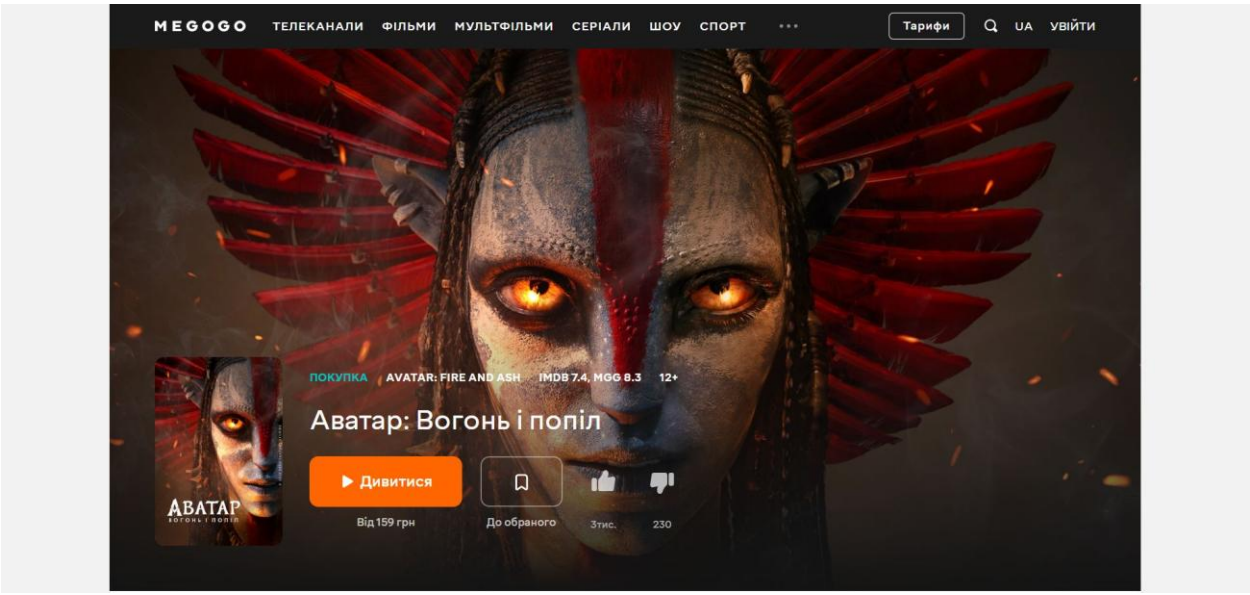

Рисунок 1.3 – Демонстрація роботи Megogo [3]

Порівняльна характеристика досліджених засобів дистрибуції кіноконтенту наведена у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика досліджених засобів дистрибуції кіноконтенту

Критерій порівняння	UASerials	Kinoukr.tv	Megogo	Розроблений вебпортал
Архітектура системи та БД	Неоптимізована, важкий DOM	Монолітна «коробкова» CMS, реляційна БД	Закрита корпоративна платформа	Клієнт-серверна, документо-орієнтована
Швидкодія клієнтської частини	Низька, затримки через сторонні скрипти	Середня	Швидка передача відео	Висока
Панель адміністрування	Стандартна	Універсальна CMS, наявність надлишкового функціоналу	Закрита корпоративна система	Кастомна та лаконічна
Соціальна взаємодія	Система коментарів	Обмежена	Відсутня	Система коментарів, рейтинги, закладки
Навігація та каталогізація	Розгалужена	Проста та інтуїтивно зрозуміла	Якісна та швидка структуризація	Динамічний пошук та фільтрація без перезавантаження сторінок

1.3 Висновки з першого розділу та постановка задачі

У першому розділі було проаналізовано предметну область розробки вебпорталів для розповсюдження кіноконтенту. У результаті дослідження встановлено, що основною проблемою існуючих рішень на базі універсальних CMS є неефективна робота з великими обсягами ієрархічних даних, а також незручні та перевантажені адмін-панелі, що ускладнює наповнення сайту.

Аналіз таких сервісів, як UASerials, Kinoukr.com та Megogo, показав, що вони не забезпечують оптимального балансу між швидкодією, зручністю інтерфейсу та функціональністю. Частина платформ має перевантажений інтерфейс і велику кількість зайвих елементів, інші повільно обробляють складні запити до бази даних. Також у деяких випадках обмежена взаємодія користувачів, зокрема відсутні коментарі чи інші соціальні функції.

З урахуванням виявлених недоліків поставлено задачу розробити власний вебпортал, який буде використовувати нереляційну базу даних для підвищення швидкодії, інтеграцію із зовнішніми API для автоматичного отримання метаданих та зручну кастомну адмін-панель. Реалізація такого підходу дозволить покращити швидкість доступу до контенту та спростити процес оновлення ресурсу.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

У другому розділі, спираючись на задачі, сформульовані раніше, обґрунтовується вибір технологічного стеку, а також виконується проєктування клієнт-серверної архітектури та структури бази даних. Основна увага приділяється розробці таких архітектурних рішень, які забезпечать високу швидкодію вебпорталу та водночас спростять його подальше адміністрування.

2.1 Вимоги до програмного забезпечення

Процес аналізу вимог до вебпорталу дистрибуції кіноконтенту базувався на дослідженні недоліків існуючих рішень та визначенні потреб цільової аудиторії. Основною метою аналізу було формування чіткого переліку функціональних та нефункціональних характеристик, які забезпечать високу швидкодію та зручність експлуатації системи.

2.1.1 Функціональні вимоги

Функціональні вимоги визначають конкретні дії, які система повинна виконувати для задоволення потреб користувачів. Відповідно до архітектури порталу, вимоги розподілено за ролями доступу.

Адміністративна частина:

– наявність спеціалізованої, лаконічної панелі адміністрування для зручного ручного додавання та редагування медіаресурсів;

– збереження введених метаданих у базу даних;

– збереження шляху до відеофайлу для інтеграції плеєра.

Управління користувачами:

– реєстрація та автентифікація користувачів із використанням захищених протоколів;

– розмежування прав доступу на рівні користувач та адміністратор;

– збереження персональних налаштувань та списків закладок.

Перегляд та пошук:

– відображення загального каталогу фільмів із можливістю динамічного завантаження даних;

– фільтрація медіаресурсів за критерієм;

– пошук фільмів за назвою.

Користувацька взаємодія:

– можливість залишати текстові коментарі до фільмів;

– виставлення числових оцінок для формування загального рейтингу фільму.

2.1.2 Нефункціональні вимоги

Нефункціональні вимоги визначають основні технічні характеристики системи та особливості її роботи. Вони впливають на продуктивність вебпорталу, стабільність роботи та безпеку даних користувачів.

Швидкодія – система повинна забезпечувати швидку обробку запитів та мінімальний час завантаження сторінок. Для цього використовується нереляційна база даних MongoDB, а клієнтська частина побудована без використання великих фреймворків, що дозволяє зменшити навантаження на браузер і покращити швидкість роботи інтерфейсу.

Надійність – забезпечення безпечного зберігання паролів у хешованому вигляді та використання JWT-токенів для верифікації сесій.

Технологічні обмеження – реалізація серверної частини має базуватися на середовищі Node.js, а клієнтської на нативних вебтехнологіях для забезпечення легкої архітектури застосунку.

2.2 Вибір мови програмування та технологічного стека

2.2.1 Backend

Вибір оптимального технологічного стеку є критично важливим для розробки вебпорталу дистрибуції кіноконтенту, оскільки від нього залежить швидкодія масштабованість та простота підтримки майбутнього вебпорталу.

Для розроблення серверної частини було проведено порівняльний аналіз трьох найпоширеніших серверних технологій: PHP, Python та Node.js (середовище виконання для мови JavaScript/TypeScript) [4]. Основними критеріями вибору були: підтримка асинхронності, швидкість роботи та можливість використання строгої типізації для запобігання помилкам.

Результати порівняння наведено у таблиці 2.1 [5]-[6].

Таблиця 2.1 – Результати порівняння мов програмування [5]-[6]

Критерії порівняння	Node.js (TypeScript)	Python	PHP
1	2	3	4
Архітектура обробки запитів	Асинхронна, код виконується паралельно, без очікування завершення функцій, що забезпечує швидкість	Синхронна, використовує синхронні обчислення, що уповільнює його роботу	Синхронна, може уповільнювати роботу через тривалі операції, що блокують виконання наступного коду

Кінець таблиці 2.1

1	2	3	4
Єдина мова для клієнта та сервера	Написаний на JavaScript, тому дозволяє використовувати одну мову для фронтенду та бекенду, спрощуючи розробку	Можна ефективно використовувати як для фронтенд, так і для бекенд розробки завдяки широкому набору інструментів та бібліотек	Доводиться перемикатися між різними мовами та синтаксисом
Швидкодія та продуктивність	Висока	Середня	Середня
Сумісність із форматом даних NoSQL	Нативна	Потребує конвертації	Потребує конвертації
Оптимальна сфера застосування	Вебпортали, стрімінг, REST API з високим навантаженням	Аналіз даних, машинне навчання, скрипти	Традиційні сайти, класичні CMS

Згідно з результатами порівняння беззаперечним лідером для реалізації поставлених завдань є середовище Node.js. Воно дозволяє використовувати одну мову програмування як на сервері, так і на клієнті, а його асинхронна природа ідеальна для створення швидкого REST API, який постійно звертається до зовнішніх кінобаз [7].

2.2.2 Frontend

Для розробки клієнтської частини було вирішено використовувати HTML5 та CSS3, а також препроцесор SCSS для зручнішої організації стилів. Інтерактивність реалізована за допомогою JavaScript і TypeScript [8]-[9].

Було прийнято рішення відмовитися від використання важких фронтенд-фреймворків. Це дозволяє зменшити розмір завантажуваних файлів і забезпечити швидкий доступ до контенту навіть на малопотужних пристроях, що є важливим для користувача.

2.2.3 База даних

У якості системи керування базами даних обрано MongoDB – нереляційну документо-орієнтовану СУБД. Оскільки Node.js працює з даними у форматі JSON, використання MongoDB, яка використовує BSON, забезпечує зручну та швидку взаємодію між сервером і базою даних без необхідності застосування складних ORM [10].

Такий підхід добре підходить для зберігання гнучких структур даних, наприклад списків фільмів, обраного контенту чи закладок користувачів. Це позитивно впливає на швидкість виконання запитів. Додатковою перевагою є наявність зручних інструментів для роботи з базою даних, зокрема MongoDB Compass, що спрощує адміністрування та налагодження.

2.3 Обґрунтування вибору середовища розроблення

Для ефективного написання програмного коду необхідно обрати середовище розроблення або розширений текстовий редактор, який найкраще відповідає обраному технологічному стеку. На основі аналізу сучасних рішень для веброзробки було проведено порівняння трьох найпопулярніших інструментів: Visual Studio Code, WebStorm та Sublime Text.

Результати порівняльного аналізу наведено у таблиці 2.2 [11]-[15].

Таблиця 2.2 – Порівняльний аналіз середовищ розроблення коду [11]-[15]

Критерії порівняння	Visual Studio Code	Sublime Text	WebStorm
Швидкодія та споживання пам'яті	Висока продуктивність, помірне споживання	Екстремальна швидкодія, мінімальне споживання ресурсів	Високе навантаження на систему, потребує до 3 ГБ RAM, тривале індексування
Підтримка TypeScript та Node.js	Нативна	Базова, потребує встановлення сторонніх плагінів	Глибокий семантичний аналіз, інтелектуальне автодоповнення
Вбудовані інструменти	Повний набір	Мінімальний, інструменти потребують ручного налаштування	Максимально розширений інтегрований набір
Розширюваність	Величезна екосистема плагінів	Широка підтримка пакетів	Більшість функцій вбудовано від початку
Ліцензія та вартість	Безкоштовне	Комерційна	Комерційна (платна підписка)

Результати порівняльного аналізу показали, що редактор Sublime Text має високу швидкість роботи, проте не забезпечує повноцінного функціоналу IDE, зокрема відсутні вбудований дебагер і нативна підтримка TypeScript, що ускладнює процес налаштування середовища.

WebStorm, у свою чергу, є потужною IDE з розширеними можливостями роботи з кодом, однак споживає значну кількість оперативної пам'яті, що може негативно впливати на продуктивність на звичайних ПК.

З урахуванням цих факторів було обрано Visual Studio Code. Даний редактор поєднує в собі достатню функціональність і високу швидкість роботи, є безкоштовним та має нативну підтримку TypeScript. Крім того, наявність вбудованого термінала для роботи з Node.js і зручних інструментів налагодження робить його оптимальним вибором для розробки даного проєкту.

2.4 Проєктування архітектури програмного забезпечення

Для реалізації вебпорталу було обрано трирівневу клієнт-серверну архітектуру. Такий підхід дозволяє розділити систему на окремі рівні, що підвищує її швидкодію, стабільність та спрощує подальшу підтримку. Кожен рівень виконує свої функції, що дає змогу оптимізувати їх незалежно один від одного.

2.4.1 Представлення

Даний рівень відповідає за взаємодію з користувачем. Фронтенд реалізований з використанням HTML5, CSS3 та TypeScript. Основні його задачі – відображення контенту, обробка дій користувача та формування асинхронних запитів до сервера. Використання нативного підходу до роботи з DOM дозволяє забезпечити швидке завантаження та відображення сторінок.

2.4.2 Бізнес-логіка

Цей рівень є проміжною ланкою між інтерфейсом і базою даних та реалізований на Node.js з використанням Express. Він відповідає за обробку запитів та виконання основних функцій системи, зокрема:

– маршрутизацію HTTP-запитів;

– обробку даних – коментарів, рейтингу, фільтрації контенту;

– реалізацію механізмів авторизації та безпеки;

– взаємодію із зовнішніми API для отримання кіноконтенту.

2.4.3 Доступ до даних

Рівень даних відповідає за зберігання та отримання інформації. Використання MongoDB дозволяє ефективно працювати з даними завдяки їх збереженню у вигляді документів без жорстких зв’язків. Взаємодія між рівнями відбувається наступним чином: клієнт надсилає запит до сервера, сервер обробляє його та звертається до бази даних, після чого отримані дані у форматі JSON повертаються клієнту і відображаються на сторінці.

Діаграма трирівневої архітектури вебпорталу наведена на рисунку 2.1.

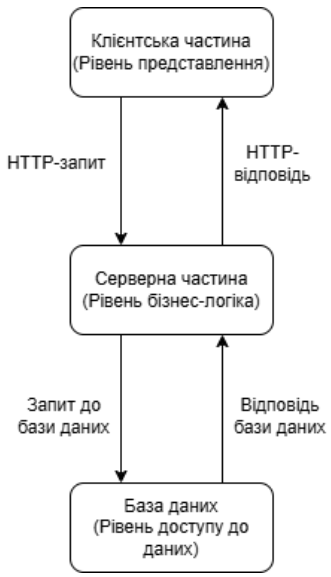

Рисунок 2.1 – Діаграма трирівневої архітектури вебпорталу

2.5 Проєктування структури бази даних

Для збереження даних вебпорталу використовується MongoDB. На відміну від реляційних баз даних, які зберігають інформацію у жорстко зв'язаних таблицях, MongoDB використовує колекції гнучких JSON-подібних документів. За допомогою цього методу можна зберігати ієрархічні дані, такі як фільм і список жанрів, в одному об’єкті. Це гарантує швидке виконання запитів на вибірку контенту.

У процесі проєктування бази даних вебпорталу було розроблено чотири основні колекції: Users, Movies, Ratings та Comments.

Колекція «Users»:

– _id: унікальний ідентифікатор користувача в системі;

– username: нікнейм користувача;

– email: електронна пошта, яка використовується як логін;

– password: криптографічний хеш пароля;

– role: рівень прав доступу;

– savedMovies: масив ідентифікаторів фільмів, доданих до улюбленого.

Колекція «Movies»:

– _id: унікальний внутрішній ідентифікатор фільму в базі порталу;

– tmdbId: ідентифікатор фільму в зовнішній базі фільмів;

– title: назва кінофільму;

– year: рік випуску фільму;

– age: вікове обмеження для перегляду контенту;

– time: загальна тривалість кіноконтенту;

– genres: назва жанру;

– studio: студія-виробник контенту;

– description: короткий опис сюжету;

– posterImg: посилання на файл постеру;

– bgImg: посилання на широкоформатне фонове зображення;

– titleImg: посилання на зображення з стилізованою назвою;

– createdAt: автоматично згенерована мітка дати та часу додавання запису до бази.

Колекція «Ratings»:

– _id: унікальний ідентифікатор запису оцінки;

– movieId: посилання на ідентифікатор оціненого фільму;

– userId: посилання на ідентифікатор користувача, який поставив оцінку;

– rating: числове значення поставленої оцінки від 1 до 5.

Колекція «Comments»:

– _id: унікальний ідентифікатор відгуку;

– movieId:ідентифікатор фільму;

– userId: посилання на ідентифікатор користувача;

– nickname: нікнейм користувача;

– text: текстовий зміст коментаря;

– date: мітка дати та часу публікації;

– isApproved: підтвердження валідності коментаря адміністратором.

Нереляційна структура дозволяє серверу отримувати достатню кількість інформації про медіаресурси за допомогою мінімальної кількості звернень до бази.

Демонстрація діаграми зв’язків між колекціями в MongoDB наведена на рисунку 2.2.

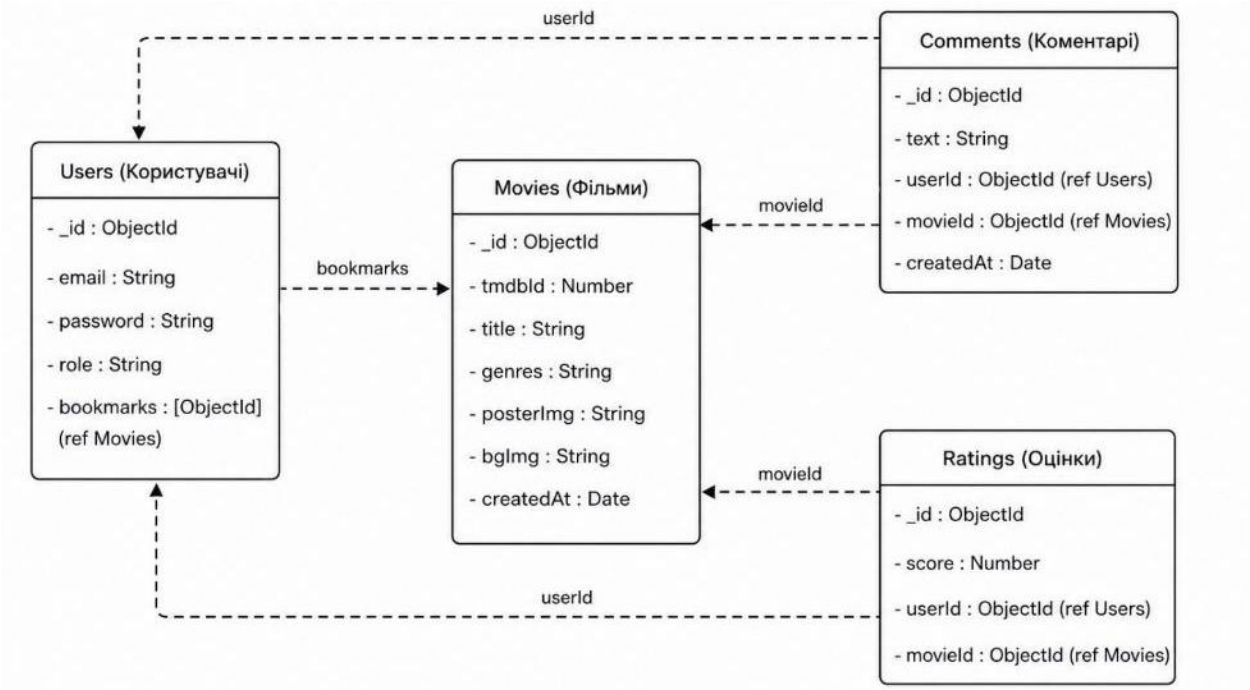

Рисунок 2.2 – Діаграма зв’язків між колекціями у БД

2.6 Висновки за розділом

У другому розділі дипломної роботи було обґрунтовано вибір технологічного стеку та виконано проєктування архітектури вебпорталу для дистрибуції кіноконтенту. Для серверної частини обрано Node.js завдяки підтримці асинхронності та можливості створення швидкого REST API, а для зберігання даних – MongoDB, яка забезпечує гнучкість і швидку обробку запитів.

У результаті було спроєктовано трирівневу клієнт-серверну архітектуру з поділом на рівень представлення, бізнес-логіки та доступу до даних. Також визначено основні інструменти розробки та адміністрування, зокрема Visual Studio Code і MongoDB Compass.

Обрані технології та архітектурні рішення відповідають вимогам до швидкодії та оптимізації системи і створюють основу для подальшої реалізації програмної частини проєкту.

3 РОЗРОБКА ПРОГРАМИ

Третій розділ дипломної роботи охоплює етап практичної реалізації спроєктованої у 2 розділі системи. У ньому описано основні алгоритми взаємодії клієнт-сервер, структуру компонентів backend і frontend програмного модуля, а також процес автоматизованого отримання метаданих про фільми за допомогою зовнішнього API. Крім того, наведено опис розробленого інтерфейсу користувача та панелі адміністрування.

3.1 Алгоритм функціонування та взаємодії компонентів системи

Розроблений вебпортал працює на основі асинхронної обробки подій. Клієнтський і адміністративний є двома основними потоками алгоритму функціонування системи.

Алгоритм клієнтського доступу до медіаресурсу:

– користувач ініціює запит;

– клієнтська частина формує HTTP-запит до відповідної кінцевої точки REST API сервера;

– серверна частина отримує запит, валідує його параметри та звертається до СУБД MongoDB за допомогою вбудованих методів;

– база даних виконує пошук відповідного документа у колекції та повертає його на сервер;

– сервер формує відповідь у форматі JSON і надсилає її клієнту;

– frontend-скрипти на базі TypeScript розбирають отриманий JSON та динамічно оновлюють DOM-дерево сторінки без повного перезавантаження вікна браузера.

Блок-схема базового алгоритму обробки клієнтського запиту наведена на рисунку 3.1.

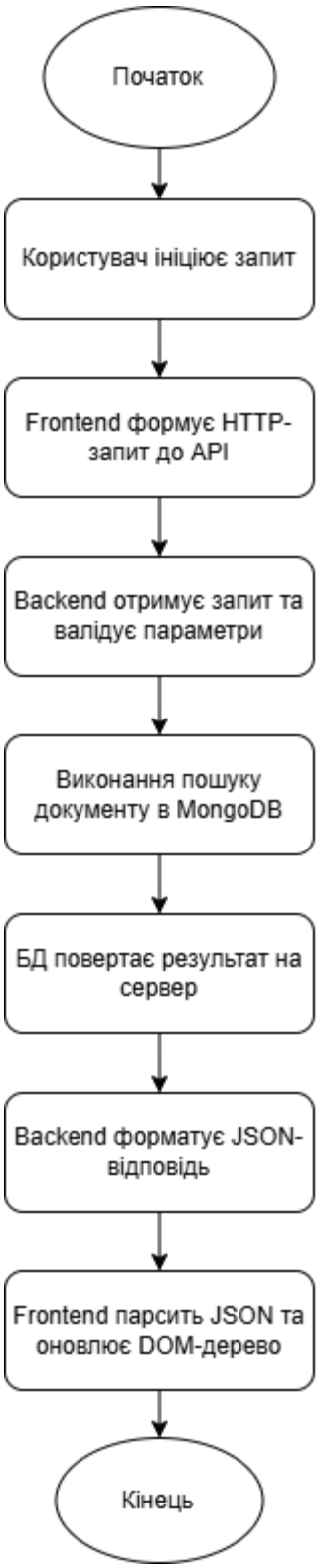

Рисунок 3.1 – Блок-схема базового алгоритму обробки клієнтського запиту

3.2 Опис модулів реалізованої програми

3.2.1 Модулі серверної частини

Програмне забезпечення побудоване за модульним принципом, що гарантує підтримку коду та легке масштабування. Файл server.js є точкою входу серверної частини і відповідає за запуску Express-застосунку, підключення до бази даних MongoDB, налаштування глобальних параметрів і роздачу статичних файлів.

Основні програмні модулі сервера:

– routes: директорія містить визначення REST API маршрутів та функції-контролери для обробки HTTP-запитів;

– models: директорія містить схеми даних, розроблені для взаємодії з MongoDB;

– middleware: директорія проміжного програмного забезпечення, складається з функцій-перехоплювачів, які виконуються в доповненні до основної логіки маршруту.

3.2.2 Модулі клієнтської частини

Клієнтська логіка була створена в директорії ts/ мовою TypeScript. Після завершення компіляції скрипти, а також HTML-розмітка та CSS-стилі (SASS), розміщуються в директорії public/.

Основними модулями інтерфейсу є:

– ts: у директорії ведеться розробка клієнтської логіки мовою TypeScript;

– public: директорія містяща усю frontend частину;

– модуль головної сторінки: відповідає за відображення каталогу фільмів, фільтрацію карток контенту та динамічне завантаження;

– модуль сторінки фільмів: забезпечує адаптивне відображення контенту про фільм за допомогою посилань на фонові зображення bgImg, інтеграцію відеоплеєра та форми для виставлення оцінок і написання коментарів;

– модуль панелі адміністрування: окремий інтерфейс, доступний лише адміністраторам та порівняно з універсальними CMS, має оптимізовані форми для швидкого ручного введення інформації про нові кінострічки, що значно пришвидшує роботу контент-менеджера.

3.3 Опис реалізації клієнтської частини та інтерфейсу користувача

Клієнтська частина вебпорталу реалізована як динамічний застосунок на базі нативних технологій HTML5, CSS3, із застосуванням препроцесора SASS, та TypeScript. Основною метою при розробці інтерфейсу було досягнення максимальної швидкості рендерингу та забезпечення інтуїтивно зрозумілої навігації. Інтерфейс розділено на частини для клієнтів і адміністратора для логічного структурування та демонстрації роботи застосунку.

3.3.1 Інтерфейс гостя та авторизованого користувача

Взаємодія з порталом починається з головної сторінки. Її інтерфейс логічно поділено на чотири ключові секції: велика вітрина останніх доданих медіаресурсів для акцентування уваги на новинках, секція загального каталогу всіх доступних фільмів, блок категорій для навігації за жанрами, блок категорій для пошуку контенту за студіями.

Демонстрація інтерфейсу головної сторінки наведена на рисунках 3.2-3.4.

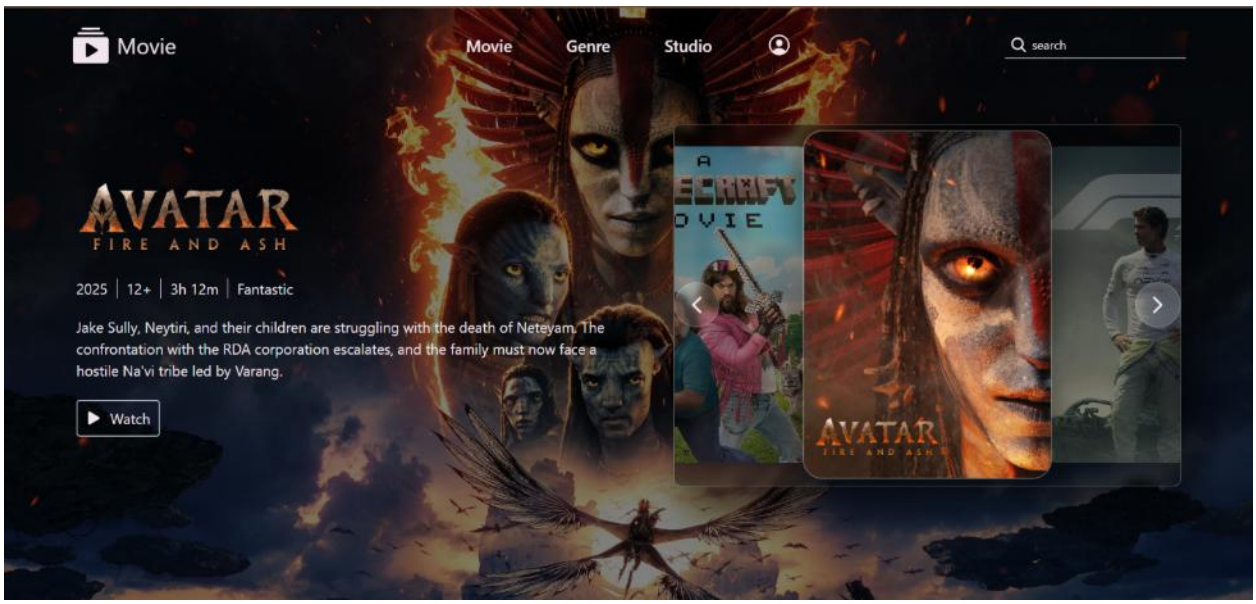

Рисунок 3.2 – Інтерфейс головної сторінки

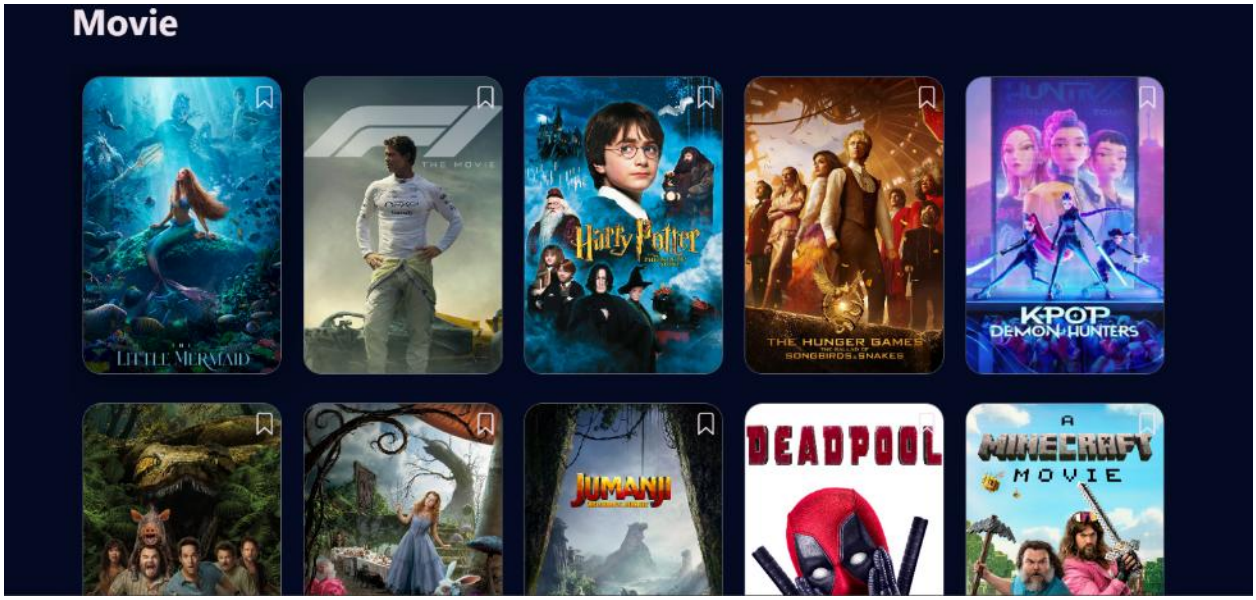

Рисунок 3.3 – Інтерфейс головної сторінки

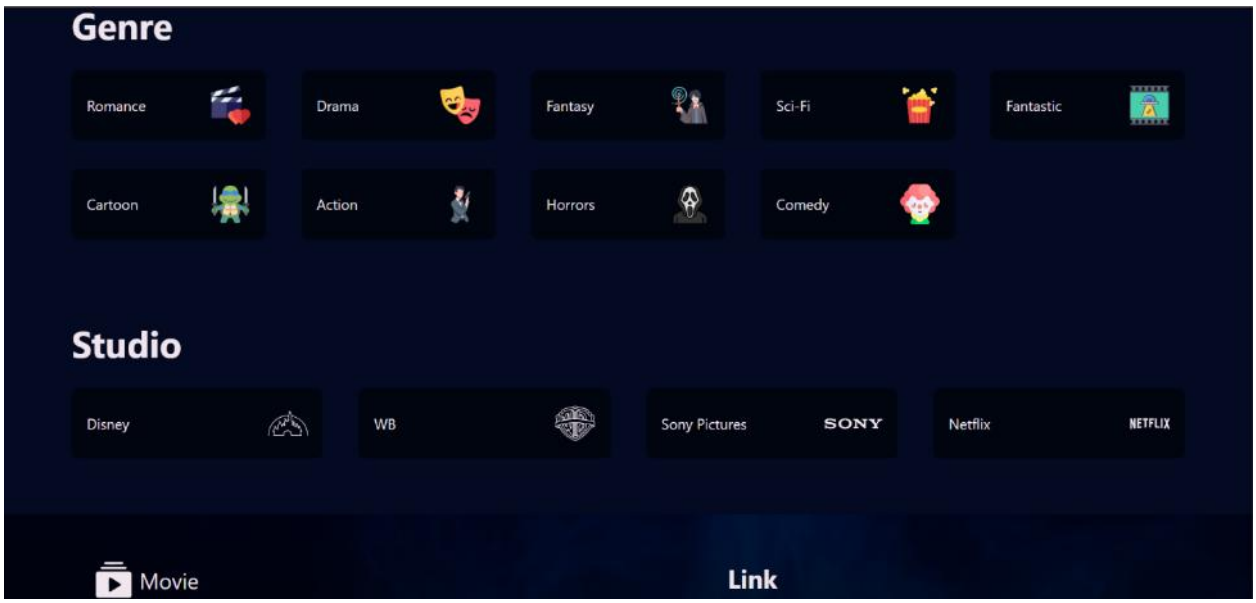

Рисунок 3.4 – Інтерфейс головної сторінки

3.3.2 Інтерфейс форми автентифікації користувача

Для використання додаткових функцій вебпорталу користувачу необхідно пройти процедуру авторизації. Після входу до системи відкривається доступ до можливості залишати коментарі під фільмами, додавати контент до списку закладок та виставляти оцінки переглянутим фільмам. Це дозволяє зробити взаємодію з вебпорталом більш персоналізованою та зручною для користувача.

Форми входу та реєстрації реалізовані у вигляді модальних вікон, які відкриваються поверх основного інтерфейсу без переходу на окремі сторінки. Такий підхід спрощує процес авторизації та робить роботу з сайтом більш комфортною.

Перед надсиланням інформації на сервер система виконує перевірку правильності введених даних на стороні клієнта. Перевіряється коректність електронної пошти, довжина пароля та заповнення обов’язкових полів. У випадку помилки користувач отримує відповідне повідомлення, що дозволяє виправити дані до відправлення форми. Це допомагає зменшити кількість некоректних запитів та підвищує стабільність роботи системи.

Демонстрація інтерфейсу форми автентифікації наведена на рисунках 3.5-3.6.

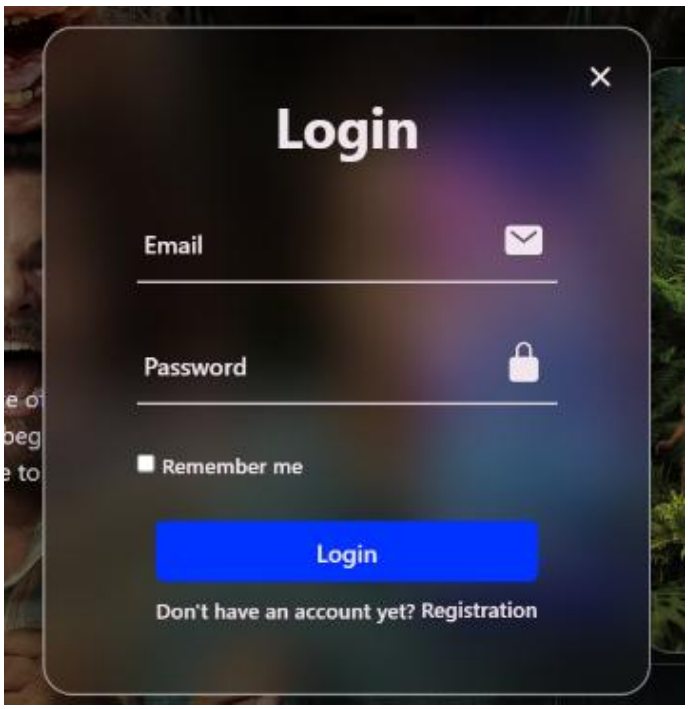

Рисунок 3.5 – Інтерфейс форми автентифікації користувача

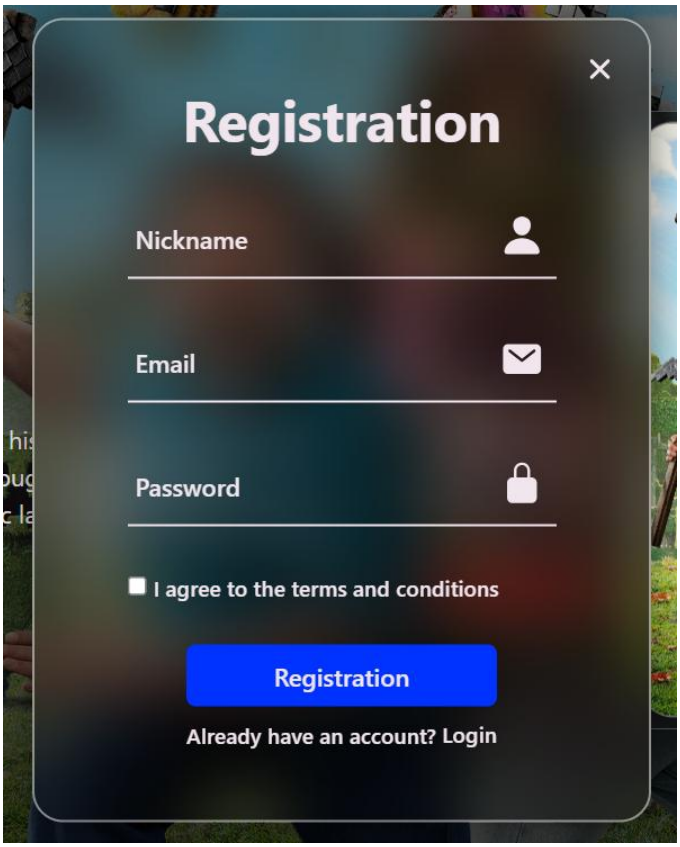

Рисунок 3.6 – Інтерфейс форми автентифікації користувача

Демонстрація інтерфейсу акаунту наведена на рисунку 3.7.

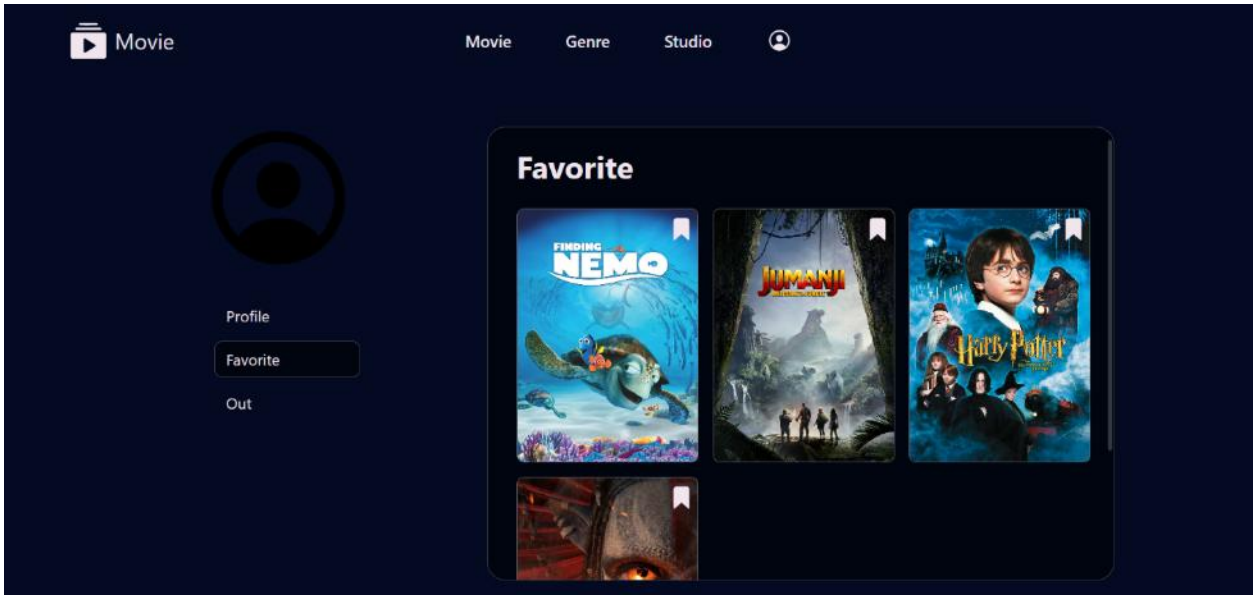

Рисунок 3.7 – Інтерфейс акаунту

3.3.3 Сторінка перегляду фільму та взаємодії з контентом

Однією з основних частин клієнтського інтерфейсу є сторінка перегляду фільму. Саме на ній користувач отримує доступ до перегляду відео, інформації про фільм та додаткових елементів взаємодії із системою.

Для покращення візуального оформлення використовується поле «bgImg», яке дозволяє автоматично змінювати фонове зображення сторінки відповідно до вибраного фільму. Завдяки цьому сторінка виглядає більш стилізовано та краще передає атмосферу кіноконтенту.

У центральній частині сторінки розташований відеоплеєр для перегляду фільму, а також текстовий опис із коротким синопсисом та додатковою інформацією про контент. Нижче знаходиться блок взаємодії користувачів, де авторизовані користувачі можуть залишати коментарі та оцінювати фільм. Це дозволяє зробити платформу більш інтерактивною та створює можливість для обговорення контенту між користувачами.

Демонстрація інтерфейсу сторінки перегляду фільму наведена на рисунках 3.8-3.9.

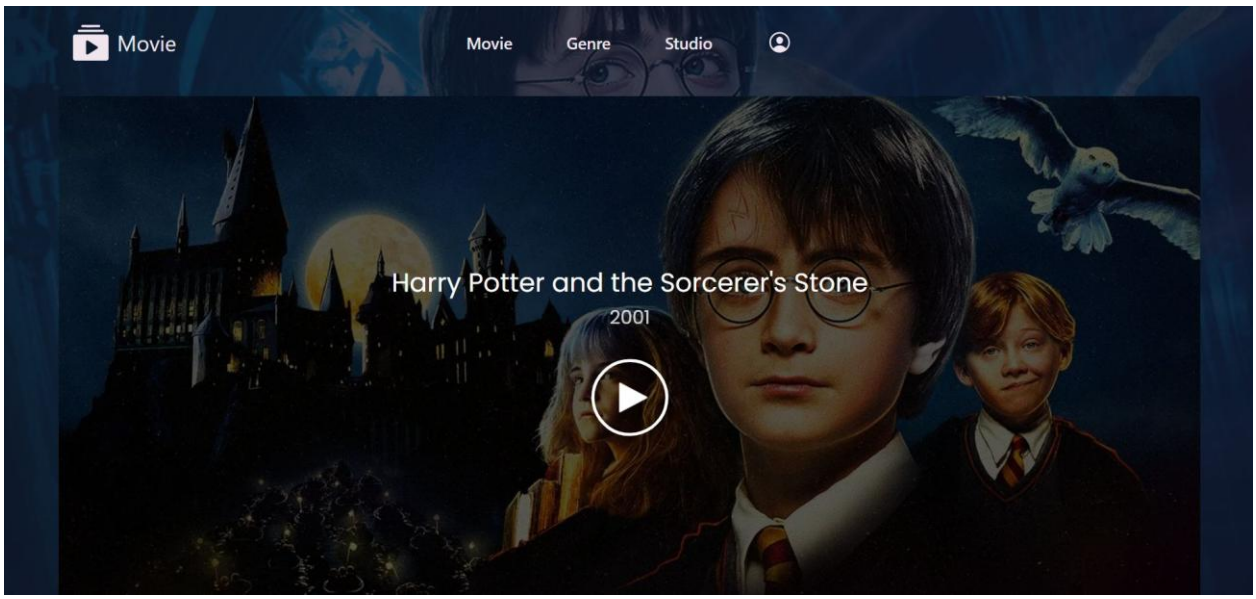

Рисунок 3.8 – Інтерфейс сторінки перегляду фільму та взаємодії з контентом

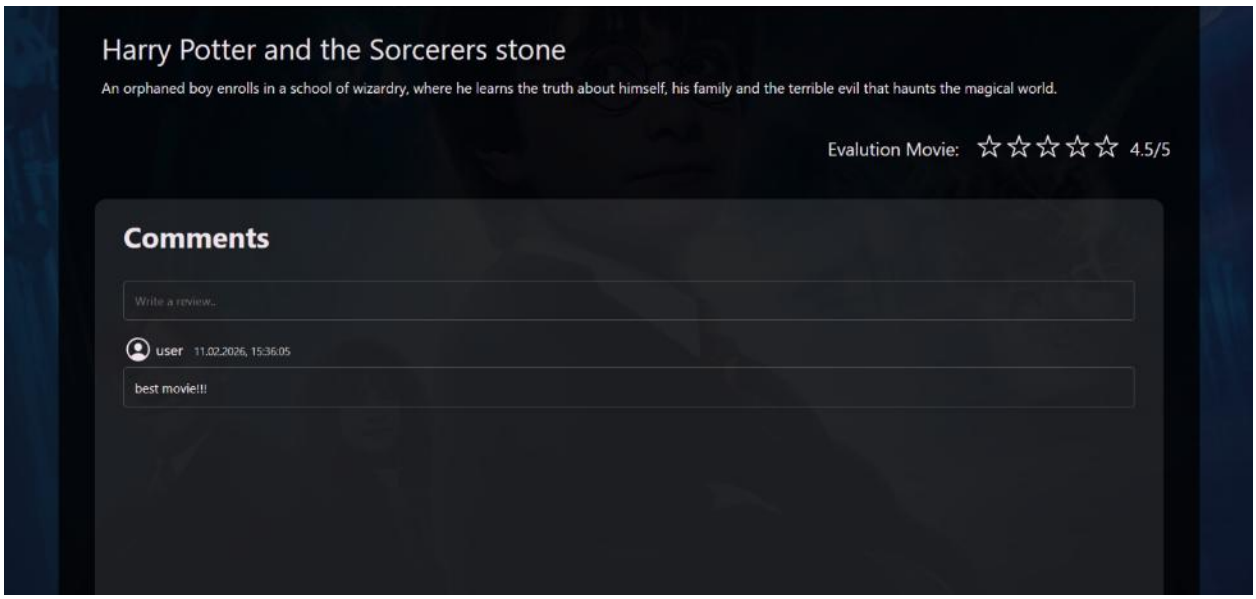

Рисунок 3.9 – Інтерфейс сторінки перегляду фільму та взаємодії з контентом

3.3.4 Навігація та інтерактивний пошук

Для забезпечення швидкого пошуку необхідного контенту на порталі реалізовано систему фільтрації та інтерактивного пошуку за назвою.

Розроблений інтерфейс відрізняється від статичних каталогів тим, що дозволяє користувачу миттєво знайти фільм, не переглядаючи всю базу даних. Алгоритм пошуку працює таким чином: Frontend-частина генерує асинхронний запит до сервера з параметром рядка після введення назви в рядок пошуку. Навіть за частиною слова сервер може знайти збіги в колекції Movies за допомогою регулярних виразів. На сторінці відображаються динамічні результати пошуку у вигляді сітки карток.

Демонстрація інтерфейсу роботи модуля пошуку наведена на рисунку 3.10.

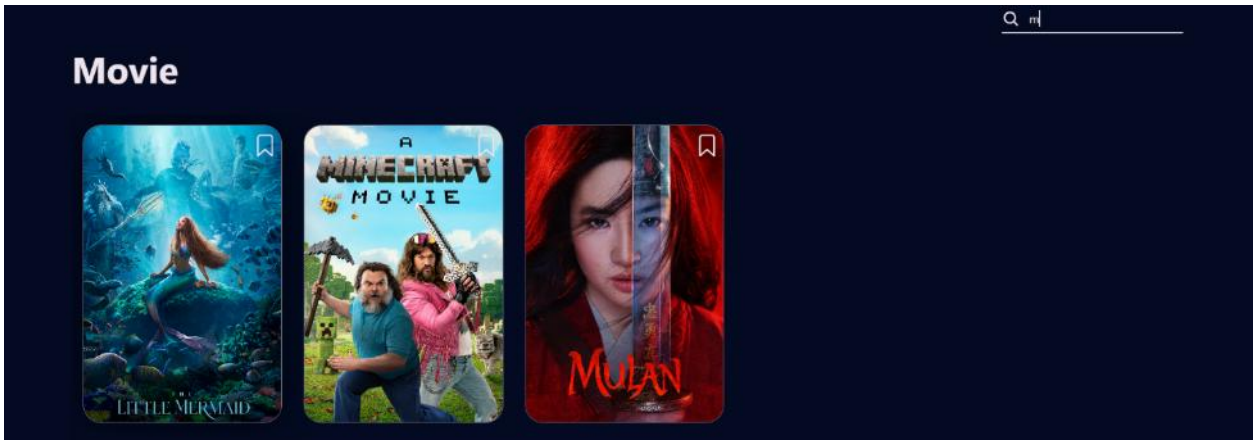

Рисунок 3.10 – Результат роботи модуля пошуку фільмів за назвою

Крім текстового пошуку, портал надає інструменти категоріальної навігації. Користувач може фільтрувати контент відповідно до жанру або студії. Скрипт додає параметри, пов’язані з категорією, до API-запиту після вибору категорії, а база даних повертає лише відповідний набір документів. Це значно спрощує користувальницький досвід для користувачів, які шукають контент у певному напрямку.

Демонстрація інтерфейсу фільтрації медіаконтенту наведена на рисунках 3.11-3.12.

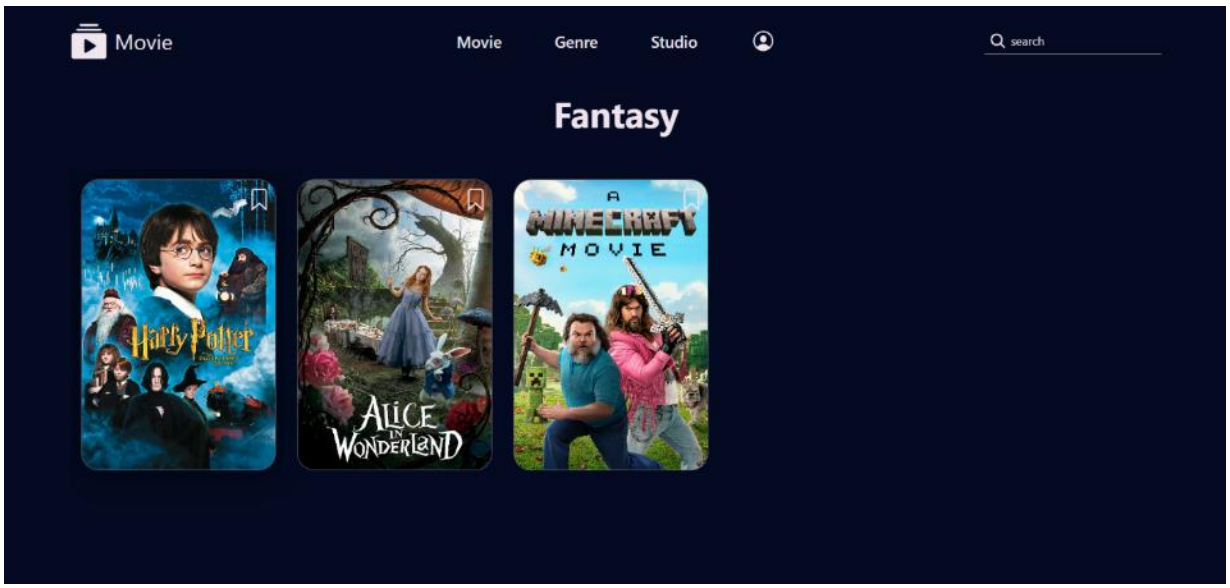

Рисунок 3.11 – Реалізація фільтрації медіаконтенту за жанрами

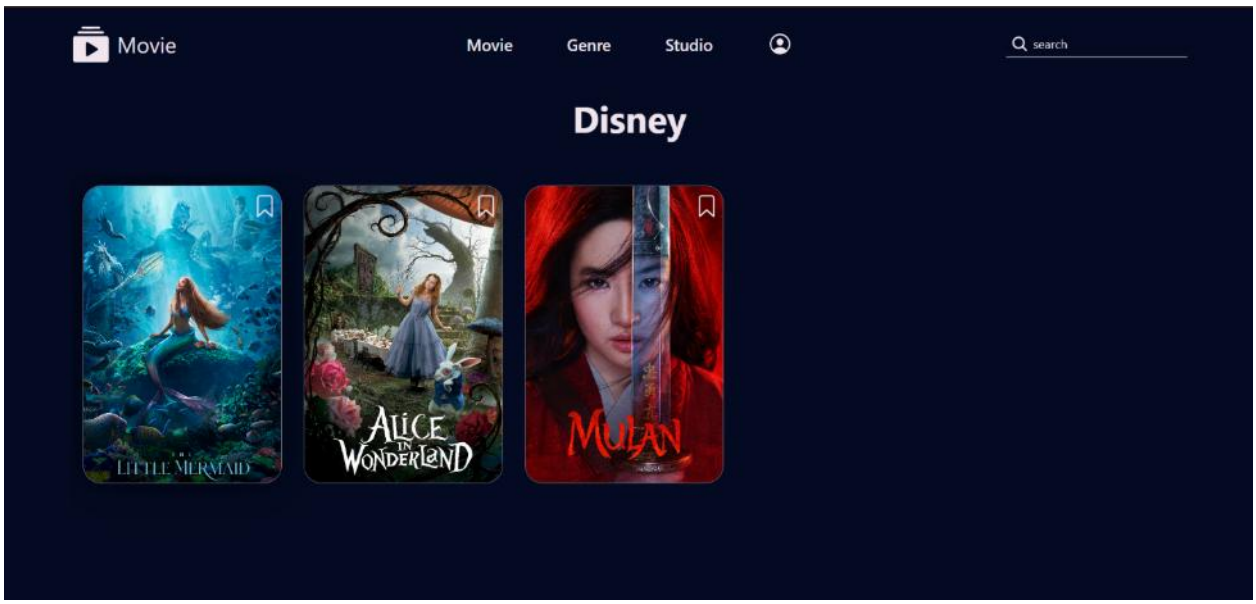

Рисунок 3.12 – Реалізація фільтрації медіаконтенту за студіями

3.3.5 Робоче середовище адміністратора для управління базою даних

Як зазначалося на етапі аналізу аналогів, однією з цілей проєкту було створення оптимізованого робочого середовища для адміністратора. Панель

адміністрування є відокремленою візуальною частиною порталу, доступною лише для юзерів, які мають роль адміністратора. Інтерфейс панелі орієнтований на швидкість роботи та не містить багато графічних елементів. Форма додавання нового фільму містить лише поля для ручного введення текстової інформації. Асинхронні запити дозволяють адміністратора миттєво зберігати нові записи в базі даних MongoDB і отримувати сповіщення про успішне виконання операції без перезавантаження сторінки.

Демонстрація інтерфейсу робочого середовища адміністратора наведена на рисунку 3.13.

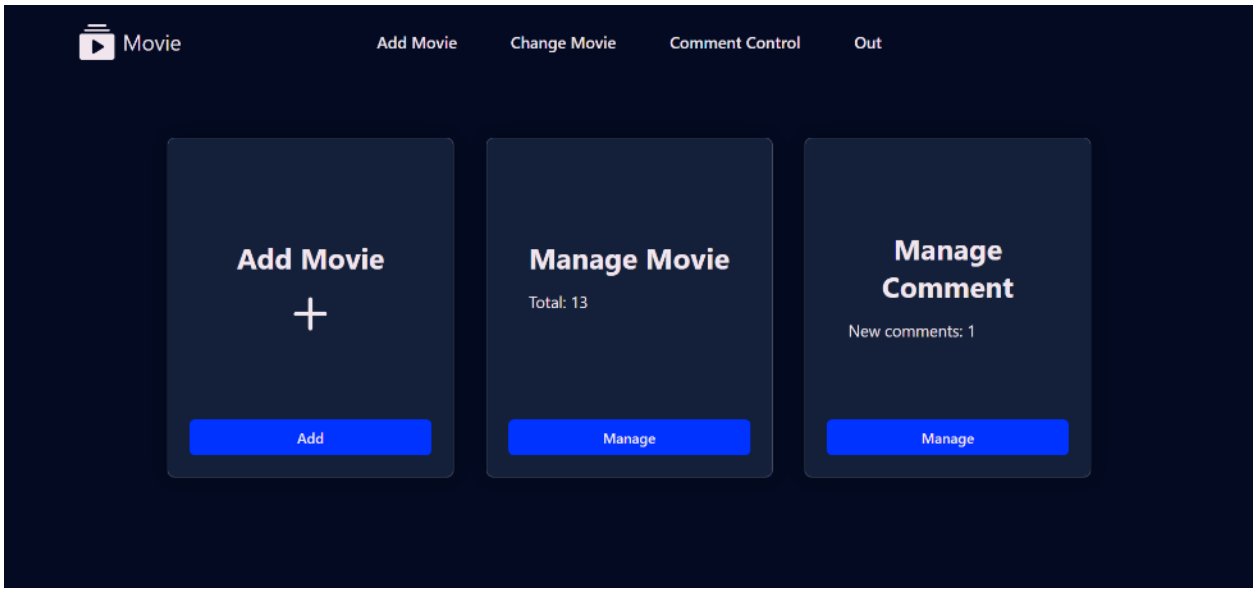

Рисунок 3.13 – Інтерфейс робочого середовища адміністратора для управління контентом

3.3.6 Інтерфейс додавання нового фільму до бази даних

Ця форма, оптимізована для ручного введення всіх полів моделі Movie. Адміністратор заповнює текстові метадані, додає посилання на медіафайли, такі як постери та фонові зображення, а також додає ідентифікатор зовнішньої бази TMDB. Після відправки форми клієнтська частина створює POST-запит, що призводить до миттєвого зберігання даних у колекції MongoDB без перезавантаження сторінки.

Демонстрація інтерфейсу додавання нового фільму наведена на рисунку 3.14.

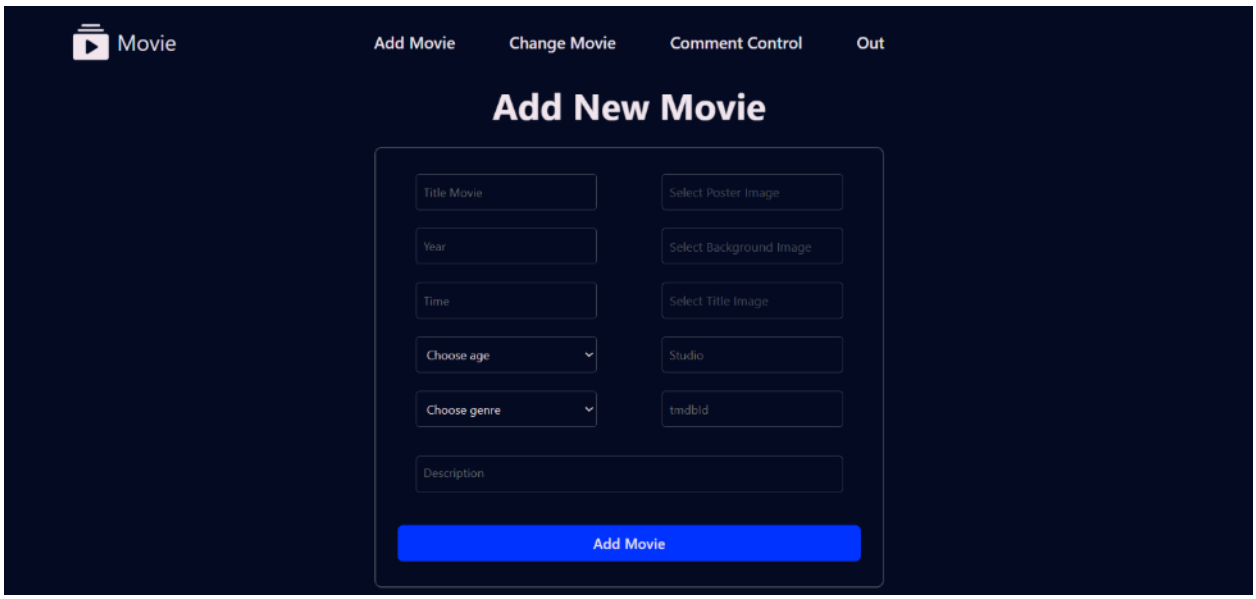

Рисунок 3.14 – Інтерфейс додавання нового фільму до бази даних

3.3.7 Сторінка управління та редагування існуючого медіаконтенту

Даний модуль використовується для керування контентом, який уже був доданий до системи. Через інтерфейс адміністратора відображається список усіх завантажених фільмів, а також автоматично показується загальна кількість контенту, що знаходиться в базі даних. Ця інформація виводиться на адміністративній панелі у вигляді окремого статистичного блоку.

Функціонал модуля дозволяє адміністратору редагувати інформацію про фільми, оновлювати посилання на відео, змінювати описи або видаляти застарілий контент. Для цього реалізовано CRUD-операції, які забезпечують створення, редагування та видалення записів без необхідності прямого втручання у базу даних.

Демонстрація інтерфейсу управління та редагування медіаконтну наведена на рисунку 3.15.

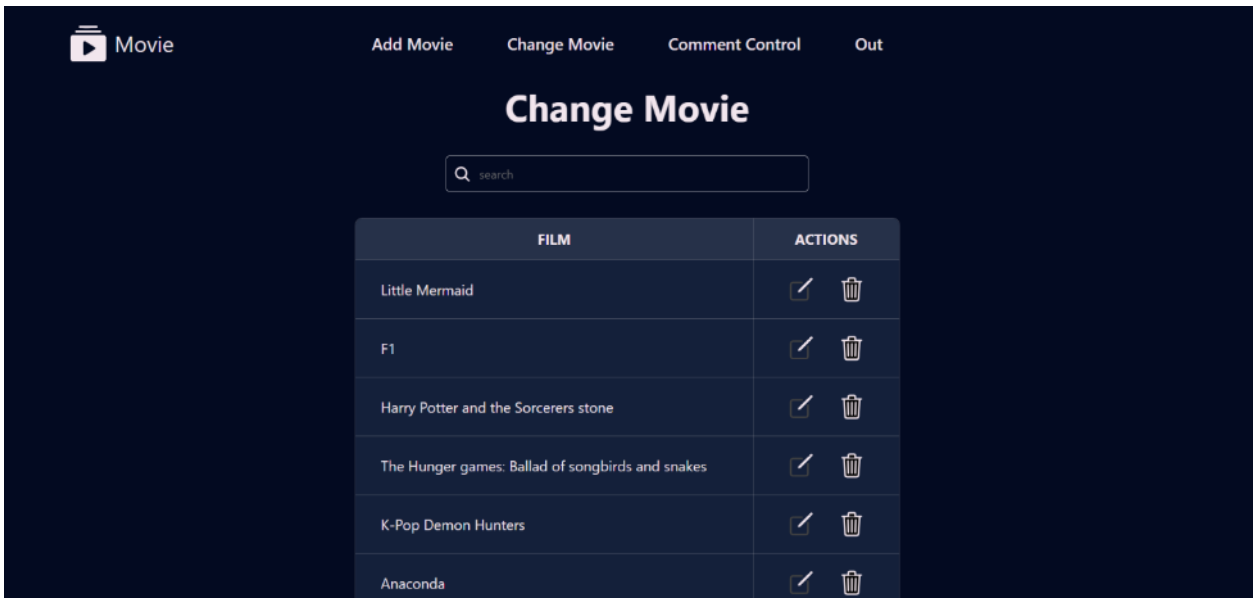

Рисунок 3.15 – Сторінка управління та редагування існуючого медіаконтенту

3.3.8 Інтерфейс модерації користувацьких коментарів

Для контролю активності користувачів у системі реалізовано окремий розділ адміністративної панелі, призначений для перевірки коментарів та модерації контенту. На головній сторінці адміністратора відображається динамічний лічильник нових коментарів, які ще не були переглянуті або оброблені. Це дозволяє швидко отримувати інформацію про нову активність на порталі.

У даному розділі адміністратор має можливість переглядати коментарі користувачів та видаляти повідомлення, що містять спам, образливий вміст або порушують правила користування вебпорталом. Такий підхід допомагає підтримувати порядок у системі та покращує якість взаємодії між користувачами платформи.

Демонстрація інтерфейсу модерації коментарів наведена на рисунку 3.16.

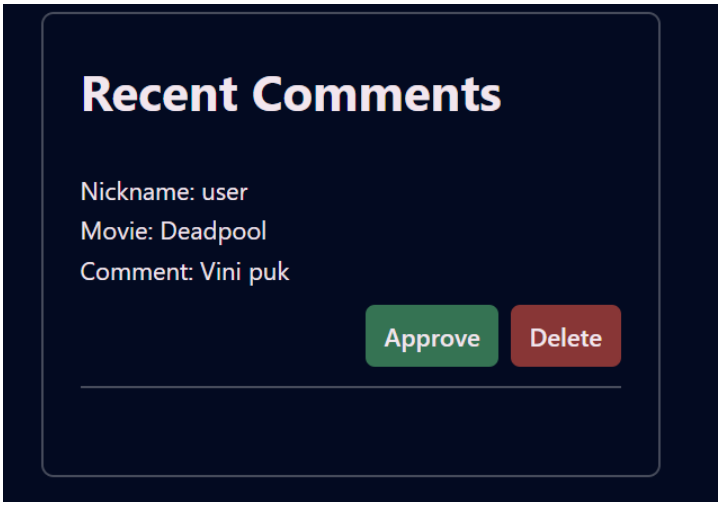

Рисунок 3.16 – Інтерфейс модерації коментарів

3.4 Висновки за розділом 3

У третьому розділі було описано процес практичної реалізації розробленого програмного забезпечення вебпорталу дистрибуції кіноконтенту. Описуються основні алгоритми функціонування системи, які використовують асинхронну взаємодію клієнт-сервер і обмін даними JSON. Детально розглядалась модульна структура серверної частини, яка базується на фреймворку Express. Бізнес-логіка, схеми баз даних і системи захисту доступу були розмежовані за допомогою використання маршрутів, моделей і середовищ. Реалізовано безпечну систему авторизації на основі хешування паролів і JWT-токенів. Описується реалізація клієнтської частини за допомогою нативних вебтехнологій без використання масивних фреймворків Frontend. Скріншоти демонструють розроблений графічний інтерфейс, який полегшує навігацію каталогом, перегляд відео та взаємодію з аудиторією. Окрему увагу приділено розробці спеціалізованої панелі адміністрування, яка допоможе контент-менеджерам швидко та лаконічно наповнити базу даних. Створене програмне забезпечення повністю задовольняє вимоги та готове до використання.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Програма є повнофункціональним вебпорталом для розповсюдження кіноконтенту. Основним завданням системи є надання користувачам швидкого та простого доступу до структурованої медіабібліотеки, яка включає пошук, фільтрацію за жанрами та студіями, перегляд фільмів тощо. Ще одним призначенням системи є надання засобів соціальної взаємодії. Портал дозволяє авторизованим користувачам створювати списки збережених фільмів, виставляти рейтингові оцінки та коментувати кіноконтент.

Адміністративному персоналу програма надає зручне робоче середовище для ручного управління контентом та модерації коментарів без використання додаткових CMS.

4.2 Умови виконання програми

Для забезпечення стабільної роботи вебпорталу та швидкого відгуку інтерфейсу необхідно дотримуватися встановлених технічних параметрів.

4.2.1 Апаратні умови

Щоб додаток працював стабільно, його апаратна частина повинна відповідати наступним параметрам:

– серверна частина: для ефективної роботи Node.js та СУБД MongoDB потрібні пристрої з багатоядерним процесором від 2.0 ГГц і обсягом оперативної пам’яті не менше 4 ГБ. Для зберігання медіафайлів та індексів бази даних потрібні SSD-накопичувачі з 10 ГБ вільного місця;

– клієнтські пристрої: для комфортної взаємодії з інтерфейсом, що містить велику кількість графічних елементів, рекомендується використовувати пристрої з оперативною пам’яттю від 2 ГБ;

– інтернет-з’єднання: швидкість інтернету є життєво важливою, оскільки портал передбачає потокове відтворення відео та завантаження високоякісних зображень, затримки в мережі можуть вплинути на швидкість завантаження сторінок і якість відтворення контенту.

4.2.2 Програмні умови та експлуатаційні вимоги

Щоб забезпечити продуктивність системи, необхідно дотримуватися певних умов програмного середовища:

– програмне забезпечення: середовище Node.js має бути встановлене на сервері. Наявність сучасного браузера з підтримкою стандартів HTML5 та CSS Grid є обов’язковою для клієнті;

– оновлення середовища: рекомендується регулярно оновлювати браузери та серверні залежності, щоб забезпечити безпеку даних і сумісність із сучасними протоколами передачі даних;

– мережеві умови: оптимальна робота передбачає наявність стабільного Wi-Fi, що дозволить серверній частині безпечно взаємодіяти з зовнішнім API TMDB під час адміністрування, а користувачам уникнути помилок під час завантаження медіаконтенту.

Таким чином, дотримання наведених програмних і апаратних вимог є умовою стабільної роботи вебпорталу, швидкого відгуку інтерфейсу та надійності збереження даних. Незалежно від типу використовуваного пристрою, комфортна взаємодія як для користувачів, так і для адміністраторів системи гарантується завдяки ефективній комбінації відповідного технологічного стека та обчислювальної потужності.

4.3 Виконання програми

Запуск вебпорталу здійснюється за наступним алгоритмом:

– підготовка середовища: встановлення всіх необхідних пакетів за допомогою менеджера npm install;

– конфігурація: налаштування файлу .env з вказанням порту сервера, посилання на базу даних MongoDB та секретного ключа для авторизації;

– ініціалізація сервера: запуск програми командою npm start. При успішному старті в терміналі відображається повідомлення про підключення до бази даних;

– доступ користувача: відкриття браузера за адресою http://localhost:<порт>, після чого завантажується головна сторінка з вітриною останніх доданих фільмів.

4.4 Тестування програми

Для підтвердження надійності, безперебійної роботи та відповідності розробленого вебпорталу заявленим функціональним вимогам було проведено комплексне тестування. Основним методом було тестування методом «чорного ящика», який не втручається у внутрішній код і перевіряє реакцію системи на різні вхідні дані. Логічні модулі складають процес тестування: перевірка авторизації, тестування користувацького інтерфейсу, тестування адміністративної панелі та перевірка безпеки доступу.

Результати виконання тестових сценаріхв наведені у таблиці 4.1.

Таблиця 4.1 – Результати виконання тестових сценаріїв

№	Назва сценарію	Дія користувача	Очікуваний результат	Фактичний результат	Статус
1	2	3	4	5	6
1	Реєстрація нового користувача	Введення валідних email та пароля у форму реєстрації, натискання кнопки підтвердження	Створення запису в БД, автоматичний вхід, перенаправлення на сторінку акаунту	Запис створено, токен видано, виконано перехід	Успішно
2	Валідація некоректних даних	Спроба входу із зареєстрованим email, але невірним паролем	Відхилення запиту, поява повідомлення про помилку авторизації	Відображено повідомлення «Невірний пароль»	Успішно
3	Пошук контенту за назвою	Введення існуючої назви фільму у рядок пошуку	Динамічне оновлення сторінки, відображення карток, що відповідають запиту	Картки відфільтровано відповідно до введеного тексту	Успішно

Кінець таблиці 4.1

1	2	3	4	5	6
4	Виставлен-ня оцінки фільму	Авторизований користувач натискає на зірку рейтингу на сторінці фільму	Збереження оцінки в БД Ratings, перерахунок середнього балу фільму	Оцінку збережено, середній рейтинг миттєво оновлено на екрані	Успіш-но
5	Додавання фільму адміністра-тором	Заповнення форми додавання валідними даними та посиланнями	Створення документа в колекції Movies, поява сповіщення про успіх	Об’єкт створено, лічильник фільмів на дашборді збільшився	Успіш-но
6	Перевірка захисту маршрутів	Звичайний користувач без ролі admin намагається перейти за URL адмін-панелі	Блокування доступу сервером, помилка 403, переадресація	Доступ заблоковано, виконано перенаправ-лення на головну	Успіш-но

Критично важливим етапом перевірки є реакція системи на некоректні дії користувача. Механізми перевірки даних вбудовані в клієнтську та серверну частину. Наприклад, система перериває спробу відправити неповні дані у формі авторизації або при додаванні фільму. У результаті система видає відповідне попередження, що запобігає потраплянню пошкоджених даних у базу.

Демонстрація обробки помилки валідації введених даних наведена на рисунках 4.1-4.2.

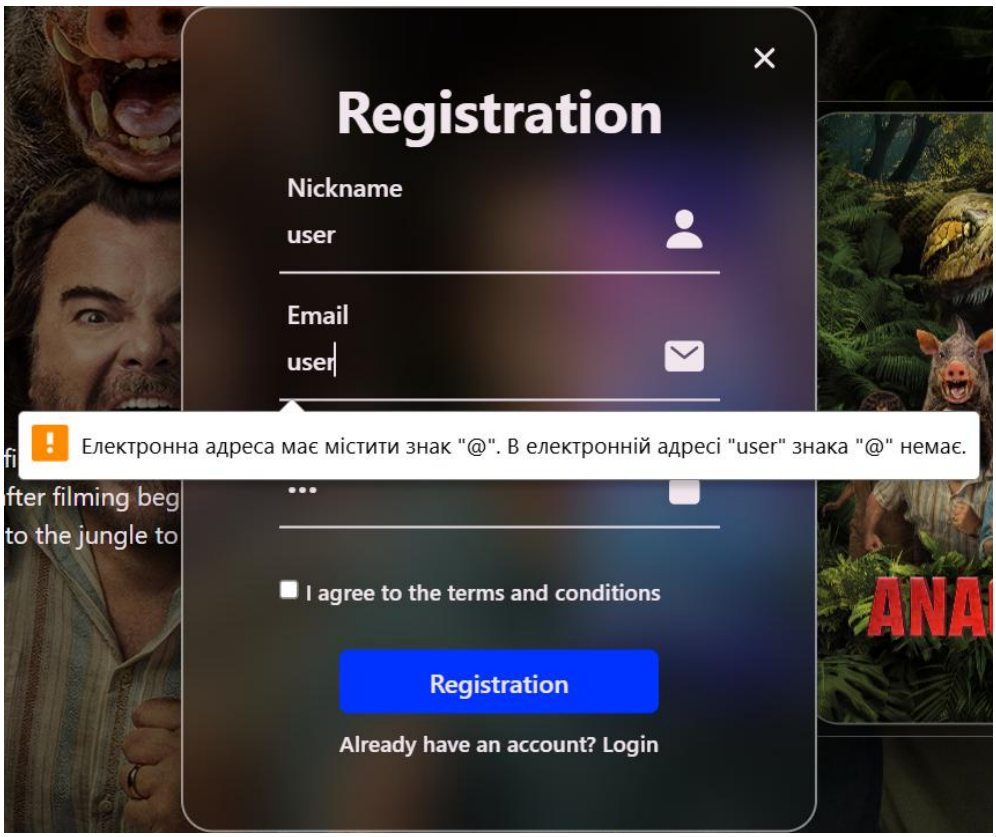

Рисунок 4.1 – Обробка помилки валідації введених даних користувачем

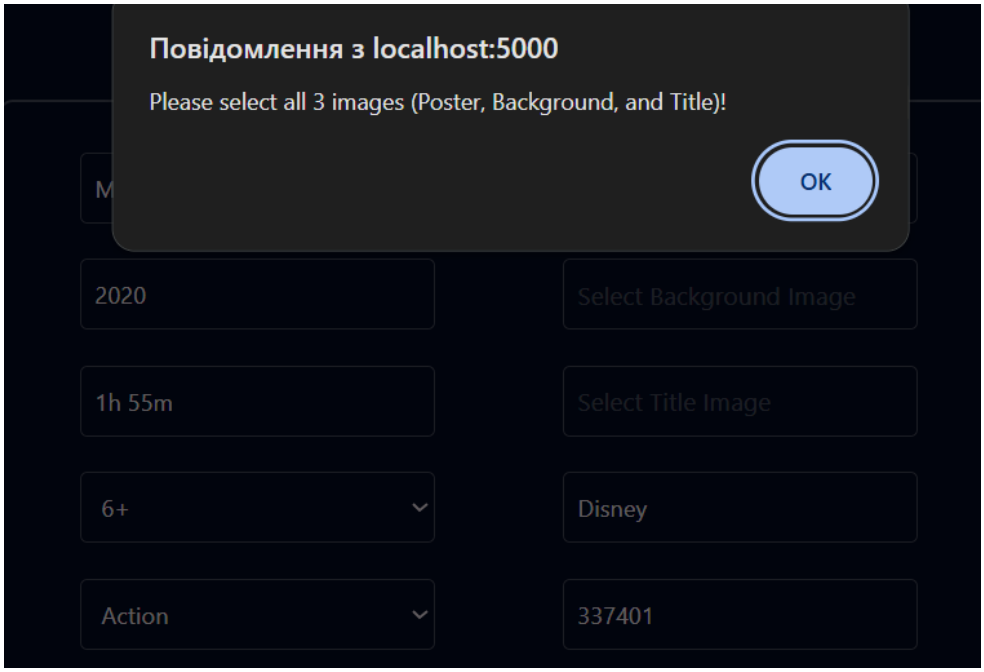

Рисунок 4.2 – Обробка помилки валідації введених даних адміністратором

При введенні правильних даних сервер успішно виконує бізнес-логіку та повертає клієнту позитивну відповідь, яка відображається в інтерфейсі негайно завдяки асинхронним запитам отримання даних.

Демонстрація інтерфейсу успішного виконання запиту до сервера наведена на рисунку 4.3.

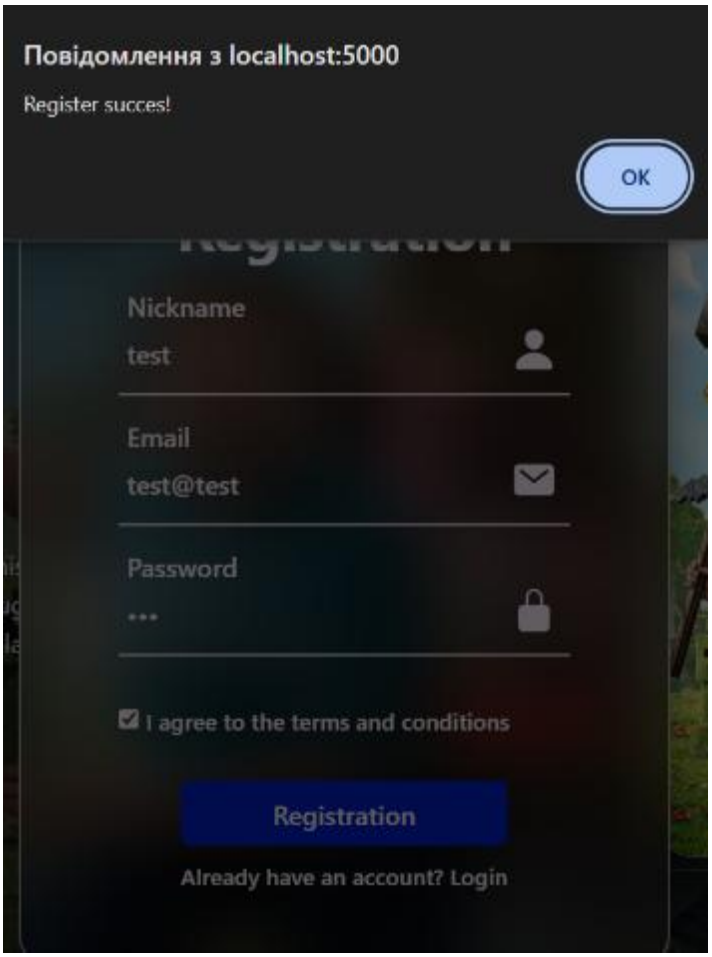

Рисунок 4.3 – Успішне виконання запиту до сервера

Імітації несанкціонованого доступу використовувалися для перевірки надійності маршрутизації та розмежування прав. Налаштовані middleware-функції на сервері Node.js блокують спроби доступу до захищених адміністративних сторінок без валидного JWT-токена адміністратора, а сервер повертає статус помилки доступу.

Демонстрація інтерфейсу реакції системи безпеки на несанкціоновану спробу доступу наведена на рисунку 4.4.

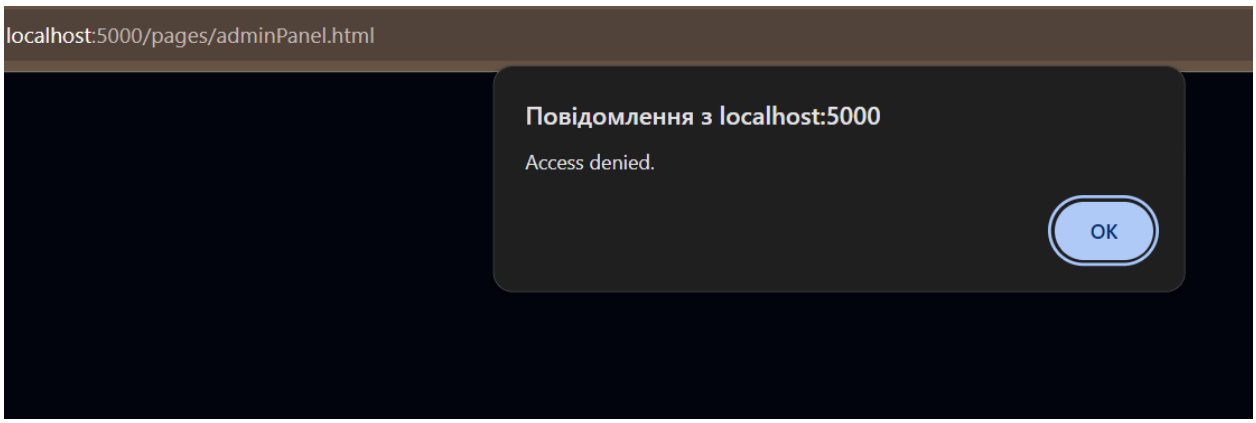

Рисунок 4.4 – Реакція системи безпеки на несанкціоновану спробу доступу

Під час тестування вебпорталу було виконано перевірку роботи інтерфейсу у браузерах Google Chrome та Microsoft Edge, оскільки саме вони найчастіше використовуються користувачами для роботи з вебзастосунками. Основною метою тестування була перевірка правильності відображення елементів інтерфейсу та стабільності роботи клієнтської частини системи.

У процесі перевірки було протестовано коректність відображення сітки карток із фільмами, роботу навігаційних елементів, кнопок керування та адаптацію інтерфейсу під різні розміри вікна браузера.

Крім цього, було перевірено швидкість завантаження сторінок, коректність оновлення контенту без перезавантаження браузера та стабільність роботи асинхронних запитів до сервера. Під час тестування не було виявлено критичних помилок у відображенні DOM-структури сторінок або проблем із позиціонуванням елементів інтерфейсу.

Демонстрація перевірки відображення інтерфейсу у браузерах наведена на рисунках 4.5-4.6.

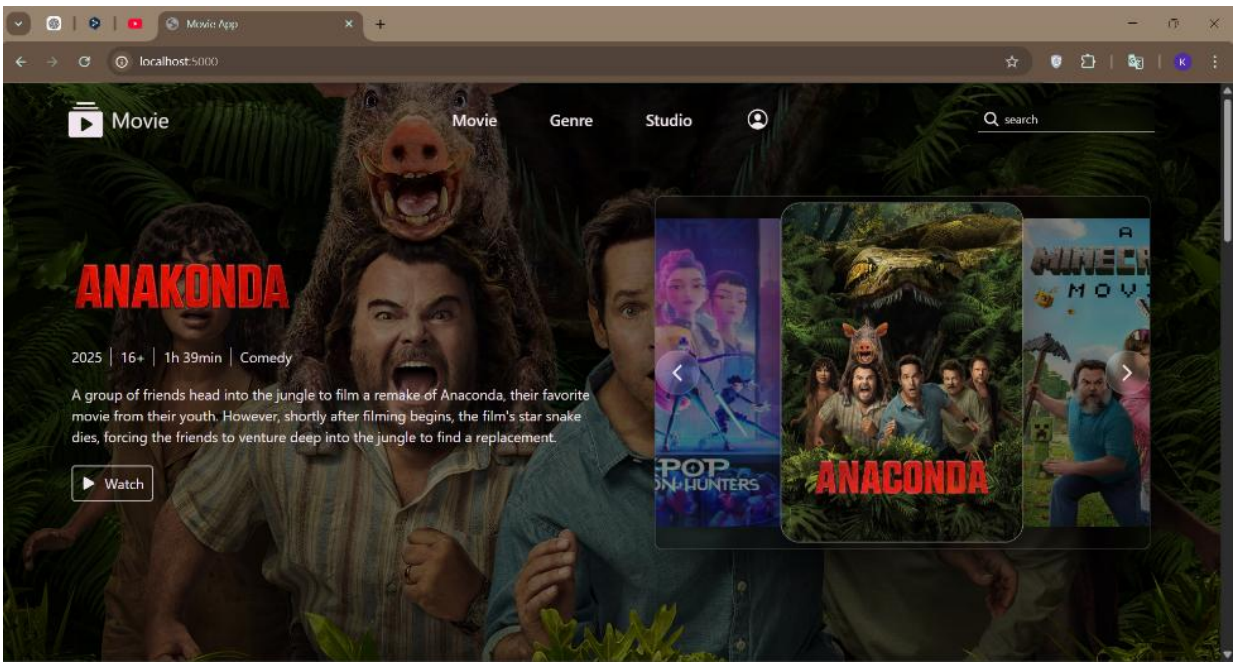

Рисунок 4.5 – Перевірка відображення інтерфейсу у Google Chrome

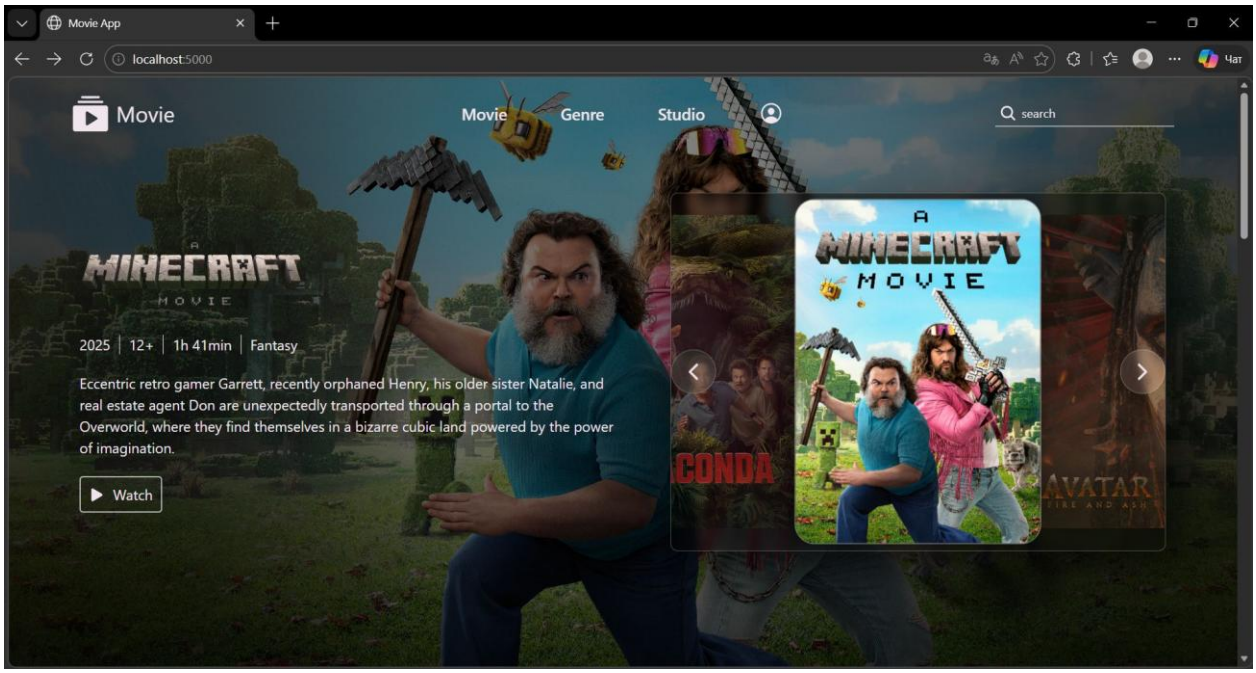

Рисунок 4.6 – Перевірка відображення інтерфейсу у Microsoft Edge

Результати комплексного тестування підтвердили, що всі програмні модулі працюють коректно, а інтерфейс вебпорталу залишається стабільним, простим і чуйним до дій користувача в усіх тестових випадках.

4.5 Висновки за розділом

У четвертому розділі детально розглянуто процедури експлуатації та тестування вебпорталу дистрибуції кіноконтенту, який було розроблено. Адміністратори отримують оптимізоване програмне забезпечення для управління, а користувачі отримують простий доступ до кіноконтенту.

Було створено детальний опис апаратних і програмних умов, необхідних для правильного запуску системи. Завдяки використанню ефективних технологічних стеків, серверна частина може працювати на базових обчислювальних потужностях, а клієнтська частина не вимагає жодних жорстких пристроїв користувачів, крім сучасного браузера та стабільного інтернет-з’єднання. Алгоритм конфігурації та запуску програми в робочому середовищі детально описано.

Особлива увага приділялася системному тестуванню інтерфейсу програми. Було зроблена таблиця тестових сценаріїв, яка охопила перевірку функціональності модулів реєстрації, валідації форм, управління контентом та соціальної взаємодії. Візуальні матеріали наочно підтверджують результати тестування. Завершуючи розділ, було показано, що розроблена система працює стабільно, повністю відповідає технічному завданню та готова до впровадження в експлуатацію в реальному житті.

ВИСНОВКИ

У дипломній роботі реалізовано повний цикл проєктування та програмної реалізації вебпорталу призначеного для дистрибуції кіноконтенту. У процесі роботи було вирішено багато науково-технічних завдань з метою створення швидкодіючої, надійної та зручної системи.

Під час аналізу встановлено, що існуючі рішення часто перевантажені зайвим функціоналом і працюють неефективно з даними. Тому було обрано сучасний стек технологій: Node.js з Express та MongoDB. Використання NoSQL-бази дало змогу гнучко працювати з медіаданими та забезпечити швидкий доступ до інформації. Відмова від важких фронтенд-фреймворків на користь TypeScript і SASS позитивно вплинула на швидкість завантаження та адаптивність інтерфейсу.

У процесі реалізації створено модульну архітектуру, яка розділяє маршрути, моделі та середовище, що полегшує підтримку та масштабування системи. Розроблено зручну адмін-панель для керування контентом. Також реалізовано інтеграцію з TMDB, що дозволило автоматизувати отримання даних про фільми.

Функціонал порталу включає пошук, фільтрацію, систему оцінок і коментарів. Безпека забезпечується за рахунок JWT-авторизації та хешування паролів.

Результати тестування показали, що система добре обробляє запити та працює стабільно навіть під навантаженням. У підсумку створено повноцінний програмний продукт, який відповідає вимогам швидкодії, надійності та безпеки і може використовуватися на практиці.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. UASerials. URL: https://uaserials.my/ (дата звернення: 07.05.2026).
2. kinoukr.tv. URL: https://kinoukr.tv/main/ (дата звернення: 07.05.2026).
3. Megogo. URL: https://megogo.net/ua (дата звернення: 07.05.2026).
4. Node.js Documentation. Node.js. URL: https://nodejs.org/docs/latest/api/ (date of access: 07.05.2026).
5. PHP VS Node.js. GeeksforGeeks. URL: https://www.geeksforgeeks.org/php/php-vs-node-js/ (date of access: 07.05.2026).
6. Node.js vs Python. GeeksforGeeks. URL: https://www.geeksforgeeks.org/blogs/node-js-vs-python/#nodejs-vs-python (date of access: 07.05.2026).
7. REST API. REST API. URL: https://restfulapi.net/ (date of access: 07.05.2026).
8. JavaScript Documentation. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (date of access: 07.05.2026).
9. TypeScript Documentation. TypeScript. URL: https://www.typescriptlang.org/docs/ (date of access: 07.05.2026).
10. MongoDB Documentation. MongoDB. URL: https://www.mongodb.com/docs/ (date of access: 07.05.2026).
11. VS Code Documentation. Visual Studio Code. URL: https://code.visualstudio.com/docs (date of access: 07.05.2026).
12. WebStorm Documentation. JetBrains. URL: https://www.jetbrains.com/help/webstorm/javascript-specific-guidelines.html (date of access: 07.05.2026).
13. Sublime Text Documentation. Sublime Text. URL: https://www.sublimetext.com/docs/ (date of access: 07.05.2026).
14. Visual Studio Code vs. Sublime Text. InfoWorld. URL:

https://www.infoworld.com/article/2263049/visual-studio-code-vs-sublime-text-which-code-editor-should-you-use.html (date of access: 07.05.2026).

15. Visual Studio Code vs WebStorm. GitHub Gist. URL: https://gist.github.com/joeytwiddle/49ecf347bf97c5a67a8d713dd437efcf (date of access: 07.05.2026).

16. JSON Web Token Best Current Practices : RFC 8725. IETF Datatracker. URL: https://datatracker.ietf.org/doc/html/rfc8725 (date of access: 07.05.2026).

ДОДАТОК А

Технічне завдання

Вступ

У сучасних умовах швидкого розвитку інтернет-технологій та переходу медіа у цифровий формат значно зросла популярність вебпорталів для перегляду та дистрибуції кіноконтенту. Користувачі очікують від таких сервісів високої швидкості роботи, зручного інтерфейсу та якісного відображення інформації. Однак більшість існуючих рішень створені на базі універсальних CMS, які часто є перевантаженими та незручними в адмініструванні.

У зв’язку з цим актуальним є створення спеціалізованого вебпорталу з простим і зрозумілим інтерфейсом, який дозволить оптимізувати процес керування контентом і забезпечити стабільний доступ до медіа. Для реалізації такого рішення використовуються сучасні технології, зокрема Node.js, Express та MongoDB, що дає змогу створити продуктивний, масштабований і надійний програмний продукт.

А.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Інженерія програмного забезпечення вебпорталу для перегляду відео з оцінюванням та адмініструванням користувацької взаємодії», затверджене наказом Національного університету «Запорізька політехніка» № 139 від 07 квітня 2026 р.

А.2 Основні вимоги до програми, що розробляється

А.2.1 Вимоги до функціональних характеристик

Програма повинна забезпечувати:

– автентифікацію та авторизацію користувачів із використанням JWT-токенів і розмежуванням прав доступу;

– відтворення відеоконтенту через інтеграцію із зовнішніми базами даних;

– можливість керування контентом через адмін-панель;

– швидкий пошук і фільтрацію фільмів за різними параметрами;

– взаємодію користувачів із системою.

А.2.2 Вимоги до інтерфейсу програми

Інтерфейс програми має відповідати таким вимогам:

– простота та лаконічність без зайвих елементів;

– інтуїтивно зрозуміла навігація між основними розділами;

– динамічне оформлення сторінок залежно від обраного контенту;

– коректна робота в сучасних браузерах незалежно від операційної системи.

А.2.3 Вимоги до надійності

Система повинна забезпечувати стабільну роботу серверної частини та надійне з’єднання з базою даних MongoDB. Важливою вимогою є коректна обробка асинхронних запитів і запобігання помилкам при взаємодії із зовнішніми API, що гарантує безперебійну роботу порталу.

А.2.4 Вимоги до технічних засобів

Доступ до вебпорталу має здійснюватися з персональних комп’ютерів або ноутбуків із сучасними браузерами. Мінімальні вимоги до клієнтських пристроїв:

– 2 ГБ оперативної пам’яті;

– підтримка HTML5 та JavaScript.

Серверна частина повинна працювати в середовищі, що підтримує Node.js. Мінімальні вимоги до сервера:

– процесор від 2.0 ГГц;

– 4 ГБ оперативної пам’яті;

– SSD-накопичувач для швидкого доступу до даних.

Також необхідне стабільне інтернет-з’єднання для взаємодії із зовнішнім API TMDB та забезпечення роботи сервісу.

ДОДАТОК Б

Текст програми

Б.1 Текст програми з файлу server.js

```
const express = require("express");
const mongoose = require("mongoose");
const cors = require("cors");
const path = require("path");
require("dotenv").config();

const moviesRoute = require("./routes/movies");
const userRoutes = require("./routes/users");
const ratingRoutes = require("./routes/raitings");
const commentsRoutes = require("./routes/comments");

const HOST = "localhost";
const PORT = process.env.PORT || 5000;
const MONGO_DB = process.env.MONGO_URL;

const app = express();
app.use(cors());
app.use(express.json());
app.use(express.static(path.join(__dirname, "public")));

app.use("/api/movies", moviesRoute);
app.use("/api/users", userRoutes);
app.use("/api/ratings", ratingRoutes);
app.use("/api/comments", commentsRoutes);

app.get("/", (req, res) => {
  const filePath = path.join(__dirname, "public", "pages",
"index.html");
  res.sendFile(filePath);
});

app.use((req, res) => {
  res.status(404).sendFile(path.join(__dirname, "public", "pages",
"404.html"));
});

mongoose
  .connect(MONGO_DB)
  .then(() => {
    console.log("Connected to DB");
```

```
  app.listen(PORT, HOST, () => {
    console.log(`Server started on http://${HOST}:${PORT}`);
  });
})
.catch((err) => {
  console.log("DB Error:", err);
});
```

Б.2 Текст програми з файлу watch.ts

```
const urlParams = new URLSearchParams(window.location.search);
const movieId = urlParams.get("id");

document.addEventListener("DOMContentLoaded", () => {
  loadMoviePlayer();
  const videoWrapper = document.querySelector(".video-wrapper") as
HTMLElement;
  const fullscreenBtn = document.getElementById(
    "fullscreen-btn"
  ) as HTMLButtonElement;

  const toggleFullscreen = () => {
    if (!document.fullscreenElement) {
      if (videoWrapper) {
        videoWrapper.requestFullscreen().catch((err) => {
          console.error(`Error attempting to enable fullscreen:
${err.message}`);
        });
      }
    } else {
      document.exitFullscreen();
    }
  };

  if (fullscreenBtn && videoWrapper) {
    fullscreenBtn.addEventListener("click", toggleFullscreen);
  }

  const starsContainer = document.getElementById("stars-container");
  const stars = starsContainer?.querySelectorAll(".icon-star");
```

```
const ratingValueDisplay = document.getElementById("rating-value");

let currentRating = 0;

const fillStars = (index: number) => {
  stars?.forEach((star, i) => {
    if (i <= index) {
      star.classList.add("active");
    } else {
      star.classList.remove("active");
    }
  });
};

const loadRatings = async () => {
  try {
    if (!movieId) return;

    const token = getToken();

    const headers: any = {
      "Content-Type": "application/json",
    };
    if (token) {
      headers["Authorization"] = "Bearer " + token;
    }

    const response = await fetch(`/api/ratings/${movieId}`, {
      method: "GET",
      headers: headers,
    });

    if (response.ok) {
      const data = await response.json();

      if (ratingValueDisplay) {
        ratingValueDisplay.textContent = data.average || "0";
      }

      currentRating = data.userRating || 0;
      fillStars(currentRating - 1);
    }
  } catch (error) {
```

```
    console.error("Error loading rating:", error);
  }
};

loadRatings();

starsContainer?.addEventListener("mouseleave", () => {
  fillStars(currentRating - 1);
});

if (stars) {
  stars.forEach((star, index) => {
    star.addEventListener("mouseenter", () => {
      fillStars(index);
    });

    star.addEventListener("click", async () => {
      const token = getToken();
      if (!token) {
        alert("Log in to rate!");
        return;
      }

      const clickedValue = index + 1;

      try {
        if (currentRating === clickedValue) {
          const response = await fetch("/api/ratings", {
            method: "DELETE",
            headers: {
              "Content-Type": "application/json",
              Authorization: "Bearer " + token,
            },
            body: JSON.stringify({ movieId: movieId }),
          });

          if (response.ok) {
            currentRating = 0;
          }
        } else {
          const response = await fetch("/api/ratings", {
            method: "POST",
            headers: {
```

```
                    "Content-Type": "application/json",
                    Authorization: "Bearer " + token,
                  },
                  body: JSON.stringify({
                    movieId: movieId,
                    rating: clickedValue,
                  }),
                });

                if (response.ok) {
                  currentRating = clickedValue;
                }
              }

              await loadRatings();
            } catch (error) {
              console.error("Voting error:", error);
            }
          });
        });
      }

      const inpComment = document.querySelector(".input-comment") as
HTMLInputElement;
      const sendBtn = document.getElementById("send-comment");

      inpComment?.addEventListener("focus", () => {
        sendBtn?.classList.remove("visualy-hidden");
      });

      inpComment?.addEventListener("blur", () => {
        setTimeout(() => {
          sendBtn?.classList.add("visualy-hidden");
        }, 200);
      });

      sendBtn?.addEventListener("click", async () => {
        const text = inpComment?.value;
        if (!text) {
          alert("You are trying to post an empty comment.");
          return;
        }
```

```
            const token = getToken();
            if (!token) {
              alert("Nema tokena");
              return;
            }

            const newCom = {
              movieId: movieId,
              text: text,
            };

            const response = await fetch("/api/comments", {
              method: "POST",
              headers: {
                "Content-Type": "application/json",
                Authorization: "Bearer " + token,
              },
              body: JSON.stringify(newCom),
            });

            if (response.ok) {
              inpComment.value = "";
              renderComments();
            }
          });

          if (typeof renderComments === "function") {
            renderComments();
          }
        });

        async function loadMoviePlayer() {
          const iframe = document.querySelector(".video-frame") as
HTMLIFrameElement;
          const urlParams = new URLSearchParams(window.location.search);
          const movieId = urlParams.get("id");

          try {
            const response = await fetch("/api/movies");
            const movies = await response.json();

            const body = document.querySelector("body");
```

```
            const currentMovie = movies.find(
              (movie: { tmdbId: string | null }) => movie.tmdbId === movieId
            );

            if (currentMovie) {
              const titleMovie = document.querySelector(".title-movie");
              const descriptionMovie = document.querySelector(".description-
movie");
              if (titleMovie) titleMovie.textContent = currentMovie.title;
              if (descriptionMovie) descriptionMovie.textContent =
currentMovie.description;
              if (body) {
                body.style.backgroundImage = `
            linear-gradient(to bottom, rgba(15, 23, 42, 0.8), rgba(15, 23, 42,
0.9)),
            url('${currentMovie.bgImg}')
          `;

                body.style.backgroundSize = "cover";
                body.style.backgroundPosition = "center";
                body.style.backgroundRepeat = "no-repeat";
                body.style.backgroundAttachment = "fixed";
              }
            }

            if (movieId && iframe) {
              iframe.allow = "autoplay; encrypted-media; fullscreen; picture-
in-picture";
              iframe.src =
`https://multiembed.mov/?video_id=${movieId}&tmdb=1`;
            } else {
              alert("Movie not found!");
              window.location.href = "/";
              return;
            }
          } catch (error) {
            console.error(error);
          }
        }

        async function renderComments() {
          const commentsContainer = document.querySelector(".comments");
          if (!commentsContainer) return;
```

```
        commentsContainer.innerHTML = "";

        const response = await fetch(`/api/comments/${movieId}`);
        const comments = await response.json();

        if (response.ok) {
          comments.forEach((comment: { nickname: string; date: string; text:
string }) => {
            const commentHTML = `<li class="comment-item">
              <div class="commentator-info">
                <div class="user-info">
                  <div class="avatar">
                    <svg
                    class="icon-person-circle brand-color"
                    width="32"
                    height="32"
                  >
                    <use href="../images/icons.svg#icon-person-circle"></use>
                  </svg>
                </div>
                <p class="nickname">${comment.nickname}</p>
              </div>
              <p                                    class="date-comment">${new
Date(comment.date).toLocaleString()}</p>
            </div>
            <div class="comment">${comment.text}</div>
          </li>`;

            commentsContainer.innerHTML += commentHTML;
          });
        }
      }
```

Б.3 Текст програми з файлу adminGuard.ts

```
      (function () {
        console.log("Checking admin rights...");

        const currentUserStr =
          localStorage.getItem("currentUser")                                ||
sessionStorage.getItem("currentUser");
```

```
  if (!currentUserStr) {
    window.location.href = "/";
    return;
  }

  try {
    const user = JSON.parse(currentUserStr);

    if (user && user.role && user.role.includes("admin")) {
      window.addEventListener("DOMContentLoaded", () => {
        document.body.style.display = "block";
      });
      if (document.readyState !== "loading") {
        document.body.style.display = "block";
      }
    } else {
      alert("Access denied.");
      window.location.href = "/";
    }
  } catch (e) {
    console.error("Rights verification error:", e);
    window.location.href = "/";
  }
})();
```

Б.4 Текст програми з файлу filter.ts

```
const allGenre = document.querySelectorAll(".genre-item");

allGenre.forEach((genre) => {
  genre.addEventListener("click", (e) => {
    e.preventDefault();
    const selectedGenre = (genre.querySelector(".title-genre") as
HTMLElement)
      .innerText;
    localStorage.setItem("filterType", JSON.stringify(selectedGenre));
    window.location.href = "/pages/genre-studio.html";
  });
});
```

```
const allStudio = document.querySelectorAll(".studio-item");

allStudio?.forEach((studio) => {
  studio.addEventListener("click", (e) => {
    e.preventDefault();
    const selectedStudio = (studio.querySelector(".title-studio") as
HTMLElement)
      .innerText;
    localStorage.setItem("filterType", JSON.stringify(selectedStudio));
    window.location.href = "/pages/genre-studio.html";
  });
});
```

Б.5 Текст програми з файлу search.ts

```
const search = document.getElementById("input-search") as
HTMLInputElement | null;
const searchContainer = document.querySelector(".search");
let searchTimeout: any;

search?.addEventListener("input", (e) => {
  const target = e.target as HTMLInputElement;
  const inpValue = target.value.toLowerCase();

  const swiperContainer = document.getElementById("movie-swiper") as
any;
  const movieSwiper = swiperContainer?.swiper;
  const wrapper = document.querySelector(
    "#movie-swiper .swiper-wrapper"
  ) as HTMLElement;

  if (!wrapper) return;

  wrapper.style.transition = "opacity 0.1s ease";
  wrapper.style.opacity = "0";

  clearTimeout(searchTimeout);

  searchTimeout = setTimeout(() => {
    if (movieSwiper && movieSwiper.params.loop) {
      movieSwiper.loopDestroy?.();
```

```
        }

        const allItems = Array.from(wrapper.children) as HTMLElement[];

        allItems.forEach((item) => {
          if (item.classList.contains("swiper-slide-duplicate")) return;

          const titleElement = item.querySelector(".title-movie") as
HTMLElement;
          if (!titleElement) return;

          const titleMovie = titleElement.innerText.toLowerCase();

          const isMatch =
            titleMovie.startsWith(inpValue) || titleMovie.includes(" " +
inpValue);

          item.style.transition = "none";

          if (isMatch) {
            item.classList.add("swiper-slide");
            item.style.display = "";
          } else {
            item.classList.remove("swiper-slide");
            item.style.display = "none";
          }
        });

        if (movieSwiper) {
          movieSwiper.update();
          movieSwiper.slideTo(0, 0);

          if (inpValue === "" && movieSwiper.params.loop) {
            movieSwiper.loopCreate?.();
            movieSwiper.update();
          }
        }

        setTimeout(() => {
          allItems.forEach((item) => (item.style.transition = ""));
        }, 50);

        wrapper.style.opacity = "1";
```

```
  }, 100);
});

search?.addEventListener("focus", (e) => {
  searchContainer?.classList.add("sticky-active");
  const movieSection = document.getElementById("movie-section");
  if (movieSection) {
    movieSection.scrollIntoView({ behavior: "smooth" });
  }
});

search?.addEventListener("blur", (e) => {
  searchContainer?.classList.remove("sticky-active");
});
```

Б.6 Текст програми з файлу manage-comments.ts

```
const adminCommentContainer = document.querySelector(".admin-comment-container");
const token = getToken();

document.addEventListener("DOMContentLoaded", async () => {
  const response = await fetch("/api/comments/moderation", {
    method: "GET",
    headers: {
      "Content-Type": "application/json",
      Authorization: "Bearer " + token,
    },
  });

  if (response.ok) {
    const commentList = await response.json();

    for (const comment of commentList) {
      const response = await fetch(`/api/movies/${comment.movieId}`, {
        method: "GET",
        headers: {
          "Content-Type": "application/json",
          Authorization: "Bearer " + token,
        },
      });
```

```
            const foundMovie = await response.json();

            if (foundMovie) {
              const commentHTML = `<div class="comment-info" >
              <div class="comment-content">
                <p class="nickname">Nickname: ${comment.nickname}</p>
                <p class="movie">Movie: ${foundMovie.title}</p>
                <p class="movie">Comment: ${comment.text}</p>
              </div>
              <div class="control-btn">
                <button         class="approve-btn         btn"         data-
id=${comment._id}>Approve</button>
                <button         class="delete-btn         btn"         data-
id=${comment._id}>Delete</button>
              </div>
            </div>
      `;
              if  (adminCommentContainer)  adminCommentContainer.innerHTML  +=
commentHTML;
            }
          }
        }
      });

      adminCommentContainer?.addEventListener("click", async (e) => {
        const target = e.target as HTMLElement;
        const approveClick = target.closest(".approve-btn") as HTMLElement;
        const deleteClick = target.closest(".delete-btn") as HTMLElement;

        if (approveClick) {
          const commentId = approveClick.dataset.id;
          const response = await fetch(`/api/comments/moderation/${commentId}`,
{
            method: "PUT",
            headers: {
              "Content-Type": "application/json",
              Authorization: "Bearer " + token,
            },
          });

          if (response.ok) {
            location.reload();
          }
```

```
    }

    if (deleteClick) {
      const commentId = deleteClick.dataset.id;
      const response = await fetch(`/api/comments/${commentId}`, {
        method: "DELETE",
        headers: {
          "Content-Type": "application/json",
          Authorization: "Bearer " + token,
        },
      });

      if (response.ok) {
        location.reload();
      }
    }
  });
```

Б.7 Текст програми з файлу account.ts

```
  const currentUser =
    localStorage.getItem("currentUser") ||
sessionStorage.getItem("currentUser");
  const user = currentUser ? JSON.parse(currentUser) : [];

  const nicknameInp = document.getElementById("nickname") as
HTMLInputElement;
  const emailInp = document.getElementById("email") as HTMLInputElement;
  const oldPasswordInp = document.getElementById("old-password") as
HTMLInputElement;
  const newPasswordInp = document.getElementById("new-password") as
HTMLInputElement;
  const passwordAgaininp = document.getElementById(
    "password-again"
  ) as HTMLInputElement;
  const saveChangesBtn = document.getElementById("save-changes");

  const profileBtn = document.getElementById("profile-btn");
  const profileContainer = document.querySelector(".profile-container");

  const favoriteBtn = document.getElementById("favorite-btn");
```

```
const favoriteContainer = document.querySelector(".favorite-container");
const favoriteBlock = document.querySelector(".favorite-block");

profileBtn?.addEventListener("click", (e) => {
  e.preventDefault();
  favoriteBtn?.classList.remove("active");
  favoriteContainer?.classList.add("visualy-hidden");

  profileBtn.classList.add("active");
  profileContainer?.classList.remove("visualy-hidden");
  profileContainer?.classList.add("active");
});

favoriteBtn?.addEventListener("click", (e) => {
  e.preventDefault();

  profileBtn?.classList.remove("active");
  profileContainer?.classList.add("visualy-hidden");

  favoriteBtn.classList.add("active");
  favoriteContainer?.classList.remove("visualy-hidden");
  favoriteContainer?.classList.add("active");
});

saveChangesBtn?.addEventListener("click", async () => {
  if (oldPasswordInp.value) {
    if (newPasswordInp.value !== passwordAgaininp.value) {
      alert("The passwords don't match");
      return;
    }
  }

  const updateData = {
    newNickname: nicknameInp.value || "",
    newEmail: emailInp.value || "",
    oldPassword: oldPasswordInp.value || "",
    newPassword: newPasswordInp.value || "",
  };

  try {
    const response = await fetch("/api/users/profile", {
      method: "PUT",
      headers: {
```

```
            "Content-Type": "application/json",
            Authorization: "Bearer " + user.token,
          },
          body: JSON.stringify(updateData),
        });

        const data = await response.json();

        if (response.ok) {
          const userToSave = {
            ...data.user,
            token: data.token,
          };

          if (localStorage.getItem("currentUser"))
            localStorage.setItem("currentUser", JSON.stringify(userToSave));

          if (sessionStorage.getItem("currentUser"))
            sessionStorage.setItem("currentUser",
JSON.stringify(userToSave));

          oldPasswordInp.value = "";
          newPasswordInp.value = "";
          passwordAgaininp.value = "";
          alert("Profile updated successfully!");
        } else {
          alert(data.message);
        }
      } catch (error) {
        console.error(error);
        alert("Server connection error");
      }
    });

    favoriteBlock?.addEventListener("click", async (e) => {
      const target = e.target as HTMLElement;
      const bookmarkBtn = target.closest(".bookmark");

      if (bookmarkBtn) {
        e.preventDefault();

        const movieItem = bookmarkBtn.closest(".movie-item");
        const idToRemove = movieItem?.getAttribute("data-id");
```

```
    if (!idToRemove) return;

    try {
      const response = await fetch("/api/users/favorites", {
        method: "PUT",
        headers: {
          "Content-Type": "application/json",
          Authorization: "Bearer " + user.token,
        },
        body: JSON.stringify({ movieId: idToRemove }),
      });

      if (response.ok) {
        const data = await response.json();

        movieItem?.remove();
        user.savedMovies = data.savedMovies;

        localStorage.setItem("currentUser", JSON.stringify(user));
        if (sessionStorage.getItem("currentUser")) {
          sessionStorage.setItem("currentUser", JSON.stringify(user));
        }

        if (favoriteBlock.children.length === 0) {
          favoriteBlock.innerHTML = `<p>No saved movies</p>`;
        }
      }
    } catch (error) {
      console.error(error);
    }
  }
});

document.addEventListener("DOMContentLoaded", async () => {
  if (!currentUser) {
    window.location.href = "/";
    return;
  }

  nicknameInp.value = user.nickname;
  emailInp.value = user.email;
```

```
        if (!favoriteBlock) return;

        const response = await fetch("/api/users/favorites", {
          headers: {
            Authorization: "Bearer " + user.token,
          },
        });

        const favoriteMovies = await response.json();

        if (favoriteMovies.length === 0) {
          favoriteBlock.innerHTML = `<p>No saved movies</p>`;
          return;
        }

        favoriteMovies.forEach(
          (movie: {
            tmdbId: any;
            posterImg: string;
            year: string;
            age: string;
            genre: string;
          }) => {
            const    movieHTML    =    `<div    class="movie-item"    data-
id="${movie.tmdbId}">
          <img
            src="${movie.posterImg}"
            alt=""
            class="poster-img"
          />
          <div class="movie-section__category-container">
            <p class="year">${movie.year}</p>
            <p class="age">${movie.age}</p>
            <p class="genre">${movie.genre}</p>
          </div>
          <button type="button" class="bookmark active">
            <svg class="icon-bookmark" width="28" height="28">
              <use href="../images/icons.svg#icon-bookmark"></use>
            </svg>
          </button>
        </div>`;

            favoriteBlock.innerHTML += movieHTML;
```

```
      }
    );
});
```

ДОДАТОК В

Слайди презентації

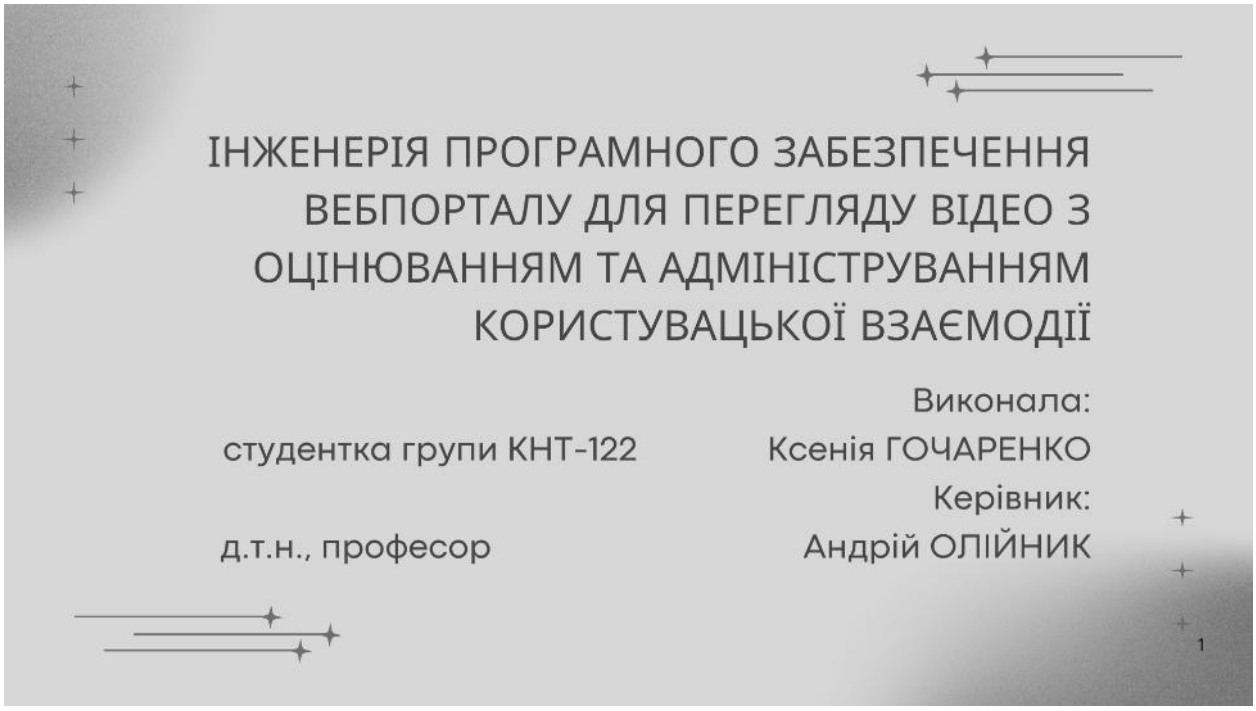

Рисунок В.1 – Слайд

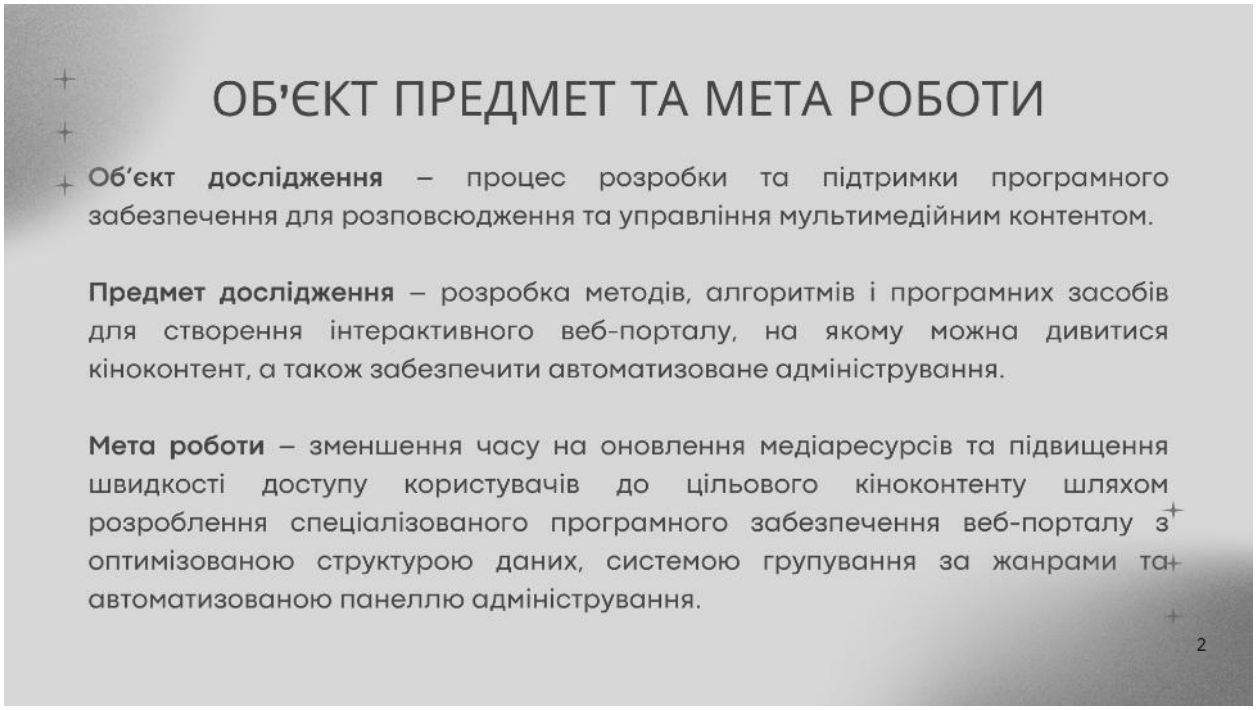

Рисунок В.2 – Слайд 2

Рисунок В.3 – Слайд 3

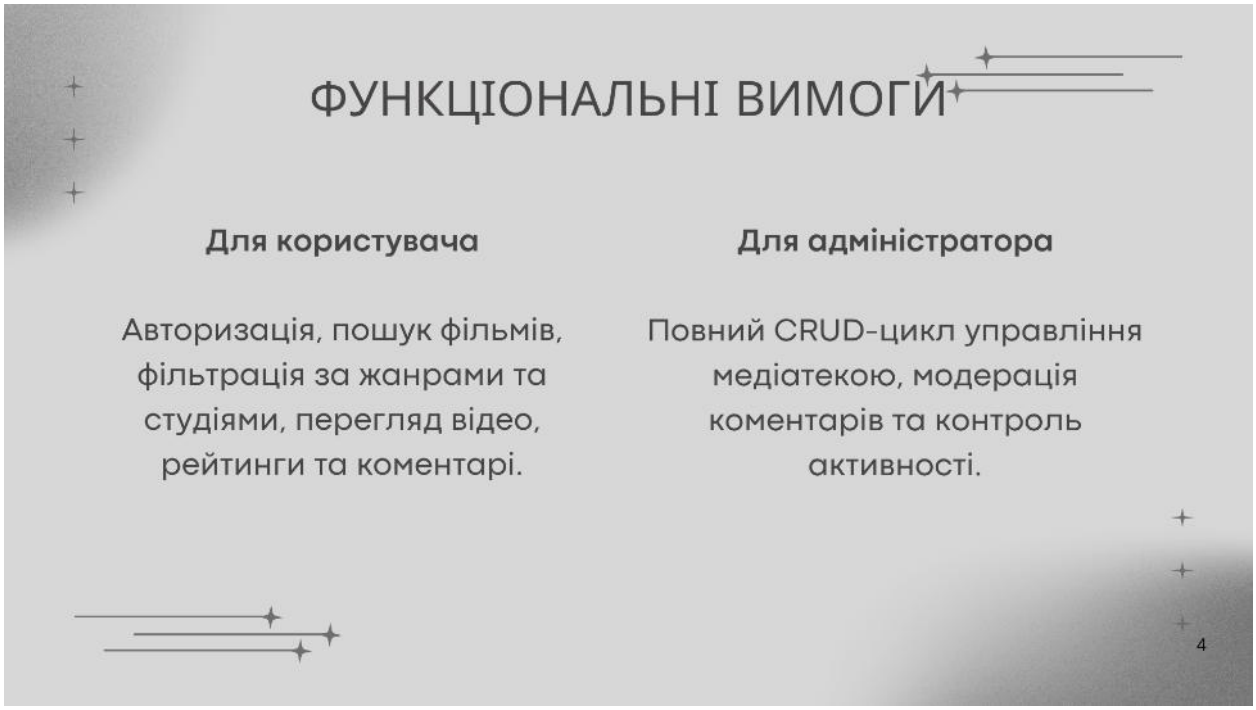

Рисунок В.4 – Слайд 4

ПОРІВНЯННЯ ІСНУЮЧИХ АНАЛОГІВ

Критерій порівняння	UASerials	Kinoukr.tv	Megogo	Розроблений вебпортал
Архітектура системи та БД	Неоптимізована, важкий DOM	Монолітна «коробкова» CMS, реляційна БД	Закрита корпоративна платформа	Клієнт-серверна, документо-орієнтована
Швидкодія клієнтської частини	Низька, затримки через сторонні скрипти	Середня	Швидка передача відео	Висока
Панель адміністрування	Стандартна	Універсальна CMS, наявність надлишкового функціоналу	Закрита корпоративна система	Кастомна та лаконічна
Соціальна взаємодія	Система коментарів	Обмежена	Відсутня	Система коментарів, рейтинги, закладки
Навігація та каталогізація	Розгалужена	Проста та інтуїтивно зрозуміла	Якісна та швидка структуризація	Динамічний пошук та фільтрація без перезавантаження сторінок

5

Рисунок В.5 – Слайд 5

ВИБІР МОВИ ПРОГРАМУВАННЯ

Критерії порівняння	Node.js (TypeScript)	Python	PHP
Архітектура обробки запитів	Асинхронна, код виконується паралельно, без очікування завершення функцій, що забезпечує швидкість	Синхронна, використовує синхронні обчислення, що уповільнює його роботу	Синхронна, може уповільнювати роботу через тривалі операції, що блокують виконання наступного коду
Єдина мова для клієнта та сервера	Написаний на JavaScript, тому дозволяє використовувати одну мову для фронтенду та бекенду, спрощуючи розробку	Можна ефективно використовувати як для фронтенд, так і для бекенд розробки завдяки широкому набору інструментів та	Доводиться перемикатися між різними мовами та синтаксисом
Швидкодія та продуктивність	Висока	Середня	Середня
Сумісність із форматом даних NoSQL	Нативна	Потребує конвертації	Потребує конвертації
Оптимальна сфера застосування	Вебпортали, стрімінг, REST API з високим навантаженням	Аналіз даних, машинне навчання, скрипти	Традиційні сайти, класичні CMS

6

Рисунок В.6 – Слайд 6

ВИБІР СЕРЕДОВИЩА РОЗРОБКИ

Критерії порівняння	Visual Studio Code	Sublime Text	WebStorm
Швидкодія та споживання пам'яті	Висока продуктивність, помірне споживання	Екстремальна швидкодія, мінімальне споживання ресурсів	Високе навантаження на систему, потребує до 3 ГБ RAM, тривале індексування
Підтримка TypeScript та Node.js	Нативна	Базова, потребує встановлення сторонніх плагінів	Глибокий семантичний аналіз, інтелектуальне автодоповнення
Вбудовані інструменти	Повний набір	Мінімальний, інструменти потребують ручного налаштування	Максимально розширений інтегрований набір
Розширюваність	Величезна екосистема плагінів	Широка підтримка пакетів	Більшість функцій вбудовано від початку
Ліцензія та вартість	Безкоштовне	Комерційна	Комерційна (платна підписка)

7

Рисунок В.7 – Слайд 7

СХЕМА СИСТЕМИ

8

Рисунок В.8 – Слайд 8

Рисунок В.9 – Слайд 9

Рисунок В.10 – Слайд 10

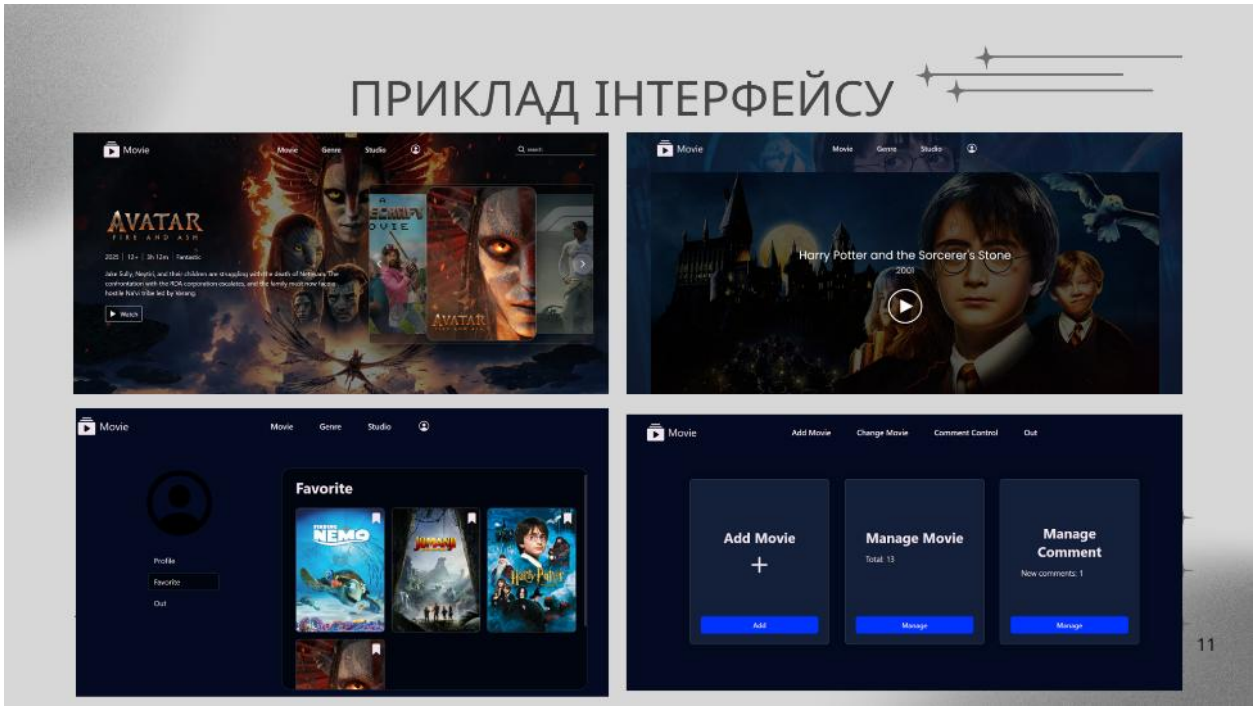

Рисунок В.11 – Слайд 11

№	Назва сценарію	Дія користувача	Очікуваний результат	Фактичний результат	Статус
1	Реєстрація нового користувача	Введення валідних email та пароля у форму реєстрації, натискання кнопки підтвердження	Створення запису в БД, автоматичний вхід, перенаправлення на сторінку акаунту	Запис створено, токен видано, виконано перехід	Успішно
2	Валідація некоректних даних	Спроба входу із зареєстрованим email, але невірним паролем	Відхилення запиту, поява повідомлення про помилку авторизації	Відображено повідомлення «Невірний пароль»	Успішно
3	Пошук контенту за назвою	Введення існуючої назви фільму у рядок пошуку	Динамічне оновлення сторінки, відображення карток, що відповідають запиту	Картки відфільтровано відповідно до введеного тексту	Успішно
4	Виставлення оцінки фільму	Авторизований користувач натискає на зірку рейтингу на сторінці фільму	Збереження оцінки в БД Ratings, перерахунок середнього балу фільму	Оцінку збережено, середній рейтинг миттєво оновлено на екрані	Успішно
5	Додавання фільму адміністратором	Заповнення форми додавання валідними даними та посиланнями	Створення документа в колекції Movies, поява сповіщення про успіх	Об'єкт створено, лічильник фільмів на дашборді збільшився	Успішно
6	Перевірка захисту маршрутів	Звичайний користувач без ролі admin намагається перейти за URL адмін-панелі	Блокування доступу сервером, помилка 403, переадресація	Доступ заблоковано, виконано перенаправлення на головну	Успішно

Рисунок В.12 – Слайд 12

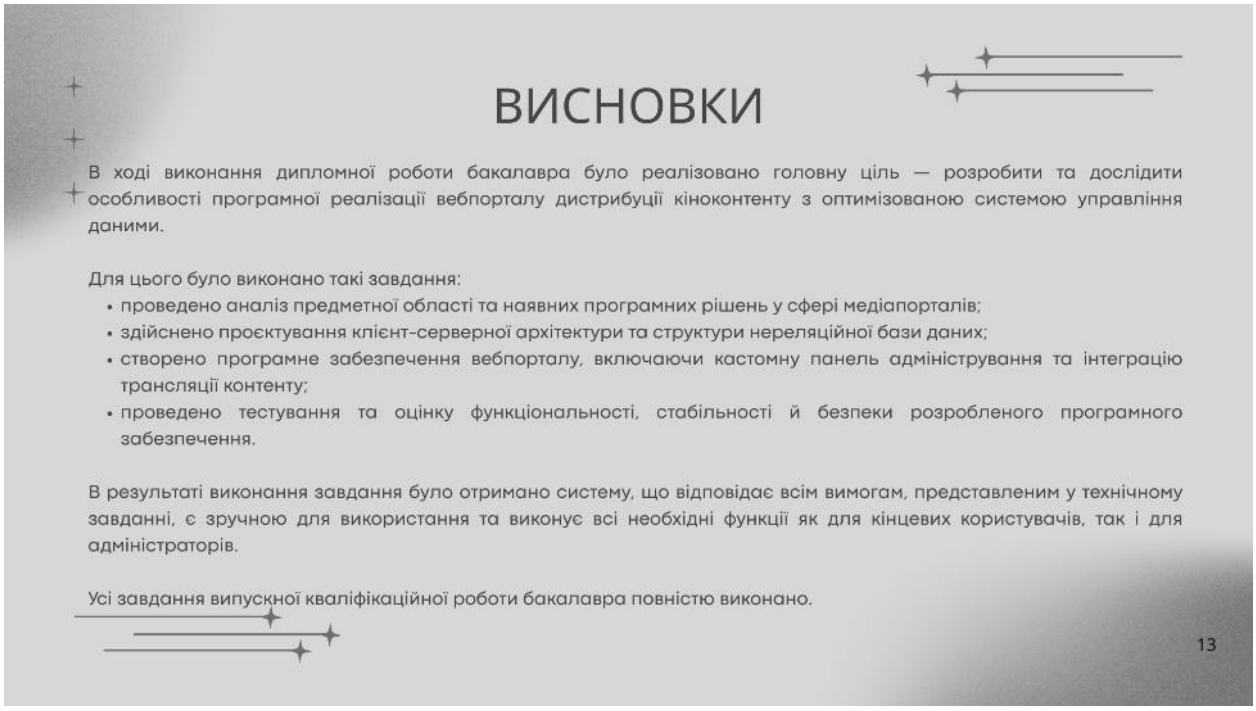

Рисунок В.13 – Слайд 13