МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«РОЗРОБКА ВЕБ-ДОДАТКУ ДЛЯ ПЕРЕГЛЯДУ ІНФОРМАЦІЇ ПРО ФІЛЬМИ ТА СЕРІАЛИ»**

Виконала: студентка 4 курсу, групи 6.1262
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)
освітньої програми Інформаційні системи та технології
(назва освітньої програми)
К.В. Мунтян
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2026

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 126 Інформаційні системи та технології
(шифр і назва)

Освітня програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри комп'ютерних наук, к.пед.н., доцент

_____ Пшенична О.С.
(підпис)

" 08 " грудня 2025 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Мунтян Крістіні Володимирівні
(прізвище, ім'я та по-батькові)

1. Тема роботи Розробка веб-додатку для перегляду інформації про фільми та серіали

керівник роботи Решевська Катерина Сергіївна, к.т.н., доцент
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 02 » грудня 2025 року № 1741-с

2. Строк подання студентом роботи 04.06.2026

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Розробка веб-додатку для перегляду інформації про фільми та серіали.

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 05.12.2025

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	15.01.2026	
2.	Збір вихідних даних.	17.02.2026	
3.	Обробка методичних та теоретичних	10.03.2026	
	джерел.		
4.	Розробка першого та другого розділу.	16.04.2026	
5.	Розробка третього розділу.	27.05.2026	
6.	Оформлення та нормоконтроль	04.06.2026	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	20.06.2026	

Студент _____ К.В. Мунтян
(підпис) (ініціали та прізвище)

Керівник роботи _____ К.С. Решевська
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Розробка веб-додатку для перегляду інформації про фільми та серіали»: 59 с., 54 рис., 7 табл., 16 джерел.

БАЗА ДАНИХ, БЕКЕНД, ВЕБДОДАТОК, ВЕБПРОГРАМУВАННЯ, СУБД, ФРЕЙМВОРК, ФРОНТЕНД.

Об’єкт дослідження – технології веброзробки.

Предмет дослідження – розробка вебдодатку за допомогою JavaScript-фреймворків та інших технологій веброзробки.

Мета роботи: розробити вебдодаток, який дозволятиме користувачам переглядати інформацію про фільми та серіали, залишати відгуки з реєстрацією або анонімно, мати особистий кабінет користувача.

Методи дослідження – аналіз, порівняння, експеримент.

У ході роботи було розроблено вебдодаток, який дозволить користувачам переглядати інформацію про фільми та серіали, реєструватися, авторизуватися, залишати відгуки, які можуть переглядати інші користувачі, та переглядати свої коментарі в особистому кабінеті.

SUMMARY

Bachelor's Qualifying Thesis «Development of a Web Application for Viewing Information About Movies and Series»: 59 pages, 54 figures, 7 tables, 16 references.

DATABASE, BACKEND, WEB APPLICATION, WEB PROGRAMMING, DBMS, FRAMEWORK, FRONTEND.

Object of the study – web development technologies.

Subject of the study – web application development using JavaScript frameworks and other web development technologies.

Aim of the study: develop a web application that will allow users to view information about movies and series, leave reviews with registration or anonymously, and have a personal user account.

Methods of research – analysis, comparison, experiment.

In the course of the work, a web application was developed that will allow users to view information about films and series, register, log in, leave reviews that other users can view, and view their comments in their personal account.

ЗМІСТ

Завдання на кваліфікаційну роботу……..2
Реферат 4
Summary 5
Вступ..... 8
1 Дослідження основ веброзробки 9
1.1 Огляд основних технологій веброзробки 9
1.2 Аналіз систем управління базами даних 11
1.2.1 Огляд існуючих СУБД 12
1.3 Висновок до першого розділу..... 14
2 Проєктування та обґрунтування вибору технології реалізації вебдодатку . 15
2.1 Архітектура вебдодатку 15
2.2 Обґрунтування вибору технологій 17
2.2.1 Вибір способу реалізації серверної частини 17
2.2.2 Виріб способу реалізації клієнтської частини 19
2.3 Проєктування серверної частини 20
2.3.1 Забезпечення безпеки серверної частини..... 21
2.4 Організація взаємодії клієнтської та серверної частин..... 22
2.5 Проєктування бази даних..... 24
2.6 Висновок до другого розділу 26
3 Програмна реалізація вебдодатку..... 28
3.1 Створення бази даних..... 28
3.2 Реалізація серверної частини 30
3.2.1 Забезпечення безпеки серверної частини..... 31
3.2.2 Реалізація маршрутизації REST API..... 33
3.2.3 Реалізація реєстрації та авторизації користувачів..... 35
3.3 Реалізація клієнтської частини 38
3.3.1 Реалізація захисту 38

3.3.2 Реалізація маршрутизації 39
3.3.3 Реалізація функцій отримання даних 40
3.3.4 Реалізація функцій відображення даних 43
3.3.5 Реалізація функцій реєстрації та авторизації 47
3.4 Демонстрація роботи вебдодатку 49
3.5 Висновок до третього розділу 55
Висновки 57
Перелік посилань 58

ВСТУП

У світі щороку виходить величезна кількість фільмів та серіалів, що сприяло появі таких вебресурсів, як IMDb, MyDramaList та AsianWiki, які допомагають глядачам у будь-який зручний для них час отримати опис сюжету, перелік акторського складу, жанрів чи прочитати відгуки інших людей про обране медіа.

Вебдодатки забезпечують зручне зберігання, обробку, відображення та отримання інформації. Для взаємодії з ними користувачу не потрібно мати якісь просунуті знання з інформаційних технологій чи встановлювати додаткові програми, оскільки буде достатньо звичайного запиту у рядку пошуку браузера, що значно спрощує доступ до сервісу. Завдяки швидкому розвитку технологій веброзробки можна створювати зручні та функціональні системи, які відповідають потребам користувачів.

Актуальність кваліфікаційної роботи полягає у створенні вебдодатку, який буде забезпечувати користувачам можливість зручно отримувати інформацію про фільми та серіали: переглядати дані, виконувати пошук, фільтрувати результати згідно потрібних критеріїв (за роками, жанрами чи країнами) та залишати відгуки. Для реалізації поставленої мети були використані набуті знання з сучасних технологій вебпрограмування, проєктування бази даних та створення інтуїтивно зрозумілого інтерфейсу користувача.

До виконання кваліфікаційної роботи були поставлені такі завдання:

- проаналізувати існуючі технології веброзробки;
- обрати оптимальні засоби реалізації;
- спроєктувати структуру додатку та бази даних;
- реалізувати серверну та клієнтську частини і їхню взаємодію;
- перевірити правильність роботи виконаного проєкту.

1 ДОСЛІДЖЕННЯ ОСНОВ ВЕБРОЗРОБКИ

1.1 Огляд основних технологій веброзробки

Веброзробка – це процес побудови вебдодатків та сервісів, що складається з наступних етапів згідно з джерелом [1]:

- аналіз цілей проєкту та вподобань аудиторії;
- розробка клієнтської частини (верстка сторінок за допомогою HTML, CSS, JavaScript тощо);
- розробка серверної частини для програмування логіки додатку та роботи з базами даних;
- тестування і усунення помилок.

Вебзастосунок – це розподілений застосунок, в якому клієнтом виступає браузер, а сервером – вебсервер [2].

Основними перевагами вебзастосунків згідно з джерелом [2] є:

- можливість роботи на будь-якій операційній системі завдяки підтримці різних веббраузерів;
- простота супроводу та оновлення, оскільки вебдодаток використовує єдину кодову базу, що спрощує підтримку порівняно з десктопними програмами;
- відсутність необхідності встановлення на пристрій користувача.

Вебдодатки поділяються на такі види згідно з джерелом [2]:

- SPA (Single Page Application) – це односторінкові застосунки зі статичною структурою, у яких під час роботи оновлюються лише окремі елементи сторінки без її повного перезавантаження;
- MPA (Multi Page Application) – це багатосторінкові застосунки, що взаємодіють із сервером та виконують повне оновлення сторінки після кожної дії користувача;

– PWA (Progressive Web App) – прогресивні вебдодатки, які поєднують можливості веб- і мобільних застосунків: вони можуть відкриватися у браузері або встановлюватися на пристрій у вигляді окремої іконки.

Одним з основних інструментів веброзробки є мови програмування, які поділяються на бекенд та фронтенд, які більш детально будуть розглянуті в наступних підрозділах.

Фронтенд – це клієнтська частина, яка охоплює всі елементи, що відображаються користувачеві та забезпечують взаємодію з системою. Його основним завданням є забезпечення зручного, інтуїтивно зрозумілого та естетично привабливого інтерфейсу, що базово можна реалізувати за допомогою HTML і CSS для структури та оформлення сторінок, а також мов програмування, на кшталт JavaScript, для забезпечення інтерактивності [3].

Бекенд – це серверна частина додатка, яка відповідає за логіку обробки запитів користувачів (отримання, додавання, редагування та видалення даних), управління обліковими записами, збереження інформації в базі даних, а також автентифікацію й авторизацію користувачів. Для розробки серверної частини широко застосовуються фреймворки – набори інструментів, бібліотек і правил, які використовуються для спрощення створення програмних додатків, а саме реалізації маршрутизації запитів, взаємодії з базами даних, налаштування безпеки та організації їхньої роботи [3].

Невід’ємною складовою серверної розробки є інтеграція з базами даних – впорядкованими сукупностями даних, призначеними для зберігання значних обсягів інформації. Окрім самого зберігання, важливими вимогами є швидкий доступ до інформації в будь-який момент часу, її впорядкування та управління, при цьому не порушуючи структурованості та цілісності інформації, що призвело до створення систем управління базами даних (СУБД) – спеціалізованого комплексу програмних засобів, що призначені для створення, зберігання, зберігання, обробки та ефективного використання баз даних, надаючи зручні засоби для роботи з інформацією [4].

1.2 Аналіз систем управління базами даних

При виборі типу використовуваних баз даних важливо розуміти їхні переваги і недоліки, а найважливіше – відповідність потребам завдання та характеру даних, які будуть зберігатися й оброблятися. Від правильного вибору системи керування базами даних залежать продуктивність, масштабованість, надійність і зручність подальшого супроводу вебдодатка. Різні типи баз даних мають власні особливості організації даних, механізми забезпечення цілісності інформації та підходи до обробки запитів, що робить їх більш або менш придатними для певних завдань. У розробці вебдодатків найчастіше використовуються два основні типи: реляційні та нереляційні бази даних, які було обрано для порівняння в цьому підрозділі.

Реляційна база даних (SQL) – це база, у якій дані організовані у вигляді таблиць, що суворо структуровані та пов’язані одна з одною. Таблиці містять рядки, кожен з яких має тип даних, та стовпчики – поля з призначеними їм типами даних. Також ще однією ключовою ознакою цієї моделі організації даних є використання мови запитів SQL, яка використовується для управління інформацією. Одними з найпоширеніших СУБД для таких баз даних є MySQL, PostgreSQL та Microsoft SQL Server [4].

Нереляційна база даних (NoSQL) зберігає інформацію без чіткої структури та зв’язків між собою, не застосовуючи SQL-запитів. Замість структурованих таблиць всередині знаходиться безліч різнорідних документів: зображення, відео та навіть публікації у соціальних мережах, що робить цей тип популярним для соціальних мереж та месенджерів. Одними з найпоширеніших СУБД для таких баз даних є Cassandra, MongoDB та Redis [4]. У табл. 1.1 наведено порівняння реляційних та нереляційних баз даних за даними джерела [4].

Отже, оскільки для розробки мого вебдодатку важлива структурованість даних, описи фільмів та серіалів містять кілька параметрів, а тому будуть наявні зв’язки, повинна забезпечуватись цілісність даних та важливими є фільтрація, пошук та сортування за різними критеріями, то доцільніше обрати реляційну тип, який дозволить виконати всі поставлені вимоги.

Таблиця 1.1 – Порівняння реляційних та нереляційних БД

Тип	Переваги	Недоліки
Реляційні	Дані мають чітку структуру, що дає змогу ефективно організовувати складні взаємозв'язки між ними.	За значного збільшення обсягу інформації швидкодія системи може знижуватися.
	Логічне представлення даних не залежить від способу їх фізичного розміщення на носіях.	Реалізація та відображення складних ієрархічних структур даних може бути досить складною.
	Підтримка мови SQL забезпечує зручне виконання запитів і операцій над даними.	
Нереляційні	Дані можуть зберігатися без заздалегідь визначеної структури, що дозволяє гнучко змінювати їх під час роботи системи.	У багатьох системах використовується принцип остаточної узгодженості, через що можливі тимчасові розбіжності між копіями даних.
	Відсутність складних зв'язків між записами сприяє швидшому виконанню операцій запису та пошуку.	Забезпечення цілісності даних під час одночасного виконання багатьох операцій є складнішим, а у разі збоїв існує ризик втрати частини інформації.
	Існує можливість обрати тип бази даних відповідно до конкретних потреб проєкту.	Виконання складних аналітичних запитів із об'єднанням даних часто є обмеженим або потребує додаткових механізмів реалізації.

1.2.1 Огляд існуючих СУБД

Оскільки було обрано доцільним використання реляційного типу, то в цьому підрозділі будуть розглянуті та порівняні дві найпопулярніші системи: MySQL та PostgreSQL.

PostgreSQL – це об'єктно-реляційна система управління базами даних з відкритим вихідним кодом, яка поєднує можливості реляційних баз даних та принципи об'єктно-орієнтованого програмування, надаючи можливість розширення стандартних типів даних і роботи з різними базами даних [14]. Вона активно використовується у багатьох сферах: у веброзробці – для створення динамічних вебсайтів і застосунків, в аналітиці даних – для ефективної обробки великих обсягів інформації, а у масштабних системах,

зокрема соціальних мережах, цінується за високу надійність і можливість масштабування [15].

MySQL – це реляційна система управління базами даних з відкритим кодом, побудована на основі клієнт-серверної архітектури. Серверна частина функціонує безперервно на комп’ютері, тоді як клієнтські застосунки надсилають SQL-запити для взаємодії з базою даних. Після обробки запитів сервер повертає клієнту відповідний результат, забезпечуючи обмін даними між процесами [16].

У таблиці 1.2 наведено порівняння СУБД за певними критеріями, взятими з джерел [14], [15] та [16].

Таблиця 1.2 – Порівняння MySQL та PostgreSQL

Категорія	**MySQL**	**PostgreSQL**
Комерційне використання	Безкоштовне, якщо не інтегрується в закритий продукт.	Безкоштовно для будь-яких цілей.
Простота використання	Проста для новачків.	Більш складна у налаштуванні.
Запис та оновлення	Вища швидкість запису потоків даних.	Обробляє великі обсяги записів з підвищеною надійністю.
Операції читання	Працює швидше завдяки оптимізації рушія InnoDB під прості вибірки.	Поступається MySQL, але має власні інструменти оптимізації кешування.
Підтримка складних SQL-запитів	Має слабшу підтримку складних запитів.	Має більш потужний аналізатор запитів.
Продуктивність	Використовується для вебпроєктів, яким потрібна база даних для простої обробки даних.	Широко використовується у великих системах для аналітики.
Обробка з’єднань	Обробляє кожне з'єднання через окремий потік операційної системи, що вимагає менше оперативної пам'яті за умови великої кількості одночасних підключень.	Виділяє окремий процес під кожне з'єднання, що забезпечує кращу стабільність та ізоляцію, але потребує більше ресурсів
Масштабованість	Швидкість на операціях читання.	Підтримка складних запитів, великої кількості одночасних записів та аналітики.

Отже, оскільки для вебдодатку для перегляду інформації про фільми та серіали важливою є швидкість обробки запитів, швидке кешування запитів, він буде виконувати здебільшого прості операції читання та не потребує складні математичні обчислення, то доцільним є вибір СУБД MySQL.

1.3 Висновок до першого розділу

У першому розділі було розглянуто основні поняття веброзробки та переглянуто технології розробки бекенду і фронтенду з видами СУБД – всі ці методи в подальшому будуть використані при реалізації мого вебдодатку. В результаті аналізу та порівняння існуючих СУБД було обрано доцільним використати MySQL, яка дозволить швидко обробляти запити без потреби в складних математичних обчисленнях.

Отже, веброзробка є комплексною задачею, яка повинна передбачати використання різних сучасних методів для створення повноцінного робочого продукту. Вона включає в себе розробку клієнтської та серверної частин, організацію коректної взаємодії між ними, а також проєктування з подальшим впровадженням бази даних для управління даними.

2 ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ ВИБОРУ ТЕХНОЛОГІЇ РЕАЛІЗАЦІЇ ВЕБДОДАТКУ

Оскільки в попередньому розділі були розглянуті основні поняття веброзробки, то в другому розділі будуть проаналізовані та порівняні сучасні засоби реалізації бекенду та фронтенду для вибору більш доцільних для реалізації поставленої задачі. Особливу увагу буде приділено дослідженню їхніх функціональних можливостей, переваг і недоліків, а також особливостям використання під час розробки вебдодатків. На основі проведеного аналізу буде здійснено вибір технологій, які найбільше відповідають вимогам поставленої задачі та забезпечать ефективну реалізацію програмного продукту.

2.1 Архітектура вебдодатку

Однією із загальноприйнятих моделей побудови вебдодатків, яка також буде застосована у даній розробці, є клієнт-серверна архітектура, яка складається з двох основних компонентів, що взаємодіють між собою згідно з джерелом [7]:

- клієнт – це програмне забезпечення на стороні користувача, наприклад веббраузер або мобільний застосунок, яке надсилає запити до сервера та забезпечує взаємодію користувача із системою;
- сервер – це програмне або апаратне забезпечення, призначене для обробки запитів клієнта, взаємодії з базами даних і надсилання готових результатів у відповідь.

Принцип роботи такої архітектури полягає в тому, що клієнт формує та надсилає запит серверу, після чого сервер обробляє його і повертає відповідь із результатом виконання. Один сервер може одночасно обслуговувати декілька клієнтів, а якщо надходить кілька запитів одночасно, то вони поміщаються у

чергу та виконуються відповідно до встановлених пріоритетів: запити з вищим пріоритетом обробляються першими [7].

Далі у таблиці 2.1 продемонстровано переваги та недоліки використання клієнт-серверної архітектури за даними джерела [7].

Таблиця 2.1 – Переваги та недоліки клієнт-серверної архітектури

Переваги	**Недоліки**
Підтримує функціонування масштабних комп'ютерних мереж.	Відмова центрального сервера може негативно вплинути на доступність мережевих ресурсів.
Дає змогу здійснювати єдине адміністрування користувачів і механізмів безпеки.	Для адміністрування необхідний кваліфікований персонал.
Забезпечує оперативний доступ до спільних ресурсів мережі.	Витрати на розгортання та підтримку мережі є відносно високими.
Передбачає використання єдиних облікових даних для роботи з мережевими сервісами.	

Для розробки мого вебдодатку було обрано використання саме цієї архітектури, бо вона допоможе вирішити критичні завдання, описані далі. По-перше, враховуючи, що в базі даних буде міститися велика кількість фільмів, серіалів та акторів, які не можуть зберігатися на пристрої користувача, є доцільною наявність централізованого зберігання, що гарантує цілісність даних та їхню доступність для всіх користувачів після оновлення бази даних. Такий підхід також спрощує адміністрування даних і дозволяє швидко вносити зміни без необхідності оновлення клієнтської частини додатка. По-друге, ця архітектура допоможе забезпечити взаємодію між користувачами: вони зможуть бачити коментарі та оцінки один одного в режимі реального часу через збереження запитів у базі даних та серверне опрацювання.

2.2 Обґрунтування вибору технологій

При виборі оптимальних рішень для створення вебдодатку важливо розуміти їхні переваги, недоліки, обмеження та відповідність потребам задачі, тому для цього у наступних підрозділах будуть розглянуті та порівняні різні вебтехнології для вибору найдоречніших.

2.2.1 Вибір способу реалізації серверної частини

Для вибору методу реалізації серверної частини проєкту для порівняння доцільно обрати такі найпоширеніші мови програмування: Python, JavaScript, та PHP.

Python – це об’єктно-орієнтована мова програмування загального призначення, яка широко застосовується в різних сферах, зокрема у бекенд-розробці, аналізі даних, машинному навчанні, тестуванні, автоматизації процесів та системному адмініструванні. Вона належить до найпопулярніших мов програмування у світі [5]. У таблиці 2.2 наведено переваги та недоліки використання цієї мови за даними джерела [5].

Таблиця 2.2 – Переваги і недоліки використання Python

Переваги	**Недоліки**
Має простий та інтуїтивно зрозумілий синтаксис, що полегшує процес навчання і розробки програм.	Швидкодія може бути нижчою порівняно з мовами, які використовують попередню компіляцію у машинний код.
Наявність великої кількості бібліотек і фреймворків дозволяє застосовувати мову для вирішення різноманітних завдань.	Реалізація ефективної багатопотокової обробки даних має певні обмеження.
Підтримує роботу на різних програмно-апаратних платформах та використовує динамічну типізацію даних.	Під час спроби доступу до ресурсів без необхідних дозволів можуть виникати помилки виконання.
Для вивчення доступна значна кількість документації, курсів і навчальних матеріалів.	Використання механізму автоматичного керування пам’яттю може призводити до додаткових витрат системних ресурсів і зниження продуктивності в окремих випадках.

Далі для аналізу розглядається JavaScript – скриптова мова програмування, яка активно використовується для створення фронтенду та бекенду, розробки односторінкових застосунків, браузерних ігор, а також для автоматизації різноманітних процесів [5]. У таблиці 2.3 наведено переваги та недоліки використання JavaScript за даними джерела [5].

Таблиця 2.3 – Переваги і недоліки використання JavaScript

Переваги	**Недоліки**
Застосовується для створення клієнтської та серверної частин.	Деякі конструкції можуть бути складними для новачків.
Дозволяє легко реалізовувати інтерактивні елементи вебсторінок завдяки інтеграції з HTML та CSS.	За неналежної реалізації механізмів захисту можливі вразливості до XSS-атак.
Відзначається кросплатформністю та широкою екосистемою додаткових інструментів.	Менша продуктивність в ресурсомістких завданнях в порівнянні з компільованими мовами.
Наявність асинхронної моделі виконання коду, що покращує швидкість обробки запитів.	
Наявність динамічної типізації.	

Наступною для порівняння є PHP – серверна мова програмування з відкритим вихідним кодом, що застосовується переважно для розробки динамічних вебсайтів і вебзастосунків [5]. У таблиці 2.4 наведено переваги та недоліки використання цієї мови за даними джерела [5].

Таблиця 2.4 – Переваги і недоліки використання PHP

Переваги	**Недоліки**
Широкий спектр готових фреймворків для пришвидшення розробки.	Різні бібліотеки можуть мати відмінні назви функцій та порядок передавання параметрів, що ускладнює їх використання.
Підтримує роботу на більшості сучасних операційних систем.	Низький поріг входження може призводити до створення коду з потенційними вразливостями.
Сучасні версії мови демонструють підвищений рівень продуктивності виконання програм.	Характеризується слабкою типізацією, що може ускладнювати виявлення помилок на етапі розробки.

Отже, проаналізувавши переваги та недоліки декількох мов програмування, для реалізації проєкту доцільно обрати JavaScript, оскільки це дозволить забезпечити можливість використання єдиного середовища розробки як на стороні клієнта, так і на стороні сервера, що полегшить процес розробки, пришвидшить виконання завдання і зробить цю технологію практичним і доцільним вибором для реалізації вебзастосунку.

2.2.2 Виріб способу реалізації клієнтської частини

Розробка клієнтської частини вебзастосунку є важливим етапом, оскільки на користувацький інтерфейс, а саме його вигляд та зручність, в першу чергу звертає увагу користувач при першому візиті на сайт. На цьому етапі визначається зручність навігації, доступність контенту та загальне враження від ресурсу, оскільки інтуїтивно зрозумілий та естетично привабливий інтерфейс формує гарний досвід користувачу.

Базовими технологіями для реалізації фронтенду є HTML, CSS та JavaScript, які будуть використані в даному проєкті.

HTML (Hypertext Markup Language) – це мова розмітки, яка визначає структуру вебсторінки та описує її елементи, такі як абзаци, кнопки, заголовки, таблиці та інші компоненти, що відображаються у браузері. CSS (Cascading Style Sheets) використовується для оформлення зовнішнього вигляду вебсторінок, роблячи їх більш привабливими та зручними для сприйняття користувачем. Термін «каскадні» означає, що стилі можуть задаватися з різних джерел і рівнів пріоритету, а остаточний вигляд елементів формується на основі правил специфічності та механізму наслідування від батьківських елементів. JavaScript, яка вже була розглянута у попередньому підрозділі, додає вебсторінкам інтерактивності, дозволяючи реалізовувати функціональні можливості, що забезпечують взаємодію користувача з інтерфейсом у режимі реального часу [6].

Отже, застосування HTML, CSS та JavaScript є важливим етапом для розробки мого проєкту, бо вони є фундаментом для розробки користувацької частини: HTML виділить всі елементи сторінки та забезпечить відображення різних елементів, CSS додасть візуальну привабливість, а JavaScript дозволить додати додатку інтерактивності.

2.3 Проєктування серверної частини

Проєктування серверної частини вебдодатку включає в себе налаштування внутрішньої логіки, яка повинна коректно працювати на сервері, обробляти запити користувачів, організовувати взаємодію з базою даних та забезпечувати безпеку даних.

Оскільки в попередньому підрозділі було обрано доцільним використання JavaScript, то для розробки серверної частини буде ефективним використання таких популярних технологій, як Node.js та Express.

Node.js – це середовище виконання JavaScript, яке дозволяє запускати код поза межами браузера. Його головною особливістю є асинхронна, подієво-орієнтована архітектура: замість блокування виконання програми під час очікування відповіді від бази даних використовуються неблокувальні операції, що дає змогу одночасно обробляти велику кількість запитів. Додатковою перевагою є вбудований менеджер пакетів npm – один із найбільших репозиторіїв програмного забезпечення, який забезпечує доступ до великої кількості відкритих бібліотек і спрощує процес розробки [8].

Node.js також ефективно поєднується з фреймворком Express.js – легким інструментом для створення веб- та мобільних застосунків, побудованим на базі стандартних можливостей Node.js. Він надає розширений набір функцій для розробки вебдодатків, зокрема систему маршрутизації для обробки HTTP-запитів (GET, POST, PUT, DELETE), підтримку роботи з базами даних (MongoDB, PostgreSQL, MySQL), а також інструменти для тестування, обробки

помилок і використання проміжного програмного забезпечення (middleware). Express.js спрощує організацію серверного коду, забезпечує модульність структури проєкту та дозволяє швидко реалізовувати REST API для взаємодії між клієнтською та серверною частинами вебдодатку. Завдяки своїй простоті, високій продуктивності та великій кількості готових модулів він є одним із найпопулярніших фреймворків для розробки серверної частини сучасних вебзастосунків [9].

Отже, ці технології будуть корисними для розробки мого вебдодатку завдяки використанню асинхронної архітектури, яка допоможе серверу обробляти безліч одночасних запитів без необхідності очікування завершення минулих операцій, а Express.js надасть прості та ефективні механізми обробки різних HTTP-методів, які створять зрозумілу структуру взаємодії між клієнтом та сервером.

2.3.1 Забезпечення безпеки серверної частини

Одним з головних етапів розробки серверної частини є забезпечення безпеки вебдодатку від різних зловмисних атак, оскільки недотримання стандартів безпеки може призвести до витоку даних та конфіденційної інформації користувачів. Далі будуть розглянуті найпоширеніші типи атак, захист від яких буде варто врахувати при розробці мого вебдодатку: SQL-ін’єкції, міжсайтовий скриптинг (XSS-атака) та міжсайтова підробка запиту (CSRF-атака) [13].

SQL-ін’єкція – це вид атаки, при якій зловмисник вводить шкідливі SQL-запитів (наприклад, ` OR 1=1 --) через поля вводу даних для маніпуляції базою даних. Для захисту слід застосовувати параметризовані запити, в яких введені користувачем запити не розпізнаються як код, екранування спецсимволів (‘, `, “, \ тощо), валідацію вхідних даних та застосування ORM-систем, які автоматично використовуються безпечні запити [13].

XSS-атака – введення зловмисником шкідливого коду у вебсторінку, який потім виконується у браузері клієнта і дозволяє перехоплювати акаунти, перенаправляти на фішингові сайти чи змінювати контент сторінки. Найважливішим способом їхнього запобігання є екранування вхідних даних – метод кодування всіх даних, отриманих від користувача, щоб браузер інтерпретував їх як текст, а не виконуваний код [13].

Міжсайтова підробка запиту (CSRF-атака) – ця атака використовує довіру вебдодатка до браузера користувача, змушуючи його надсилати шкідливі запити без відома власника облікового запису. Захист включає використання токенів запитів (CSRF-токени), які є випадково згенерованими рядками, і перевірку Referer-заголовка – сервер перевіряє з якої сторінки користувач перейшов на поточну [13].

Корисними практиками для запобігання великої кількості атак є валідація вхідних даних, завдяки якій перевіряється правильність введених користувачем даних (довжина, тип, допустимі символи тощо), обмеження прав доступу та використання хешування паролів за допомогою спеціальних алгоритмів, оскільки у такому випадку навіть при витоку даних зловмисник не зможе отримати справжні паролі користувачів [13].

Отже, при розробці вебдодатку варто враховувати заходи безпеки від різних кібератак: SQL-ін’єкцій, XSS-атак та CSRF-атак, які призводять до втрати даних користувачів, тому для їх профілактики варто використовувати параметризовані запити, екранування, валідацію вхідних даних та захищене зберігання паролів для мінімізації ризику несанкціонованого доступу.

2.4 Організація взаємодії клієнтської та серверної частин

Взаємодія між клієнтською та серверною частинами вебдодатку реалізується через HTTP-протокол із використанням REST API (Representational State Transfer API) – архітектурного підходу, що визначає набір правил обміну

даними між системами через інтернет за допомогою стандартних HTTP-методів, що дозволяє клієнту надсилати запити до сервера та отримувати відповідні відповіді. Процес взаємодії можна описати наступним чином згідно з джерелом [10]:

- користувач виконує дію на вебсторінці, після чого JavaScript формує HTTP-запит і надсилає його на сервер;
- сервер отримує запит і, використовуючи REST API, обробляє його та виконує необхідну операцію;
- після обробки сервер формує HTTP-відповідь (наприклад, зі статусом виконання або оновленими даними) і повертає її клієнту;
- на стороні клієнта JavaScript опрацьовує отриману відповідь і за потреби оновлює інтерфейс користувача.

Залежно від типу операції, що виконується, застосовуються різні HTTP-методи: GET, POST, PUT та DELETE. Їх призначення, згідно з джерелом [10], є наступним:

- GET використовується для отримання даних із сервера без їхньої зміни;
- POST призначений для надсилання даних на сервер з метою створення нових ресурсів або їх обробки;
- PUT застосовується для оновлення вже наявних даних;
- DELETE використовується для видалення даних.

Можна виділити низку переваг використання REST API згідно з джерелом [10]:

- простота використання завдяки стандартним HTTP-методам і URL-адресам;
- сумісність із більшістю сучасних платформ;
- підтримка різних форматів даних, зокрема JSON і XML;
- можливість кешування, що підвищує продуктивність і зменшує навантаження на сервер.

Водночас, попри переваги, існують і певні недоліки цієї технології. Зокрема, у великих проєктах із великою кількістю розробників може виникати складність у підтриманні єдиних стандартів. Також, попри загальну безпеку підходу, можливі вразливості, наприклад надмірна кількість запитів від клієнта може призвести до перевантаження сервера. Крім того, іноді серверні відповіді містять зайві дані або потребують додаткових запитів для отримання повної інформації [11].

Отже, завдяки використанню архітектури REST API в розробці мого вебдодатку можна реалізувати чітке розділення фронтенду та бекенду, що дозволить змінювати будь-яку частину незалежно одна від одної. Використання стандартних HTTP-методів створює передбачуваність системи та значно спрощує розробку, оскільки я зможу перевірити роботу сервера через звичайні браузери.

2.5 Проєктування бази даних

Процес проєктування бази даних складається з аналізу функціональних вимог проєкту, виділення основних сутностей, атрибутів та зв'язків між ними, далі виконується нормалізація, яка спрямована на зменшення дублювання, та обмеження цілісності даних – призначення кожній таблиці унікального первинного ключа та обмеження значень для атрибутів [12].

На початковому етапі після аналізу функціональних вимог створюється діаграма прецедентів (Use Case), яка буде визначати, що повинна робити система, хто буде нею користуватися та які дії вони можуть виконувати. На рис. 2.1 продемонстровано Use Case діаграму для мого завдання.

Перед розробкою схеми даних в межах реляційної моделі потрібно виділити ключові об’єкти (сутності) та їхні властивості (атрибути) [12], які відповідають предметній області. У таблиці 2.5 продемонстровано основні сутності та їхні атрибути з описом, які відповідають моїй предметній області.

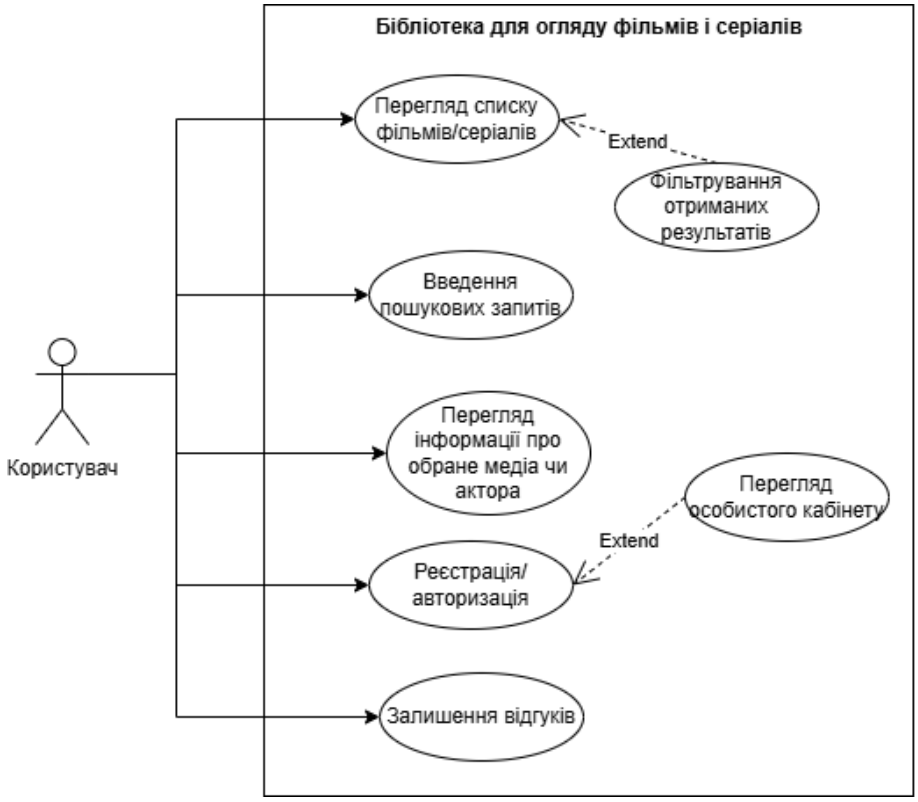


Рисунок 2.1 – Use Case діаграма

Таблиця 2.5 – Таблиця сутностей та атрибутів

Сутність	**Опис**	**Атрибути**
Фільми	Інформація про фільми, яка містить певні атрибути, які можуть знадобитися користувачу.	ID, назва українською, назва англійською, опис, країна, рік випуску, тривалість, рейтинг, актори, постер, трейлер.
Серіали	Інформація про серіали, яка містить певні атрибути, які можуть знадобитися користувачу.	ID, назва українською, назва англійською, опис, країна, рік випуску, к-ть сезонів, к-ть епізодів, рейтинг, актори, постер, трейлер.
Актори	Інформація про акторів, які грали у фільмах/серіалах.	ID, ім’я, дата народження, фото.
Користувачі	Список зареєстрованих користувачів.	ID, логін, пароль, час створення.
Відгуки	Список коментарів, які будуть відображатися під медіа для всіх користувачів.	ID, ID-користувача, ID-фільму, ID-серіалу, гостьове ім’я, виставлений рейтинг, текст відгуку, час створення.

Наступний етап включає в себе створення семантичної моделі «Сутність-зв’язок» (ER-діаграми), в якій обираються ключові об’єкти, необхідні для

функціоналу вебдодатку, та візуалізується структура БД: визначення типів зв’язків між сутностями (один-до-одного, багато-до-багатьох, один-до-багатьох, багато-до-багатьох), виділення первинних та зовнішніх ключів, нормалізація відношень [12].

Нормалізація відношень виконується для забезпечення відсутності дублювань, що досягається шляхом декомпозиції у такий спосіб, щоб у відношенні зберігалися лише первинні факти [12]. Для виконання мого проєкту буде достатньо привести дані до третьої нормальної форми, оскільки подальша нормалізація може супроводжуватися уповільненням виконання запитів до бази даних.

Для приведення даних до 1НФ потрібно, щоб в будь-якому допустимому значенні відношення кожен його рядок містив лише одне значення для кожного з атрибутів [12].

Відношення знаходиться у 2НФ, якщо воно знаходиться у 1НФ та кожен неключовий атрибут повністю функціонально залежить від первинного ключа (ПК) [12].

Для належності до 3НФ відношення повинно знаходитися у 2НФ, та кожен неключовий атрибут відношення має функціонально залежати тільки від первинного ключа [12].

На рис. 2.2 наведено нормалізовану ER-діаграму моєї бази даних.

2.6 Висновок до другого розділу

У другому розділі було обґрунтовано вибір технологій та детальніше розглянуто методи веброзробки: Node.js, Express, клієнт-серверну архітектуру та процес проєктування бази даних. Завдяки проведеним аналізам найпоширеніших технологій було обрано найдоцільніші засоби веброзробки, що будуть використані для програмної реалізації мого вебдодатку. Використання Node.js та Express.js дозволить реалізувати асинхронну обробку

запитів, що є критично важливим для продуктивності серверної частини. Клієнт-серверна архітектура у поєднанні з REST API забезпечить чітку модульність проєкту, що дозволить відокремити інтерфейс користувача від складної логіки сервера, та гарантує універсальний обмін даними, а спроєктована структура бази даних, доведена до третьої нормальної форми, забезпечить їхню цілісність.

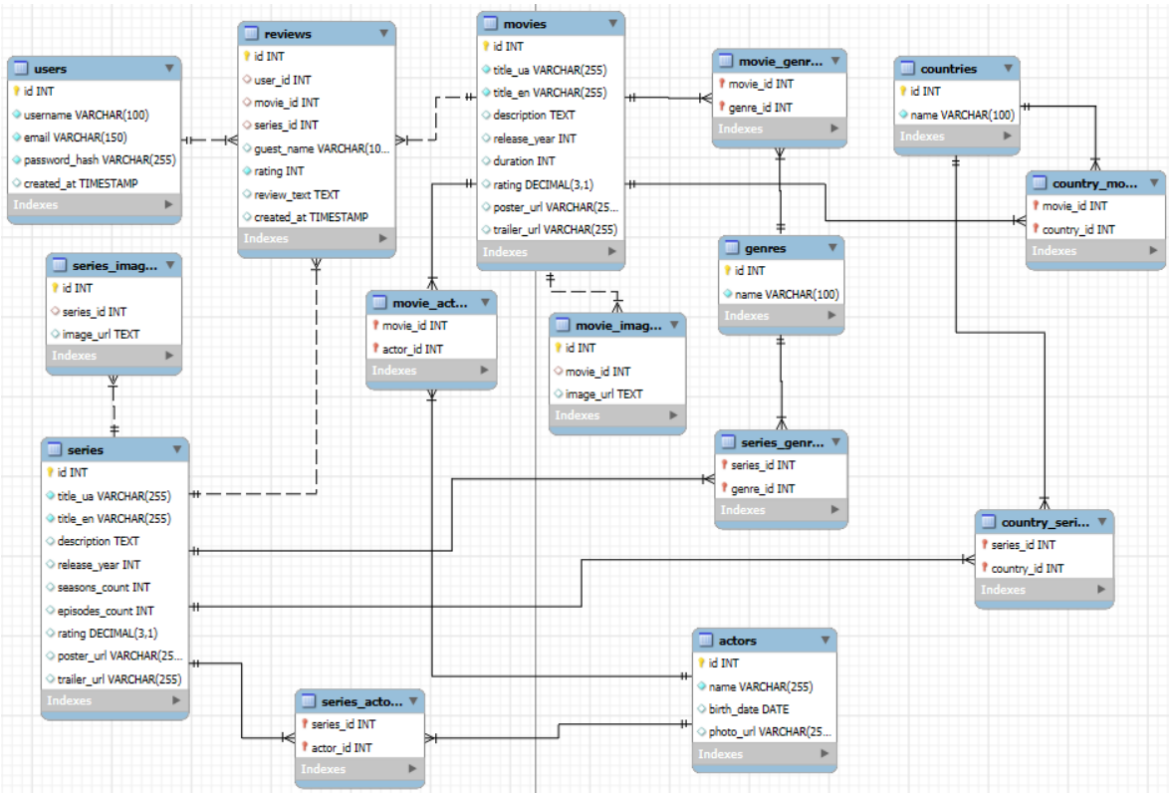


Рисунок 2.2 – ER-діаграма

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБДОДАТКУ

У цьому розділі детально описується програмна реалізація мого односторінкового вебдодатку – він завантажується один раз і динамічно оновлює контент без повного перезавантаження сторінки [2].

У серверній частині реалізовано REST API для отримання списку даних та їх детальної інформації, реєстрації, авторизації, залишення відгуків та отримання даних особистого кабінету. Для роботи з базою даних реалізовано асинхронні та параметризовані запити, фільтрація, пагінація через LIMIT і OFFSET та групування даних за допомогою методу GROUP_CONCAT.

У користувацькій частині реалізовано динамічне завантаження даних за допомогою Fetch API, асинхронну обробку запитів, пагінацію результатів, фільтрацію за різними факторами, пошук даних, горизонтальні слайдери для фільмів/серіалів/кадрів, галерею кадрів з їх відкриттям у спливаючому вікні поверх сторінки, форму реєстрації/авторизації та залишення відгуків. Для реалізації маршрутизації використовується location.hash, завдяки чому виконується динамічний перехід між сторінками фільмів, серіалів, акторів та особистого кабінету. Для відображення даних використовуються методи DOM API та генерація HTML через innerHTML.

3.1 Створення бази даних

Для створення бази даних у СУБД MySQL було використано команду CREATE DATABASE, використання якої показано на рис. 3.1.

```
CREATE DATABASE movie_app;
```

Рисунок 3.1 – Створення бази даних

Для створення таблиць було використано команду CREATE TABLE (рис. 3.2).

```
CREATE TABLE movies (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title_ua VARCHAR(255) NOT NULL,
    title_en VARCHAR(255) NOT NULL,
    description TEXT,
    release_year INT,
    duration INT,
    rating DECIMAL(3,1),
    poster_url VARCHAR(500),
    trailer_url VARCHAR(500)
);
```

Рисунок 3.2 – Створення таблиці «movies»

Для заповнення інформації в таблицях використовувалася команда INSERT INTO назва_таблиці () VALUES, що продемонстровано на рис. 3.3 на прикладі таблиці з жанрами.

```
INSERT INTO genres (name)
VALUES
('Драма'), ('Кримінал'), ('Трилер'), ('Екшн'), ('Історія дорослішання'),
('Комедія'), ('Романтика'), ('Виживання'), ('Підліткова драма'),
('Наукова фантастика'), ('Жахи'), ('Пригоди'), ('На основі реальних подій'),
('Трагедія'), ('Психологічний трилер'), ('Фантастика'), ('Фільм-катастрофа'),
('Містика'), ('Спорт'), ('Історичний'), ('Військовий'), ('Ігри на виживання');
```

Рисунок 3.3 – Заповнення таблиці «genres»

Кінцевий вигляд всіх таблиць в базі даних показано на рис. 3.4.



Рисунок 3.4 – Список таблиць в базі даних

3.2 Реалізація серверної частини

Для ініціалізації проєкту було використано команду npm -init -y, яка автоматично створює файл package.json з налаштуваннями за замовчуванням та (рис. 3.5).

```
PS D:\универ\4 курс\2 семестр\кваліфікаційна робота бакалавра\app> npm init -y
Wrote to D:\универ\4 курс\2 семестр\кваліфікаційна робота бакалавра\app\package.json:

{
  "name": "app",
  "version": "1.0.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": ""
}
```

Рисунок 3.5 – Ініціалізація проєкту

Далі для підключення всіх необхідних пакетів у папку проєкту було використано команду npm install назва_пакету, що показано на рис. 3.6 на прикладі завантаження Express.

```
PS D:\универ\4 курс\2 семестр\кваліфікаційна робота бакалавра\app> npm install express

added 65 packages, and audited 66 packages in 2s

22 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Рисунок 3.6 – Завантаження пакету Express

Серверна частина реалізована за допомогою Node.js та фреймворку Express. У ній спочатку ініціалізується порт для запуску сервера та підключаються необхідні бібліотеки (рис. 3.7).

- express.js для обробки HTTP-запитів та маршрутизації;
- express-session.js для створення сесій та підтримки авторизації користувача;

– db.js для роботи з базою даних;
– bcrypt.js для хешування паролів та їх перевірки під час авторизації;
– sanitize-html для очищення HTML-коду від потенційного небезпечного вмісту;
– csurf для захисту від CSRF-атак;
– express-validator для валідації та очищення даних у додатку.

```
const express = require('express');
const session = require('express-session');
const db = require('./db');
const bcrypt = require('bcrypt');
const sanitizeHtml = require('sanitize-html');
const csrf = require('csurf');
const { body, validationResult } = require('express-validator');
const app = express();
const PORT = 3000;
```

Рисунок 3.7 – Підключення бібліотек

Використання express.json() дозволить автоматично перетворити JSON із body HTTP-запитів у JavaScript-об'єкти, session() використовується для вказання параметрів сесії сервера, express.static() надасть браузеру доступ до статичних файлів, а інші методи застосовуються для впровадження захисту додатку.

3.2.1 Забезпечення безпеки серверної частини

На наступних рисунках продемонстровано приклад застосування захисту від CSRF-атак за допомогою використання бібліотеки csurf. Спочатку генерується унікальний токен для поточної сесії (рис. 3.8), який надалі буде використовуватись для POST-маршрутів (рис. 3.9).

На рис. 3.10 показано застосування параметризованих запитів для захисту від SQL-ін’єкцій на прикладі отримання списку жанрів для фільму за певним id.

На рис. 3.11 продемонстровано застосування функції з бібліотеки sanitize-html для захисту від XSS-атаки на прикладі очищення відгуку перед його збереженням в базу.

```
const csrfProtection = csrf();
app.use((req, res, next) => {
    res.locals.csrfToken = req.csrfToken?.();
    next();
});
app.get('/csrf-token', csrfProtection, (req, res) => {
    res.json({ csrfToken: req.csrfToken() });
});
app.use(express.static('public'));
app.use((req, res, next) => {
    res.locals.csrfToken = req.csrfToken?.();
    next();
});
```

Рисунок 3.8 – Генерація токену для поточної сесії

```
app.post('/movies/:id/reviews', csrfProtection, async (req, res) => {
```

Рисунок 3.9 – Використання csrfProtection для POST-запитів

```
const [genres] = await db.query(`
    SELECT g.name FROM genres g
    JOIN movie_genres mg ON g.id = mg.genre_id
    WHERE mg.movie_id = ?
`, [id]);
```

Рисунок 3.10 – Параметризовний запит

```
const clean = sanitizeHtml(review_text, {
    allowedTags: [],
    allowedAttributes: {}
});
```

Рисунок 3.11 – Очищення HTML

На рис. 3.12 продемонстровано хешування паролю за допомогою алгоритму bcrypt з кількістю раундів 10.

```
const hash = await bcrypt.hash(password, 10);
```

Рисунок 3.12 – Хешування паролів

На рис. 3.13 показано використання функцій з бібліотеки express-validator на приладі валідації даних під час реєстрації користувача: перевіряється довжина імені та паролю, формат пошти та виконується очищення від пробілів та нормалізація пошти.

```
app.post('/register', csrfProtection,
    [
        body('username').trim().isLength({ min: 3, max: 10 }).withMessage('Username має бути від 3 до 10 символів'),
        body('email').isEmail().withMessage('Некоректний email').normalizeEmail(),
        body('password').isLength({ min: 6 }).withMessage('Довжина паролю мінімум 6 символів')
    ],
```

Рисунок 3.13 – Валідація даних

3.2.2 Реалізація маршрутизації REST API

Маршрути було реалізовано відповідно до REST-підходу, що забезпечує розділення функціональності за ресурсами:

- GET /movies – виведення списку всіх фільмів;
- GET /movies/:id – отримання детальної інформації про конкретний фільм;
- GET /movies/:id/actors – отримання списку акторів, які брали участь у фільмі;
- GET /movies/:id/images – отримання галереї кадрів для фільму;
- GET /movies/:id/reviews – отримання списку відгуків до фільму;
- POST /movies/:id/reviews – додавання відгуку до фільму;
- GET /series – отримання списку серіалів;
- GET /series/:id – отримання інформації про конкретний серіал;
- GET /series/:id/actors – отримання списку акторів серіалу;
- GET /series/:id/images – отримання кадрів серіалу;
- GET /series/:id/reviews – отримання списку відгуків до серіалу;
- POST /series/:id/reviews – додавання відгуку до серіалу;
- GET /actors/:id – отримання інформації про актора;
- POST /register – реєстрація користувача;

– POST /login – авторизація користувача;

– POST /logout – вихід користувача з системи;

– GET /me – перегляд особистого кабінету користувача;

– GET /me/reviews – перегляд списку власних відгуків користувача.

На рис. 3.14 продемонстровано використання методу GET для отримання деталей про актора: імені, фото, рік народження, списку фільмів та серіалів, які потім формуються у JSON-об’єкт для відображення інформації.

```
app.get('/actors/:id', async (req, res) => {
    try {
    const id = req.params.id;

    const [[actor]] = await db.query(
        'SELECT * FROM actors WHERE id=?',
        [id]
    );

    const [movies] = await db.query(`
        SELECT m.id, m.title_ua, m.poster_url, m.release_year
        FROM movies m
        JOIN movie_actors ma ON m.id = ma.movie_id
        WHERE ma.actor_id = ?
    `, [id]);

    const [series] = await db.query(`
        SELECT s.id, s.title_ua, s.poster_url, s.release_year
        FROM series s
        JOIN series_actors sa ON s.id = sa.series_id
        WHERE sa.actor_id = ?
    `, [id]);

    res.json({ ...actor, movies, series });
} catch (err) {
        res.status(500).json({ message: err.message });
    }
});
```

Рисунок 3.14 – Приклад використання методу GET

На рис. 3.15 продемонстровано фрагмент використання методу POST для додавання відгуку до фільму: спочатку отримуються дані відгуку та перевіряється текст коментаря, надалі визначається, чи авторизований користувач, і створюється новий запис у таблиці reviews.

```
app.post('/movies/:id/reviews', csrfProtection, async (req, res) => {
    try {
        const movieId = req.params.id;
        const { guest_name, rating, review_text} = req.body;
        const clean = sanitizeHtml(review_text, {
            allowedTags: [],
            allowedAttributes: {}
        });
        const userId = req.session.user?.id || null;
        if (!clean?.trim()) {
            return res.status(400).json({
                message: 'Введіть текст відгуку'
            });
        }
        await db.query(`
            INSERT INTO reviews(
                user_id,
                movie_id,
                guest_name,
                rating,
                review_text
            )
            VALUES (?, ?, ?, ?, ?)
        `, [
            userId,
            movieId,
            guest_name || null,
            rating,
            clean
        ]);

        res.json({
            success: true
        });
```

Рисунок 3.15 – Приклад використання методу POST

3.2.3 Реалізація реєстрації та авторизації користувачів

На рис. 3.16 продемонстровано основний фрагмент методу реєстрації нового користувача: спочатку перевіряється заповнення полів, їхня відповідність умовам та чи вже існує цей користувач в базі даних, далі пароль хешується та створюється запис у відповідну таблицю, користувач автоматично авторизується після реєстрації та створюється сесія. Спосіб авторизації відбувається за схожим принципом (рис. 3.17), але там порівнюються отримані email та хеш паролю з наявними значеннями в базі даних.

```
app.post('/register', csrfProtection,
    [
        body('username').trim().isLength({ min: 3, max: 10 }).withMessage('Username має бути від 3 до 10 символів'),
        body('email').isEmail().withMessage('Некоректний email').normalizeEmail(),
        body('password').isLength({ min: 6 }).withMessage('Довжина паролю мінімум 6 символів')
    ],
    async (req, res) => {
    try {
        const errors = validationResult(req);
        if (!errors.isEmpty()) {
            return res.status(400).json({
                message: errors.array()[0].msg
            });
        }
        const { username, email, password } = req.body;
        const [exists] = await db.query('SELECT id FROM users WHERE email=?', [email] );
        if (exists.length) {
            return res.status(400).json({
                message: 'Користувач вже існує'
            });
        }
        const hash = await bcrypt.hash(password, 10);

        const [result] = await db.query(`
            INSERT INTO users(username, email, password_hash)
            VALUES (?, ?, ?)
        `, [username, email, hash]);

        req.session.user = {
            id: result.insertId,
            username
        };

        res.json({
            success: true,
            user: req.session.user
        });
```

Рисунок 3.16 – Реєстрація користувача

```
app.post('/login', csrfProtection,
    [
        body('email').isEmail().withMessage('Некоректний email').normalizeEmail(),
        body('password').notEmpty().withMessage('Пароль не може бути пустим')
    ],
    async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
            return res.status(400).json({
                message: errors.array()[0].msg
            });
        }
    try {
        const { email, password } = req.body;
        const [[user]] = await db.query('SELECT * FROM users WHERE email=?', [email] );
        if (!user) {
            return res.status(400).json({
                message: 'Користувача не знайдено'
            });
        }
        const valid = await bcrypt.compare( password, user.password_hash);
        if (!valid) {
            return res.status(400).json({
                message: 'Невірний пароль'
            });
        }
        req.session.user = {
            id: user.id,
            username: user.username
        };
        res.json({
            success: true,
            user: req.session.user
        });
```

Рисунок 3.17 – Авторизація користувача

На рис. 3.18 наведено маршрут отримання інформації про поточного авторизованого користувача та маршрут отримання відгуків: перевіряється сесія та, якщо користувач авторизований, повертається його ім’я та список відгуків, який сортується за датою.

```
app.get('/me', (req, res) => {
    res.json(req.session.user || null);
});

app.get('/me/reviews', async (req, res) => {
    try{
        if (!req.session.user) {
            return res.json([]);
        }

        const userId = req.session.user.id;
        const [rows] = await db.query(`
            SELECT
                r.id,
                r.rating,
                r.review_text,
                r.created_at,

                m.id AS movie_id,
                m.title_ua AS movie_title,

                s.id AS series_id,
                s.title_ua AS series_title
            FROM reviews r
            LEFT JOIN movies m ON r.movie_id = m.id
            LEFT JOIN series s ON r.series_id = s.id
            WHERE r.user_id = ?
            ORDER BY r.created_at DESC
        `, [userId]);

        res.json(rows);
    } catch (err) {
        res.status(500).json({
            message: err.message
        });
    }
});
```

Рисунок 3.18 – Маршрут отримання інформації про користувача та відгуків

На рис. 3.19 продемонстровано маршрут для виходу користувача з особистого кабінету: видаляється поточна сесія, завершається авторизація та очищаються дані користувача.

```
app.post('/logout', csrfProtection, (req, res) => {
    req.session.destroy(err => {
        if (err) {
            return res.status(500).json({ message: 'Помилка виходу з акаунту.' });
        }

        res.clearCookie('connect.sid');
        res.json({ success: true });
    });
});
```

Рисунок 3.19 – Маршрут виходу з акаунту

3.3 Реалізація клієнтської частини

Спочатку отримуються основні HTML-елементи сторінки для подальшої роботи з інтерфейсом (рис. 3.20). Через ці змінні здійснюються виведення списків фільмів та серіалів, робота з пагінацією, пошук показ та приховування фільтрів, навігація по сторінках застосунку.

```
const listContainer = document.getElementById('listContainer');
const paginationContainer = document.getElementById('pagination');
const searchBtn = document.getElementById('searchBtn');
const searchInput = document.getElementById('searchInput');
const filters = document.querySelector('.filters');
const nav = document.querySelector('nav');
```

Рисунок 3.20 – Ініціалізація змінних

3.3.1 Реалізація захисту

На рис. 3.21 продемонстровано спосіб автоматичного завантаження актуального токену через loadCsrf() і підстановка його в заголовок csrf-token при кожному POST-запиті.

На рис. 3.22 продемонстровано функцію escapeHtml() для перетворення спеціальних символів на безпечні HTML-сутності, яка використовується при рендерингу карток, деталей фільмів чи відгуків.

```
let csrfToken = '';

async function loadCsrf() {
    const res = await fetch('/csrf-token');
    const data = await res.json();
    csrfToken = data.csrfToken;
}

loadCsrf();
```

Рисунок 3.21 – Передача CSRF токена

```
function escapeHtml(text = '') {
    return String(text)
        .replaceAll('&', '&amp;')
        .replaceAll('<', '&lt;')
        .replaceAll('>', '&gt;')
        .replaceAll('"', '&quot;')
        .replaceAll("'", '&#039;');
}
```

Рисунок 3.22 – Функція escapeHtml()

Функція safeUrl() (рис. 3.23) перевіряє посилання перед тим, як вставити їх в атрибути src тегів <img> та <iframe>, що захищає від атаки, коли замість посилання на картинку зловмисник може вставити конструкцію типу javascript:alert('XSS').

```
function safeUrl(url = '') {
    return /^(https?:\/\/|\/)/.test(url) ? url : '';
}
```

Рисунок 3.23 – Функція safeUrl()

3.3.2 Реалізація маршрутизації

Функція parseHash() (рис. 3.24) реалізує механізм маршрутизації без перезавантаження сторінки. Вона повертає об'єкт маршруту з інформацією про тип сторінки та медіа (фільм/серіал), id елемента та номер сторінки пагінації.

```
function parseHash() {
    const hash = window.location.hash.slice(1);
    if (!hash) return { page: 'list', type: 'all', pageNum: 1 };
    if (hash.includes('-') && !hash.includes('?')) {
        const [type, id] = hash.split('-');
        if (type === 'movies' || type === 'series') {
            return { page: 'details', type, id };
        }
        if (type === 'actor') {
            return { page: 'actor', id };
        }
    }
    if (hash === 'profile') {
        return { page: 'profile' };
    }
    if (hash.includes('?')) {
        const [type, query] = hash.split('?');
        const params = new URLSearchParams(query);
        return {
            page: 'list',
            type: type || 'all',
            pageNum: Number(params.get('page')) || 1
        };
    }
    return { page: 'list', type: hash, pageNum: 1 };
}
```

Рисунок 3.24 – Функція parseHash()

Функція handleRoute() (рис. 3.25) є головним маршрутизатором і відповідає за відкриття списків фільмів/серіалів, сторінки з деталями, сторінки акторів, особистого кабінету та автоматичне прокручування сторінки вгору при навігації.

3.3.3 Реалізація функцій отримання даних

Функція renderList() (рис. 3.26) відповідає за відображення списку контенту у вигляді карток на яких показано постер, назва, оцінка та рік.

```
async function handleRoute() {
    const route = parseHash();
    if (route.page === 'list') {
        currentType = route.type || 'all';
        currentPage = route.pageNum || 1;
        filters.classList.remove('hidden');
        nav.classList.remove('hidden');
        paginationContainer.classList.remove('hidden');
        await loadData();
        return;
    }
    filters.classList.add('hidden');
    nav.classList.add('hidden');
    paginationContainer.classList.add('hidden');
    if (route.page === 'details') {
        window.scrollTo({ top: 0, behavior: 'auto' });
        currentType = route.type;
        openItem(route.id);
        return;
    }
    if (route.page === 'actor') {
        window.scrollTo({ top: 0, behavior: 'auto' });
        openActor(route.id);
        return;
    }
    if (route.page === 'profile') {
        window.scrollTo({ top: 0, behavior: 'auto' });
        openProfile();
        return;
    }
    location.hash = '';
}
```

Рисунок 3.25 – Функція handleRoute()

```
function renderList(data) {
    try {
        listContainer.classList.add('grid-mode');
        listContainer.innerHTML = '';
        if (!data.length) {

            listContainer.innerHTML = '<p>Нічого не знайдено</p>';
            return;
        }
        listContainer.innerHTML = data.map(item => `
            <div class="movie-card" onclick="location.hash='${item.type}-${item.id}'">
              <img src="${safeUrl(item.poster_url)}">
              <h3>${escapeHtml(item.title)}</h3>
              <p>Рік: ${item.release_year}</p>
              <p>Рейтинг: ${item.rating}</p>
            </div>
            `).join('');
            renderPagination();
    } catch (e) {
        console.error(e);
        listContainer.innerHTML = 'Помилка завантаження';
    }
}
```

Рисунок 3.26 – Функція renderList ()

Функція loadData() (рис. 3.27) є основною для завантаження контенту та виконує асинхронне завантаження списку фільмів і серіалів із сервера. Її основним функціоналом є: формування query string, додавання пошуку та фільтрів, надсилання fetch-запитів, обробка JSON-відповіді, завантаження фільмів і серіалів паралельно, передача даних у renderList() та обробка помилок API.

```
async function loadData(search = '') {
    const filters = getFilters();
    const queryParts = [];
    const ITEMS_PER_PAGE = 16;
    if (search) queryParts.push(`search=${encodeURIComponent(search)}`);
    if (filters) queryParts.push(filters);
    if (currentSearch) {
        queryParts.push(`search=${encodeURIComponent(currentSearch)}`);
    }
    const queryString = queryParts.join('&');
    try {
        if (currentType === 'all') {
            const [moviesRes, seriesRes] = await Promise.all([
                fetch(`/movies?page=${currentPage}&limit=${ITEMS_PER_PAGE/2}&${queryString}`),
                fetch(`/series?page=${currentPage}&limit=${ITEMS_PER_PAGE/2}&${queryString}`)
            ]);
            if (!moviesRes.ok || !seriesRes.ok) throw new Error('API помилка');
            const movies = (await moviesRes.json());
            const series = (await seriesRes.json());
            let combined = [];
            const maxLength = Math.max(movies.length, series.length);
            for (let i = 0; i < maxLength; i++) {
                if (movies[i]) combined.push(movies[i]);
                if (series[i]) combined.push(series[i]);
            }
            renderList(combined);
        } else {
            const res = await fetch(`/${currentType}?page=${currentPage}&limit=${ITEMS_PER_PAGE}&${queryString}`);
            if (!res.ok) throw new Error('API помилка');
            const data = await res.json();
            renderList(data);
        }
    } catch (e) {
        console.error(e);
        listContainer.innerHTML = 'Помилка завантаження';
    }
}
```

Рисунок 3.27 – Функція loadData()

Наступні функції (рис. 3.28) використовуються для створення елементів пагінації сторінки, завдяки яким виконується створення кнопок для переходу на наступну чи минулу сторінку.

```
function renderPagination() {
    paginationContainer.innerHTML = `
        <div style="text-align:center; margin:20px;">
        <button id="prevBtn">⬅</button>
        <span>Сторінка ${currentPage}</span>
        <button id="nextBtn">➡</button>
    </div>
    `;
    document.getElementById('prevBtn').onclick = prevPage;
    document.getElementById('nextBtn').onclick = nextPage;
}

function nextPage() {
    currentPage++;
    loadData();
}

function prevPage() {
    if (currentPage > 1) {
        currentPage--;
        loadData();
    }
}
```

Рисунок 3.28 – Функції пагінації

3.3.4 Реалізація функцій відображення даних

Функція openItem() відкриває детальну сторінку фільму або серіалу на якій виводиться: назва, постер, більш детальна інформація (рік, жанри, країни, опис тощо), кадри, трейлер та відгуки. На рисунках 3.29, 3.30 та 3.31 показано фрагменти з реалізацією основного функціоналу.

```
async function openItem(id) {

        listContainer.classList.remove('grid-mode');
        listContainer.innerHTML = `
            <div class="details">
                <div class="poster">
                    <img src="${safeUrl(item.poster_url)}">
                </div>

                <div class="details-info">
                    <h2>${escapeHtml(item.title_ua)}</h2>
                    <p><b>EN:</b> ${escapeHtml(item.title_en || '')}</p>
                    <p><b>Рік:</b> ${item.release_year}</p>
                    <p><b>Жанр:</b> ${item.genres?.map(escapeHtml).join(', ') || ''}</p>
                    <p><b>Країна:</b> ${item.countries?.map(escapeHtml).join(', ') || ''}</p>

                    ${currentType === 'movies'
                        ? `<p><b>Тривалість:</b> ${item.duration} хв</p>`
                        : `<p><b>Сезони:</b> ${item.seasons_count}</p>
                           <p><b>Епізоди:</b> ${item.episodes_count}</p>`
                    }

                    <p><b>Рейтинг:</b> ${item.rating}</p>
                    <p>${escapeHtml(item.description || '')}</p>

                    <h3>Актори:</h3>
                    <div class="actors">
                        ${actors.map(a => `
                            <div class="actor-card" onclick="location.hash='actor-${a.id}'">
                                <img src="${safeUrl(a.photo_url)}">
                                <p>${escapeHtml(a.name)}</p>
                            </div>
                        `).join('')}
                    </div>
                </div>
            </div>
```

Рисунок 3.29 – Вивід загальної інформації

```
async function openItem(id) {
            <h2>Кадри</h2>

            <button class="gallery-btn left" onclick="scrollContainer('galleryList', -500)"> ⬅ </button>

            <div class="gallery-wrapper">
            <div class="gallery-list" id="galleryList">
               ${images.map(img => `
                 <img src="${safeUrl(img.image_url)}" onclick="openLightbox('${safeUrl(img.image_url)}')">`).join('')}
                 </div>
                 </div>

            <button class="gallery-btn right" onclick="scrollContainer('galleryList', 500)"> ➡ </button>
            </div>


            ${item.trailer_url ? `
                <div class="trailer-block">
                    <h2>Трейлер</h2>
                    <div class="trailer">
                    <iframe src="${safeUrl(item.trailer_url)}" allowfullscreen></iframe>
                </div>
             </div>
             ` : ''}
```

Рисунок 3.30 – Фрагмент виводу кадрів та трейлеру

```
async function openItem(id) {
            <h2>Відгуки</h2>

            <div id="reviewsList"></div>
            <h2>Залишити відгук</h2>

            <div class="review-form">

            <input type="text" id="guestName" placeholder="Ваше ім'я">

            <p>Оцінка:
            <select id="reviewRating">
                <option value="10">10</option>
                <option value="9">9</option>
                <option value="8">8</option>
                <option value="7">7</option>
                <option value="6">6</option>
                <option value="5">5</option>
                <option value="4">4</option>
                <option value="3">3</option>
                <option value="2">2</option>
                <option value="1">1</option>
            </select></p>

            <textarea id="reviewText" placeholder="Ваш коментар"></textarea>

            <button onclick="addReview(${item.id})"> Надіслати </button>
            </div>
            </div>

            <div id="lightbox" class="lightbox" onclick="closeLightbox()">
              <img id="lightboxImg">
            </div>
            <center><button onclick="history.back()">⬅ Назад</button></center>
        </div>
        `;
```

Рисунок 3.31 – Фрагмент виводу списку відгуків та форми для їх написання

Функція openProfile() відкриває особистий кабінет користувача. В ній перевіряється авторизація, отримується список залишених відгуків, формується сторінка профілю та є можливість вийти з акаунту. На рисунках 3.32 та 3.33 продемонстровано фрагменти цієї функції, коли користувач не авторизовний та авторизований.

```
async function openProfile() {
    try{
        listContainer.classList.remove('grid-mode');

        const res = await fetch('/me');
        const user = await res.json();

        if (!user) {
          listContainer.innerHTML = `
              <div class="profile-page not-auth">
                  <h2>Ви не авторизовані</h2>
                  <button onclick="openLogin()"> Увійти </button>
              </div>
          `;
          return;
        }
```

Рисунок 3.32 – Користувач не авторизований

```
async function openProfile() {
        const reviewsRes = await fetch('/me/reviews');
        const reviews = await reviewsRes.json();
        listContainer.innerHTML = `
        <div class="profile-page">
        <h2>${escapeHtml(user.username)}</h2>
        <h3>Ваші відгуки</h3>
        ${reviews.length ? reviews.map(r => {
            const title = r.movie_title || r.series_title || 'Невідомо';
            const link = r.movie_id ? `movies-${r.movie_id}` : `series-${r.series_id}`;
            return `
                <div class="review-item">
                    <a href="#${link}" class="review-title">
                        ${escapeHtml(title)}
                    </a>
                    <p>Рейтинг: ${r.rating}/10</p>
                    <p>${escapeHtml(r.review_text)}</p>
                    <small>${new Date(r.created_at).toLocaleDateString('uk-UA')}</small>
                </div>
            `;
        }).join('') : '<p>Відгуків ще немає</p>'}

        <div class="profile-actions">
        <button onclick="logout()">Вийти</button>
        </div>
        </div>
        `; } catch (e) {
            console.error(e);
            listContainer.innerHTML = 'Помилка завантаження';
        }
}
```

Рисунок 3.33 – Користувач авторизований

Функція openActor() відкриває сторінку актора, де виводиться фото та дата народження, формуються списки фільмів і серіалів та створюються горизонтальні слайдери. На рисунку 3.34 показано фрагмент коду цієї функції для виводу списку фільмів.

```
async function openActor(id) {
    try {
        const res = await fetch(`/actors/${id}`);
        const actor = await res.json();
        if (!res.ok) {
            listContainer.innerHTML = 'Актора не знайдено';
            return;
        }
        listContainer.classList.remove('grid-mode');
        listContainer.innerHTML = `
        <div class="details">
            <div class="poster">
                <img src="${safeUrl(actor.photo_url)}">
            </div>
            <div class="details-info">
                <h2>${escapeHtml(actor.name)}</h2>
                <p> Дата народження: ${actor.birth_date ? new Date(actor.birth_date).toLocaleDateString('uk-UA') : '-'} </p>

                <h3>Фільми:</h3>
                <div class="media-slider">
                    <button class="scroll-btn left" onclick="scrollContainer('moviesRow', -300)"> ⬅ </button>
                    <div class="horizontal-scroll" id="moviesRow">
                        ${actor.movies.map(m => `
                            <div class="movie-card"
                                 onclick="location.hash='movies-${m.id}'">
                                <img src="${safeUrl(m.poster_url)}">
                                <h3>${escapeHtml(m.title_ua)}</h3>
                                <p>Рік: ${m.release_year || ''}</p>
                            </div>
                        `).join('')}
                    </div>
                    <button class="scroll-btn right" onclick="scrollContainer('moviesRow', 300)"> ➡ </button>
                </div>
```

Рисунок 3.34 – Фрагмент функції openActor()

3.3.5 Реалізація функцій реєстрації та авторизації

Функція login() (рис. 3.35) відкриває виконує авторизацію користувача. В ній отримується email та пароль, надсилається POST-запит, перевіряється відповідь сервера і виконується перехід у профіль. Функція реєстрації реалізована схожим чином (рис. 3.36).

```
async function login() {
    const email = document.getElementById('loginEmail').value;
    const password = document.getElementById('loginPassword').value;

    const res = await fetch('/login', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            email,
            password
        })
    });

    const data = await res.json();

    if (!res.ok) {
        alert(data.message);
        return;
    }

    alert('Успішний вхід');

    location.hash = 'profile';
}
```

Рисунок 3.35 – Функція login()

```
async function register() {
    const username = document.getElementById('registerUsername').value;
    const email = document.getElementById('registerEmail').value;
    const password = document.getElementById('registerPassword').value;

    const res = await fetch('/register', {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },
        body: JSON.stringify({
            username,
            email,
            password
        })
    });

    const data = await res.json();

    if (!res.ok) {
        alert(data.message);
        return;
    }

    alert('Реєстрація успішна');

    location.hash = 'profile';
}
```

Рисунок 3.36 – Функція register()

Функція addReview() (рис. 3.37) виконує додавання нового відгуку. Вона отримує отримує дані форми, надсилає POST-запит, виконує валідацію відповіді, очищує форму та оновлює список відгуків.

```
async function addReview(id) {
    const guest_name = document.getElementById('guestName').value;
    const rating = document.getElementById('reviewRating').value;
    const review_text = document.getElementById('reviewText').value;

    const res = await fetch(`/${currentType}/${id}/reviews`, {
        method: 'POST',
        headers: {
            'Content-Type': 'application/json'
        },

        body: JSON.stringify({
            guest_name,
            rating,
            review_text
        })
    });

    if (!res.ok) {
        const error = await res.json();
        alert(error.message);
        return;
    }

    document.getElementById('reviewText').value = '';
    document.getElementById('reviewRating').value = '10';

    await loadReviews(id);
}
```

Рисунок 3.37 – Функція addReview()

3.4 Демонстрація роботи вебдодатку

На рис. 3.38 Продемонстрована головна сторінка вебдодатка з наведенням курсору на певне медіа.

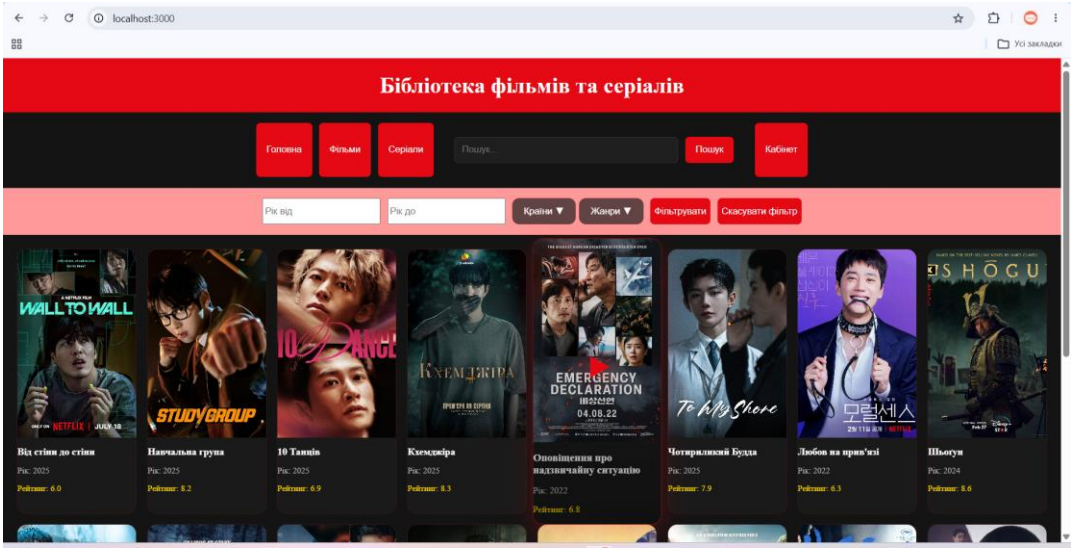


Рисунок 3.38 – Головна сторінка

При наведенні курсору на кнопки вони підсвічуються (рис. 3.39, 3.40).

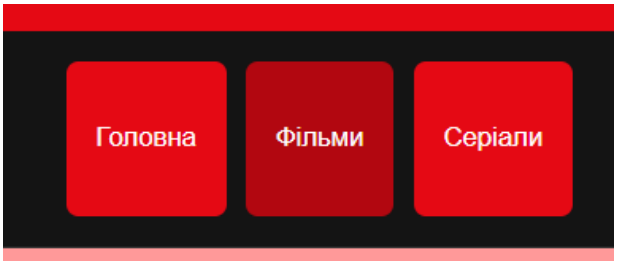


Рисунок 3.39 – Наведення курсору на кнопку «Фільми»

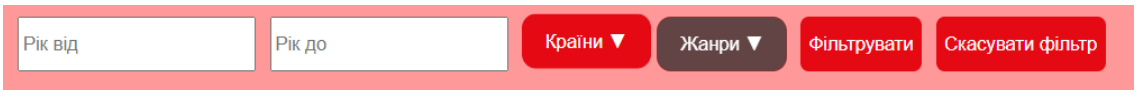


Рисунок 3.40 – Наведення курсору на кнопку «Країни»

Результат застосування фільтрів (рис. 3.41).

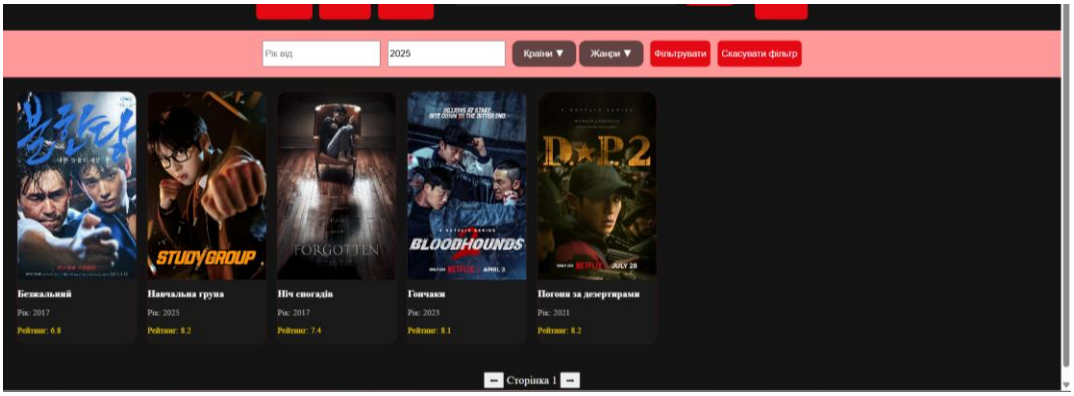


Рисунок 3.41 – Застосування фільтрів

Результат пошуку медіа за назвою (рис. 3.42).

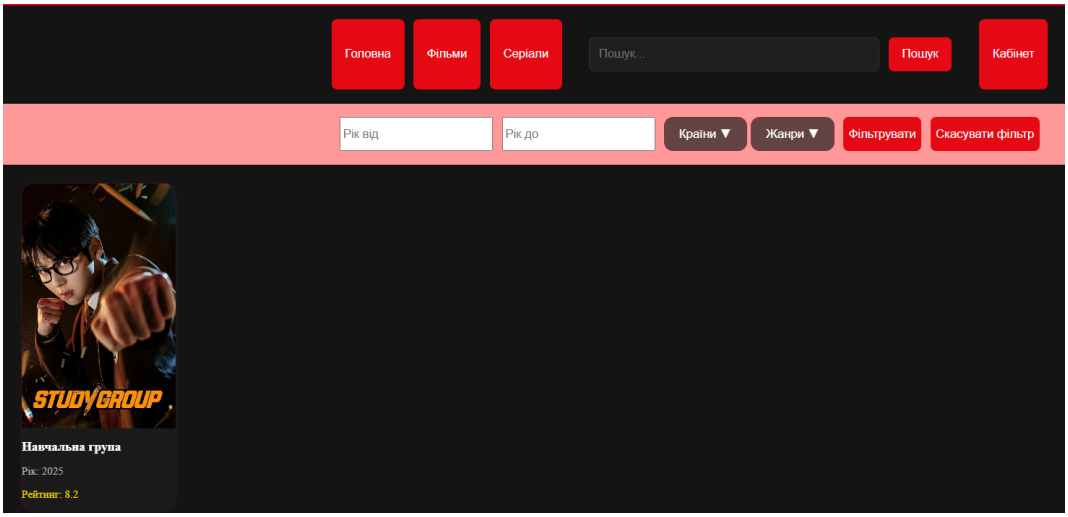


Рисунок 3.42 – Пошук за назвою

На рис. 3.43, 3.44 та 3.45 продемонстровано вигляд відкритої сторінки з обраним медіа.

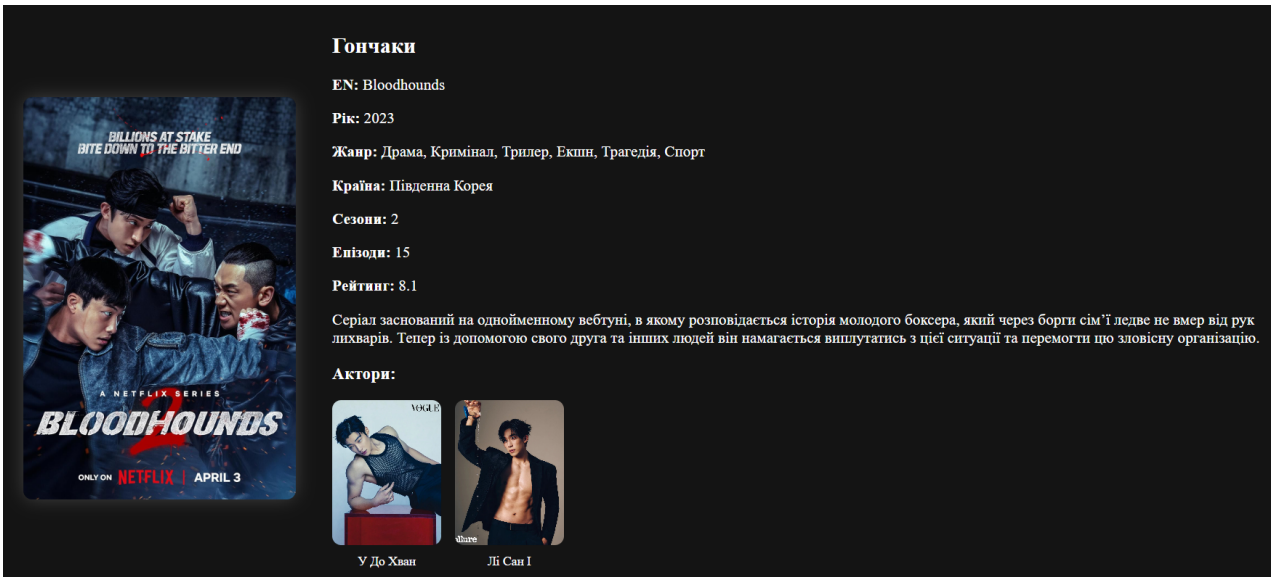


Рисунок 3.43 – Частина сторінки з описом

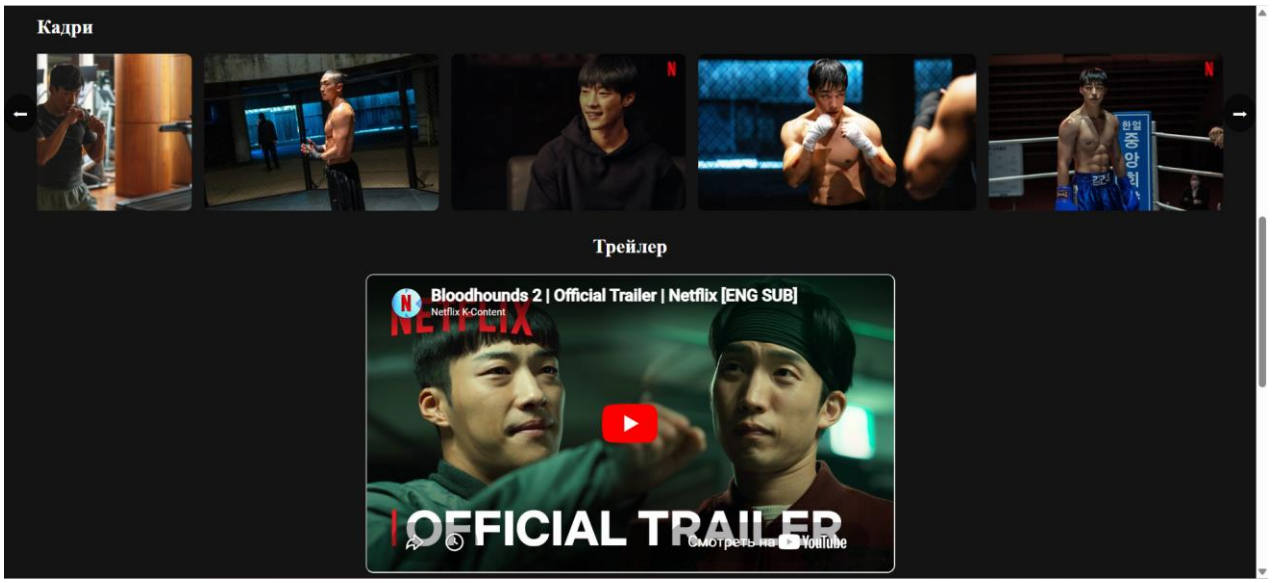


Рисунок 3.44 – Частина сторінки з кадрами та трейлером

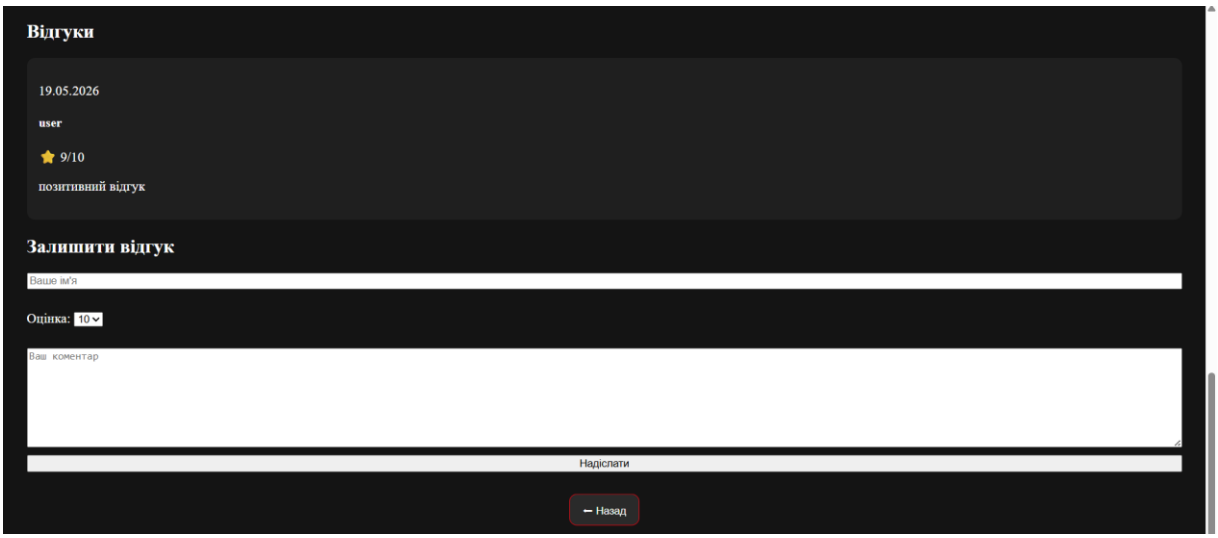


Рисунок 3.45 – Частина сторінки з відгуками

На рис. 3.46 показано приклад розгортання кадра з медіа.



Рисунок 3.46 – Розгорнутий кадр

Приклад сторінки актора (рис. 3.47, 3.48).



Рисунок 3.47 – Сторінка актора (перша половина)

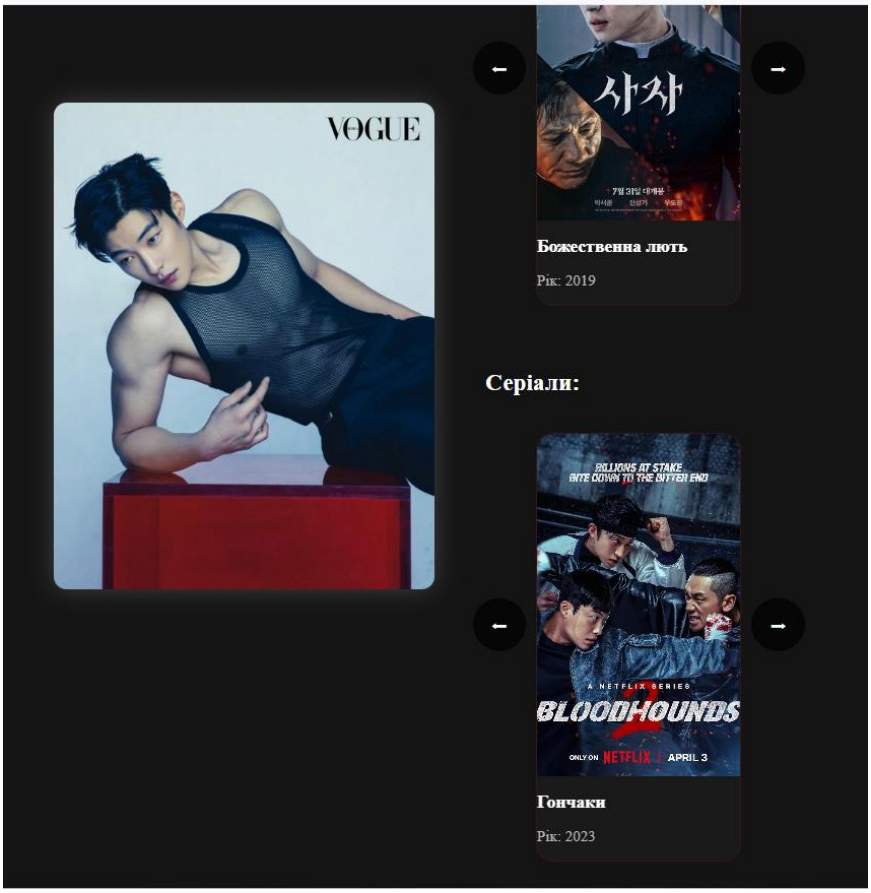


Рисунок 3.48 – Сторінка актора (нижня половина)

Перехід до особистого кабінету, коли користувач ще не авторизувався (рис. 3.49).

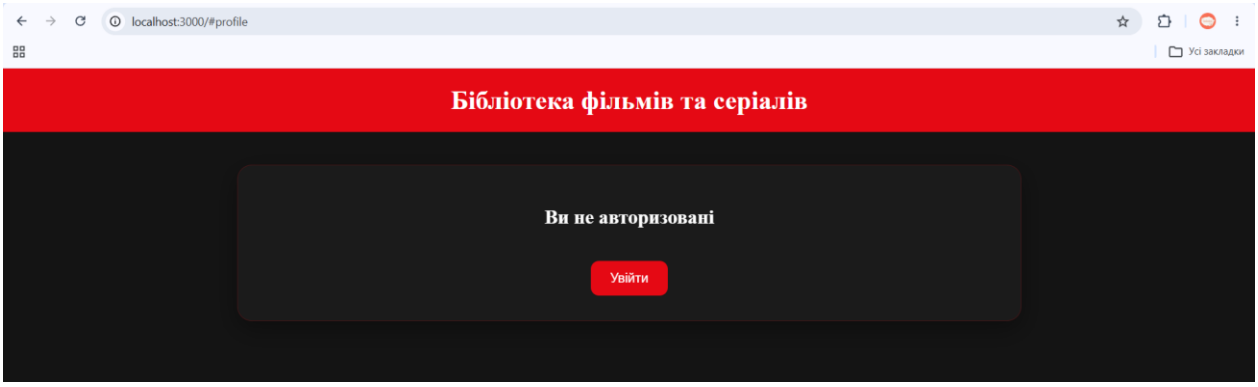


Рисунок 3.49 – Особистий кабінет без авторизації

Форми реєстрації (рис. 3.50) та входу в особистий кабінет (рис. 3.51).

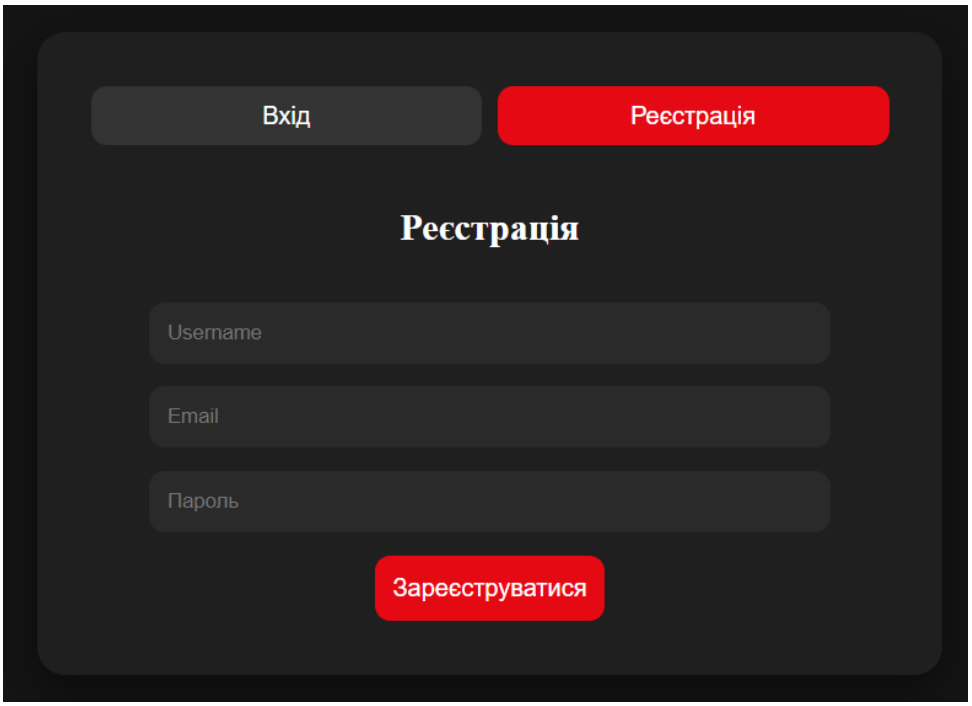


Рисунок 3.50 – Форма реєстрації

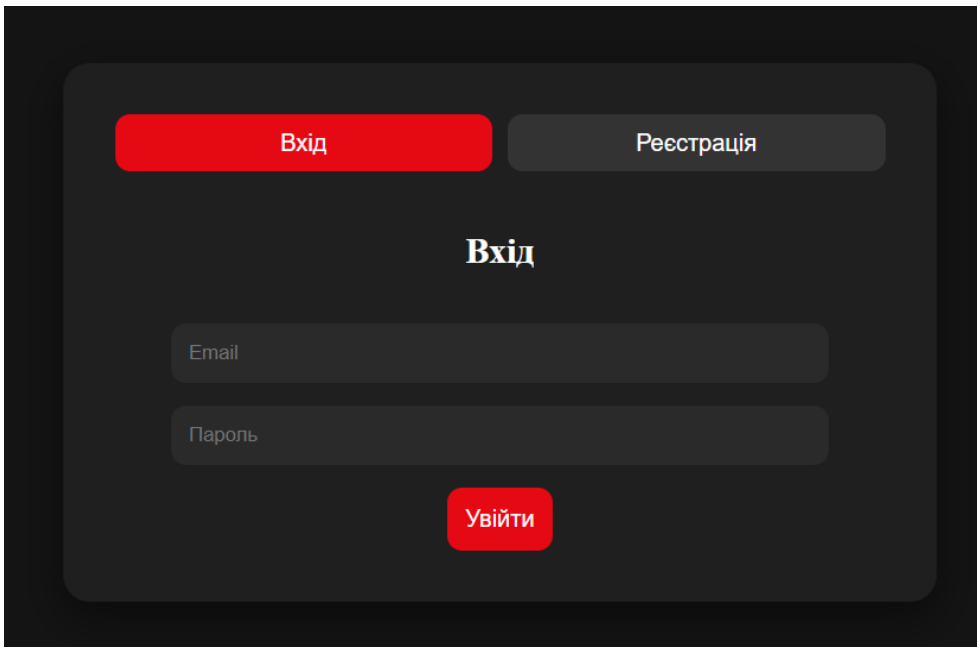


Рисунок 3.51 – Форма авторизації

Вигляд особистого кабінету після авторизації (рис. 3.52).



Рисунок 3.52 – Особистий кабінет

3.5 Висновок до третього розділу

Отже, в третьому розділі було описано програмну реалізацію розробленого клієнт-серверного односторінкового вебдодаку: розроблено

серверну та клієнтську частини системи, реалізовано взаємодію між ними за допомогою HTTP-запитів та REST API.

На серверній стороні було створено маршрути для отримання списків фільмів, серіалів та акторів, перегляду детальної інформації, роботи з відгуками користувачів, а також реалізовано систему реєстрації та авторизації із використанням сесій та хешування паролів. Для взаємодії з базою даних використовувались SQL-запити.

На клієнтській стороні було реалізовано односторінковий вебдодаток із динамічним оновленням контенту без перезавантаження сторінки: було створено механізми пошуку, фільтрації, пагінації, перегляду детальної інформації, роботи з відгуками, перегляду кадрів і трейлерів, а також особистий кабінет користувача.

ВИСНОВКИ

Отже, в процесі виконання кваліфікаційної роботи бакалавра було проаналізовано та порівняно сучасні технології веброзробки, які в результаті були використані для розробки клієнт-серверного односторінкового вебдодатку.

По-перше, було розглянуто основні поняття веброзробки: бекенд, фронтенд, фреймворки, бази даних та системи управління базами даних.

По-друге, було проаналізовано принципи створення клієнт-серверних вебдодатків, оглянуто та порівняно засоби розробки інтерфейсу користувача, серверної логіки та організації бази даних. На основі проведеного аналізу було обрано найдоцільніші технології: HTML, CSS, JavaScript, Node.js, Express та MySQL.

Результатом виконання дипломної роботи є розроблений клієнт-серверний односторінковий вебдодаток для перегляду інформації про фільми та серіали, який має такий функціонал: перегляд інформації про фільми/серіали/акторів, пошук потрібного медіа, можливість реєстрації та авторизації користувача, залишення відгуків. Крім того, було забезпечено базовий рівень захисту вебдодатку від поширених вебзагроз.

ПЕРЕЛІК ПОСИЛАНЬ

1. Веброзробка. Wikipedia. URL : https://uk.wikipedia.org/wiki/%D0%92%D0%B5%D0%B1%D1%80%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0 (дата звернення : 22.03.2026).

2. Що таке вебдодаток? Webcase. URL : https://webcase.com.ua/uk/blog/cho-takoe-web-prilozhenie-vse-vidy/ (дата звернення : 24.03.2026).

3. Що таке бекенд. Wezom.com.ua. URL : https://wezom.com.ua/ua/blog/chto-takoe-back-end-razrabotka (дата звернення : 24.03.2026).

4. Що таке СУБД та для чого вони потрібні. Foxminded. URL : https://foxminded.ua/systema-upravlinnia-bazamy-danykh/ (дата звернення : 28.03.2026).

5. Backend-розробка: яку мову програмування вивчити новачку. GoIT Global. URL : https://goit.global/ua/articles/backend-rozrobka-iaku-movu-prohramuvannia-vchyty-novachku/ (дата звернення : 24.03.2026).

6. HTML, CSS та JavaScript: ключові аспекти веб-розробки. Tto-studio.com.ua. URL : https://tto-studio.com.ua/html-css-ta-javascript-klyuchovi-aspekty-veb-rozrobky/ (дата звернення : 28.03.2026).

7. Клієнт-серверна архітектура. QATestLab training center. URL : https://training.qatestlab.com/blog/technical-articles/client-server-architecture/ (дата звернення : 28.03.2026).

8. Що таке Node.js: визначення, особливості та застосування. Wezom.com.ua. URL : https://wezom.com.ua/ua/blog/vse-chto-nuzhno-znat-o-nodejs (дата звернення : 02.04.2026).

9. Express.js: встановлення, налаштування та особливості. FoxmindEd. URL : https://foxminded.ua/express-js/ (дата звернення : 02.04.2026).

10. REST API як спосіб спілкування компонентів веб-додатку. FoxmindEd. URL : https://foxminded.ua/shcho-take-rest-api/ (дата звернення : 20.04.2026).

11. Вступ до REST API. Robot_dreams. URL : https://robotdreams.cc/uk/blog/466-vstup-do-rest-api-restful-vebservisi (дата звернення :21.04.2026).

12. Погромська Г. С., Махровська Н. А. Бази даних : проектування та реалізація. Миколаїв :Миколаївський національний університет імені В. О. Сухомлинського, 2019. 183 с. URL : https://files.znu.edu.ua/files/Bibliobooks/Inshi78/0058251.pdf (дата звернення : 30.04.2026).

13. Безпека веб-додатків : найкращі практики та вразливості. itProger.com. URL : https://itproger.com/ua/news/bezopasnost-veb-prilozheniy-luchshie-praktiki-i-uyazvimosti (дата звернення : 10.05.2026).

14. PostgreSQL vs MySQL : Explore Their 12 Critical Differences. Kinsta.com. URL : https://kinsta.com/blog/postgresql-vs-mysql/ (дата звернення : 18.05.2026).

15. Postgresql що це і як застосовується в різних напрямках. FoxmindEd.com. URL : https://foxminded.ua/postgresql-shcho-tse/ (дата звернення : 18.05.2026).

16. Що таке MySQL? FREEhost.UA. URL : https://freehost.com.ua/ukr/faq/wiki/chto-takoe-mysql/ (дата звернення : 18.05.2026).