**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**
**НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ**
**ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ**
**КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ**

# КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення для рекомендації кінофільмів за вподобанням користувачів»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

*Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело*

_____ Павло КУЗНЕЦОВ
*(підпис)*

Виконав: здобувач вищої освіти групи ПД-41

Павло КУЗНЕЦОВ

Керівник: Данило КОВАЛЕНКО
*доктор філософії (PhD)*

Рецензент: _____

**Київ 2026**

**ДЕРЖАВНИЙ УНІВЕРСИТЕТ**
**ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ**

**Навчально-науковий інститут інформаційних технологій**

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

**ЗАТВЕРДЖУЮ**
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2026 р.

# ЗАВДАННЯ
# НА КВАЛІФІКАЦІЙНУ РОБОТУ

Кузнецову Павлу Олександровичу

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення для рекомендації кінофільмів за вподобанням користувачів»
керівник кваліфікаційної роботи доктор філософії (PhD) Данило КОВАЛЕНКО,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «20» лютого 2026 року № 51.

2. Строк подання кваліфікаційної роботи «29» травня 2026 року.

3. Вихідні дані до кваліфікаційної роботи: теоретичні відомості про сучасні web-технології та принципи побудови клієнт-серверних застосунків, опис методів роботи з REST API та зовнішніми сервісами для отримання даних про кінофільми, API TMDB для інтеграції інформації про фільми та реалізації функціоналу пошуку і відображення контенту.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Аналіз сучасних web-застосунків для пошуку та перегляду інформації про кінофільми, визначення їхніх переваг і недоліків.
2. Проєктування клієнт-серверної архітектури застосунка «Кіно Хвиля».

3. Програмна реалізація web-застосунка «Кіно Хвиля» із використанням сучасних веб-технологій та зовнішнього TMDB.
4. Реалізація механізмів авторизації користувачів та управління персональним списком перегляду фільмів.
5. Тестування функціональних можливостей web-застосунка та перевірка коректності роботи основних модулів системи.

5. Перелік графічного матеріалу: *презентація*
   1. Аналіз аналогів.
   2. Вимоги до додатку.
   3. Програмні засоби реалізації.
   4. Діаграма Use Case.
   5. Діаграма послідовності.
   6. Тестування
   7. Екрані форми.
   8. Відео роботи додатку.
   9. Апробація результатів дослідження

6. Дата видачі завдання «23» лютого 2026 року.

## КАЛЕНДАРНИЙ ПЛАН

| № з/п | Назва етапів кваліфікаційної роботи | Строк виконання етапів роботи | Примітка |
|---|---|---|---|
| 1 | Підбір та аналіз науково-технічної літератури | 23.02-28.02.2026 | |
| 2 | Аналіз та дослідження існуючих аналогів | 01.03-07.03.2026 | |
| 3 | Огляд існуючих технологій, підходів та API | 08.03-14.03.2026 | |
| 4 | Проєктування web-застосунка «Кіно Хвиля» | 15.03-28.03.2026 | |
| 5 | Програмна реалізація клієнтської частини | 29.03-18.04.2026 | |
| 6 | Тестування web-застосунка та виправлення помилок | 19.04-03.05.2026 | |
| 7 | Оформлення роботи: вступ, висновки, реферат | 04.05-08.05.2026 | |
| 8 | Розробка демонстраційних матеріалів | 09.05-10.05.2026 | |
| 9 | Попередній захист роботи | 11.05-22.05.2026 | |
| 10 | Подання кваліфікаційної роботи | 23.05-29.05.2026 | |

Здобувач вищої освіти _____ Павло КУЗНЕЦОВ
*(підпис)*

Керівник
кваліфікаційної роботи _____ Данило КОВАЛЕНКО
*(підпис)*

# РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: __42__ стор., __ 2 __ табл., __21__ рис., __23__ джерел.

*Мета роботи* – спрощення процесу пошуку, перегляду інформації та формування персонального списку за вподобаннями користувача, перегляду кінофільмів шляхом створення сучасного web-застосунка.

*Об'єкт дослідження* – процес пошуку, фільтрації рекомендації мультимедійного контенту, ведення власного списку перегляду.

*Предмет дослідження* – технології, методи та програмні засоби реалізації клієнт-серверного web-застосунка для роботи з кінофільмами.

*Короткий зміст роботи*: У роботі проаналізовано існуючі сервіси для пошуку фільмів Netflix, IMDb, Letterboxd, TMDB та визначено вимоги до власного застосунка. Спроектовано клієнт-серверну архітектуру системи з використанням REST API та зовнішнього сервісу TMDB.

Реалізовано функціонал пошуку та фільтрації фільмів, авторизації користувачів, а також персонального списку перегляду. Серверна частина побудована на Node.js та Express.js з використанням JWT та SQLite. Клієнтська - на HTML, CSS та JavaScript. Проведено тестування основних функцій системи та перевірено коректність взаємодії між компонентами.

Сферою використання розробленого веб-застосунка є повсякденний пошук фільмів, організація власного списку перегляду та спрощення вибору контенту для користувачів.

КЛЮЧОВІ СЛОВА: WEB-ЗАСТОСУНОК, КІНОФІЛЬМИ, TMDB API, JAVASCRIPT, NODE.JS, EXPRESS, SQLITE, JWT, СИСТЕМА ПЕРЕГЛЯДУ

**ЗМІСТ**

# ВСТУП

*Актуальність дослідження*

Актуальність теми зумовлена тим, що зараз стрімко розвиваються сервіси пов'язані з кіноіндустрією. Та постійним збільшенням кількості доступного медіаконтенту. У сучасних умовах користувачі часто стикаються з проблемою швидкого пошуку фільмів та організації персонального списку перегляду. Це трапляється через перевантаженість існуючих платформ або обмежений функціонал безкоштовних сервісів. Тому виникає потреба у створенні простого, зручного та функціонального веб-застосунка. В якому буде швидкий доступ до інформації про фільми, систему фільтрації та персоналізований список перегляду.

*Ступінь наукової розробки*

Питання розробки веб-застосунків та систем пошуку медіаконтенту широко досліджуються у сфері веб-технологій, та клієнт-серверних систем. Зараз існує велика кількість крутих популярних сервісів. Таких як Netflix, IMDb, Letterboxd та TMDB  які реалізують окремі функції пошуку, рекомендацій та персоналізації контенту. Проте і вони мають свої нюанси типу складного інтерфейсу, платний функціонал чи недостатню гнучкість у роботі зі списками перегляду. Це створює передумови для розробки власного веб-застосунка із акцентом на простоту та зручність використання.

*Практичне значення одержаних результатів*

Практичне значення роботи полягає у створенні функціонального веб-застосунка «Кіно Хвиля». Він може використовуватися користувачами для пошуку фільмів, перегляду інформації про них та формування персонального списку перегляду. Розроблений застосунок забезпечує інтеграцію із зовнішнім API від TMDB. Підтримує систему авторизації користувачів та дозволяє зберігати персональні дані у базі. Отримані результати можуть бути використані як основа для подальшого розвитку системи. Такі як реалізації рекомендаційних алгоритмів, системи оцінювання фільмів та інтеграції зі стрімінговими платформами.

*Мета дослідження*

Забезпечення зручного та ефективного процесу пошуку, перегляду інформації та формування персонального списку перегляду кінофільмів. Шляхом створення сучасного web-застосунка з використанням клієнт-серверної архітектури та інтеграції із зовнішнім API.

*Результати дослідження*

У результаті виконання роботи було проаналізовано популярні веб-застосунки для пошуку фільмів, визначено функціональні вимоги до системи. Також спроєктовано архітектуру веб-застосунка «Кіно Хвиля». Реалізовано клієнтську та серверну частини застосунка з використанням HTML, CSS, JavaScript, Node.js, Express.js та SQLite. У системі реалізовано пошук і фільтрацію фільмів через API від TMDB. Систему реєстрації та авторизації користувачів на основі JWT і також персональний список перегляду. Тестування підтвердило, що основні функцій працюють коректно а взаємодії між компонентами системи стабільні.

# 1 АНАЛІЗ WEB-ЗАСТОСУНКА «КІНО ХВИЛЯ»

## 1.1 Аналіз та загальна характеристика додатка

Розроблений застосунок це сучасний сервіс, який орієнтований на роботу з фільмами серіалами та інше. Має можливість ведення власного списку перегляду, для зареєстрованих користувачів. Основна мета спростити пошук кінофільмів за вподобаннями користувача, отримання інформації про фільм та ведення власного списку перегляду. Застосунок вирішить проблему великої частини кінолюбителів, які постійно зберігають фільми скріншотами або в замітках та потім гублять їх.

У сучасних умовах цифрового розвитку обсяг кіно-контенту зростає щодня. Тому процес вибору фільму на вечір чи ваш інший вільний час дуже ускладнюється з кожним разом. Тому виникає потреба розробити застосунок який може надати структуровану інформацію про фільм у швидкому доступі.

Тож, застосунок поєднує в собі функції, як інформаційного так і персоналізованої системи з управлінням списку перегляду.

Основний функціональна додатку:

1. Пошук кінофільмів за назвою;
2. Перегляд інформації про них;
3. Відображення постерів та описів;
4. Перегляд рейтингу, жанрів та року випуску;
5. Ведення власного списку перегляду;
6. Додавання та видалення фільмів зі списку;
7. Фільтрацію та сортування контенту за новизною, популярностю та кращими за оцінками.

Увагу приділив механізму фільтрації та сортування кінофільмів, користувач може відібрати за жанром, роком випуску та мінімальним рейтингом за оцінками глядачів, що дозволить скорити витрачений на пошук час.

З приводу персоналізації застосунку, було реалізовано реєстрацію та авторизацію користувачів. Коли користувач зареєструвався та увійшов до системи єму надається можливість формування власного списку перегляду. В нього можна додавати та видаляти кінофільми.

Система побудована за клієнт-серверною архітектурою:

1. Клієнтська частина відповідає за інтерфейс користувача і взаємодію з ним;

2. Серверна частина обробляє запити, працює з базою та забезпечує авторизацію користувача;

3. База даних використовується для збереження інформації про користувачів та їх списки перегляду.

Щоб отримувати свіжу інформацію про фільми використовується зовнішнє API від TMDB.

Тож, застосунок є функціональним рішенням для більшості, в якому швидкий та зручний пошук за вподобаннями користувача інформація по фільму та ведення власного списку перегляду.

### 1.2 Цільова аудиторія

Цільова аудиторія додатку дуже широка, тому що кіноіндустрія та кінолюбителі охоплюють величезну кількість людей,  різного віку та інтересів. Спершу застосунок орієнтований на людей які регулярно дивляться та шукають фільми, коли в них є вільний час їм потрібен інструмент, в якому вони могли б швидко знати  фільм та подивитись інформацію до нього і якщо він сподобався додати до власного списку перегляду. Окрема аудиторія це люди які ведуть власні списки в замітках телефону, або скріншоти які займають купу місця у галереї.

Також з плюсів додатку є простий та інтуїтивний інтерфейс, тому нові користувачі які можливо не мали досвіду з такими додатками, зможуть без проблем ним користуватись.

### 1.3 Дослідження існуючих аналогів

Перед початком розробки була проведено аналіз конкурентів, існуючих рішень які вже давно працюють у цій сфері. Метою аналізу було знайти сильні та слабкі сторони популярних сервісів, так як вони давно працюють з кіно-контенотом, рейтингами та рекомендаціями фільмів. Після аналізу вдалось краще зрозуміти сучасні підходи в інтерфейсі користувача, обробці даних та взаємодії з користувачем. І допомогло сформувати вимоги до системи.

Приділялась увага як вони реалізували пошук контенту, систему рекомендацій та списки перегляду. В цілому зручність інтерфейсу, наскільки швидко користувач міг знайти потрібний йому контент, чи не витрачається на це багато часу.

Структура інформації про фільм, які персоналізації використовуються. І на основі цього аналізу було виявлено, що найбільш ефективним буде поєднати простий інтерфейс з можливостями фільтрації та рекомендацій. Так як окремого простого інструменту для пошуку та ведення списків часто бракує. Тому саме цю нішу обрав як основу для концепції застосунку і дав йому назву «Кіно Хвиля»

Для дослідження було взято наступні популярні на даний момент аналоги:

1. Netflix
2. IMDb
3. Letterboxd
4. TMDB

Трохи детальніше про кожного з них:

#### 1.3.1 Netflix

Netflix - це напевно так стрімінгова платформа яку згадують першою. Вона дає користувачу доступ до великої та постійно оновлюваної бази з фільмами, серіалами та їх власними проєктами. Головна особливість Netflix є використання складних алгоритмів рекомендацій. Вони аналізують поведінку користувача його історію перегляду, оцінки та уподобання та після цього формують персональну

стрічку для нього. Цим вони значно спрощують процес пошуку для свого користувача.

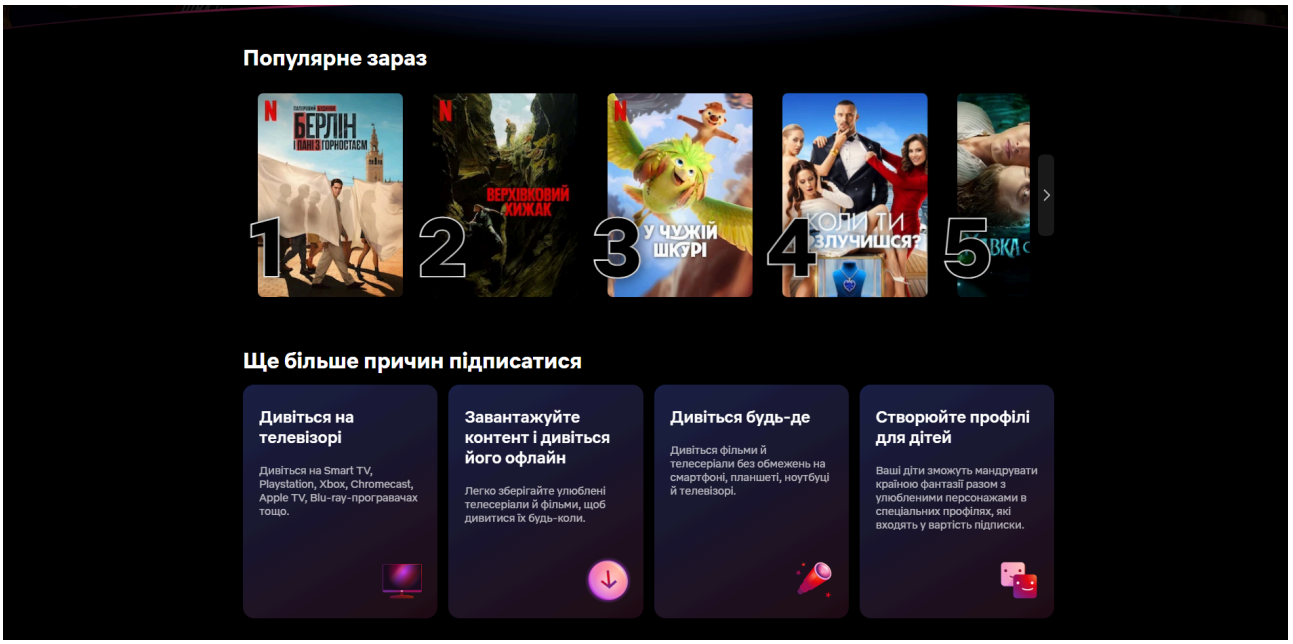



Рис. 1.1 Скріншот Netflix

Основні переваги Netflix:

1. Високий рівень зручності та продуманості інтерфейсу користувача;
2. Адаптація під різні пристрої смартфони, пк, планшети та телевізори;
3. Велика бібліотека з контентом та наявність оригінальних проєктів;
4. Розвинена система персоналізованих рекомендацій для користувача;
5. Стабільна робота сервісу та швидкий доступ до контенту.

Основні недоліки Netflix:

1. Платна модель доступу;
2. Обмеження доступності контенту залежно від вашого регіону;
3. Потенційне перевантаження нового користувача, бо контенту дуже багато;

### 1.3.2 IMDb

IMDb - одна з найбільших онлайн баз даних кіноіндустрії світу, вона містить інформацію про фільми, серіали, акторів, режисерів і так далі. Сервіс виступає як

інформаційний портал, що дозволяє переглядати вище сказану інформацію. Також можна залишати власні відгуки та оцінки. Завдяки цьому IMDb являється одним із ключових джерел інформації про кіноіндустрію.

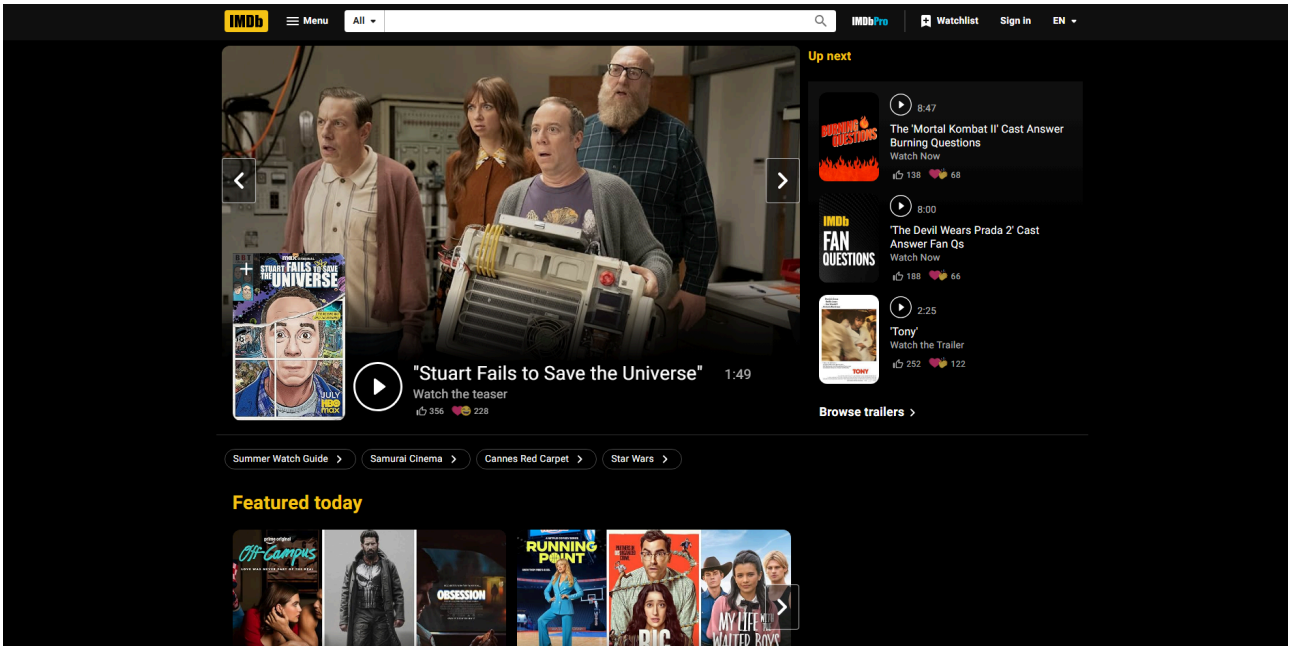



Рис. 1.2 Скріншот IMDb

Основні переваги IMDb:

1. Велика та структурована онлайн база;

2. Наявність детальної інформації про кожен фільм. Така як: акторський склад, команда, жанри, технічні дані;

3. Доступ до рейтингів і відгуків користувачів;

4. Охоплення як самих свіжих релізів, так і класичних фільмів;

5. Можливість отримання довідкової інформації про кіноіндустрію в цілому.

Основні недоліки IMDb:

1. Перевантажений інтерфейс великою кількість інформації на сторінках, таким чином новий користувач може заблукати;

2. Складність навігації для нових користувачів;

3. Менш зручна система персоналізованих списків перегляду;

### 1.3.3 Letterboxd

Letterboxd - соціальна платформа орієнтована на кінолюбителів всього світу, яким цікаво обговорити або подивитися думку інших про фільм, серіал та інше. На платформі можна створювати власні списки, ділитись ними та дивитись списки інших користувачів. Тобто, хороший сервіс для тих кому треба бачити думки інших про переглянуте чи поділитись власним враженнями після перегляду. Має простий та мінімалістичний інтерфейс, він зрозумілий та не буде перевантажувати нового користувача зайвою інформацією.

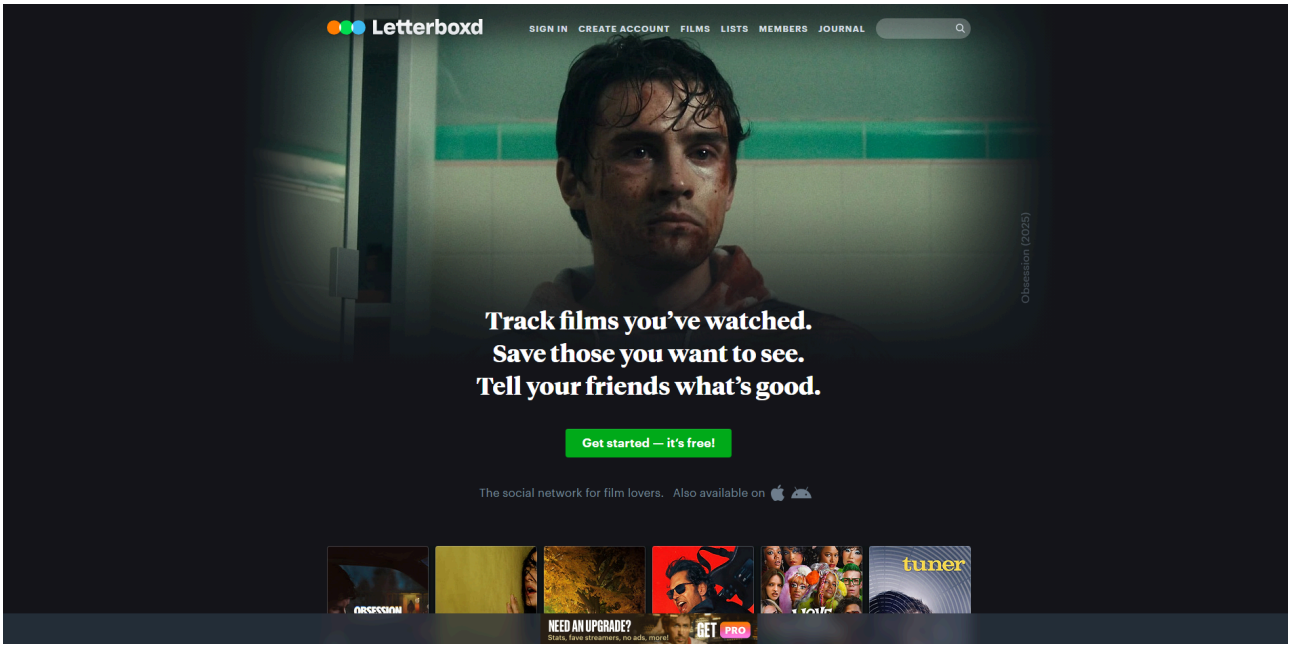



Рис. 1.3 Скріншот Letterboxd

Переваги Letterboxd:

1. Соціальна складова та взаємодії між користувачами;
2. Можливість створення персональних списків перегляду;
3. Простий і мінімалістичний інтерфейс;
4. Активна спільнота користувачів;

Недоліки Letterboxd:

1. Можливості пошуку та фільтрації у порівнянні з конкурентами;
2. Менш гнучка робота з великими каталогами фільмів;
3. Акцент все ж таки йде на соціальну взаємодію

### 1.3.4 TMDB

TMDB - надає безкоштовний API, який надає доступ до великої та постійно оновлюваної бази даних з фільмами, серіалами та інше. Його використовує велика кількість розробників веб та мобільних застосунків, як джерело даних про кіно.

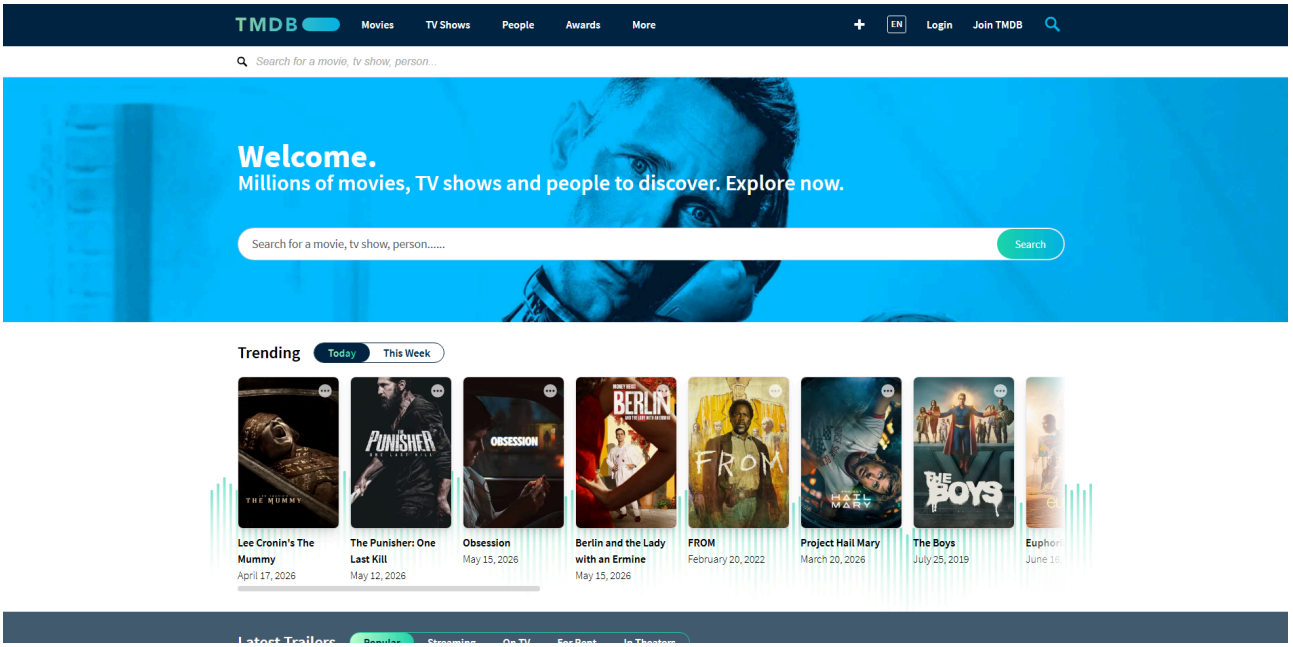



Рис. 1.4 Скріншот TMDB

Основними перевами TMDB є:

1. Безкоштовний доступ до API.
2. Велика база даних яку постійно оновлюють.
3. Наявність детальної інформації: описи, рейтинги, жанри, постери.
4. Можливість інтеграції у власні застосунки.
5. Гнучкість використання даних.

Основними недоліками TMDB є:

1. Відсутність готового користувацького інтерфейсу.
2. Необхідність самостійної реалізації клієнтської частини.
3. Залежність від зовнішнього API.
4. Необхідність обробки та структурування даних зі сторони розробника.

У таблиці 1.1 було порівняно аналоги розробленого застосунку за такими характеристиками:

1. Персональні списки.
2. Рекомендації.
3. Авторизація користувачів.
4. Детальна інформація про фільми.
5. Адаптивність.
6. API для розробників.
7. Безкоштовний доступ.
8. Швидкий підбір фільмів.

Таблиця 1.1

Порівняльна таблиця

| **Характерис-тика** | **Netflix** | **IMDb** | **Letterboxd** | **TMDB** |
|---|---|---|---|---|
| Персональні списки | + | + | + | Лише через сторонні сервіси |
| Рекомендації | + | + | + | - |
| Авторизація користувачів | + | + | + | - |
| Детальна інформація про фільми | Основна інформація | Дуже детально | Огляди + Базові дані | Повні метадані |
| Адаптивність | + | + | + | - |
| API для розробників | Закритий | Обмежений | Немає | Повноцінне API |
| Безкоштовний доступ | Підписка | Частково | Freemium | Безкоштовний API |
| Швидкий підбір фільмів | Рекомендації | Пошук вручну | Через списки | Через API |

## 1.4 Вимоги до системи

Перед початком проєктування будь якого застосунку треба визначати вимоги до нього. Це допоможе чітко сформувати межі для функціоналу, та мати очікувану поведінку додатка, для перевірки між задуманим та реалізованим. Тому сформував функціональні та нефункціональні вимоги проєкту.

### 1.4.1 Функціональні вимоги

Функціональні вимоги визначають основні можливості системи:

1. Пошук фільмів за назвою.
2. Реєстрація нових користувачів.
3. Авторизація користувачів.
4. Відображення результатів пошуку у вигляді карток фільмів.
5. Перегляд інформації про обраний фільм. Має містити: опис, рейтинг, жанри, рік випуску та постер.
6. Персональний список перегляду.
7. Додавання фільмів до списку.
8. Видалення фільмів зі списку.
9. Можливість вести список перегляду лише у зареєстрованих користувачів.
10. Фільтрація за жанрами, роком випуску та мінімальним рейтингом.
11. Сортування нові, популярні та найкращі.

### 1.4.2 Нефункціональні вимоги

Нефункціональні вимоги визначають якість роботи та технічні характеристики системи:

1. Висока швидкість обробки запитів менше 2 секунд.
2. Асинхронна обробка запитів до зовнішнього сервісу.
3. Логічна структура інтерфейсу без перевантаження елементами.
4. Адаптивність під різні пристрої такі як пк, планшети та смартфони.
5. Коректне відображення на різних розмірах екранів від 320 px.

6. Забезпечення безпеки користувацьких даних.

7. Зберігання паролів у захешованому вигляді.

8. Використання JWT-токенів для авторизації.

9. Масштабованість системи для подальшого розширення функціоналу.

### 1.5 UML-опис системи

Було проведено UML аналіз для того щоб, краще зрозуміти структуру, логіку та поведінку застосунку. Вони використовуються в програмній інженерії на етапі проєктування. Метою є візуалізація ключових аспектів ще до реалізації, це дозволяє краще зрозуміти взаємодію компонентів та зменшити шанс допускання помилок у майбутньому.

У проєкті використовуються наступні UML-діаграми:

1. Діаграма варіантів використання.

2. Діаграма компонентів.

3. Діаграма послідовностей.

#### 1.5.1 Діаграма варіантів використання

Use Case діаграма відображає основні функції користувача, такі як пошук фільмів, перегляд детальної інформації, авторизація та управління списком перегляду. Основним актором системи є користувач. Він взаємодіє з системою через браузер і має доступ до основних функцій.

До основних варіантів використання належать:

1. Пошук фільмів, серіалів та інше.

2. Перегляд інформації про них.

3. Перегляд схожих фільмів.

4. Реєстрація в системі.

5. Авторизація користувача.

6. Додавання фільму до списку перегляду.

7. Видалення фільму зі списку.

8. Застосування фільтрів.

9. Сортування результатів.

Взаємодія користувача з системою відбувається через інтерфейс клієнтської частини. А она взаємодіє з сервером через HTTP-запити.

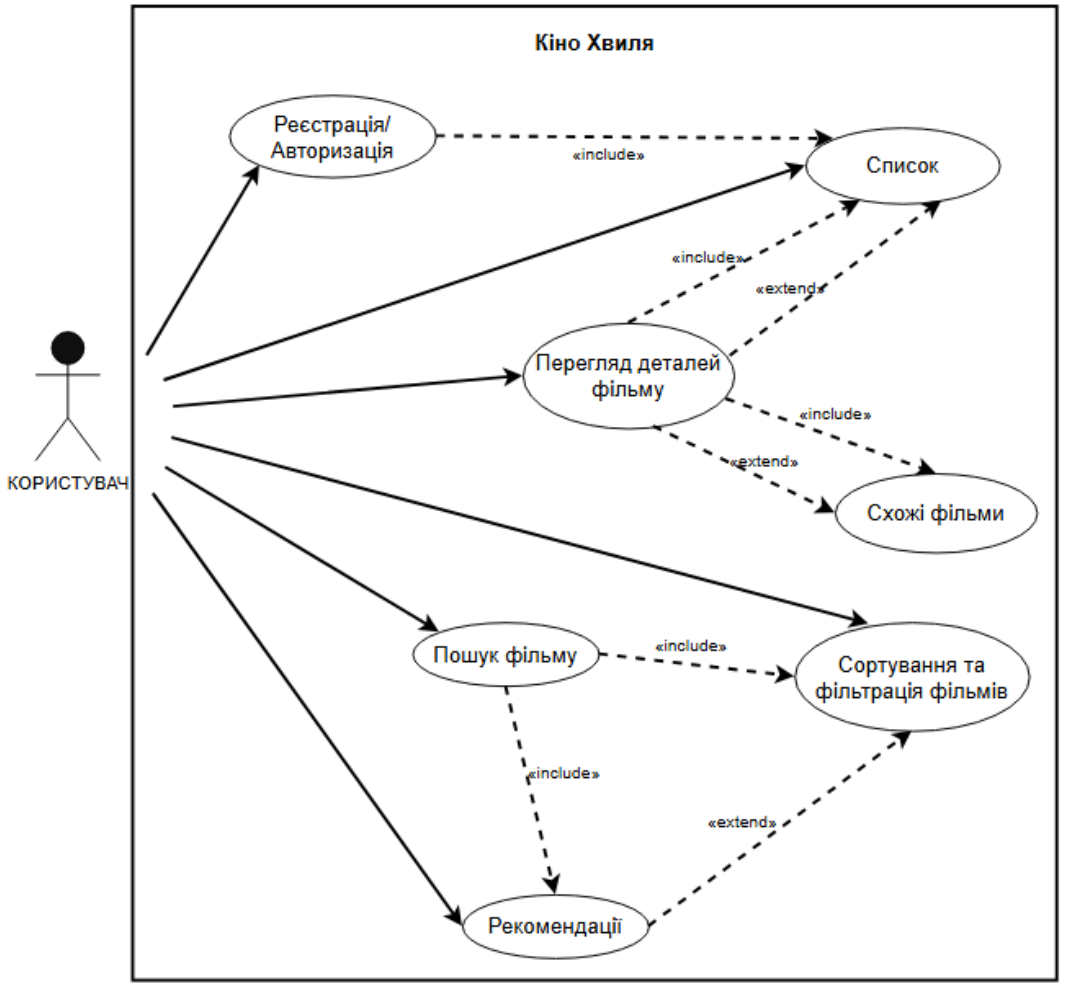



Рис. 1.5 Діаграма варіантів використання

### 1.5.2 Діаграма компонентів

Діаграма компонентів демонструє загальну архітектуру застосунка та взаємодію між його основними частинами. Архітектура застосунка побудована за принципом клієнт-серверної моделі. Клієнтська частина відповідає за відображення інтерфейсу та обробку взаємодії користувача. Вона реалізована на JavaScript та працює в браузері. Серверна частина обробляє запити від

користувача працює з базою даних та передає інформацію клієнтській частині. База даних зберігає в собі користувачів і списки перегляду.

API TMDB використовується для отримання актуальної інформації про фільми, серіали та інше.

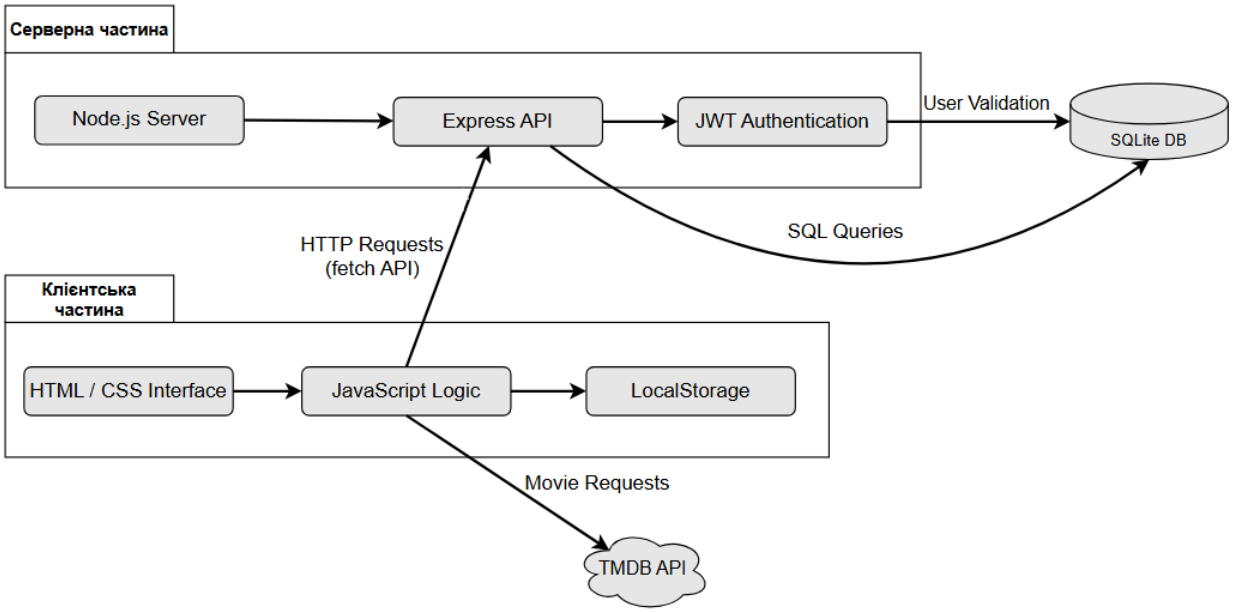



Рис. 1.6 Діаграма компонентів

### 1.5.3 Діаграма послідовності

Діаграма послідовності показує послідовність запитів між користувачем, клієнтською, серверною частиною та базою даних.

Роботу системи можна описати таким сценарієм:

1. Користувач відкриває головну сторінку.

2. Клієнтська частина надсилає запит до API для отримання списку фільмів.

3. Система відображає отримані дані у вигляді карток з фільмами.

4. Користувач може вибрати фільм для перегляду інформації.

5. Якщо натискаєш на кнопку <Додати до списку> система перевіряє, чи авторизований користувач.

6. Якщо не авторизований, відкривається форма Входу / Реєстрації.

7. Якщо авторизований, дані про фільм передаються на сервер і зберігаються в базі даних.

8. При наступному вході користувача система завантажує список перегляду з бази даних.

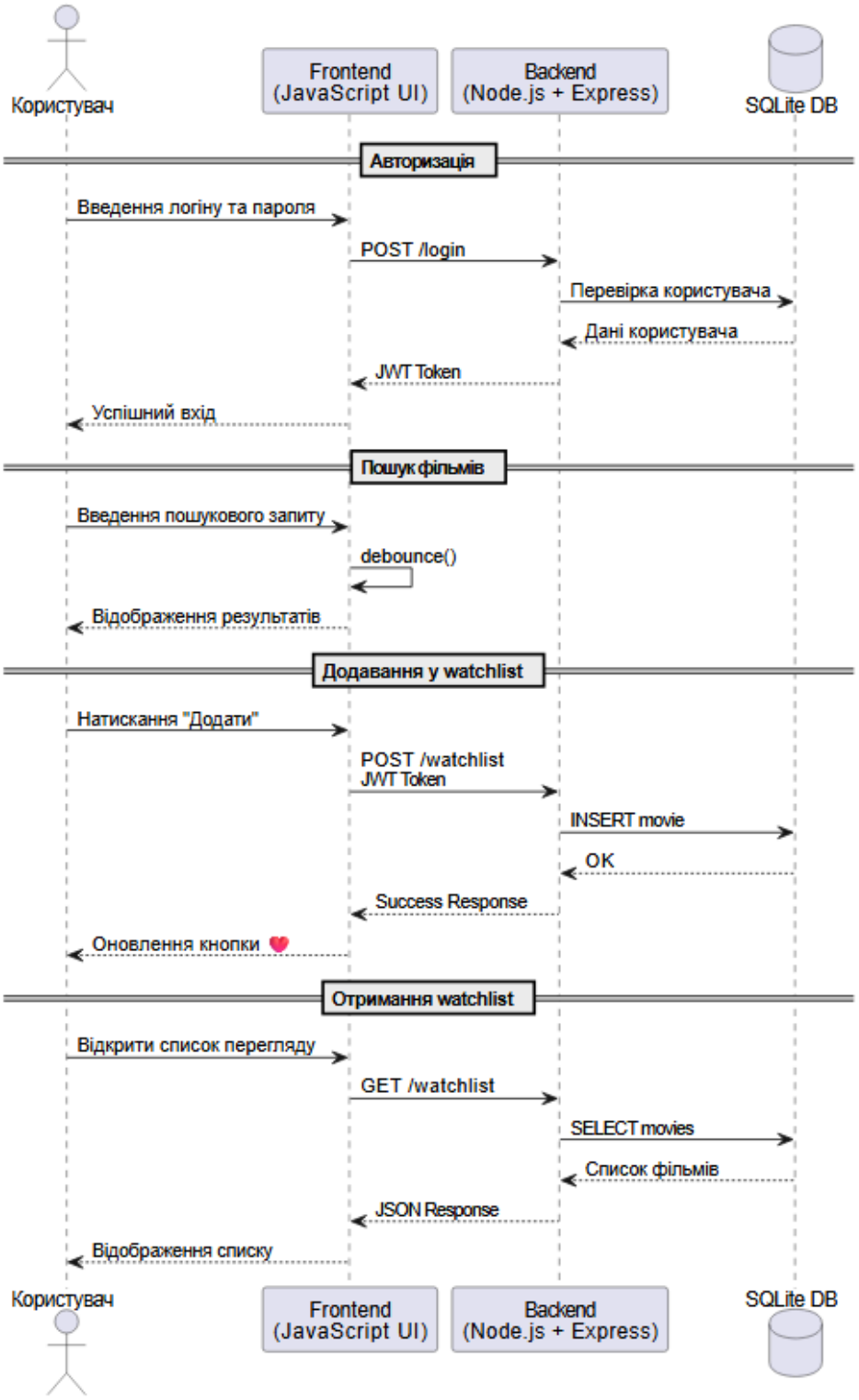



Рис. 1.7 Діаграма послідовності

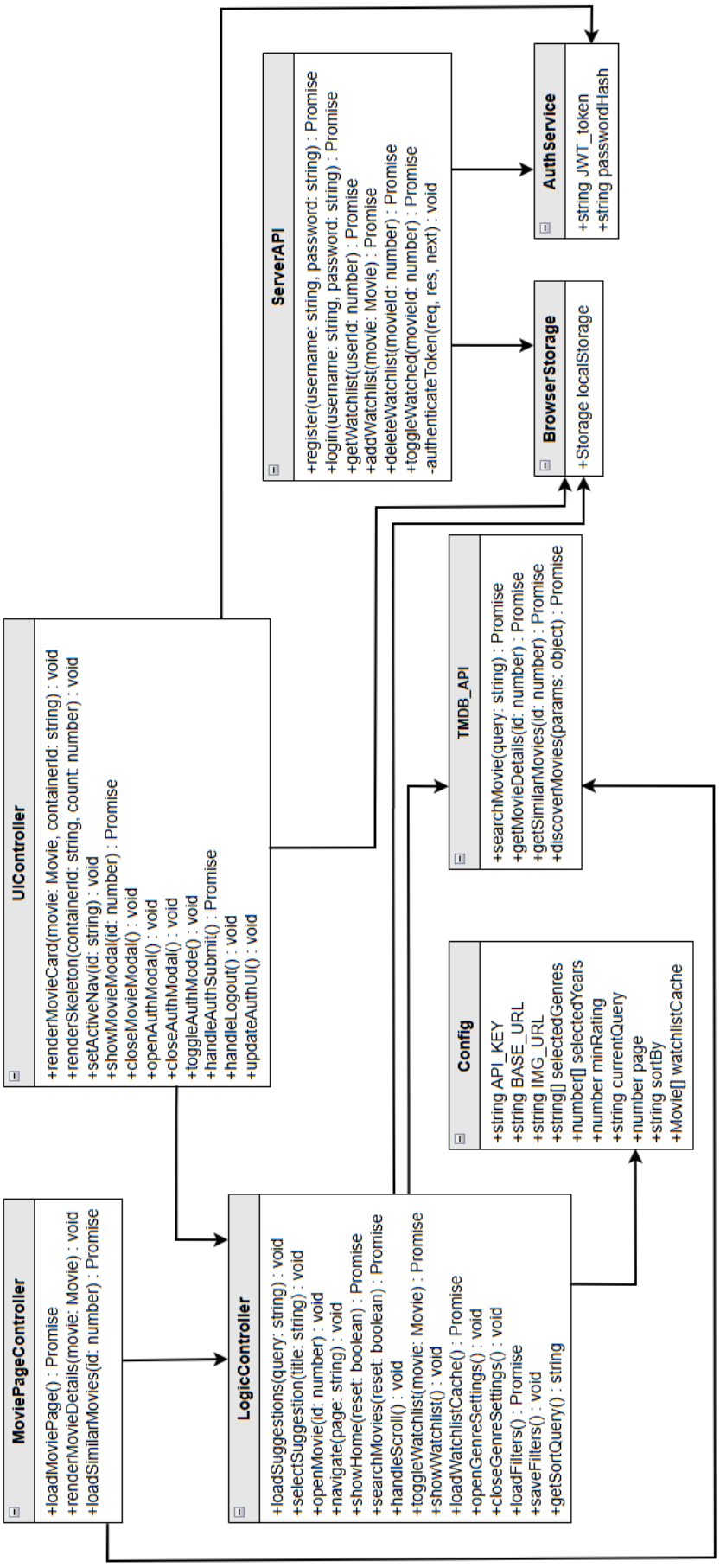



Рис. 1.8 Діаграма логічних модулів системи

# 2 ЗАСОБИ РЕАЛІЗАЦІЇ

## 2.1 Середовище розробки

Процес розробки вимагав сучасного, гнучкого та багатофункціонального середовища для розробки. В якому можна було б ефективно реалізувати як серверну, так і клієнтську частину. Тому, середовищем розробки обрав Visual Studio Code, який є найпопулярнішим редактором коду. Його використовують як професіонали, так і новачки, тому що має гнучкі налаштування, високу продуктивність та велику кількість розширень для спрощення розробки.

В процесі розробки використовувалися такі можливості та інструменти в Visual Studio Code:

1. Розширення для роботи з JavaScript та Node.js.
2. Live Server для швидкого перегляду змін у браузері.
3. Вбудований термінал для запуску серверної частини.

```
PS C:\Users\thelman\Desktop\movie 19.05> node server.js
Server running on http://localhost:3000
Connected to SQLite database
```

Рис. 2.1 node server.js

Використання цих інструментів дозволило прискорити процес розробки та зробити код зручнішим для підтримки. Node.js важлива частина середовища розробки, так як використовується для реалізації серверної частини. Завдяки ньому сервер обробляє HTTP-запити, працює з базою та виконує основну бізнес-логіку системи. Його асинхронна архітектура дозволяє ефективно працювати з великою кількістю запитів.

Тож, таке поєднання забезпечує зручне середовище для розробки, тестування та підтримки застосунку.

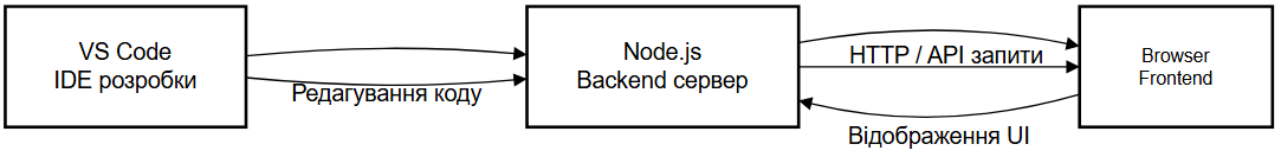



Рис. 2.2 Схема середовища розробки

## 2.2 Мови програмування

Для розробки додатку використовувались сучасні мови, вони є стандартом на даний момент та дозволять створити повноцінний додаток. Вибір технічного стеку був обумовлений необхідністю створення швидкого та зручного застосунку. Використання однієї мови програмування дозволяє зменшити складність архітектури та підтримки коду у майбутньому.

### 2.2.1 JavaScript

Основна мова програмування проєкту JavaScript, вона використовується для клієнтської та серверної частини. Це дозволяє використовувати її для всієї системи і спрощує як розробку, так і підтримку. На клієнтській стороні відповідає за роботу інтерфейсу, натискання кнопок, введення тексту, вибір фільтрів та відкриття модальних вікон. Дає можливість оновлення сторінки без її перезавантаження. Щоб отримувати дані використання використовуються fetch API, дозволяє звертатись к зовнішньому сервісу TMDB. На серверній стороні через Node.js, використовується для створення API, обробки запитів та реалізації реєстрації/авторизації і керування списком перегляду.

### 2.2.2 HTML та CSS

Для створення структури сторінок використовується мова розмітки HTML (HyperText Markup Language). Вона визначає логічну організацію елементів інтерфейсу. Таких як заголовки, кнопки, поля введення, списки фільмів, модальні вікна та інші компоненти інтерфейсу. HTML забезпечує основу всієї клієнтської

частини застосунка, оскільки саме він визначає, які елементи будуть відображатися користувачу та в якій послідовності.

Для оформлення зовнішнього вигляду використовується CSS (Cascading Style Sheets), який відповідає за візуальне представлення елементів інтерфейсу сторінок. У межах даного проєкту CSS використовується для створення сучасного та адаптивного дизайну. Який однаково коректно відображається на різних пристроях пк чи смартфонах. Було реалізовано стилізацію карток фільмів, кнопок взаємодії, панелі фільтрів, модальних вікон. А також ефектів наведення та анімацій завантаження. Особлива увага приділялася щоб новому користувачу все було інтуїтивно зрозуміло тобто UX-дизайну - зручності використання інтерфейсу.

Поєднання HTML та CSS дозволило створити повноцінний користувацький інтерфейс, який є інтуїтивно зрозумілим та не потребує додаткового навчання.

### 2.3 Веб-фреймворки

У серверній частині застосунка використовується Express.js - мінімалістичний та гнучкий фреймворк для Node.js. Який значно спрощує створення веб-серверів та REST API.

Express.js є простим та модульним. Розробник самостійно визначає структуру застосунка, маршрути та логіку обробки запитів. Це дозволяє створювати легкі та продуктивні серверні рішення без зайвого перевантаження функціоналом. В межах проєкту Express.js використовується для реалізації API-ендпоінтів таких як:

1. Реєстрація користувача.
2. Авторизація.
3. Отримання списку.
4. Додавання фільму до списку.
5. Видалення фільму зі списку.

У проєкті використовується бібліотека JSON Web Token (JWT). Забезпечує механізм автентифікації користувачів. Після успішного входу користувач отримує токен, який передається у кожному запиті до сервера. Це дозволяє ідентифікувати користувача та забезпечити захист його даних. Використання Express.js у поєднанні з JWT дозволяє реалізувати безпечну та масштабовану серверну архітектуру.

### 2.4 База даних

Для збереження даних у проєкті використовується SQLite - легка файлова реляційна база даних. Не потребує окремого серверного програмного забезпечення. Це робить її оптимальним рішенням для навчальних та середніх за масштабом веб-застосунків.

SQLite є простим в налаштувані та використані. База даних зберігається у вигляді одного файлу, що значно спрощує розгортання проєкту та його перенесення на інші пристрої. У межах застосунка використовується для зберігання двох основних сутностей:

1. Користувачів
2. Списків перегляду

Кожен запис у таблиці списку перегляду має зв’язок із конкретним користувачем через його id, що дозволяє реалізувати персоналізацію даних. Таким чином, кожен користувач бачить лише свій власний список фільмів.

Паролі користувачів не зберігаються у відкритому вигляді. Для їх захисту була використана бібліотека bcryptjs. Виконує хешування паролів перед записом у базу. Це означає, що навіть у випадку доступу до бази даних сторонні особи не зможуть отримати оригінальні паролі користувачів.

Також база даних підтримує унікальність користувачів за username, що запобігає дублюванню акаунтів.

# 3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ

## 3.1 Проєктування архітектури застосунка

Архітектура застосунка побудована за моделлю клієнт-сервер, яка є поширеною у сучасній веб-розробці. В ній клієнтська частина відповідає за взаємодію з користувачем. А серверна, за обробку бізнес-логіки та роботу з даними. Клієнтська частина виконується у браузері користувача і реалізована за допомогою HTML, CSS та JavaScript. Відповідає за відображення інтерфейсу, динамічне оновлення сторінок, обробку подій користувача, та взаємодію з сервером через HTTP-запити.

Серверна частина побудована на платформі Node.js з використанням фреймворку Express.js. Вона виконує роль API-шару. Обробляє запити клієнта, виконує операції з базою SQLite та забезпечує механізм авторизації користувачів.

Зовнішнім компонентом є API TMDB, який використовується як основне джерело даних про фільми. Завдяки цьому застосунок не потребує власного сховища медіаданих, що значно зменшує навантаження на сервер і спрощує архітектуру.

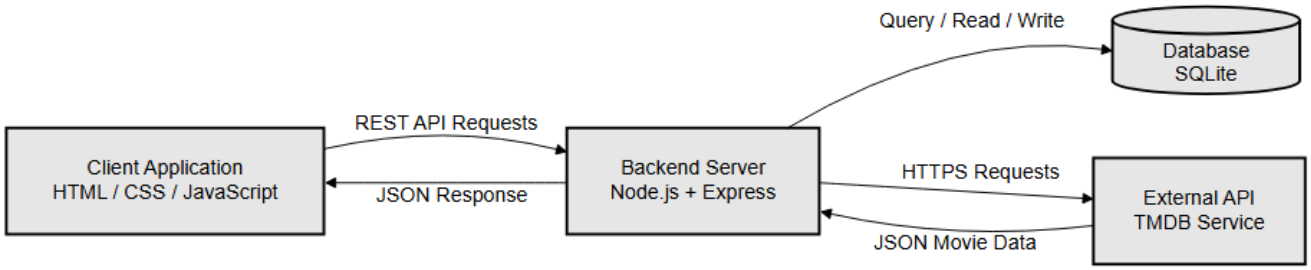



Рис. 3.1 Діаграма компонентів системи

### 3.1.1 Архітектура клієнтської частини

Клієнтська частина застосунка є найбільш об’ємною з точки зору логіки взаємодії з користувачем. Вона реалізована у вигляді набору JavaScript-модулів.

Кожен з яких виконує окрему функціональну роль. Такий підхід дозволяє реалізувати принцип розділення відповідальностей, що є важливим для підтримки та масштабування проєкту. Кожен з модулів відповідає за власну частину функціоналу та не дублює логіку інших компонентів системи.

Основні модулі клієнтської частини мають наступне призначення:

- config.js - містить глобальні конфігураційні параметри застосунка. Сюди входять ключ API TMDB, базові URL-адреси, а також змінні стану. Такі як поточна сторінка, активні фільтри, параметри сортування та пошуковий запит.
- logic.js - відповідає за основну бізнес-логіку застосунка. У ньому реалізовано функції пошуку фільмів, завантаження даних з API, роботу з фільтрами, пагінацію та нескінченну прокрутку. Саме тут формується логіка запитів до TMDB та обробка отриманих результатів.
- ui.js - відповідає за формування та оновлення інтерфейсу користувача. У цьому модулі створюються картки фільмів, модальні вікна, елементи завантаження та обробка взаємодії користувача з інтерфейсом. Відокремлення UI-логіки від бізнес-логіки дозволяє спростити підтримку та зміну дизайну.
- script.js - виконує роль ініціалізаційного модуля. Він відповідає за запуск основних процесів після завантаження сторінки, підключення обробників подій, ініціалізацію пошуку, фільтрів, сортування та авторизації користувача.

Механізм збереження стану фільтрів у localStorage. Це дозволяє зберігати налаштування користувача навіть після перезавантаження сторінки, забезпечуючи більш зручний та персоналізований досвід використання застосунка.

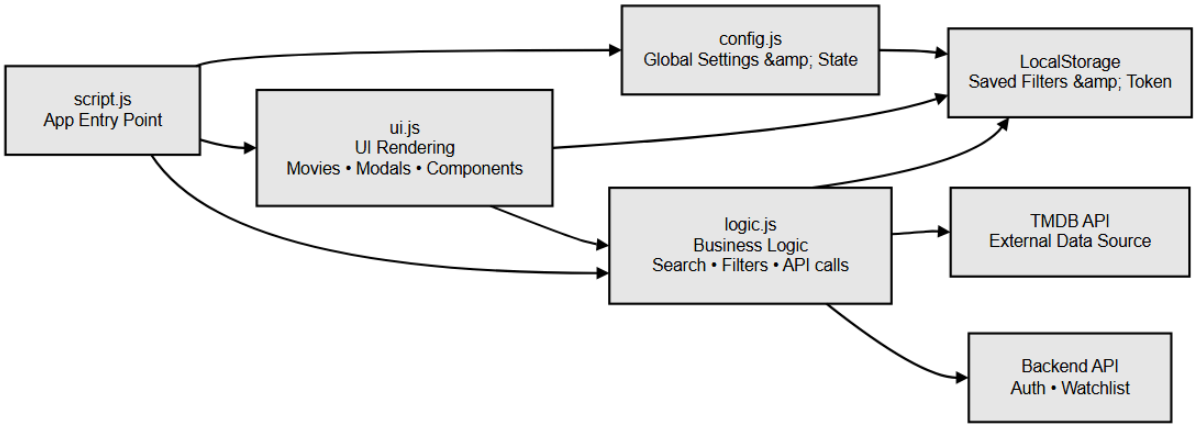



Рис. 3.2 Діаграма компонентів клієнтської частини

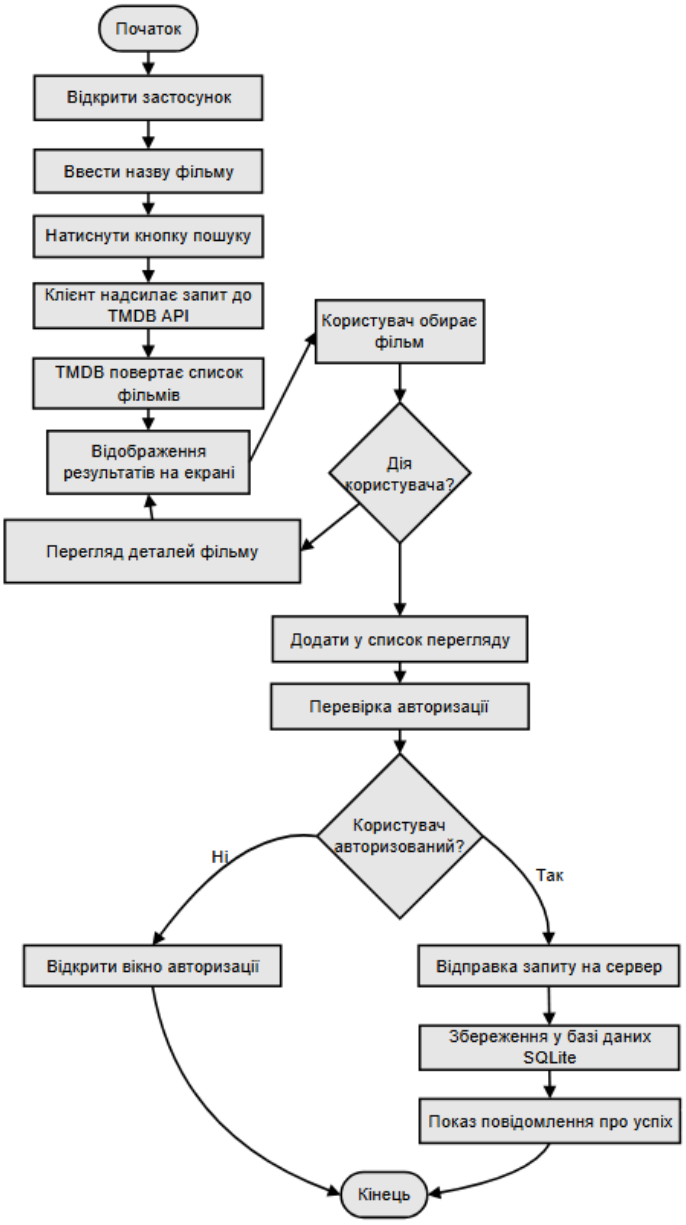



Рис. 3.3 Діаграма активностей

### 3.1.2 Архітектура серверної частини

Серверна частина застосунка побудована на базі Node.js та фреймворку Express.js. Це забезпечує легкість створення REST API та високу продуктивність при обробці HTTP-запитів. Основною задачею серверної частини є забезпечення логіки роботи з користувачами та їх персональними списками. Сервер реалізує такі основні функції: реєстрація користувачів, авторизація, перевірка JWT-токенів, додавання фільмів до списку перегляду, видалення фільмів та отримання списку перегляду для конкретного користувача додатку.

Для безпеки використовується механізм JSON Web Token (JWT). Дозволяє ідентифікувати користувача без необхідності постійного звернення до бази для перевірки сесії. Токен передається у заголовках HTTP-запитів і перевіряється middleware функцією authenticateToken. Обробка паролів користувачів. Вони не зберігаються у відкритому вигляді, а хешуються за допомогою бібліотеки bcryptjs. Це дозволяє забезпечити базовий рівень захисту персональних даних користувачів. Сервер також виконує роль посередника між клієнтом та базою даних SQLite. Він не зберігає складної логіки, а лише координує операції читання та запису.

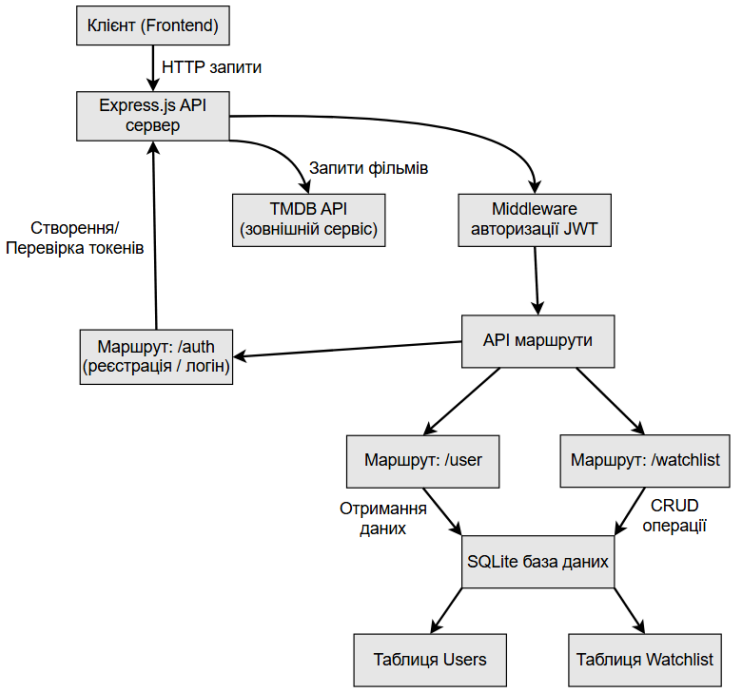



Рис. 3.4 Діаграма серверної архітектури

### 3.1.3 Архітектура бази даних

База даних застосунка реалізована за допомогою SQLite, що є компактним та простим у використанні рішенням для навчальних і невеликих комерційних проєктів. Основною перевагою SQLite є відсутність необхідності запуску окремого серверу. Оскільки база зберігається у вигляді локального файлу.

Структура бази даних складається з двох основних таблиць: users та watchlist. Така структура є мінімально достатньою для реалізації функціоналу авторизації та персонального списку фільмів.

Таблиця users містить унікальний ідентифікатор користувача, ім’я користувача та пароль у захешованому вигляді. Обмеження унікальності поля username гарантує, що кожен користувач у системі має унікальний обліковий запис. Таблиця watchlist зберігає інформацію про фільми, які користувач додає до свого списку перегляду. Вона містить зв’язок із таблицею користувачів через поле user_id. Що забезпечує логічну ізоляцію даних між різними акаунтами.

Додатково у таблиці передбачено поле watched, яке дозволяє розширити функціональність системи в майбутньому. Наприклад для реалізації переглянутих фільмів чи створення рекомендаційної системи.

Таким чином, база має просту, але ефективну структуру, яка повністю відповідає вимогам застосунка і може бути легко розширена при необхідності.

## 3.2 Проєктування UI/UX дизайну

Проєктування важливий етап, так як маємо надати користувачу зручну взаємодію із застосунком. Зараз у сучасних веб-застосунках важлива не лише функціональність, а також простота, швидкість роботи та візуальна зрозумілість.

UI (User Interface) відповідає за зовнішній вигляд системи. UX (User Experience) - за зручність використання. Тому під час розробки увага приділялася простій навігації, щоб розташування елементів було логічним та мінімізувати дії користувача.

Метою було створення простого та інтуїтивно зрозумілого інтерфейсу. Який не потребує додаткового навчання, користувач повинен швидко отримувати доступ до основних функцій:

1. Пошуку фільмів.
2. Перегляду інформації.
4. Фільтрації.
5. Списку перегляду.

Використано мінімалістичний підхід. Інтерфейс не перевантажений зайвими елементами, акцент зроблено на контенті. Для відображення фільмів використано формат карток, що дозволяє швидко переглядати постери, назву та рейтинг.

У застосунку використовується темна тема, вона є зручною для тривалого використання без навантаження на очі та краще підкреслює візуальний частину контенту. Акцентні елементи виділяються червоним кольором. Це допомагає користувачу швидко знаходити основні дії. Інтерфейс єдиний за стилем всі поля, сторінки модальні вікна оформленні за однаковим стилем.

Для взаємодії з контентом використовуються модальні вікна, вони відкриваються без переходу на інші сторінки. Це дозволяє користувачу швидше отримувати інформацію. Реалізовано нескінченну прокрутку (infinite scroll) замість пагінації, це робить перегляд контенту більш зручним. Пошук оптимізований за допомогою debounce, щоб зменшити кількість запитів до API. Також використовується індикація завантаження (skeleton loading), показує процес отримання даних. Застосунок є адаптивним під різні розміри екранів: пк, планшетах і смартфонах. Для цього використано CSS media queries, flexbox і grid. На мобільних пристроях інтерфейс перебудовується під вертикальний формат для зручності використання.

Реалізований UI/UX дозволив створити простий, сучасний і зручний інтерфейс, який забезпечує комфортну роботу із застосунком.

### 3.2.1 Порівняння Desktop та Mobile версії

Екран головної сторінки розроблений таким чином щоб не відволікати користувача від контенту. Для цього використовуються доволі великі картки фільмів з постером, назвою, роком випуску та рейтингом.

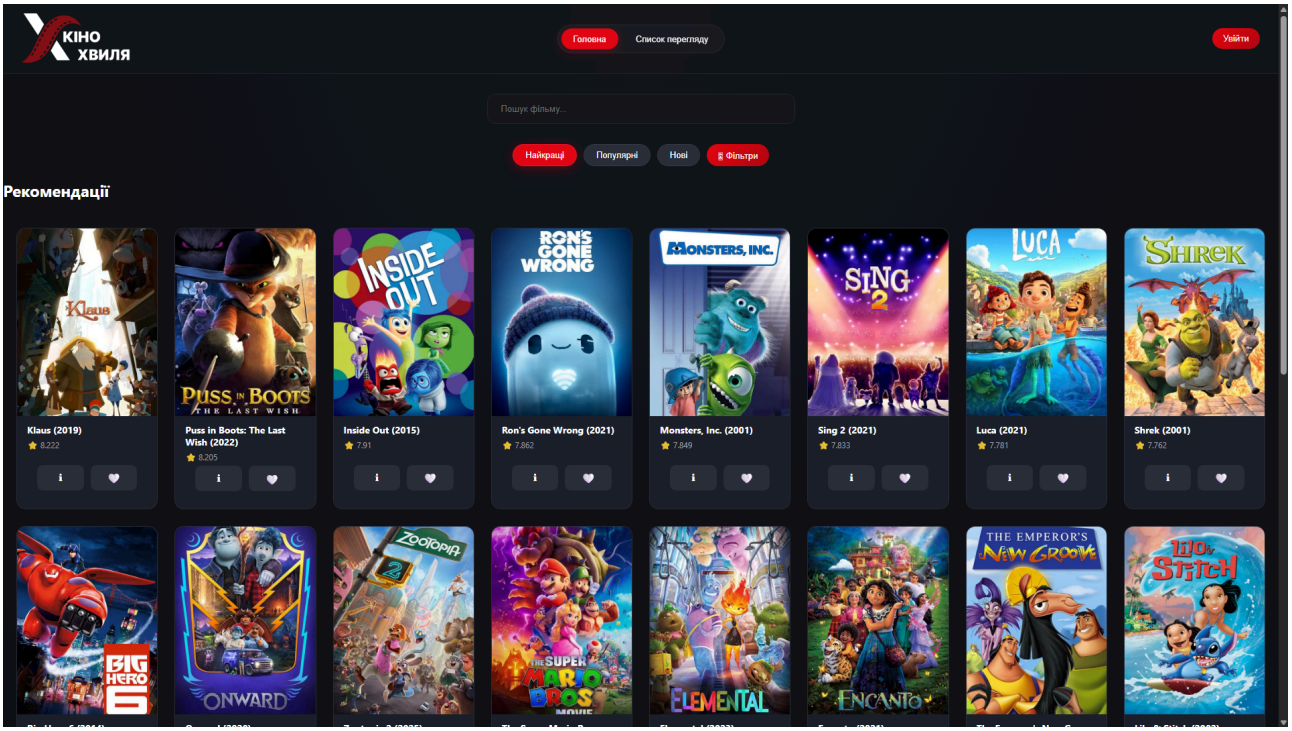



Рис. 3.5 Головна сторінка Desktop

Оскільки левова частина користувачів заходить у застосунок із мобільних пристроїв, особливу увагу приділено адаптивності інтерфейсу для них. Мобільна версія сторінки розроблена таким чином, щоб забезпечити зручний та практичний перегляд контенту на невеликих екранах без втрати функціональності.

Акцент також зроблено на відображенні постерів. Люди звертають увагу на яскраві постери більше уваги, тому фільми подаються у вигляді великих карток, що дозволяє зберегти візуальну привабливість і спростити вибір навіть під час використання на смартфоні.

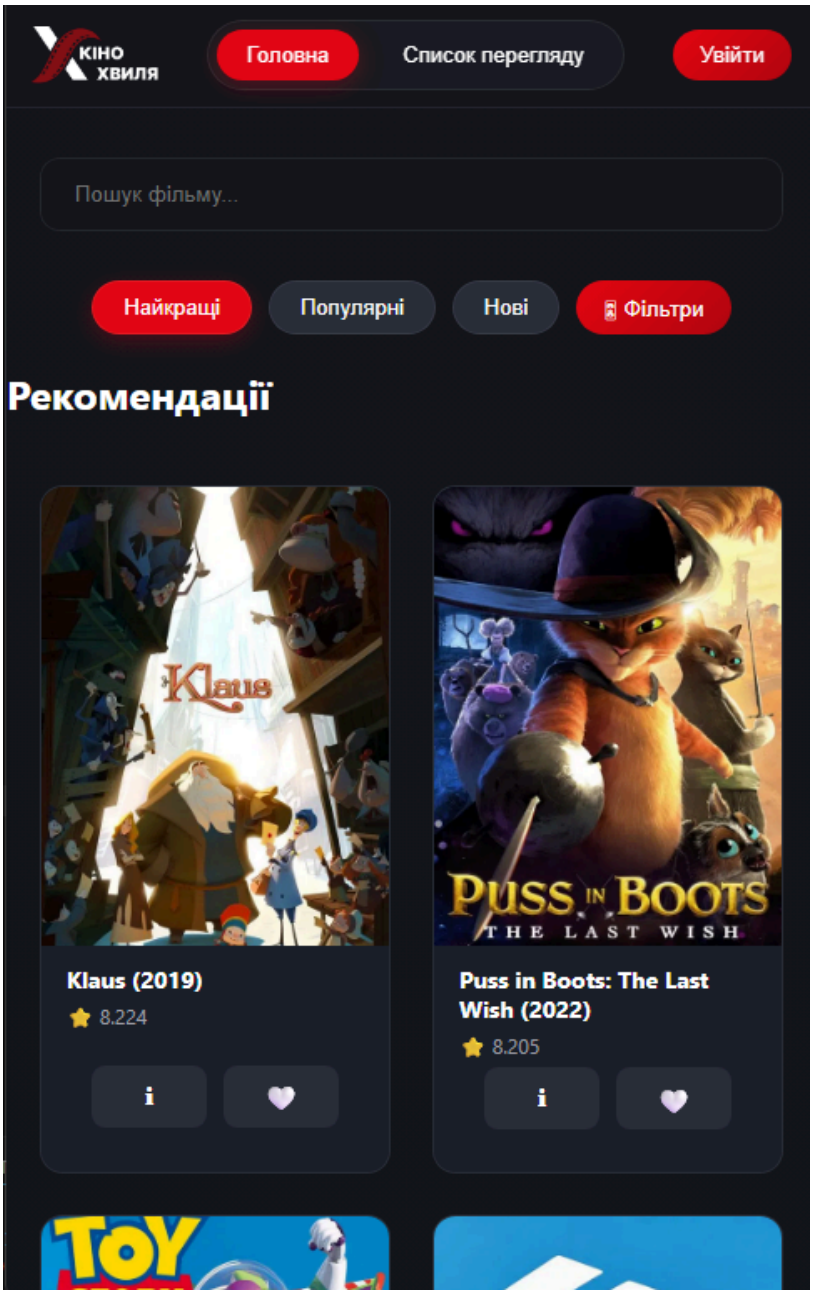



Рис. 3.6 Головна сторінка Mobile

### 3.2.2 UI-компоненти Desktop та Mobile версії

При натисканні кнопки <Інфо> на картці фільму відкривається модальне вікно. В ньому користувач може переглянути основний опис фільму. Це рішення реалізовано для того, щоб користувач мав можливість швидко ознайомитися зі змістом фільму без переходу на окрему сторінку.

У модальному вікні відображаються постер та текстовий опис на темному фоні, що дозволяє зосередити увагу саме на інформації та зменшити відволікання.

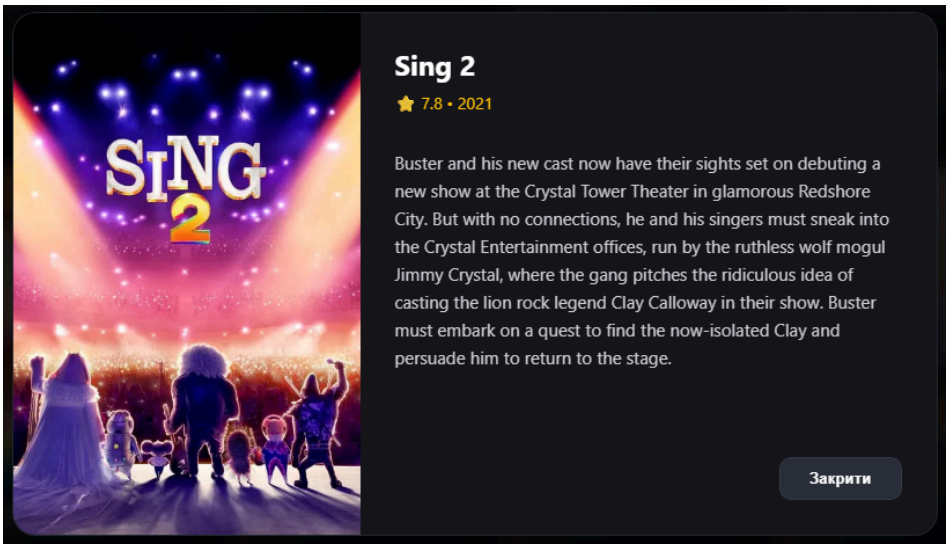



Рис. 3.7 Картка фільму Desktop

У мобільній версії інтерфейс зберігає ту саму логіку, але додатково передбачено можливість прокручування вікна. До прикладу, у випадку якщо опис фільму є занадто довгим. Це забезпечує зручність перегляду та комфортну взаємодію з контентом на невеликих екранах.

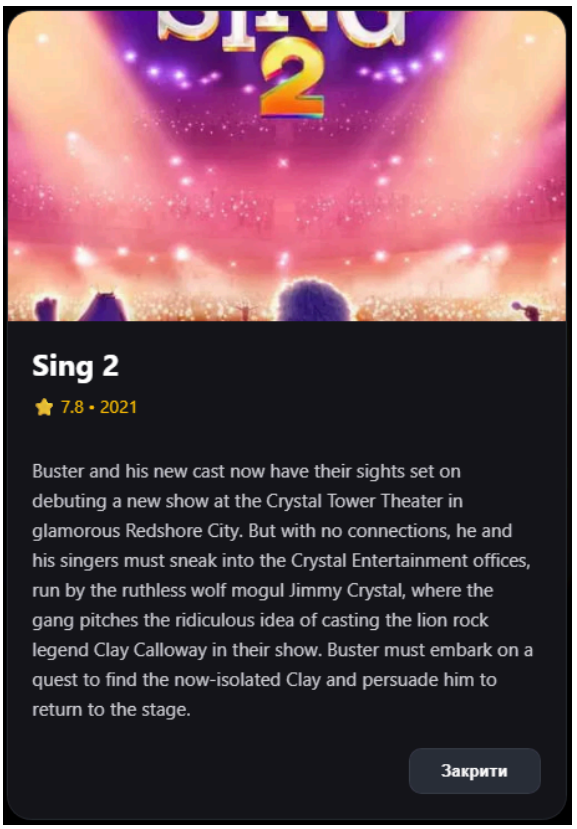



Рис. 3.8 Картка фільму Mobile

Модальне вікно фільтрів поділено на три основні блоки: жанри, мінімальна оцінка та роки випуску. Такий поділ дозволяє користувачу швидко орієнтуватися і зручно налаштовувати відображення фільмів до власних уподобань.

У стандартному стані кнопки мають нейтральний сірий колір, що не перевантажує інтерфейс візуально. Після вибору певного параметра кнопка змінює колір на червоний, для того щоб користувач одразу бачив активні фільтри. В пк версії елементи фільтрації мають збільшений розмір для кращої читабельності та зручності взаємодії. У мобільній версії збережено аналогічний підхід до відображення елементів, але реалізовано вертикальну прокрутку.

Це дозволяє уникнути надмірного зменшення кнопок. Тому що, вони могло негативно вплинути на зручність читання та використання інтерфейсу на невеликих екранах.

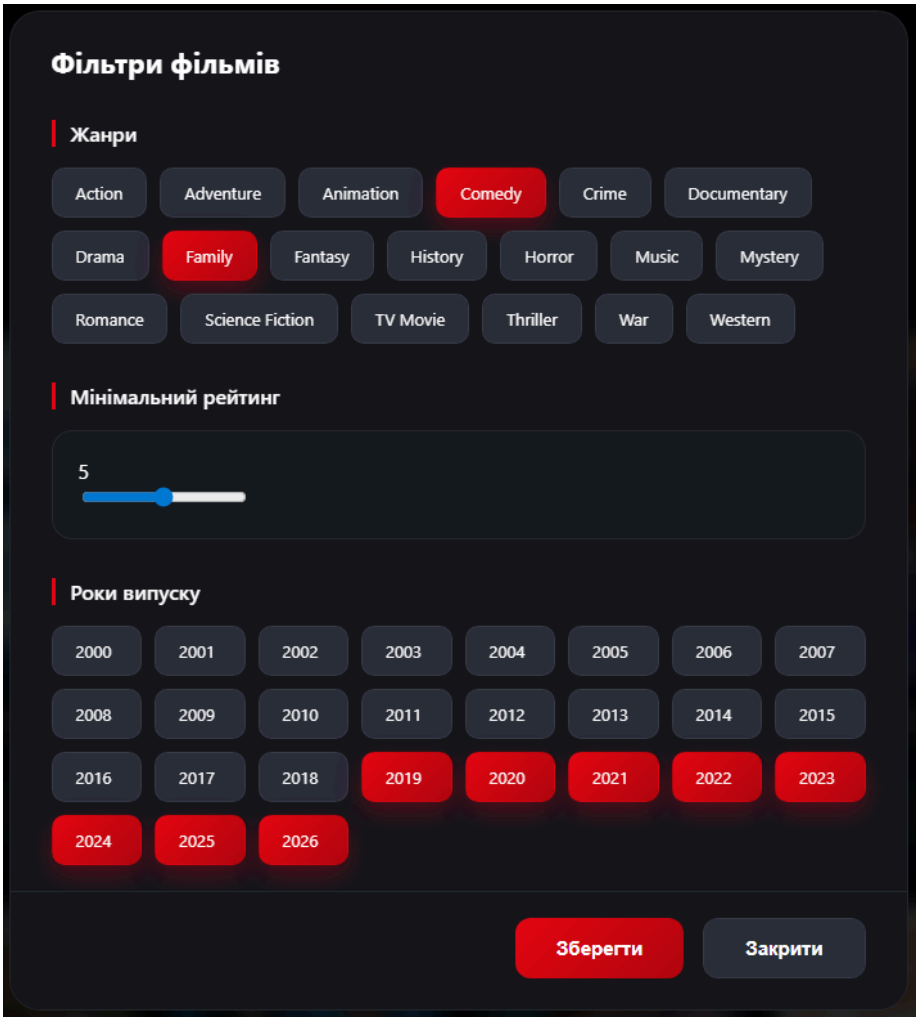



Рис. 3.9 Модальне вікно Фільтри Desktop

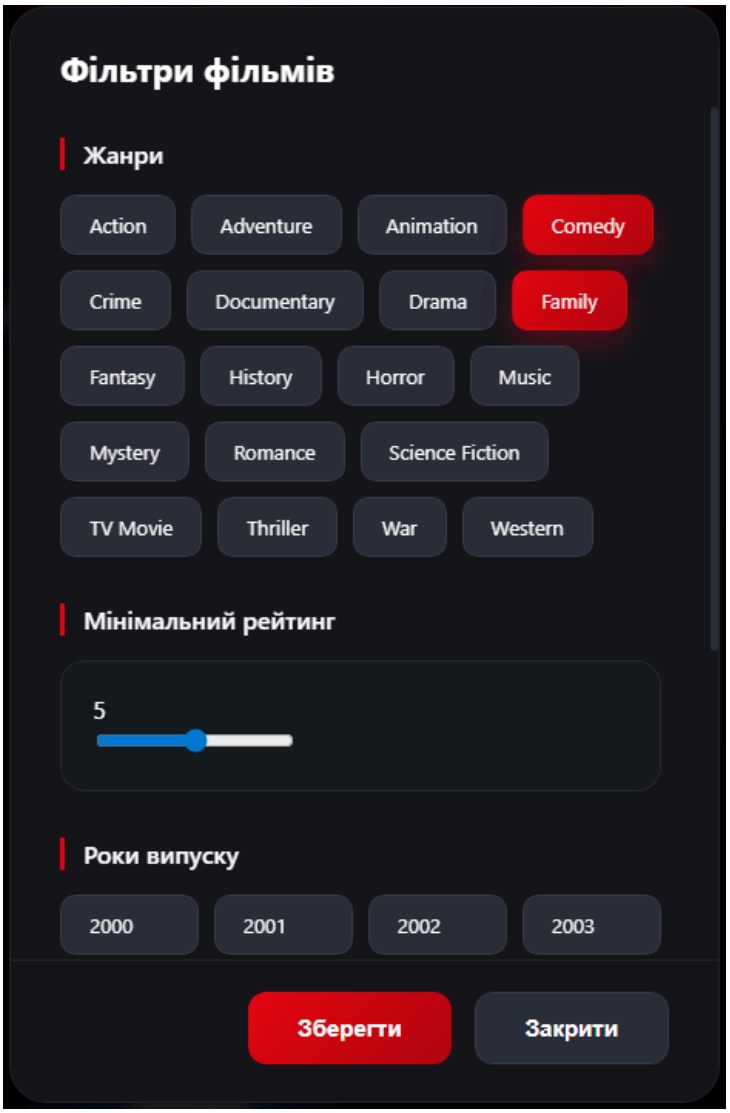



Рис. 3.10 Модальне вікно Фільтри Mobil

### 3.3 Реалізація

Реалізація застосунка була здійснена по попередньо розробленій архітектурі клієнт-серверної системи. У процесі реалізації увага приділялася практичному втіленню проєктних рішень. Забезпеченню стабільної взаємодії між клієнтською та серверною частинами, а також інтеграції зовнішнього API для отримання актуальної інформації про фільми.

Загальний процес розробки передбачав поетапне створення функціональних модулів. Починаючи від базової структури інтерфейсу і завершуючи реалізацією авторизації, роботи з базою даних та механізмом персонального списку перегляду.

Важливим принципом під час реалізації було забезпечення безперервної роботи інтерфейсу без перезавантаження сторінок. Це дозволило досягти поведінки, наближеної до SPA-застосунків.

Особливістю реалізованого рішення є те, що більшість операцій виконується асинхронно. Це дозволяє користувачу взаємодіяти із системою без затримок навіть під час виконання мережевих запитів. Це особливо важливо для операцій пошуку, фільтрації та завантаження списків фільмів, які напряму залежать від швидкості відповіді зовнішнього API.

### 3.3.1 Підготовка середовища

На початку реалізації було проведено налаштування програмного середовища, необхідного для розробки. Основою серверної частини було обрано платформу Node.js, яка дозволяє виконувати JavaScript-код поза межами браузера. Це дало можливість створити єдину мову програмування для всієї системи, як на клієнті, так і на сервері. Для створення веб-сервера було використано фреймворк Express.js, який значно спрощує процес визначення маршрутів API та обробки HTTP-запитів. Завдяки йому серверна частина отримала структурований вигляд і стала легко масштабованою.

У якості системи збереження даних було обрано SQLite, яка є легкою файловою базою даних і не потребує встановлення окремого серверного програмного забезпечення. Вона є оптимальним рішенням для навчальних проєктів, оскільки дозволяє швидко розгорнути систему без складної конфігурації.

У проєкті було використано бібліотеки bcryptjs та jsonwebtoken. Перша забезпечує надійне хешування паролів користувачів, друга реалізує механізм JWT-авторизації, що дозволяє захищати маршрути API та ідентифікувати користувача без постійного звернення до бази даних.

Клієнтська частина була реалізована без використання важких фреймворків. Дозволило максимально наблизити проєкт до чистої веб-розробки. HTML, CSS та JavaScript використовуються у класичному вигляді, що сприяє кращому розумінню принципів роботи DOM, подій та асинхронних запитів.

### 3.3.2 Реалізація клієнтської частини

Клієнтська частина веб-застосунка відповідає за повну взаємодію користувача з системою. Відображення інтерфейсу, обробку подій, відправку запитів до API та відображення отриманих даних. Вона є найбільш динамічною частиною застосунка і безпосередньо впливає на користувацький досвід.

Основна сторінка застосунка реалізована у файлі - index.html -, який містить структуру інтерфейсу, включаючи пошуковий рядок, блоки фільтрів, сортування та контейнер для відображення списку фільмів. Всі елементи сторінки створені таким чином, щоб користувач міг виконувати основні дії без необхідності переходу між різними сторінками.

Відображення контенту реалізовано динамічно за допомогою JavaScript. Після отримання даних з API система створює DOM-елементи на льоту. Дозволяє оновлювати інтерфейс без перезавантаження сторінки. Такий підхід значно покращує швидкодію та створює відчуття живого інтерфейсу.

Використовується механізм нескінченної прокрутки infinite scroll. При досягненні користувачем нижньої частини сторінки система автоматично завантажує наступну порцію даних. Це дозволяє уникнути перевантаження інтерфейсу великою кількістю контенту та забезпечує плавний користувацький досвід.

Пошукова система реалізована на основі API TMDB і використовує механізм debounce, який дозволяє зменшити кількість запитів до сервера. Пошук активується лише після короткої паузи у введенні тексту. Значно оптимізує продуктивність системи та знижує навантаження на API.

Реалізовано систему підказок пошуку, яка у реальному часі відображає релевантні фільми. Користувач може обрати одну з підказок, після чого система автоматично формує запит і відображає результати пошуку.

Фільтрація фільмів здійснюється за кількома параметрами, включаючи жанри, роки випуску та мінімальний рейтинг. Всі обрані фільтри зберігаються у

локальному сховищі браузера localStorage, що дозволяє зберігати налаштування між сесіями використання застосунка.

Система списку перегляду, дозволяє користувачу зберігати обрані фільми. Ця функція інтегрована із серверною частиною, що забезпечує збереження даних у базі та доступ до них з будь-якого пристрою після авторизації.

### 3.3.3 Index.html

- index.html - є головною точкою входу у веб-застосунок і визначає базову структуру всієї системи. У ньому розташовані основні елементи інтерфейсу, включаючи навігаційну панель, поле пошуку, блоки сортування, фільтри та контейнер для відображення списку фільмів. Інтерфейс побудований таким чином, щоб користувач міг інтуїтивно взаємодіяти з системою без додаткового навчання. Всі основні елементи логічно згруповані та розміщені відповідно до принципів UX-дизайну.

Використовуються модальні вікна для додаткових функцій, таких як фільтри, авторизація та перегляд детальної інформації про фільм. Це дозволяє не перевантажувати основну сторінку та зберігати фокус користувача на основному контенті.

### 3.3.4 Вікно авторизації

Система авторизації реалізована у вигляді модального вікна, яке підтримує два режими роботи: реєстрацію нового користувача та вхід у вже існуючий акаунт. Перемикання між режимами здійснюється без перезавантаження сторінки, що забезпечує швидку взаємодію.

При реєстрації користувача введений пароль передається на сервер, де він хешується за допомогою bcryptjs і лише після цього зберігається у базі даних. Це забезпечує захист персональної інформації. Після успішного входу користувач отримує JWT-токен, який зберігається у localStorage браузера. Цей токен використовується для авторизації всіх наступних запитів до серверної частини системи.

У випадку відсутності авторизації система обмежує доступ до функції додавання фільмів у список перегляду. При спробі виконати таку дію автоматично відкривається вікно входу, що забезпечує логічну цілісність системи.

### 3.3.5 Робота з API та фільтрація

Основним джерелом даних про фільми є зовнішній API TMDB. Через нього реалізовано всі основні функції, включаючи пошук фільмів, отримання популярних записів та завантаження інформації про фільми.

Всі запити виконуються асинхронно за допомогою fetch, що дозволяє не блокувати інтерфейс користувача під час завантаження даних. Для підвищення продуктивності використовується механізм пагінації. Дозволяє завантажувати дані частинами. Фільтрація реалізована як комбінація серверної та клієнтської обробки. Частина фільтрів застосовується безпосередньо у запиті до API, а частина - після отримання даних на клієнтській стороні.

### 3.3.6 Реалізація серверної частини

Серверна частина веб-застосунка виконує роль центрального логічного ядра системи та забезпечує взаємодію між клієнтською частиною, базою даних і зовнішніми сервісами. Вона побудована на основі платформи Node.js. Це дозволяє виконувати JavaScript-код на стороні сервера, та фреймворку Express.js, який значно спрощує створення REST API та обробку HTTP-запитів.

Архітектурно сервер реалізує принцип розділення. Кожен модуль відповідає за окрему функціональну частину системи. Це дозволяє підвищити читабельність коду, спростити тестування та забезпечити можливість подальшого масштабування проєкту.

Основною задачею серверної частини є обробка запитів від клієнта, які включають реєстрацію нових користувачів, авторизацію, отримання та зміну даних списку перегляду фільмів. Для цього реалізовано набір REST API-ендпоінтів, які приймають HTTP-запити та повертають відповіді у форматі JSON. Важливим компонентом серверної логіки є система автентифікації

користувачів. Вона побудована на основі JWT (JSON Web Token). Після успішної авторизації сервер генерує токен, який містить ідентифікатор користувача та підписується секретним ключем. Цей токен передається клієнту та зберігається у localStorage, після чого використовується для доступу до захищених маршрутів. При кожному запиті до захищених API-ендпоінтів сервер виконує перевірку токена через middleware. Якщо токен дійсний, користувач отримує доступ до відповідних ресурсів, а якщо ні - запит відхиляється. Такий підхід дозволяє забезпечити безпеку даних і виключає необхідність повторної авторизації при кожній операції.

Важливою частиною серверної реалізації є захист паролів користувачів. Для цього використовується алгоритм хешування bcryptjs, який перетворює пароль у незворотний хеш перед збереженням у базі даних.

Також сервер виконує роль посередника між клієнтською частиною та зовнішнім API TMDB. Через нього можуть передаватися або проксуватися запити до сервісу фільмів. Це дозволяє централізувати логіку роботи з даними та уникнути прямої взаємодії клієнта із зовнішніми сервісами у певних сценаріях.

### 3.3.7 Реалізація бази даних

Для збереження інформації у проєкті використовується SQLite - легка файлова реляційна база даних, яка не потребує окремого серверного процесу. Такий вибір є обґрунтованим для проєкту, оскільки дозволяє швидко розгорнути систему без складної конфігурації серверної інфраструктури. База даних виконує функцію довготривалого збереження інформації про користувачів та їхні персональні списки перегляду фільмів. Вся інформація організована у вигляді таблиць.

Основними таблицями системи є users та watchlist. Таблиця users містить облікові дані користувачів, включаючи унікальний ідентифікатор, ім’я користувача та хешований пароль. Обмеження унікальності логіна гарантує неможливість створення двох однакових акаунтів. Таблиця watchlist зберігає інформацію про фільми, які користувач додає до персонального списку перегляду. У ній містяться

дані про ідентифікатор користувача, ідентифікатор фільму з TMDB, назву фільму, постер, рейтинг, рік випуску та статус перегляду.

Зв’язок між таблицями реалізовано через зовнішній ключ user_id, який встановлює відповідність між користувачем і його списком фільмів. Це дозволяє реалізувати багатокористувацьку систему, де кожен користувач має повністю ізольовані дані.

Така структура бази даних забезпечує можливість масштабування системи в майбутньому, наприклад, шляхом додавання нових таблиць для рейтингових оцінок, коментарів або історії переглядів.

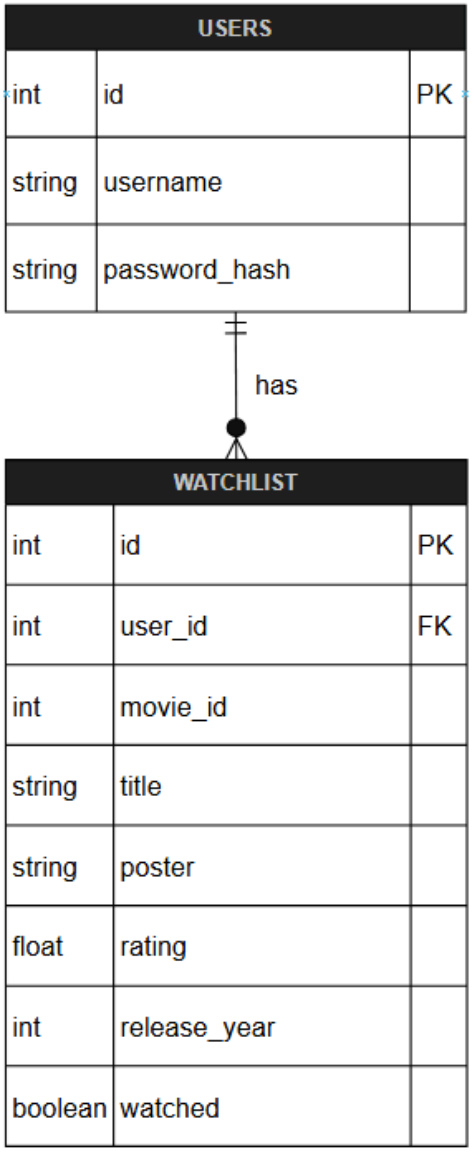



Рис. 3.11 ER-діаграма

# 4 ТЕСТУВАННЯ

Тестування веб-застосунка проводилося на всіх етапах його розробки з метою своєчасного виявлення помилок. Перевірки коректності роботи функціональних модулів та забезпечення стабільності всієї системи в різних сценаріях використання. Такий підхід дозволив поступово перевіряти окремі частини системи ще до їх повної інтеграції, що значно спростило процес налагодження та зменшило кількість критичних помилок на фінальному етапі.

Під час тестування приділялася увага перевірці взаємодії між клієнтською та серверною частинами застосунка, а також стабільності роботи REST API-запитів до зовнішнього сервісу TMDB. Перевірялась коректність обробки відповідей від API, включаючи ситуації з затримкою відповіді, відсутністю даних або некоректними параметрами запиту.

Було протестовано сценарії реєстрації та авторизації користувачів, зокрема перевірка роботи JWT-токенів, їх збереження на стороні клієнта та коректність обмеження доступу до захищених маршрутів. Увага приділялася безпеці даних, зокрема перевірці правильності хешування паролів та неможливості отримання доступу до даних іншого користувача.

Окремим етапом тестування стало перевірення функціоналу додавання та видалення фільмів зі списку перегляду. У цьому контексті перевірялося не лише коректне збереження даних у базі SQLite, але й своєчасне оновлення інтерфейсу користувача без необхідності перезавантаження сторінки. Це є важливим для забезпечення зручності використання застосунка.

Система пошуку та фільтрації тестувалася шляхом введення різних типів запитів, включаючи коректні, некоректні, часткові та порожні значення. Це дозволило перевірити стійкість системи до неправильного вводу та переконатися у коректній обробці виняткових ситуацій без виникнення помилок виконання.

У результаті проведеного тестування було встановлено, що застосунок функціонує стабільно у всіх основних сценаріях використання, забезпечує

коректну взаємодію між усіма компонентами системи, а також демонструє достатній рівень продуктивності та стійкості до некоректних вхідних даних.

Таблиця 4.1

Тестові сценарії

| № | Тестовий сценарій | Очікуваний результат | Наявний результат |
|---|---|---|---|
| 1 | Пошук фільму за назвою | Система відображає список фільмів відповідно до запиту | Працює коректно |
| 2 | Перегляд інформації про фільм | Відкривається модальне вікно з описом, рейтингом та постером | Працює коректно |
| 3 | Відкриття сторінки фільму | Відкривається сторінка з детальною інформацією | Працює коректно |
| 4 | Фільтрація за жанром | Відображаються лише фільми вибраного жанру | Працює коректно |
| 5 | Фільтрація за рейтингом | Система показує фільми з рейтингом не нижче вказаного | Працює коректно |
| 6 | Фільтрація за роком випуску | Відображаються фільми за вибраний період | Працює коректно |
| 7 | Сортування фільмів | Фільми змінюють порядок відображення | Працює коректно |
| 8 | Реєстрація користувача | Створюється новий обліковий запис | Працює коректно |
| 9 | Авторизація користувача | Користувач успішно входить у систему | Працює коректно |
| 10 | Невірний логін або пароль | Система виводить повідомлення про помилку | Працює коректно |
| 11 | Додавання фільму до списку | Фільм зберігається у списку перегляду | Працює коректно |

Продовження Таблиці 4.1

Тестові сценарії

| № | Тестовий сценарій | Очікуваний результат | Наявний результат |
|---|---|---|---|
| 12 | Видалення фільму зі списку | Фільм видаляється зі списку перегляду | Працює коректно |
| 13 | Перевірка збереження списку | Дані списку залишаються збереженими | Працює коректно |
| 14 | Нескінченна прокрутка | Автоматично завантажуються нові фільми | Працює коректно |
| 15 | Адаптивність інтерфейсу | Інтерфейс коректно адаптується під різні розміри екранів | Працює коректно |
| 16 | Перевірка роботи модального вікна | Вікно коректно відкривається та закривається | Працює коректно |
| 17 | Перевірка API-запитів | Дані успішно отримуються з TMDB API | Працює коректно |
| 18 | Перевірка захищених маршрутів | Система вимагає авторизацію | Працює коректно |
| 19 | Перевірка швидкості завантаження | Контент завантажується без критичних затримок | Працює коректно |

# ВИСНОВКИ

1. У процесі виконання курсового проєкту було розроблено веб-застосунок «Кіно Хвиля», який призначений для пошуку фільмів, перегляду інформації про них та формування персонального списку перегляду користувача. Метою проєкту було спростити пошук фільмів за вподобаннями користувачів та ведення власного списку перегляду.

2. У ході виконання роботи було проведено детальний аналіз предметної області та досліджено існуючі популярні аналоги. Такі як Netflix, IMDb, Letterboxd та TMDB. Аналіз дозволив виявити основні переваги та недоліки існуючих рішень, зокрема високу якість рекомендаційних систем, але водночас складність інтерфейсів або обмеженість безкоштовного функціоналу. Отримані результати стали основою для формування вимог до власного застосунка. Серед яких ключовими стали простота використання, швидкий пошук контенту та можливість персоналізації даних.

3. Було приділено увагу проєктуванню архітектури системи. Розроблено клієнт-серверну модель взаємодії, у якій клієнтська частина відповідає за відображення інтерфейсу та взаємодію з користувачем. Серверна частина забезпечує обробку запитів, управління базою даних та реалізацію механізмів авторизації. Використання REST API дозволило організувати структуровану та логічну взаємодію між компонентами системи, а застосування JWT-токенів забезпечило захищений доступ до персональних даних користувачів та контроль автентифікації.

4. Реалізація застосунка була виконана із використанням сучасних веб-технологій. На клієнтській стороні застосовано HTML, CSS та JavaScript, що дозволило створити динамічний та адаптивний інтерфейс користувача. Серверна частина реалізована на базі Node.js із використанням фреймворку Express.js, що забезпечило гнучкість та простоту побудови API. Для збереження даних обрано

SQLite як легку та зручну реляційну базу даних. Не потребує складного адміністрування. Додатково було інтегровано зовнішнє API TMDB, що забезпечує доступ до актуальної та великої бази даних з фільмами у реального часу.

5. У процесі тестування було перевірено основні функціональні можливості застосунка, включаючи пошук та фільтрацію фільмів, роботу системи авторизації, взаємодію з базою даних, а також додавання та видалення фільмів зі списку перегляду. Отримані результати підтвердили коректність роботи всіх основних модулів системи та стабільність взаємодії між клієнтською і серверною частинами.

6. Усі поставлені в роботі завдання були виконані повністю, а мета проєкту досягнута. Розроблений веб-застосунок «Кіно Хвиля» є функціональним, простим у використанні та відповідає сучасним вимогам до веб-додатків подібного типу.

7. Отримані результати також підтверджують можливість подальшого розвитку системи шляхом впровадження рекомендаційних алгоритмів, соціальних функцій, системи оцінювання фільмів та аналітики поведінки користувачів.

Робота пройшла апробацію на двох наукових конференціях:

1. Кузнецов П.О., Коваленко Д.С., Розробка веб-сервісу для рекомендації кінофільмів за вподобанням користувачів. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії», 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025. С.75-76.

2. Кузнецов П.О., Коваленко Д.С., Розробка та супровід web-застосунку “Програмне забезпечення для рекомендації кінофільмів за вподобанням користувачів”. Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація» , 1-3 травня 2026 року, Київ, ДУІКТ. Збірник тез. (подано до друку).

# ПЕРЕЛІК ПОСИЛАНЬ


1. Netflix Official Website. URL: https://www.netflix.com
2. IMDb - Internet Movie Database. URL: https://www.imdb.com
3. Letterboxd. URL: https://letterboxd.com
4. The Movie Database (TMDB). URL: https://www.themoviedb.org
5. Мартін Роберт Чистий код: створення, аналіз і рефакторинг - Київ: Фабула, 2019, с. 45-52.
6. TMDB API Documentation. URL: https://developer.themoviedb.org/docs
7. MDN Web Docs - JavaScript Guide. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript
8. MDN Web Docs - HTML Reference. URL: https://developer.mozilla.org/en-US/docs/Web/HTML
9. MDN Web Docs - CSS Reference. URL: https://developer.mozilla.org/en-US/docs/Web/CSS
10. Node.js Official Documentation. URL: https://nodejs.org/en/docs
11. Express.js Guide. URL: https://expressjs.com
12. SQLite Official Documentation. URL: https://www.sqlite.org/docs.html
13. JWT Introduction (Auth0). URL: https://auth0.com/learn/json-web-tokens/
14. bcrypt.js GitHub Repository. URL: https://github.com/dcodeIO/bcrypt.js
15. REST API Design Guidelines (Microsoft Learn). URL: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design
16. Web Development Basics (MDN Learn). URL: https://developer.mozilla.org/en-US/docs/Learn
17. Fetch API Documentation (MDN). URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
18. HTTP Overview (MDN). URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

19. Client-Server Architecture Explanation (GeeksforGeeks). URL: https://www.geeksforgeeks.org/client-server-model/
20. JavaScript Asynchronous Programming (MDN). URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous
21. Web Application Architecture Overview (AWS). URL: https://aws.amazon.com/what-is/web-application/
22. RESTful API Tutorial (REST API Concepts). URL: https://restfulapi.net
23. Bootstrap Documentation (Responsive Web Design Concepts). URL: https://getbootstrap.com/docs/

# ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО–НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра інженерії програмного забезпечення

**Розробка програмного забезпечення для рекомендації кінофільмів за вподобанням користувачів**

Виконав: студент 4 курсу
групи ПД-41
Кузнецов Павло Олександрович
Керівник роботи
доктор філософії (PhD) Коваленко Данил Сергійович

Київ - 2026

1

## МЕТА, ОБ'ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

**Мета роботи**: спрощення процесу пошуку, перегляду інформації та формування персонального списку перегляду кінофільмів за вподобаннями користувача, шляхом створення сучасного web-застосунка.

**Об'єкт дослідження:** процес пошуку, фільтрації, рекомендації мультимедійного контенту, ведення власного списку перегляду.

**Предмет дослідження**: технології, методи та програмні засоби реалізації клієнт-серверного web-застосунка для роботи з кінофільмами.

2

## ЗАДАЧІ КВАЛІФІКАЦІЙНОЇ РОБОТИ

1. Провести аналіз сучасних web-застосунків для пошуку та перегляду інформації про кінофільми.
2. Визначити функціональні та нефункціональні вимоги до web-застосунка.
3. Обрати технології та засоби реалізації клієнтської та серверної частини застосунка.
4. Спроектувати архітектуру web-застосунка, структуру бази даних та взаємодію між компонентами системи.
5. Реалізувати систему пошуку та фільтрації фільмів за вподобанням користувачів із використанням API (TMDB).
6. Реалізувати механізм реєстрації та авторизації користувачів із використанням JWT-токенів.
7. Створити систему персонального списку перегляду фільмів для зареєстрованих користувачів.
8. Забезпечити адаптивний користувацький інтерфейс web-застосунка для різних розмірів екранів.
9. Тестування, перевірка основного функціоналу застосунку.

3

## АНАЛІЗ АНАЛОГІВ

| Характеристика | Netflix | IMDb | Letterboxd | TMDB | Кіно Хвиля |
|---|---|---|---|---|---|
| Персональні списки | + | + | + | Лише через сторонні сервіси | + |
| Рекомендації | + | + | + | - | + |
| Авторизація користувачів | + | + | + | - | + |
| Інформація про фільми | Основна інформація | Дуже детальна | Огляди + Базові дані | Повні метадані | + |
| Адаптивність | + | + | + | - | + |
| API для розробників | Закритий | Закритий | Немає | + | TMDB API |
| Швидкий підбір фільмів | Рекомендації | Пошук вручну | Через списки | Через API | + |

4

## ВИМОГИ ДО ДОДАТКУ

**Функціональні:**

1. Система повинна забезпечувати можливість пошуку за назвою.
2. Фільтрації фільмів за заданими параметрами (жанр, рік, рейтинг, мін. оцінкою).
3. Можливість формування персонального списку перегляду.
4. Відображення інформації про фільми у вигляді карток.
5. Система авторизації та реєстрації.
6. Збереження даних користувача.

**Нефункціональні:**

1. Асинхронна обробка запитів до зовнішнього сервісу.
2. Адаптивність під різні розміри екранів від 320 px.
3. Забезпечення безпеки користувацьких даних.
4. Зберігання паролів у захешованому вигляді.
5. Використання JWT-токенів для авторизації.

5

## ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

VS Code

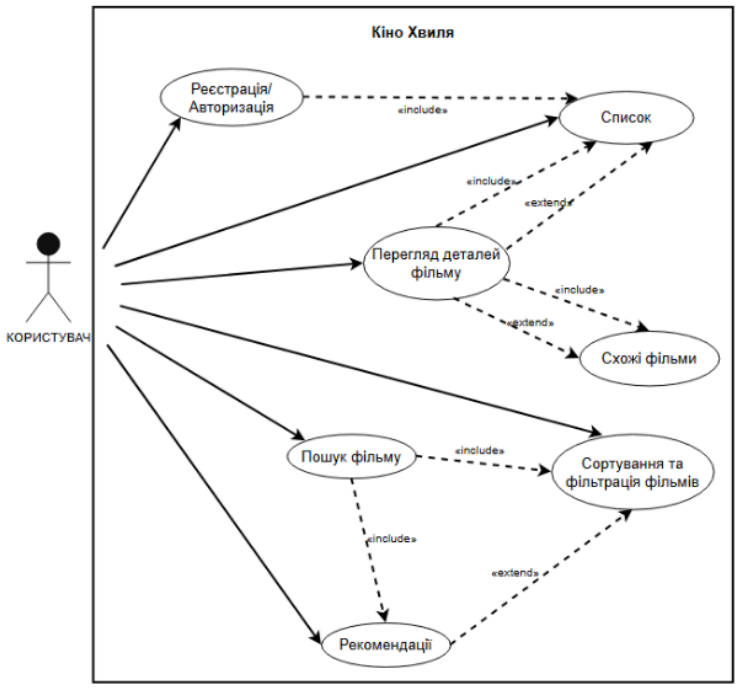

Express.js

Figma



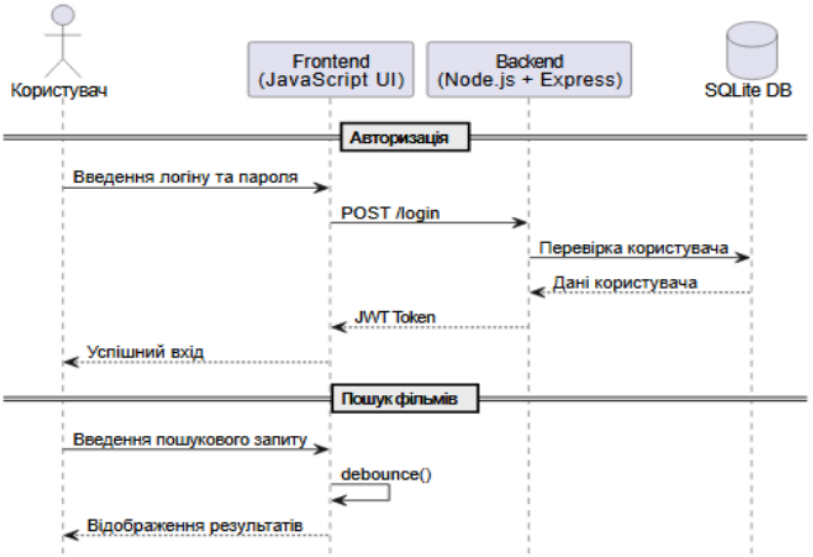

6

# USE CASE ДІАГРАМА



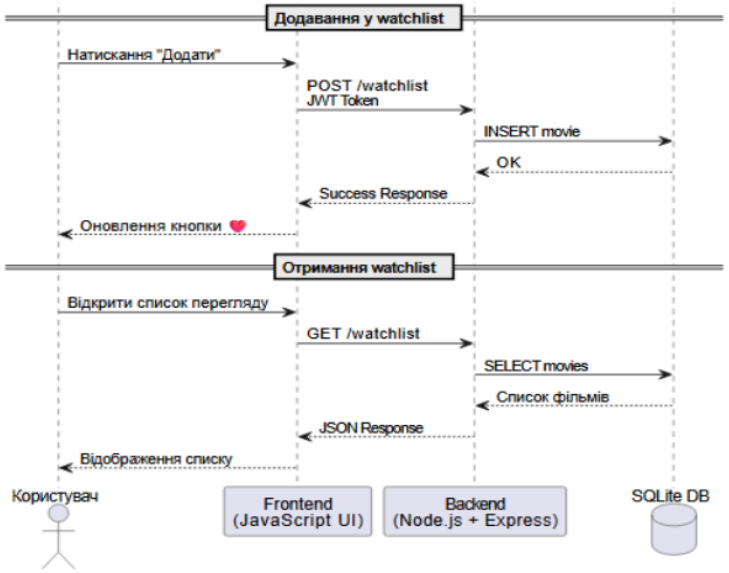

7

# ДІАГРАМА ПОСЛІДОВНОСТІ




8

# ДІАГРАМА ПОСЛІДОВНОСТІ



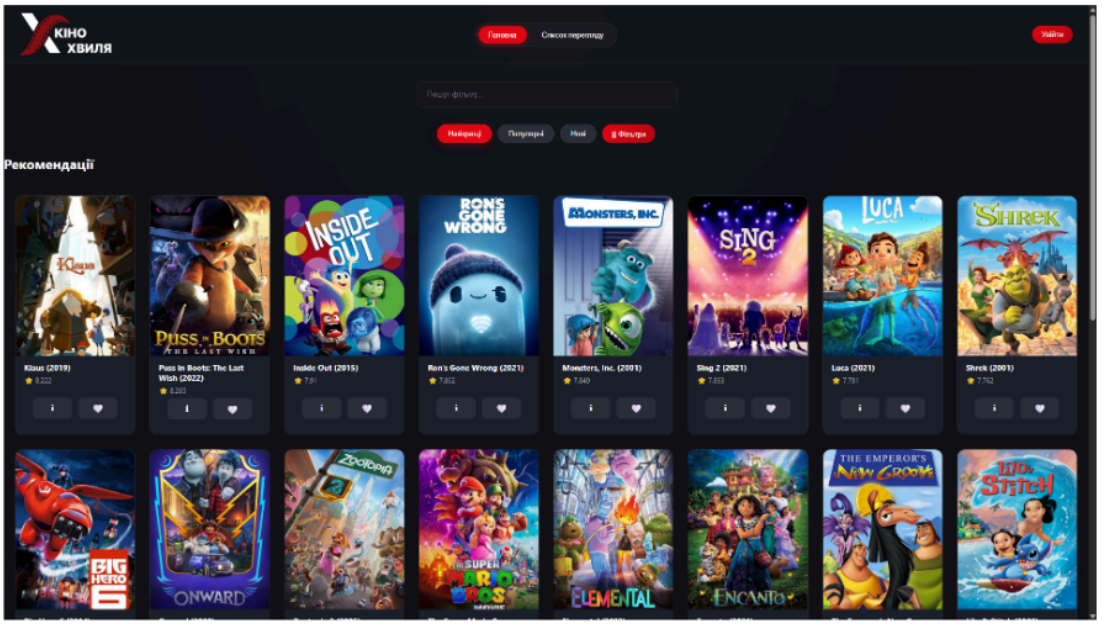

9

# ТЕСТУВАННЯ

| № | Тестовий сценарій | Очікуваний результат | Наявний результат |
|---|---|---|---|
| 1 | Пошук фільму за назвою | Система відображає список фільмів відповідно до запиту | Працює коректно |
| 2 | Перегляд інформації про фільм | Відкривається модальне вікно з описом, рейтингом та постером | Працює коректно |
| 3 | Відкриття сторінки фільму | Відкривається сторінка з детальною інформацією | Працює коректно |
| 4 | Фільтрація за жанром | Відображаються лише фільми вибраного жанру | Працює коректно |
| 5 | Фільтрація за рейтингом | Система показує фільми з рейтингом не нижче вказаного | Працює коректно |
| 6 | Фільтрація за роком випуску | Відображаються фільми за вибраний період | Працює коректно |
| 7 | Сортування фільмів | Фільми змінюють порядок відображення | Працює коректно |
| 8 | Реєстрація/Авторизація користувача | Створюється новий обліковий запис | Працює коректно |

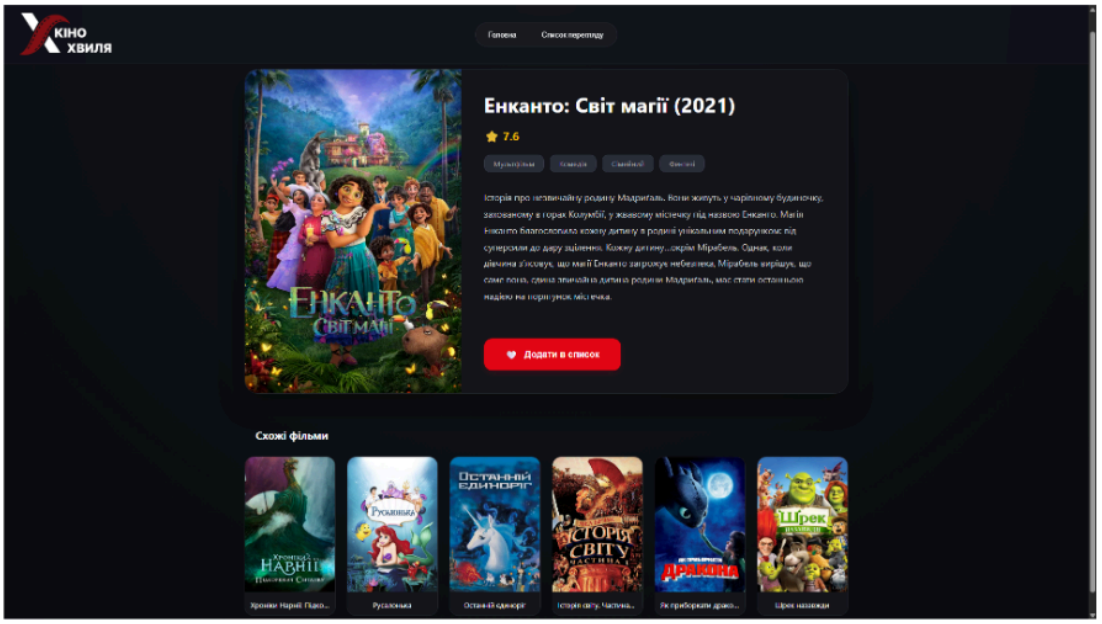
10

## ЕКРАНІ ФОРМИ



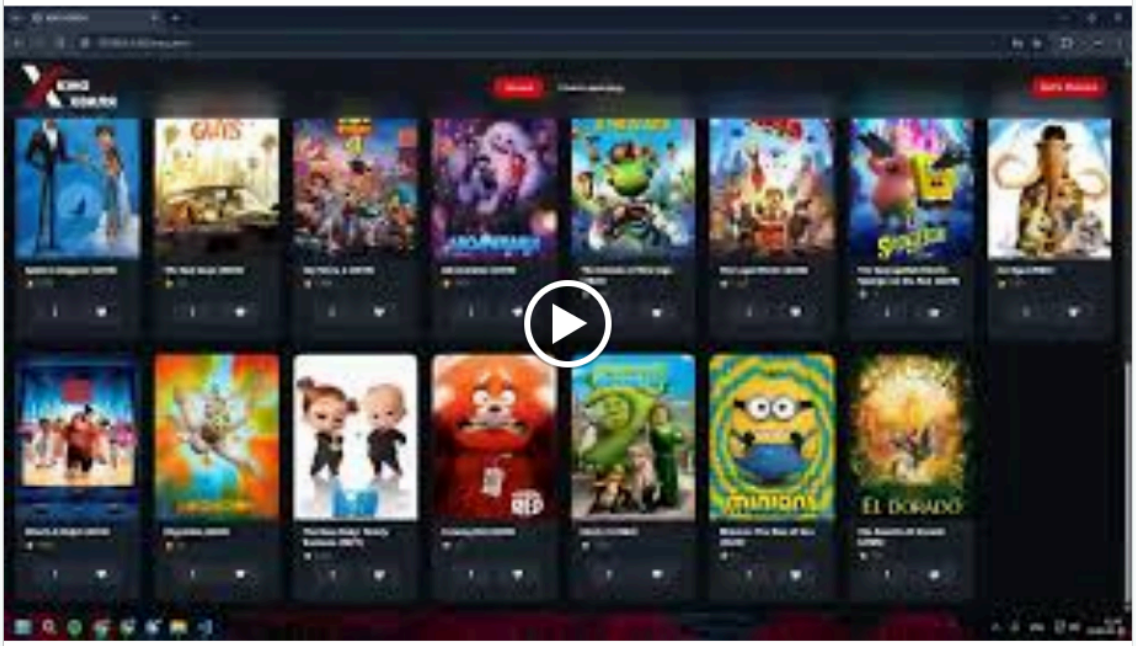

11

## ЕКРАНІ ФОРМИ




12

## ВІДЕО РОБОТИ ДОДАТКУ



13

## АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

**Тези доповідей:**

1. Кузнецов П.О., Коваленко Д.С., Розробка веб-сервісу для рекомендації кінофільмів за вподобанням користувачів. Матеріали II Всеукраїнської науково-технічної конференції «Виклики та рішення в програмній інженерії », 26 листопада 2025 року, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. – К.: ДУІКТ, 2025, С.75-76.
2. Кузнецов П.О., Коваленко Д.С., Розробка та супровід web-застосунку “Програмне забезпечення для рекомендації кінофільмів за вподобанням користувачів”. Матеріали Всеукраїнської науково-технічної конференції «Технології цифрового розвитку: програмні системи та децентралізація» , 1–3 травня 2026 року, Київ, ДУІКТ. Збірник тез. *(подано до друку).*

14

# ВИСНОВКИ

1. Проведено аналіз сучасних web-застосунків для пошуку та перегляду інформації про кінофільми, визначено їхні основні можливості, переваги та недоліки.
2. Визначено функціональні та нефункціональні вимоги до web-застосунка «Кіно Хвиля», сформовано ключові вимоги до системи.
3. Обрано сучасні технології та засоби реалізації клієнтської та серверної частини застосунка, що забезпечують ефективну роботу системи.
4. Спроєктовано архітектуру web-застосунка, структуру бази даних та взаємодію між основними компонентами системи.
5. Реалізовано систему пошуку та фільтрації фільмів за вподобаннями користувачів із використанням API (TMDB).
6. Реалізовано механізм реєстрації та авторизації користувачів із використанням JWT-токенів.
7. Створено систему персонального списку перегляду для зареєстрованих користувачів.
8. Забезпечено адаптивність під різні розміри екранів.
9. Були виконані тестові сценарії, які підтвердили роботу всіх основних функцій застосунку

# ДОДАТОК Б. ЛІСТИНГИ ПРОГРАМНИХ МОДУЛІВ

server.js

```
//БД
const db = new sqlite3.Database("./database.db", (err) =>
{
  if (err) {
    console.error("DB connection error:", err.message);
  } else {
    console.log("Connected to SQLite database");
  }
});

db.serialize(() => {
  db.run(`
    CREATE TABLE IF NOT EXISTS users (
      id INTEGER PRIMARY KEY
AUTOINCREMENT,
      username TEXT UNIQUE,
      password TEXT
    )
  `);

  db.run(`
    CREATE TABLE IF NOT EXISTS watchlist (
      id INTEGER PRIMARY KEY
AUTOINCREMENT,
      user_id INTEGER,
      movieId INTEGER,
      title TEXT,
      poster_path TEXT,
      vote_average REAL,
      release_date TEXT,
      watched INTEGER DEFAULT 0,
      UNIQUE(user_id, movieId),
      FOREIGN KEY(user_id) REFERENCES
users(id)
    )
  `);
});

const authenticateToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  const token = authHeader && authHeader.split(' ')[1];

  if (!token) return res.status(401).json({ error: "Access
denied" });

  jwt.verify(token, SECRET_KEY, (err, user) => {
    if (err) return res.status(403).json({ error: "Invalid
token" });
    req.user = user;
    next();
  });
};

//РЕЄСТРАЦІЯ ЕНДПОІНТ
app.post("/register", async (req, res) => {
  const { username, password } = req.body;

  if (!username || !password) return
res.status(400).json({ error: "Username and password
required" });

  try {
    const hashedPassword = await
bcrypt.hash(password, 10);
    db.run(
      "INSERT INTO users (username, password)
VALUES (?, ?)",
      [username, hashedPassword],
      function (err) {
        if (err) {
          if (err.message.includes("UNIQUE
constraint failed")) {
            res.status(400).json({ error: "Username
already exists" });
          } else {
            res.status(500).json({ error: err.message
});
          }
        } else {
          res.json({ success: true, message: "User
registered successfully" });
        }
      }
    );
  } catch (err) {
    res.status(500).json({ error: "Error hashing
password" });
  }
});

//ВХІД ЕНДПОІНТ
app.post("/login", (req, res) => {
  const { username, password } = req.body;

  db.get("SELECT * FROM users WHERE username =
?", [username], async (err, user) => {
    if (err) return res.status(500).json({ error:
err.message });
    if (!user) return res.status(400).json({ error: "User
not found" });

    const validPassword = await
bcrypt.compare(password, user.password);
    if (!validPassword) return res.status(400).json({
error: "Invalid password" });

    const token = jwt.sign({ id: user.id, username:
user.username }, SECRET_KEY, { expiresIn: "24h" });
    res.json({ success: true, token, username:
user.username });
  });
});

//ОТРИМАТИ ВЕСЬ СПИСОК ДЛЯ КОР.
app.get("/watchlist", authenticateToken, (req, res) => {
```

```
    db.all("SELECT * FROM watchlist WHERE user_id
= ?", [req.user.id], (err, rows) => {
        if (err) {
            res.status(500).json({ error: err.message });
        } else {
            res.json(rows);
        }
    });
});

//ДОДАТИ ФІЛЬМ ДО СПИСКУ
app.post("/watchlist", authenticateToken, (req, res) => {
    const { movieId, title, poster_path, vote_average,
release_date } = req.body;

    db.run(
        `
        INSERT INTO watchlist
        (user_id, movieId, title, poster_path, vote_average,
release_date)
        VALUES (?, ?, ?, ?, ?, ?)
        `,
        [req.user.id, movieId, title, poster_path,
vote_average, release_date],
        function (err) {
            if (err) {
                res.status(500).json({ error: err.message });
            } else {
                res.json({
                    success: true,
                    id: this.lastID,
                    movieId
                });
            }
        }
    );
});

//ВИДАЛИТИ ФІЛЬМ ЗА movieId
app.delete("/watchlist/:movieId", authenticateToken,
(req, res) => {
    db.run(
        "DELETE FROM watchlist WHERE user_id = ?
AND movieId = ?",
        [req.user.id, req.params.movieId],
        function (err) {
            if (err) {
                res.status(500).json({ error: err.message });
            } else {
                res.json({ success: true });
            }
        }
    );
});

//ПЕРЕКЛ СТАН ПЕРЕГЛЯНУТОГО
app.put("/watchlist/watched/:movieId",
authenticateToken, (req, res) => {
    db.run(
        `
        UPDATE watchlist
        SET watched = CASE WHEN watched = 1 THEN
0 ELSE 1 END
        WHERE user_id = ? AND movieId = ?
        `,
        [req.user.id, req.params.movieId],
        function (err) {
            if (err) {
                res.status(500).json({ error: err.message });
            } else {
                res.json({ success: true });
            }
        }
    );
});

//START SERVER
app.listen(PORT, () => {
    console.log(`Server running on
http://localhost:${PORT}`);
});

srcipt.js
document.addEventListener("DOMContentLoaded",
async () => {

    document.addEventListener("submit", (e) => {
    e.preventDefault();
    });

    const searchInput =
document.getElementById("searchInput");

    await loadWatchlistCache();

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

    //ПРОПОЗИЦІЇ ПОШУКУ
    searchInput.addEventListener("input", (e) => {
        clearTimeout(debounceTimer);
        clearTimeout(suggestionsTimer);

        const query = e.target.value.trim();
        currentQuery = query;

        suggestionsTimer = setTimeout(() => {
            if (query.length > 1) loadSuggestions(query);
            else clearSuggestions();
        }, 200);

        debounceTimer = setTimeout(() => {
            page = 1;

            if (query.length === 0) {

document.getElementById("pageTitle").textContent =
"Рекомендовані";
                showHome(true);
            } else {
```

```
document.getElementById("pageTitle").textContent =
"Результати пошуку";
            searchMovies(true);
        }
    }, 400);

});

//SORT
document.querySelectorAll(".sort-btn").forEach(btn
=> {
    btn.addEventListener("click", () => {

document.querySelectorAll(".sort-btn").forEach(b =>
b.classList.remove("active"));
        btn.classList.add("active");

        sortBy = btn.dataset.sort;
        page = 1;

        showHome(true);
    });
});

//NAV

document.getElementById("homeBtn").addEventListene
r("click", () => {
    setActiveNav("homeBtn");
    showHome(true);
});


document.getElementById("watchlistBtn").addEventList
ener("click", () => {
    setActiveNav("watchlistBtn");
    showWatchlist();
});

//ФІЛЬТРИ

document.getElementById("filterBtn").addEventListener
("click", openGenreSettings);

document.getElementById("saveFiltersBtn").addEventLi
stener("click", saveFilters);

document.getElementById("closeFiltersBtn").addEventL
istener("click", closeGenreSettings);

//AUTH

document.getElementById("authBtn").addEventListener(
"click", () => {
    if (localStorage.getItem("token")) {
        handleLogout();
    } else {
        openAuthModal();
    }
});

document.getElementById("closeAuthBtn").addEventLis
tener("click", closeAuthModal);

document.getElementById("authToggleBtn").addEventL
istener("click", toggleAuthMode);

document.getElementById("authSubmitBtn").addEventL
istener("click", handleAuthSubmit);

window.addEventListener("scroll", handleScroll);

setActiveNav("homeBtn");

//старт
if (selectedGenres.length === 0) {
    openGenreSettings();
} else {
    showHome(true);
}
});
```

ui.js

```
//SKELETON
function renderSkeleton(containerId = "results", count =
8) {
    const container =
document.getElementById(containerId);
    container.innerHTML = "";

    for (let i = 0; i < count; i++) {
        const div = document.createElement("div");
        div.className = "movie-card skeleton-card";
        div.innerHTML = `
            <div class="skeleton skeleton-img"></div>
            <div class="movie-info">
                <div class="skeleton skeleton-text"></div>
                <div class="skeleton skeleton-text"></div>
                <div class="movie-buttons">
                    <div class="skeleton skeleton-btn"></div>
                    <div class="skeleton skeleton-btn"></div>
                </div>
            </div>
        `;
        container.appendChild(div);
    }
}

//CARD
function renderMovieCard(movie, containerId =
"results") {

    const container =
document.getElementById(containerId);

    const id = Number(movie.id);

    const isInWatchlist = watchlistCache.some(
        m => Number(m.movieId) === id
    );

    const year = movie.release_date
```

```
    ? movie.release_date.slice(0, 4)
    : "—";

  const card = document.createElement("div");
  card.className = "movie-card";

  card.innerHTML = `
    <img src="${IMG_URL + movie.poster_path}">

    <div class="movie-info">

      <div class="movie-text">
        <h3>${movie.title} (${year})</h3>
        <p>⭐ ${movie.vote_average}</p>
      </div>

      <div class="movie-buttons">

        <button type="button" class="btn-details">
          ℹ
        </button>

        <button type="button"
                class="btn-watchlist ${isInWatchlist ?
"added" : ""}">
          ${isInWatchlist ? "❤️" : "🤍"}
        </button>

      </div>

    </div>
  `;

  const watchBtn = card.querySelector(".btn-watchlist");
  const infoBtn = card.querySelector(".btn-details");

  //OPEN MOVIE
  card.addEventListener("click", (e) => {
  if (e.target.closest("button")) return;
  openMovie(id);
  });

  //INFO BTN
  infoBtn.addEventListener("click", (e) => {

    e.preventDefault();
    e.stopPropagation();

    showMovieModal(id);

  });

  //WATCHLIST BTN
  watchBtn.addEventListener("click", async (e) => {

    e.preventDefault();
    e.stopPropagation();

    const nowInList = await toggleWatchlist(movie);

    watchBtn.classList.toggle("added", nowInList);

    watchBtn.innerHTML = nowInList
      ? "❤️"
      : "🤍";

  });

  container.appendChild(card);
}

//MOVIE MODAL
async function showMovieModal(id) {
  const modal =
document.getElementById("movieModal");
  const content =
document.getElementById("movieModalContent");

  modal.style.display = "flex";
  content.innerHTML = "Loading...";

  const res = await
fetch(`${BASE_URL}/movie/${id}?api_key=${API_K
EY}`);
  const movie = await res.json();

  content.innerHTML = `
    <img src="${IMG_URL + movie.poster_path}"
alt="${movie.title}">
    <div class="modal-body">
      <h2>${movie.title}</h2>
      <div class="meta-info">
        ⭐ ${movie.vote_average.toFixed(1)} •
${movie.release_date?.slice(0,4)}
      </div>
      <p class="description">${movie.overview ||
"Опис відсутній."}</p>
      <button
onclick="closeMovieModal()">Закрити</button>
    </div>
  `;
}

function closeMovieModal() {

document.getElementById("movieModal").style.display
= "none";
}

window.onclick = function (e) {
  const modal =
document.getElementById("movieModal");
  const authModal =
document.getElementById("authModal");
  if (e.target === modal) {
    modal.style.display = "none";
  }
  if (e.target === authModal) {
    authModal.style.display = "none";
  }
};
```