Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

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

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

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

на тему РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ОЦІНЮВАННЯ ТА ОБГОВОРЕННЯ КІНОФІЛЬМІВ
DEVELOPMENT OF SOFTWARE FOR MOVIES REVIEWING AND DISCUSSING

Виконав(ла): студент(ка) 4 курсу, групи КНТ-122
Спеціальності 121 Інженерія програмного
(код і найменування спеціальності)
забезпечення

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

КОВАЛЕНКО В.П .
(ПРІЗВИЩЕ та ініціали)
Керівник ФЕДОРОНЧАК Т.В.
(ПРІЗВИЩЕ та ініціали)
Рецензент БАБЕНКО Н.В.
(ПРІЗВИЩЕ та ініціали)

2026

Форма № 25

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

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

ЗАТВЕРДЖУЮ

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

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КОВАЛЕНКА Владислава Павловича
(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи) Розроблення програмного забезпечення для оцінювання та обговорення кінофільмів. Development of Software for Movies Reviewing and Discussing

керівник проєкту (роботи) к.т.н., доцент, ФЕДОРОНЧАК Тетяна Василівна,
(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові)
затверджені наказом закладу вищої освіти від “ 07 ” квітня 2026 року № 139
2. Строк подання студентом проєкту (роботи) 10 червня 2026 року
3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

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

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

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ФЕДОРОНЧАК Т. В., доцент		
Нормоконтроль	ДЕЙНЕГА Л.Ю., ст. викладач		

7. Дата видачі завдання « 20 » квітня 2026 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка) _____ Владислав КОВАЛЕНКО
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Тетяна ФЕДОРОНЧАК
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до випускної кваліфікаційної роботи бакалавра: 99с., 8 табл., 20 рис., 2 дод., 13 джерел.

ПОШУК ФІЛЬМІВ, ВЕБЗАСТОСУНОК, АВТЕНТИФІКАЦІЯ, ПЕРСОНАЛІЗАЦІЯ, ВІДГУКИ, MONGODB, REACT, NODE.JS.

Об’єкт дослідження – процес інженерії програмного забезпечення для взаємодії користувачів із цифровим кіноконтентом у вебсередовищі.

Предмет дослідження – програмні системи для оцінювання та обговорення кінофільмів.

Мета роботи – підвищення якості інформаційної підтримки користувачів при виборі кіноконтенту шляхом розроблення вебзастосунку для оцінювання та обговорення фільмів, що забезпечує систематизацію глядацького досвіду та інтерактивну взаємодію між учасниками спільноти.

Матеріали, методи та технічні засоби: використано методи клієнт-серверної розробки; React і Vite для клієнтської частини; Node.js і Express для серверної логіки; MongoDB і Mongoose для зберігання даних; JWT для автентифікації та авторизації; TMDB API як зовнішнє джерело даних про фільми.

Результати. Розроблено вебзастосунок CineRate, що реалізує реєстрацію та вхід користувачів, пошук фільмів за назвою, фільтрацію за жанрами, перегляд детальної інформації про фільм, ведення персональних списків (обране, вотчліст), додавання оцінок і коментарів, перегляд відгуків спільноти, а також базову персоналізацію рекомендацій.

Висновки. Поставлену мету досягнуто: створено працездатний вебзастосунок для персоналізованої роботи з кіноконтентом.

Галузь використання – онлайн-сервіси для пошуку та рекомендації фільмів.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 99 pages, 8 tables, 20 figures, 2 appendixes, 13 sources.

MOVIE SEARCH, WEB APPLICATION, AUTHENTICATION, PERSONALIZATION, REVIEWS, MONGODB, REACT, NODE.JS.

The object of research is the software engineering process for user interaction with digital movie content in a web environment.

The subject of research is software systems for rating and discussing movies.

The purpose of the work is to improve the quality of informational support for users when choosing movie content by developing a web application for rating and discussing films, which provides systematization of viewers' experience and interactive interaction between community members.

Materials, methods, and technical tools: client-server development methods were used; React and Vite for the client side; Node.js and Express for server-side logic; MongoDB and Mongoose for data storage; JWT for authentication and authorization; TMDB API as an external source of movie data.

Results. The CineRate web application was developed, implementing user registration and login, movie search by title, genre filtering, viewing detailed movie information, managing personal lists (favorites, watchlist), adding ratings and comments, viewing community reviews, as well as basic recommendation personalization.

Conclusions. The stated goal has been achieved: a functional web application for personalized interaction with movie content has been created.

Field of application – online services for movie search and recommendation.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

CRUD – Create, Read, Update, Delete;

HTTP – HyperText Transfer Protocol;

JSON – JavaScript Object Notation;

JWT – JSON Web Token;

NoSQL – Not Only SQL;

TMDB – The Movie Database;

UI – User Interface;

URL – Uniform Resource Locator;

UX – User Experience.

ВСТУП

Сучасний стан розвитку інформаційних технологій характеризується стрімким поширенням цифрових платформ та мобільних застосунків, які охоплюють практично всі сфери повсякденного життя користувачів. Однією з таких сфер є індустрія кіно та розваг, де користувачі активно споживають контент, оцінюють його та обмінюються враженнями. У зв’язку з цим виникає потреба у зручних інструментах для систематизації переглянутих фільмів, виставлення оцінок і ведення обговорень.

Проблема організації та зручного представлення інформації про кінофільми, а також взаємодії між користувачами залишається актуальною. Незважаючи на наявність популярних платформ для оцінювання фільмів, багато з них мають або надлишковий функціонал, або обмежені можливості для персоналізації, соціальної взаємодії та зручного ведення власної історії перегляду. Часто користувачі змушені використовувати декілька сервісів одночасно, що знижує зручність і ефективність використання таких систем.

Проблема полягає у необхідності створення програмного засобу, який поєднує в собі можливості перегляду інформації про фільми, оцінювання, ведення власних списків та організації обговорень між користувачами в межах єдиного середовища. Існуючі рішення частково вирішують ці задачі, однак не завжди забезпечують оптимальний баланс між функціональністю, простотою використання та швидкодією.

Для досягнення поставленої мети в роботі необхідно вирішити такі завдання:

- провести аналіз предметної області та існуючих програмних рішень для оцінювання кінофільмів;
- визначити функціональні та нефункціональні вимоги до застосунку;
- розробити архітектуру програмного забезпечення;

- реалізувати основний функціонал застосунку, включаючи систему оцінок, коментарів та пошуку;
- провести тестування програмного продукту та оцінити його ефективність.

Методика дослідження включає аналіз літературних джерел і програмних аналогів, застосування принципів об’єктно-орієнтованого програмування, використання сучасних технологій розробки програмного забезпечення, а також тестування застосунку в умовах, наближених до реальних.

Розроблений застосунок може бути використаний широким колом користувачів для ведення власної колекції переглянутих фільмів, обміну думками та отримання рекомендацій. Практична цінність роботи полягає у створенні зручного інструменту для взаємодії з кіно-контентом, що підвищує якість користувацького досвіду та спрощує процес вибору фільмів для перегляду.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Короткий опис досліджуваної предметної області

Сфера онлайн-кіноконтенту та його оцінювання користувачами є важливою складовою сучасного цифрового середовища. З розвитком стримінгових платформ та збільшенням кількості кінофільмів користувачі стикаються з проблемою вибору контенту для перегляду, а також необхідністю збереження власного досвіду взаємодії з фільмами. У зв’язку з цим зростає потреба у вебзастосунках, які дозволяють не лише переглядати інформацію про фільми, а й вести персоналізований облік переглянутого контенту, оцінювати його та формувати рекомендації.

Основними поняттями даної предметної області є фільм як одиниця контенту, користувач як суб’єкт взаємодії із системою, оцінка як числове відображення враження від перегляду, а також відгук як текстове представлення думки користувача. Важливу роль відіграють також списки фільмів, такі як переглянуті або обрані, що дозволяють структурувати взаємодію користувача з контентом. Окремим аспектом є рекомендаційна система, яка аналізує дії користувача та пропонує нові фільми відповідно до його вподобань.

Сучасні вебсервіси у сфері кіно стикаються з низкою викликів. По-перше, це надлишок інформації, через який користувачу складно знайти релевантний контент. По-друге, відсутність персоналізації у багатьох сервісах, що знижує якість користувацького досвіду. По-третє, обмежені можливості для соціальної взаємодії, такі як обмін відгуками або оцінками між користувачами. Крім того, важливим є питання зручності інтерфейсу та швидкості доступу до інформації, оскільки користувачі очікують інтуїтивно зрозумілий та швидкий сервіс.

Серед сучасних тенденцій розвитку цієї галузі можна виділити активне використання зовнішніх API для отримання актуальної інформації про фільми, впровадження рекомендаційних алгоритмів на основі

переглянутих фільмів, а також розвиток соціальних функцій, таких як рейтинги та обговорення фільмів. Окремо слід відзначити зростання популярності вебзастосунків, які поєднують функції каталогу фільмів, особистого щоденника перегляду та платформи для обговорення контенту.

1.2 Огляд існуючих рішень проблеми

Розглянемо три популярні вебсервіси для обліку переглянутого кіноконтенту, оцінювання фільмів та формування рекомендацій: Letterboxd, IMDb та Trakt. Кожен із них має свої особливості, проте не забезпечує повного поєднання персоналізованого аналізу, соціальної взаємодії та гнучкої системи рекомендацій.

1.2.1 Застосунок Letterboxd

Letterboxd — це міжнародна онлайн-платформа, створена у Новій Зеландії у 2011 році. Вона дозволяє зберігати улюблені фільми, оцінювати їх, писати рецензії, вести списки перегляду і ділитися думками з іншими глядачами. Протягом 13 років Letterboxd залишався нішевим сервісом, схожим на стильну альтернативу IMDb. Але до 2024 року він почав активно змінювати онлайн-культуру спілкування про фільми.

Переваги застосунку:

- сучасний інтерфейс;
- активна спільнота користувачів;
- можливість створення списків і ведення історії переглядів;
- зручна система оцінювання.

Недоліки:

- він обмежений лише фільмами (без повноцінної підтримки серіалів) ;

– частина функціоналу, зокрема розширена статистика переглядів та персоналізовані підбірки, доступні лише у платній версії, що обмежує можливості користувачів без підписки;

– має обмежені можливості аналітики та персоналізації.

Вигляд сторінки з фільмами застосунку Letterboxd продемонстровано на рис. 1.1.

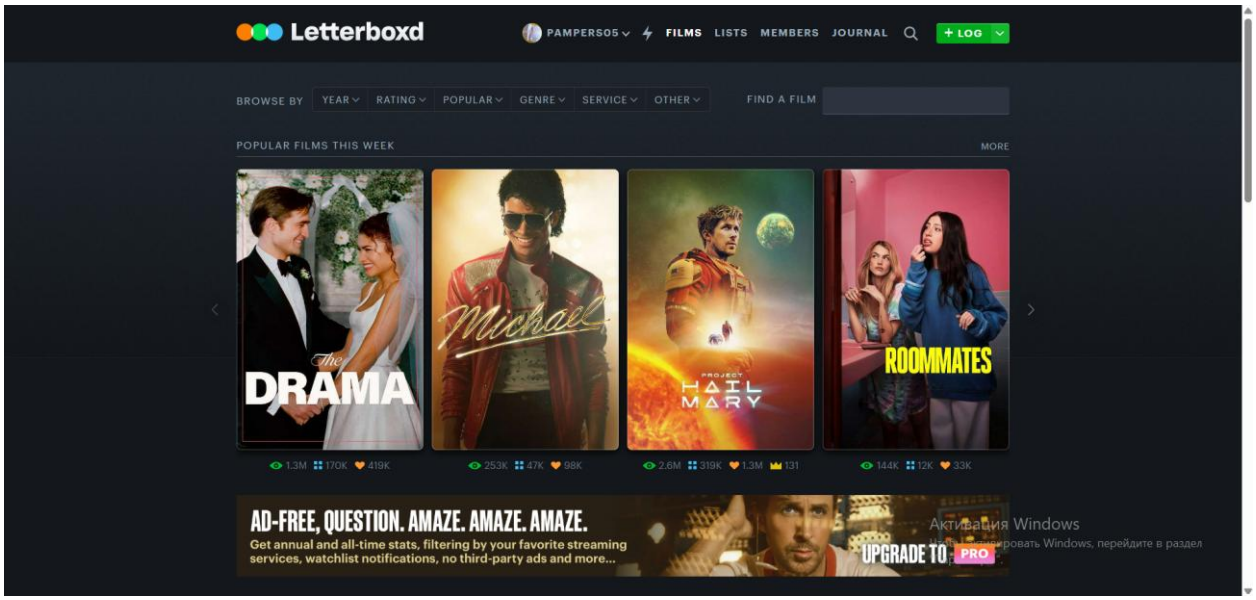


Рисунок 1.1 – Вигляд сторінки з фільмами Letterboxd [1]

1.2.2 Застосунок IMDb

Платформа IMDb є однією з найбільших у світі баз даних, що містить інформацію про кінофільми, серіали, акторів, режисерів та інші аспекти кіноіндустрії. Вона надає користувачам доступ до великого обсягу довідкової інформації, включаючи рейтинги, рецензії, трейлери та новини. Користувачі можуть створювати власні списки перегляду та оцінювати фільми, однак основний акцент зроблено саме на інформаційній складовій.

Основні переваги застосунку:

– велика база даних;

– висока достовірність інформації;

– підтримка різних типів контенту та популярність серед користувачів.

Недоліки:

– перевантажений інтерфейс;

– обмежені можливості персоналізації та слабка соціальна взаємодія;

– частина функціоналу доступна лише в межах платного сервісу IMDbPro, що обмежує можливості звичайних користувачів.

Вигляд головної сторінки застосунку IMDb продемонстровано на рис. 1.2.

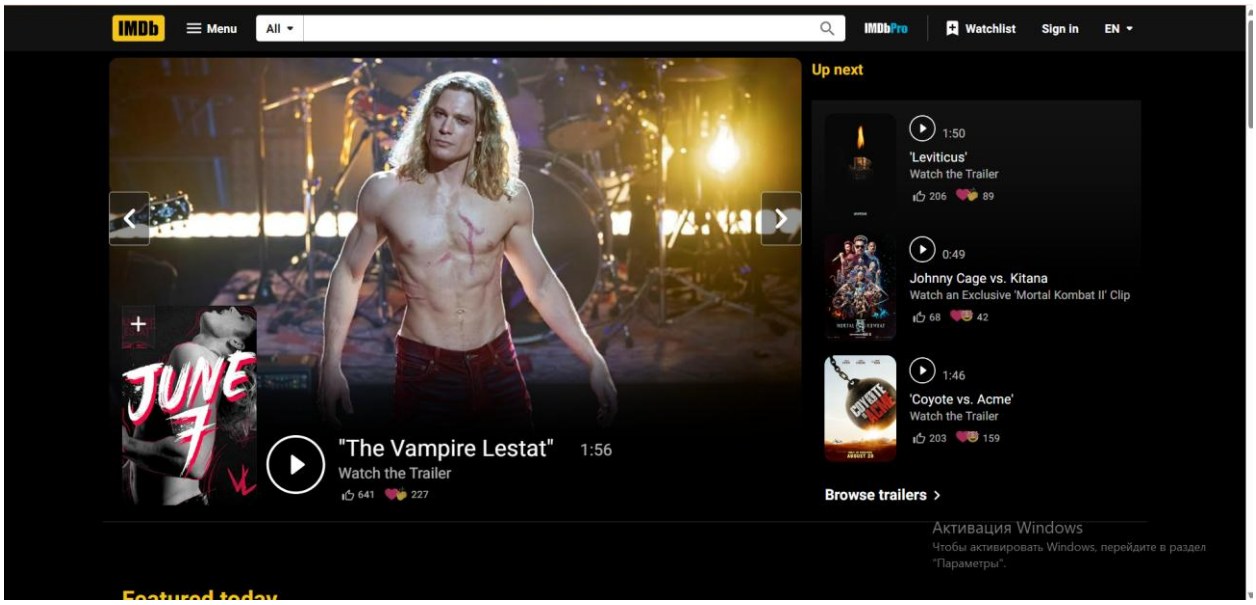


Рисунок 1.2 – Головний екран IMDb [2]

1.2.3 Застосунок Trakt

Сервіс Trakt призначений для відстеження перегляду фільмів і серіалів та синхронізації цієї інформації між різними пристроями і платформами. Користувачі можуть автоматично фіксувати переглянутий контент, формувати списки, оцінювати фільми та отримувати рекомендації. Однією з ключових особливостей сервісу є інтеграція з іншими медіаплатформами, що дозволяє автоматизувати процес обліку.

Основні переваги:

- підтримка як фільмів, так і серіалів;
- автоматизація обліку переглядів та можливість інтеграції з іншими сервісами.

Недоліки:

- складніший для сприйняття інтерфейс;
- частина функцій доступна лише у платній підписці, що знижує зручність використання для безкоштовних користувачів.

Вигляд головної сторінки застосунку Trakt продемонстровано на рис. 1.3.

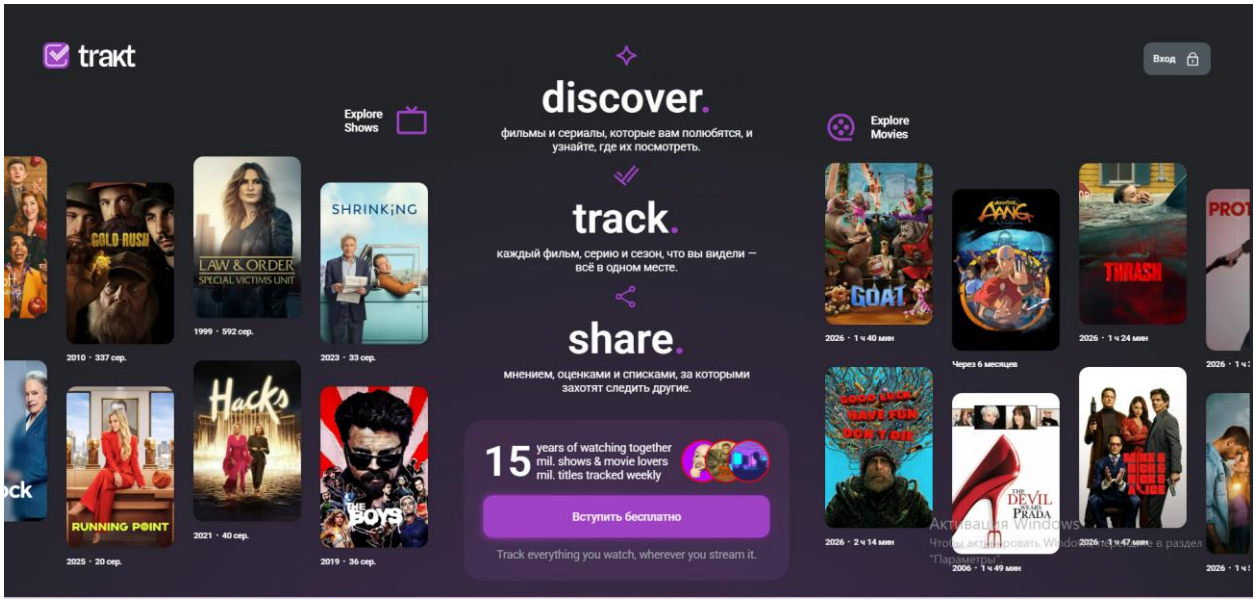


Рисунок 1.3 – Головний екран Trakt [3]

1.2.4 Порівняльна таблиця існуючих та розроблюваного програмних продуктів

Для оцінки сильних і слабких сторін розглянутих застосунків порівняємо їх із розроблюваним вебзастосунком «CineRate» за ключовими критеріями: функціональність, персоналізація рекомендацій, зручність використання, можливості взаємодії між користувачами та система оцінювання кінофільмів.

В табл. 1.1 проведено порівняння вже існуючих застосунків та програми, яка буде розроблена в ході виконання роботи.

Таблиця 1.1 — Порівняння функціональності застосунків для оцінки та обговорення кінофільмів

Критерій	IMDb	Letterboxd	Trakt	Розроблюваний застоосунок
1	2	3	4	5
Головна сторінка з популярними фільмами	✓	✓	✓	✓
Детальна сторінка фільму	✓	✓	✓	✓ (розширена + франшизи)
Пошук за назвою	✓	✓	✓	✓
Фільтрація за жанром	✓	✓	✓	✓
Персоналізовані рекомендації	✓	частково	✓	✓
Додавання в обране	✓	✓	✓	✓
Watchlist (“подивитися пізніше”)	✓	✓	✓	✓
Оцінювання фільмів	✓	✓	✓	✓
Коментар/відгук користувача	✓	✓	✓	✓(+ редагування)
Середній рейтинг + кількість оцінок	✓	✓	✓	✓

Кінець таблиці 1.1

1	2	3	4	5
Аутентифікація (JWT / токен)	✓	✓	✓	✓
Модерація коментарів	✓	✓	✓	✓ (Розширена система ролей)
Інтеграція з TMDB	□ власна БД)	□	✓	✓
Адаптивний інтерфейс	✓	✓	✓	✓
Обмеження безкоштовної версії	✓	✓	✓	□

1.3 Опис основних вимог до програми

1.3.1 Визначення функціональних вимог

Функціональні вимоги визначають, що саме повинна робити система. Для вебзастосунку «Movie App» (платформа пошуку, перегляду та персоналізованого підбору фільмів) визначено такі основні функціональні вимоги:

а) реєстрація та автентифікація користувачів:

1) реєстрація користувача через email/пароль;

2) вхід користувача через email/пароль;

3) перевірка авторизованого стану користувача при відкритті застосунку;

4) вихід із системи з очищенням сесійного токена;

б) перегляд каталогу фільмів:

1) відображення популярних фільмів на головній сторінці;

2) відображення карток фільмів із базовою інформацією (постер, назва, рейтинг, дата);

3) перехід на сторінку детальної інформації про обраний фільм;

в) пошук і фільтрація:

1) пошук фільмів за назвою;

2) фільтрація фільмів за жанром;

3) комбінування пошуку і жанрового фільтра в межах одного запиту;

г) перегляд детальної інформації про фільм:

1) відображення опису, жанрів, дати релізу, тривалості, статусу;

2) відображення акторського складу та режисера;

3) відображення рейтингу з зовнішнього джерела даних (TMDB);

4) відображення фільмів колекції/франшизи (за наявності) ;

д) персональні списки користувача:

1) додавання/видалення фільмів до списку «Обране»;

2) додавання/видалення фільмів до списку «Подивитися пізніше» (Watchlist);

3) перегляд окремих сторінок списків «Обране» та «Watchlist»;

е) оцінювання та відгуки:

1) додавання власної оцінки фільму (шкала 1-5);

2) додавання текстового відгуку до фільму;

3) редагування власного відгуку;

4) перегляд публічних відгуків інших користувачів;

5) відображення середнього рейтингу та кількості оцінок/коментарів;

ж) персоналізація:

1) формування персональних рекомендацій для авторизованого користувача;

2) врахування історії взаємодій користувача (переглянуті/оцінені, обране) для рекомендацій;

з) профіль користувача:

1) перегляд статистики користувача (активність, переглянуті, оцінені тощо);

2) перегляд і керування власними відгуками;

3) відображення персональних даних облікового запису;

и) рольова модель та модерація:

1) розмежування ролей користувачів (user, admin, superadmin);

2) надання прав модерації відгуків для ролей admin/superadmin;

3) керування адміністраторами (лише для superadmin);

1.3.2 Нефункціональні вимоги

Нефункціональні вимоги визначають характеристики та обмеження системи:

а) вимоги до безпеки:

1) передача даних лише через захищений HTTPS-канал;

2) автентифікація на основі токенів доступу (JWT);

3) контроль доступу до захищених маршрутів і дій за ролями;

4) безпечне зберігання паролів у БД (хешування);

б) вимоги до продуктивності:

1) час первинного завантаження головної сторінки — до 3 секунд за стабільного з’єднання;

2) час відгуку інтерфейсу при навігації — до 0,5 секунди для більшості дій;

3) час відповіді API для стандартних запитів — до 1 секунди;

4) коректна робота при одночасних запитах кількох користувачів без деградації критичних функцій;

в) вимоги до надійності:

1) коректна обробка помилок мережі та API з повідомленням користувача;

2) відновлення сеансу користувача після перезавантаження сторінки (за валідного токена);

3) стабільна робота основних сценаріїв (пошук, деталі, обране, відгуки);

г) вимоги до інтерфейсу:

1) інтуїтивно зрозумілий інтерфейс із мінімальною кількістю кроків до цільової дії;

2) адаптивний дизайн для смартфонів, планшетів і десктопів;

3) єдиний візуальний стиль компонентів застосунку;

4) наявність станів завантаження (скелетони/індикатори) для покращення UX;

д) вимоги до масштабованості:

1) можливість розширення функціональності без зміни базової архітектури (додавання нових сторінок/модулів);

2) можливість підключення нових джерел рекомендацій та додаткових фільтрів;

3) можливість збільшення обсягу даних користувачів і відгуків без повної переробки системи;

е) вимоги до сумісності:

1) підтримка актуальних версій браузерів Google Chrome, Mozilla Firefox, Microsoft Edge, Safari;

2) коректне відображення інтерфейсу на екранах різної роздільної здатності;

ж) вимоги до локалізації:

1) підтримка української мови інтерфейсу;

2) можливість додавання англійської та інших мов у майбутньому;

3) коректне форматування дат відповідно до локалі;

з) вимоги до підтримки:

1) структурований код і модульна організація компонентів для спрощення супроводу;

2) можливість логування та діагностики помилок;

3) документування API та ключових модулів застосунку для подальшого розвитку.

1.4 Постановка завдання роботи

Розроблення вебзастосунку «Movie App» спрямоване на вирішення проблеми незручного пошуку, відбору та персоналізованого перегляду фільмів у розрізнених сервісах. Запропоноване рішення має об’єднати в одному інтерфейсі каталог фільмів, інструменти персоналізації та соціальну взаємодію користувачів (оцінки й відгуки).

Основні завдання роботи:

- аналіз вимог: дослідити потреби користувачів щодо пошуку фільмів, формування персональних списків і рекомендацій; визначити функціональні та нефункціональні вимоги до системи;
- проєктування архітектури: спроєктувати клієнт-серверну архітектуру застосунку з поділом на frontend і backend, інтеграцією із зовнішнім API фільмів та власним API користувацьких даних;
- реалізація функціоналу: програмно реалізувати реєстрацію та автентифікацію користувачів, пошук і фільтрацію фільмів, сторінки деталей, списки «Обране» та «Watchlist», систему оцінок і відгуків, персональні рекомендації;
- забезпечення безпеки: впровадити рольову модель доступу, захищені механізми автентифікації та авторизації, безпечну взаємодію клієнта і сервера;

- тестування та оптимізація: виконати перевірку коректності основних сценаріїв, стабільності роботи API, зручності інтерфейсу й адаптивності; оптимізувати швидкодію інтерфейсу та запитів.

Цілі розробки:

- створити зручний і зрозумілий інструмент для пошуку та вибору фільмів;
- забезпечити персоналізований користувацький досвід через рекомендації та індивідуальні списки;
- надати функціонал соціальної взаємодії через оцінки, коментарі та публічні відгуки;
- забезпечити надійність, безпеку та масштабованість застосунку для подальшого розвитку.

1.5 Висновки за розділом 1

Аналіз предметної області показав, що сучасні вебзастосунки для роботи з кіно-контентом є популярними серед користувачів, однак не всі вони повною мірою задовольняють потреби персоналізації та зручної взаємодії з фільмами. Існуючі рішення, такі як IMDb, Letterboxd і Trakt, надають широкий функціонал для перегляду інформації про фільми, оцінювання та ведення списків, проте мають певні обмеження.

Зокрема, частина функцій у цих сервісах є обмеженою у безкоштовній версії, а рівень персоналізації рекомендацій або можливостей модерації контенту не завжди відповідає сучасним вимогам. Крім того, не всі системи забезпечують гнучку взаємодію з контентом, наприклад, можливість окремого додавання фільмів до обраного без необхідності їх оцінювання чи позначення як переглянутих.

Основними викликами у даній предметній області є створення зручного та інтуїтивного інтерфейсу, забезпечення персоналізованих рекомендацій, ефективна організація користувацьких даних, а також

реалізація механізмів взаємодії між користувачами, зокрема через систему відгуків і коментарів.

Розроблюваний вебзастосунок спрямований на вирішення зазначених проблем шляхом поєднання функціоналу перегляду та пошуку фільмів із можливостями персоналізації, такими як обране, watchlist, оцінювання та коментування. Додатково реалізовано систему ролей користувачів (admin та superadmin) для модерації контенту, інтеграцію з зовнішнім API для отримання актуальної інформації про фільми, а також кешування даних для підвищення продуктивності. Таким чином, запропоноване рішення є більш гнучким і орієнтованим на потреби користувача порівняно з існуючими аналогами.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Структура системи

2.1.1 Основні компоненти системи

Архітектура застосунку CineRate побудована за клієнт-серверним підходом із поділом на фронтенд (інтерфейс користувача) та бекенд (API і бізнес-логіка). На клієнтській частині використано компонентну модель React, маршрутизацію та централізоване керування станом через Context API (AuthContext). Такий підхід забезпечує модульність, повторне використання компонентів і зручний супровід системи.

Шар представлення (View, Frontend UI):

- екрани (pages): Home, MovieDetails, Favorites, Watchlist, Profile, Login, Register;
- багаторазові UI-компоненти (components): Layout, Navbar, MovieCard, SearchBar, FavoriteHeartIcon, WatchlistBookmarkIcon;
- маршрутизація реалізована через react-router-dom (публічні та захищені сторінки).

Шар керування станом і клієнтської логіки (ViewModel / State Layer):

- AuthContext (AuthProvider) зберігає поточний стан користувача, обраного (favorites), списку перегляду (watchlist) та відгуків;
- реалізує операції автентифікації (register, login, logout) і дії користувача (додавання/видалення з обраного, watchlist, робота з відгуками);
- виконує синхронізацію з бекендом через HTTP-запити та зберігає JWT-токен у localStorage.

Шар доступу до даних (API Layer):

- модуль movies.js працює із зовнішнім API TMDB (пошук, популярні фільми, деталі, жанри, колекції) [4];
- модуль reviews.js працює з власним серверним API для отримання публічних відгуків і рейтингу спільноти;

- базова URL-адреса API конфігурується через змінні середовища (VITE_API_URL).

Серверний шар (Backend / Business Logic):

- REST API на Node.js + Express;
- контролери: UserController (автентифікація, обране, watchlist, ролі користувачів), ReviewController (оцінки, коментарі, переглянуті фільми, модерація коментарів);
- middleware CheckAuth виконує перевірку JWT і авторизацію запитів;
- валідація вхідних даних виконується на рівні серверних обробників.

Шар моделі даних (Model / Database):

- MongoDB як основна база даних;
- Mongoose-моделі;
- User (ім’я, email, хеш пароля, роль, списки фільмів);
- Review (зв’язок із користувачем, movieId, стан перегляду, оцінка, коментар);
- індексація (унікальна пара user + movieId) запобігає дублюванню відгуків.

Шар зовнішніх інтеграцій:

- TMDB API — джерело метаданих фільмів (каталог, жанри, рейтинги, постери, деталі);
- Локальне сховище браузера (localStorage) — збереження JWT-токена сесії користувача.

2.1.2 Взаємодія компонентів системи

Взаємодія між компонентами системи у застосунку відбувається за такою схемою.

Користувацький інтерфейс – Контекст/стан (AuthContext, сторінки, компоненти):

- сторінки (Home, MovieDetails, Favorites, Watchlist, Profile) і віджети (MovieCard, SearchBar, Navbar) відображають дані зі стану застосунку;
- дії користувача (пошук, додавання в обране, додавання у watchlist, оцінювання, коментування) передаються у AuthContext та API-функції для обробки;
- зміни стану оновлюють інтерфейс без перезавантаження сторінки.

Контекст/стан – Сервісний шар (API):

- AuthContext викликає backend API для автентифікації, роботи з обраним, watchlist, відгуками та ролями;
- сторінки також використовують модуль movies.js для отримання даних з TMDB (популярні, топові, жанри, пошук, деталі);
- результати запитів повертаються у стан і відображаються в UI.

Сервіси – Серверна частина (Express):

- backend маршрути (/auth, /favorites, /watchlist, /reviews, /admin) приймають запити від клієнта;
- middleware авторизації перевіряє JWT токен та права доступу;
- контролери реалізують бізнес-логіку й формують уніфіковані відповіді для клієнта.

Сервіси – Зовнішні інтеграції:

- фронтенд інтегрується з TMDB API для отримання каталогу фільмів, жанрів, деталей і рейтингів;
- отримані дані комбінуються з локальними даними користувача (оцінки, обране, переглянуте) для формування персональних рекомендацій.

2.2 Функціонування системи

2.2.1 Основні процеси системи

Функціонування системи можна описати через основні процеси, які відбуваються під час використання програми.

Процес реєстрації та входу:

– користувач реєструється через ім’я, email і пароль або входить у вже створений обліковий запис;

– сервер хешує пароль, створює користувача та видає JWT-токен;

– токен зберігається локально в браузері і використовується для авторизованих запитів..

Процес перегляду та пошуку фільмів:

- користувач відкриває головну сторінку, виконує пошук або фільтрує фільми за жанром;
- система отримує дані з TMDB (popular, top rated, search, genres);
- результати відображаються у вигляді карток фільмів.

Процес формування персональних рекомендацій:

- система аналізує обране користувача та високі оцінки переглянутих фільмів;
- визначаються пріоритетні жанри, завантажуються релевантні фільми та домішуються топові фільми спільноти;
- рекомендації кешуються на 24 години для зменшення навантаження і пришвидшення повторного доступу.

Процес роботи з обраним, watchlist і відгуками:

- користувач додає/видаляє фільми з обраного та watchlist;
- користувач позначає перегляд, ставить оцінку і залишає коментар до фільму;
- система зберігає зміни в базі даних і оновлює клієнтський стан.

Процес взаємодії зі спільнотою та модерації:

- для сторінки фільму завантажуються публічні відгуки, середня оцінка та кількість коментарів;
- адміністратори/суперадміністратори мають інструменти керування ролями та модерації коментарів;
- доступ до адміністративних операцій контролюється ролями користувачів.

2.2.3 Логіка роботи системи

Логіка роботи системи базується на таких принципах.

Принцип розділення відповідальності:

- UI-компоненти відповідають за відображення та події;
- AuthContext керує станом автентифікації і діями користувача;
- API-шар і backend-контролери виконують запити та бізнес-логіку.

Принцип захищеної автентифікації:

- паролі не зберігаються у відкритому вигляді, використовується хешування;
- доступ до захищених маршрутів здійснюється лише за валідним JWT-токеном;
- ролі (user, admin, superadmin) визначають доступ до адміністративних дій.

Принцип реактивного оновлення інтерфейсу:

- після будь-якої операції (toggle favorite/watchlist, save review) стан оновлюється одразу;
- компоненти автоматично перемальовуються згідно з новими даними;
- користувач отримує швидкий зворотний зв’язок без ручного оновлення сторінки.

Принцип персоналізації контенту:

- рекомендації будуються на основі індивідуальних дій користувача (обране, оцінки, переглянуті фільми);
- для підвищення якості добірки персональні результати поєднуються з глобальним топом;
- уже переглянуті фільми виключаються з блоку рекомендацій.

Принцип оптимізації продуктивності:

- дані рекомендацій кешуються в локальному сховищі з часовим обмеженням актуальності;

– повторні звернення до API зменшуються, а швидкість завантаження інтерфейсу зростає;

– помилки мережі обробляються з поверненням зрозумілих повідомлень користувачу.

2.3 Вибір мови програмування та інструментарію для розробки програми

2.3.1 Мова програмування

В процесі вибору було проведено порівняльний аналіз мов програмування JavaScript, Python та Java.

Порівняння мов програмування наведено в табл. 2.1.

Таблиця 2.1 – Порівняння мов програмування

Критерій	JavaScript	Python	Java
Кросплатформність	Висока (веб)	Висока	Середня
Продуктивність	Середня	Середня	Висока
Синтаксис	Простий	Простий	Складніший
Підтримка	Вбудована	Обмежена	Є
Екосистема	Дуже розвинена (npm)	Розвинена	Дуже розвинена
Використання у вебі	Висока	Висока	Середня
Швидкість розробки	Основна мова	Обмежене	Обмежене

Для розробки вебзастосунку з перегляду та оцінювання фільмів було обрано мову програмування JavaScript [5], яка використовується як на клієнтській, так і на серверній частині застосунку. На фронтенді застосовується бібліотека React [6]. На бекенді — середовище виконання Node.js з фреймворком Express [7]-[8].

Вибір JavaScript зумовлений низкою факторів, що відповідають вимогам проєкту.

Однією з ключових переваг є універсальність мови. JavaScript дозволяє використовувати єдину мову програмування як для клієнтської, так і для серверної частини, що значно спрощує розробку, зменшує поріг входу та підвищує швидкість створення застосунку.

Важливим фактором є висока швидкість розробки. Завдяки використанню бібліотеки React можна будувати інтерфейс у вигляді компонентів, що дозволяє повторно використовувати код і спрощує підтримку проєкту. Також застосування сучасних інструментів, таких як Vite, забезпечує швидку збірку та оновлення застосунку під час розробки [9].

Ще однією перевагою є підтримка асинхронного програмування. Використання механізмів async/await дозволяє ефективно працювати з API (наприклад, TMDB API) та обробляти запити до серверу без блокування основного потоку виконання.

JavaScript має одну з найбільших екосистем серед мов програмування. Велика кількість бібліотек і пакетів (npm) дозволяє швидко реалізовувати функціонал, включаючи авторизацію, роботу з базами даних, обробку HTTP-запитів та створення адаптивного інтерфейсу.

2.3.1.1 Опис переваг та обмежень вибраної мови

Переваги.

Висока швидкість розробки досягається завдяки використанню React та готових бібліотек, що дозволяє швидко створювати інтерфейс і бізнес-логіку застосунку.

Єдина мова для фронтенду і бекенду значно спрощує структуру проєкту та зменшує складність підтримки.

Асинхронність JavaScript дозволяє ефективно працювати з зовнішніми API та базами даних, що є критично важливим для застосунку, який активно взаємодіє з TMDB API.

Велика спільнота розробників і розвинена екосистема забезпечують наявність великої кількості готових рішень і документації.

Гнучкість інтерфейсу завдяки React дозволяє створювати сучасний, адаптивний і динамічний UI, що відповідає вимогам до зручності користування.

Обмеження.

JavaScript є динамічно типізованою мовою, що може призводити до помилок під час виконання програми, якщо не використовуються додаткові інструменти типізації.

Велика кількість бібліотек може призводити до залежностей і ускладнювати підтримку проєкту при неправильній організації коду.

Продуктивність може бути нижчою порівняно з мовами, що компілюються у машинний код, однак для вебзастосунків це не є критичним.

Безпека потребує додаткової уваги, оскільки JavaScript широко використовується у вебі і є потенційною ціллю для атак (XSS, CSRF), тому необхідно реалізовувати захисні механізми.

2.3.2 Середовище розробки

У процесі вибору було проведено порівняльний аналіз середовищ розробки, результати якого наведено в табл. 2.2

Таблиця 2.2 – Порівняння середовищ розробки

Критерій	Visual Studio Code	WebStorm	Visual Studio
Підтримка JavaScript	Повна	Повна	Часткова
Швидкість роботи	Висока	Середня	Низька
Споживання ресурсів	Низьке	Середнє	Високе
Плагіни	Дуже багато	Багато	Обмежено
Інтеграція з Git	Вбудована	Вбудована	Вбудована
Вартість	Безкоштовний	Платний	Частково платний
Кросплатформність	Так	Так	Обмежена

У результаті Visual Studio Code було обрано як основне середовище розробки завдяки його легкості, гнучкості, широким можливостям розширення та повній підтримці технологій, що використовуються у вебзастосунку.

Однією з ключових переваг є універсальність середовища. Visual Studio Code підтримує велику кількість мов програмування, зокрема JavaScript, що дозволяє використовувати його як для фронтенд-розробки (React), так і для бекенд-розробки (Node.js, Express).

Важливим фактором є легкість і швидкість роботи. На відміну від більш важких IDE, Visual Studio Code швидко запускається, споживає менше ресурсів і забезпечує комфортну роботу навіть на середніх за продуктивністю комп’ютерах.

Середовище має потужну систему розширень. За допомогою плагінів можна значно розширити функціональність редактора, додавши підтримку ESLint, Prettier, Tailwind CSS, Git, REST Client та інших інструментів, необхідних для розробки вебзастосунку.

Ще однією перевагою є вбудована інтеграція з системами контролю версій. Visual Studio Code має зручний інтерфейс для роботи з Git, що дозволяє відслідковувати зміни в коді, створювати коміти та працювати з репозиторіями без використання сторонніх програм.

Також середовище підтримує інтегрований термінал, що дозволяє виконувати команди (npm, запуск сервера, збірка проєкту) безпосередньо в IDE, що значно прискорює процес розробки.

2.3.2.1 Обґрунтування вибору та опис його можливостей

Visual Studio Code обрано через його гнучкість, швидкість роботи та широку підтримку сучасних вебтехнологій, що повністю відповідає вимогам проєкту.

Редактор коду забезпечує інтелектуальне автодоповнення (IntelliSense), підсвічування синтаксису, рефакторинг коду та швидкий доступ до документації. Це значно підвищує продуктивність розробника.

Можливості налагодження дозволяють запускати та тестувати JavaScript-код, встановлювати точки зупину, переглядати змінні та стек викликів, що спрощує пошук і виправлення помилок.

Інтегрований термінал дає змогу виконувати всі необхідні команди для роботи з проєктом, такі як запуск сервера (Node.js), встановлення залежностей (npm) та запуск збірки (Vite).

Підтримка розширень дозволяє адаптувати середовище під конкретні потреби проєкту. Наприклад, розширення для React, Tailwind CSS та роботи з API значно спрощують розробку інтерфейсу та взаємодію з сервером.

2.3.3 База даних

У процесі вибору було проведено порівняльний аналіз баз даних, результати якого наведено в табл. 2.3

Таблиця 2.3 – Порівняння баз даних

Критерій	MongoDB	PostgreSQL
Тип бази даних	NoSQL (документоорієнтована)	Реляційна
Структура даних	Гнучка (JSON/BSON)	Строга (таблиці)
Продуктивність	Висока для CRUD-операцій	Висока для складних запитів
Масштабованість	Висока (горизонтальна)	Висока (вертикальна + горизонтальна)
Робота зі зв’язками	Обмежена	Повноцінна (JOIN)
Простота розробки	Висока	Середня
Інтеграція з Node.js	Дуже зручна	Зручна
Контроль цілісності	Низький	Високий

Для забезпечення роботи вебзастосунку обрано базу даних MongoDB — сучасну NoSQL базу даних, яка забезпечує гнучке зберігання даних та добре підходить для вебзастосунків із динамічною структурою [10].

Вибір MongoDB зумовлений особливостями проєкту, зокрема необхідністю зберігання інформації про користувачів, їхні оцінки, коментарі та списки обраних фільмів. Дані мають напівструктурований характер, що робить використання NoSQL-підходу більш доцільним порівняно з традиційними реляційними базами даних.

MongoDB використовується разом із бібліотекою Mongoose, яка забезпечує зручну взаємодію з базою даних у середовищі Node.js, включаючи опис схем, валідацію даних та роботу з колекціями.

Для порівняння було розглянуто також реляційну базу даних PostgreSQL, яка могла б використовуватись як альтернатива. PostgreSQL забезпечує строгість структури, підтримку складних запитів та високий рівень надійності. Однак для даного вебзастосунку, де дані мають гнучку структуру (відгуки, списки, взаємодії користувачів), використання MongoDB є більш доцільним, оскільки дозволяє спростити розробку та зменшити складність реалізації.

2.3.3.1 MongoDB

MongoDB — це документоорієнтована база даних, у якій дані зберігаються у форматі BSON (аналог JSON). Це дозволяє зберігати складні об’єкти без необхідності жорсткої схеми.

Переваги:

- гнучка структура даних. Відсутність жорсткої схеми дозволяє легко змінювати структуру даних у процесі розробки;
- зручна інтеграція з JavaScript. Дані у форматі JSON добре поєднуються з Node.js, що спрощує розробку;
- масштабованість. MongoDB підтримує горизонтальне масштабування, що дозволяє обробляти великі обсяги даних;

- швидкість розробки. Завдяки простій структурі та використанню Mongoose, реалізація серверної логіки значно спрощується;
- підтримка вкладених структур. Можна зберігати, наприклад, оцінку і коментар користувача в одному документі.Обмеження. Необхідність синхронізації з хмарою для обміну даними між пристроями.

Обмеження:

- відсутність складних JOIN-запитів, характерних для реляційних баз даних;
- можливість дублювання даних при неправильному проєктуванні структури;
- менша строгість у контролі цілісності даних порівняно з SQL-базами.

2.4 Вимоги до технічних і програмних засобів

Для роботи застосунку необхідний пристрій із такими мінімальними характеристиками:

– операційна система: Android 8.0+ / iOS 13+ / Windows 10+ / macOS 11+;

– процесор: частота не нижче 1.2 ГГц;

– оперативна пам’ять: мінімум 2 ГБ;

– вільне місце на диску: не менше 200 МБ;

– підключення до інтернету для завантаження даних і синхронізації;

– веб-браузер.

Апаратне забезпечення для розробки:

– процесор: Intel Core i5 або аналогічний (рекомендується для комфортної роботи із середовищем розробки та сервером);

– оперативна пам’ять: не менше 8 Гб (рекомендується 16 Гб для одночасної роботи з frontend, backend та базою даних);

– вільне місце на диску: не менше 5–10 Гб для збереження проєкту, залежностей та інструментів розробки;

– пристрій користувача: будь-який пристрій із сучасним веббраузером (Chrome, Firefox, Edge тощо).

Програмне забезпечення для розробки:

– операційна система: Windows 10/11, Linux або macOS;

– середовище розробки: Visual Studio Code;

– Node.js: середовище виконання JavaScript (версія 16 або новіша);

– npm або yarn: менеджер пакетів для встановлення залежностей;

– фреймворк React: для розробки клієнтської частини застосунку;

– Express.js: для створення серверної частини застосунку;

– MongoDB: база даних для зберігання інформації про користувачів, відгуки та списки фільмів;

– браузер: Google Chrome або аналогічний для тестування інтерфейсу;

– Git: система контролю версій для управління кодом.

2.5 Висновки за розділом 2

У другому розділі було розглянуто архітектуру вебзастосунку для оцінювання та рекомендації кінофільмів, визначено основні компоненти системи та їхню взаємодію. Застосунок реалізовано за клієнт-серверною архітектурою, де клієнтська частина відповідає за інтерфейс користувача, а серверна — за обробку запитів, бізнес-логіку та роботу з базою даних.

Вибір мови програмування JavaScript обґрунтовано її універсальністю та можливістю використання як на клієнтській, так і на серверній стороні. Використання бібліотеки React дозволяє створювати сучасний, динамічний та зручний інтерфейс користувача, тоді як Node.js і Express забезпечують ефективну реалізацію серверної логіки.

Середовище розробки Visual Studio Code обрано завдяки своїй легкості, гнучкості та широкій підтримці плагінів, що значно спрощує процес

розробки. Використання бази даних MongoDB забезпечує гнучке зберігання даних та добре підходить для роботи з напівструктурованою інформацією, такою як відгуки та взаємодії користувачів.

Таким чином, обраний стек технологій забезпечує ефективну розробку, масштабованість та зручність підтримки вебзастосунку, що повністю відповідає поставленим вимогам до програмного продукту.

3 РОЗРОБКА ПРОГРАМИ

3.1 Архітектура системи

Архітектура вебзастосунку для оцінювання та обговорення кінофільмів побудована за принципом клієнт-серверної моделі (Client–Server Architecture) з використанням сучасного підходу до розробки SPA (Single Page Application). Така архітектура забезпечує розподіл логіки між клієнтською та серверною частинами, що дозволяє підвищити продуктивність, масштабованість та зручність підтримки системи.

Основні компоненти архітектури.

Client (Клієнтська частина) — відповідає за взаємодію з користувачем та відображення даних:

- сторінки застосунку (Home, MovieDetails, Favorites, Watchlist, Profile, Login, Register);
- компоненти інтерфейсу (MovieCard, Navbar, SearchBar, ReviewForm);
- управління станом (React hooks, Context API);
- взаємодія з сервером через API-запити.

Server (Серверна частина) — реалізує бізнес-логіку застосунку:

- маршрути (authRoutes, movieRoutes, reviewRoutes, userRoutes);
- контролери (AuthController, ReviewController, UserController);
- обробка запитів і відповідей;
- автентифікація користувачів за допомогою JWT.

Model (Модель даних) — описує структуру даних і взаємодію з базою даних:

- колекції MongoDB (User, Review);
- збереження інформації про користувачів, оцінки, коментарі, обране та список перегляду;
- використання Mongoose для роботи з базою даних [11].

Services (Сервіси) — забезпечують допоміжну функціональність:

- AuthService — реєстрація та авторизація користувачів;
- ReviewService — обробка оцінок і коментарів;
- UserService — управління профілем користувача;
- TMDBService — отримання інформації про фільми із зовнішнього API;
- TokenService — генерація та перевірка JWT-токенів.

Зовнішні API — використовуються для отримання додаткових даних:

- TMDB API — отримання інформації про фільми (опис, рейтинг, постери, актори тощо).

Взаємодія компонентів системи відбувається за такою послідовністю: користувач виконує дії в інтерфейсі клієнтської частини (Client), після чого клієнт формує запити до серверної частини (Server) та, залежно від типу операції, до зовнішнього API TMDB. Сервер обробляє запити, виконує перевірку автентифікації та прав доступу, звертається до моделей даних (Model) у базі MongoDB і формує відповідь. Отримані результати повертаються клієнту, де оновлюється стан застосунку та повторно відображається інтерфейс відповідно до актуальних даних.

Схема взаємодії компонентів системи зображена на рис. 3.1.

3.1.1 Структурна схема системи

Проєкт організовано за принципом модульної декомпозиції: програмний код поділено на логічно відокремлені підсистеми клієнтської частини, серверної частини, шару даних і зовнішніх інтеграцій відповідно до їх функціонального призначення.

Структурну схему системи зображено на рис. 3.2.

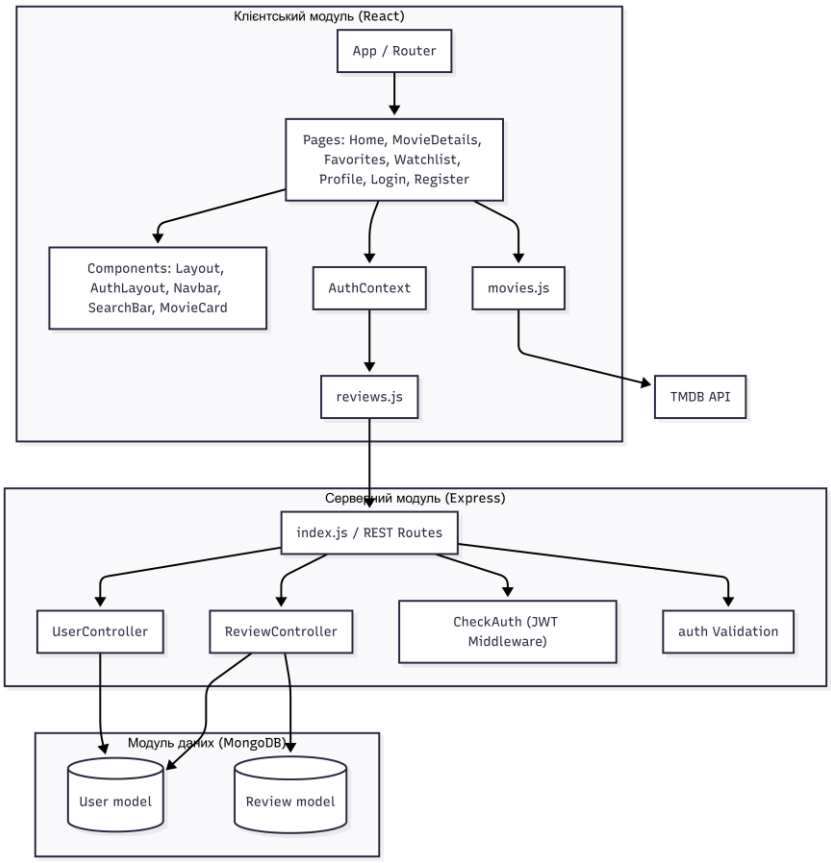


Рисунок 3.1 – Схема взаємодії компонентів системи

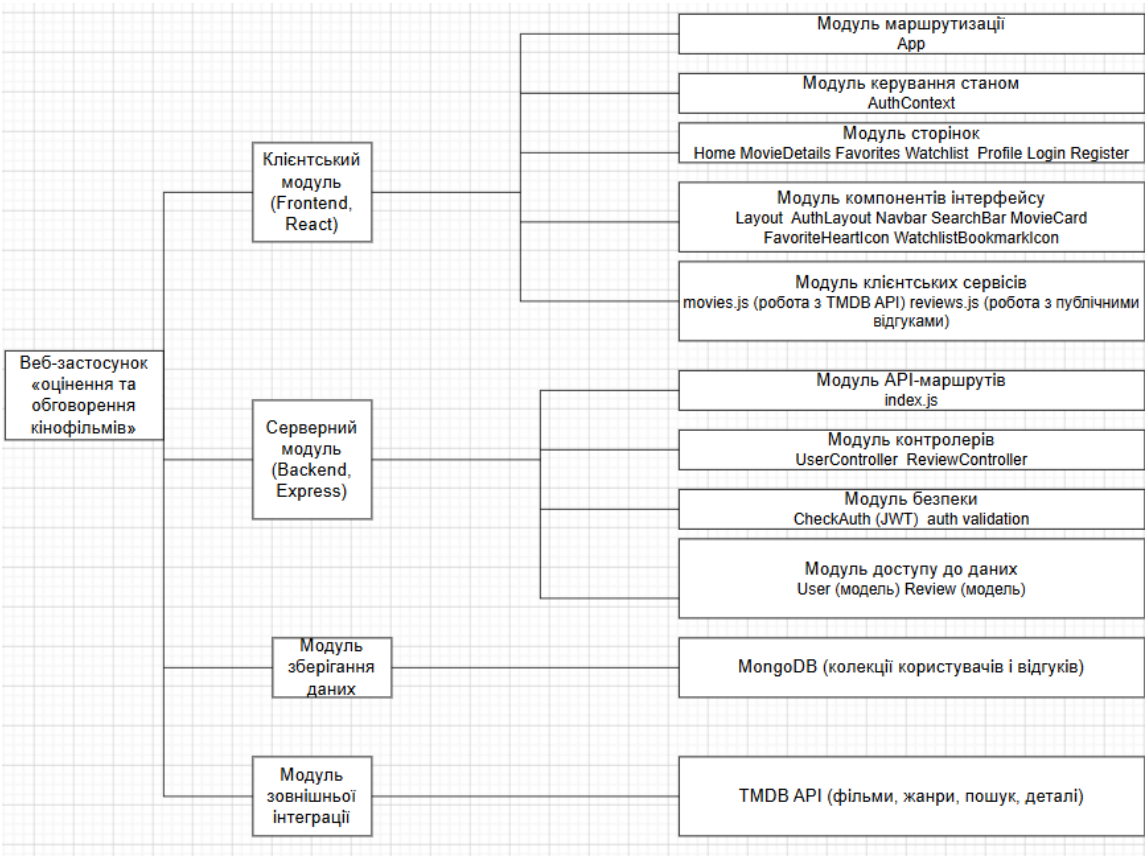


Рисунок 3.2 – Структурна схема системи

3.2 Функціонування системи

3.2.1 Функціональна схема

Вебзастосунок «CineRate» функціонує на основі взаємопов’язаних алгоритмів, що забезпечують пошук фільмів, персоналізацію рекомендацій, збереження дій користувача та стабільну роботу клієнт-серверної частини. Архітектурно система поєднує клієнтський вебінтерфейс (React), зовнішній сервіс TMDB API (каталог фільмів) і власний REST API з базою даних (облікові записи, рецензії, ролі, колекції користувача).

У режимі онлайн застосунок виконує повний цикл обробки даних: автентифікацію, завантаження фільмів, синхронізацію обраного/списку перегляду/рецензій та формування рекомендацій. Для підвищення швидкодії використовується локальне кешування рекомендацій у браузері.

Розглянемо основні модулі, що лежать в основі програми (рис. 3.3)

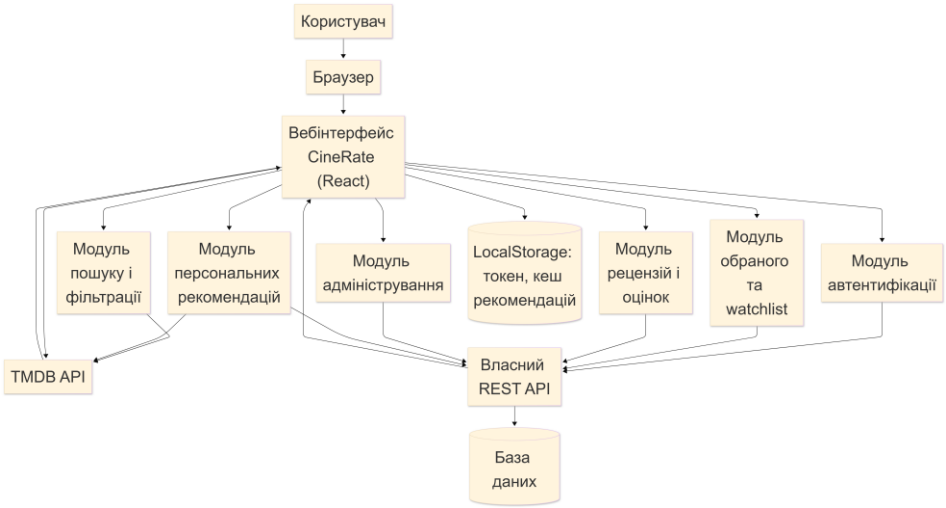


Рисунок 3.3 – Функціональна схема розробленої програми

3.2.2 Опис процесу обміну даними у системі

Процес обміну даними у вебзастосунку «CineRate» організовано за клієнт-серверною моделлю з реактивним оновленням інтерфейсу: після отримання нових даних стан компонентів автоматично змінюється, а інтерфейс повторно рендериться. Обмін виконується одночасно з двома

джерелами: TMDB API (фільми, жанри, рейтинги) та власним REST API (авторизація, обране, watchlist, рецензії, ролі користувачів).

Основні етапи потоку даних:

- введення дії користувачем: через вебінтерфейс користувач виконує дію (пошук фільму, вхід у систему, додавання в обране, виставлення оцінки);
- обробка запиту на клієнті: React-компонент або AuthContext формує запит і викликає відповідний модуль API;
- звернення до зовнішнього/внутрішнього сервісу: дані запитуються з TMDB API або з власного REST API;
- збереження та кешування: токен сесії та кеш рекомендацій зберігаються у localStorage, а серверні дані зберігаються в базі даних застосунку;
- реактивне оновлення інтерфейсу: після відповіді сервера оновлюється стан (useState, useEffect, Context), і зміни автоматично відображаються на сторінці.

Основні етапи потоку даних подано на рис. 3.4

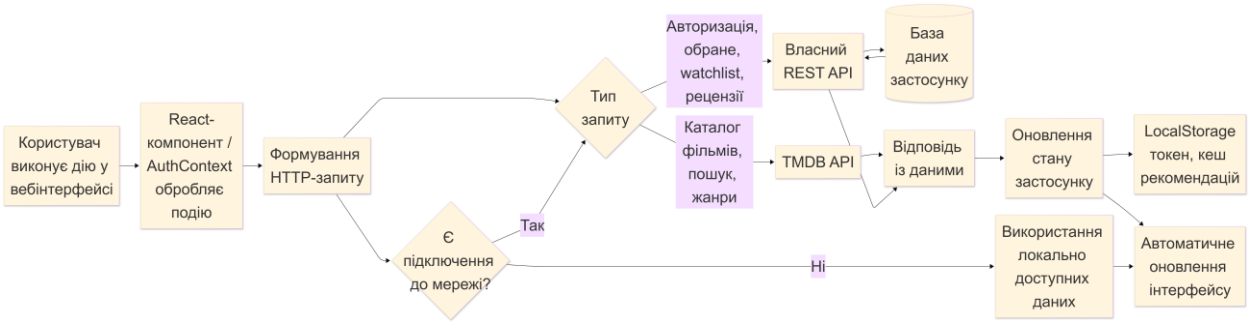


Рисунок 3.4 – Схема процесу обміну даними у системі

3.2.3 Сценарії використання

Основні сценарії використання вебзастосунку «CineRate» описують типові дії користувачів та адміністраторів під час взаємодії із системою. Сценарії використання дозволяють визначити основні функціональні

можливості застосунку, логіку роботи окремих модулів та взаємодію користувача із програмною системою. Описані сценарії охоплюють як базові можливості для гостей системи, так і персоналізовані функції для авторизованих користувачів та адміністраторів.

Сценарій «Реєстрація та вхід у систему»:

- користувач відкриває сторінку реєстрації або входу;
- користувач вводить ім’я, email і пароль (або email і пароль для входу);
- система виконує валідацію даних і надсилає запит на сервер;
- у разі успіху система зберігає токен сесії та завантажує профіль користувача;
- користувач отримує доступ до персоналізованих функцій (обране, watchlist, рецензії).

Сценарій «Пошук і перегляд інформації про фільм»:

- користувач вводить назву фільму в пошуковому рядку;
- система надсилає запит до TMDB API та отримує результати;
- користувач застосовує фільтр за жанром (за потреби);
- система відображає список знайдених фільмів;
- користувач відкриває картку фільму для перегляду детальної інформації.

Сценарій «Додавання фільму до обраного / watchlist»:

- авторизований користувач відкриває картку фільму;
- користувач натискає кнопку додавання в «Обране» або «Watchlist»;
- система надсилає запит на REST API для оновлення колекції;
- сервер зберігає зміну в базі даних;
- інтерфейс автоматично оновлює стан кнопок і списків.

Сценарій «Оцінювання та рецензування фільму»:

- користувач відкриває сторінку фільму;

- користувач виставляє оцінку та (за бажанням) додає текст рецензії;
- система зберігає відгук через API;
- дані рецензії оновлюються у профілі користувача та в перегляді фільму;
- відгук враховується в подальшому формуванні персональних рекомендацій.

Сценарій «Отримання персональних рекомендацій»:

- користувач переходить на головну сторінку;
- система аналізує обране, оцінки та переглянуті фільми;
- формується персоналізована добірка (з урахуванням популярних фільмів спільноти);
- результати кешуються локально для швидшого повторного завантаження;
- користувач отримує оновлений список «Рекомендовані для вас».

Сценарій «Адміністрування користувачів і контенту»:

- адміністратор входить у систему з відповідними правами;
- адміністратор відкриває панель керування;
- адміністратор призначає або скасовує роль адміністратора за email;
- за потреби адміністратор видаляє недоречні коментарі/рецензії;
- система фіксує зміни та оновлює дані для всіх користувачів.

Наведені сценарії використання демонструють основні можливості вебзастосунку та відображають типові варіанти взаємодії користувачів із системою. Вони також дозволяють оцінити повноту реалізованого функціоналу та забезпечують основу для подальшого тестування й удосконалення програмного продукту.

Діаграма варіантів використання зображена на рис. 3.5.

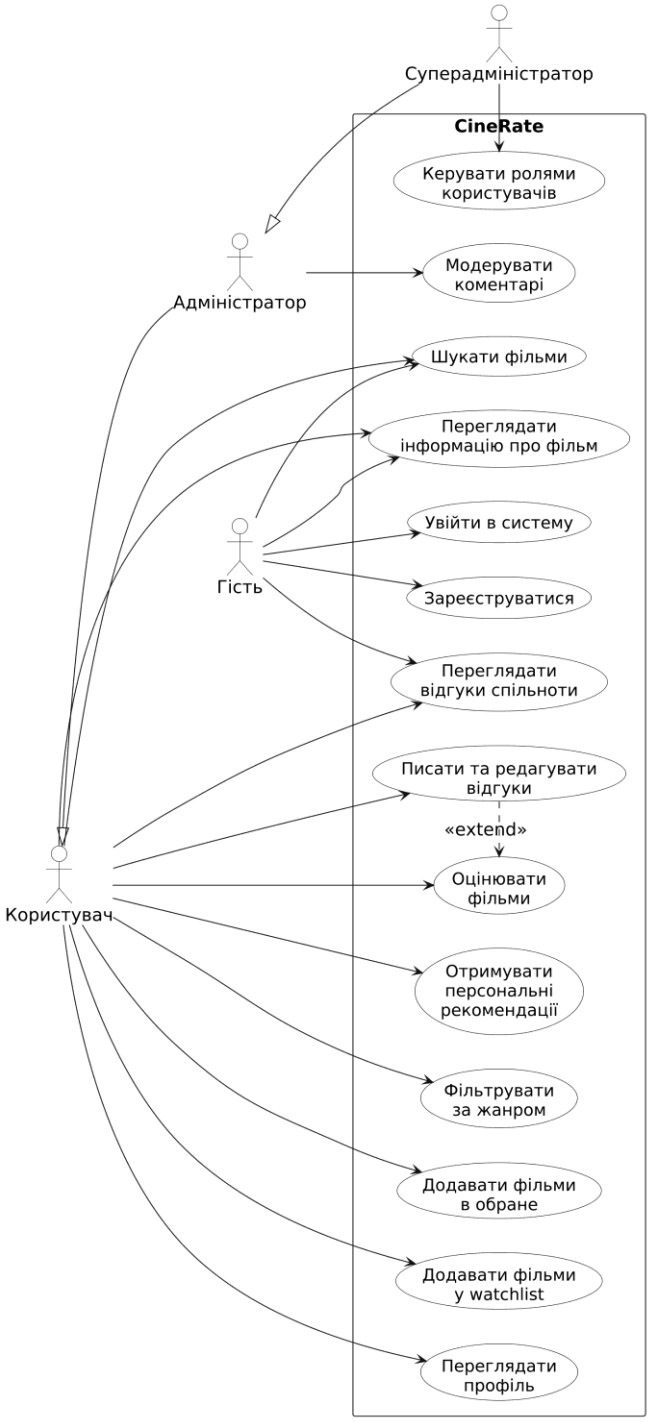


Рисунок 3.5 – Діаграма варіантів використання

3.3 Опис модулів системи

3.3.1 Модуль реєстрації та авторизації

3.3.1.1 Алгоритм автентифікації та авторизації

Алгоритм автентифікації у застосунку реалізований через перевірку JWT-сесії та вхід за email/пароль.

JWT (JSON Web Token) забезпечує надійний механізм автентифікації завдяки використанню цифрового підпису та шифрування даних. Токен містить інформацію про користувача і перевіряється сервером при кожному запиті, що унеможливлює підробку або зміну даних без відповідного секретного ключа.

Алгоритм автентифікації та авторизації зображено на рис. 3.6.

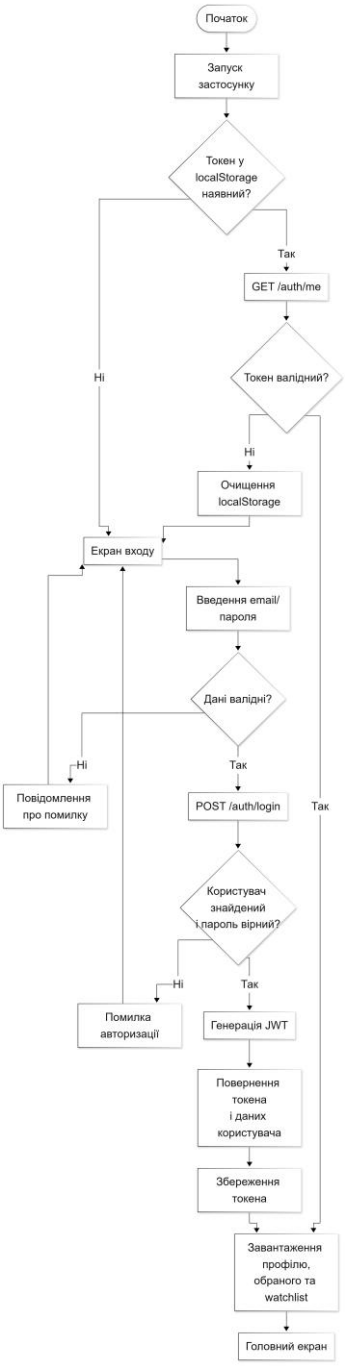


Рисунок 3.6 — Алгоритм автентифікації та авторизації користувача

3.3.1.2 Функціональність модуля

Модуль реєстрації та авторизації забезпечує такі функції.

- Реєстрація нових користувачів:
- реєстрація через email та пароль;
- валідація полів (email, мінімальна довжина пароля, обов'язкове ім'я);
- хешування пароля перед збереженням у базі даних;
- автоматична авторизація після успішної реєстрації (видача JWT).

Автентифікація існуючих користувачів:

- вхід через email та пароль;
- перевірка пароля через bcrypt;
- видача JWT-токена з часом дії 30 днів;
- збереження сесії користувача на клієнті (localStorage);
- відновлення сесії при повторному запуску застосунку.

Управління сесією та профілем:

- отримання даних поточного користувача через GET /auth/me;
- завершення сесії (видалення токена та очищення стану);
- синхронізація персональних даних після входу (обране та watchlist).

3.3.1.3 Реалізація та використані технології

Модуль реалізовано за допомогою таких технологій:

- React + Context API — керування станом автентифікації на клієнті;
- Axios — HTTP-взаємодія між клієнтом і сервером;
- Node.js + Express — серверна логіка автентифікації;
- MongoDB + Mongoose — зберігання облікових записів;
- bcrypt — безпечне хешування паролів;
- jsonwebtoken (JWT) — токен-орієнтована авторизація;

- express-validator — серверна валідація вхідних даних;
- localStorage — збереження токена на клієнті.

Основні класи та файли модуля:

- src/context/AuthContext.jsx — клієнтський сервіс авторизації (login/register/logout, відновлення сесії);
- src/pages/Login.jsx — екран входу;
- src/pages/Register.jsx — екран реєстрації;
- backend/Controllers/UserController.js — обробники register, login, getMe;
- backend/utils/CheckAuth.js — middleware перевірки JWT;
- backend/validations/auth.js — правила валідації для входу/реєстрації.

На рис. 3.7 зображено екран входу в систему.

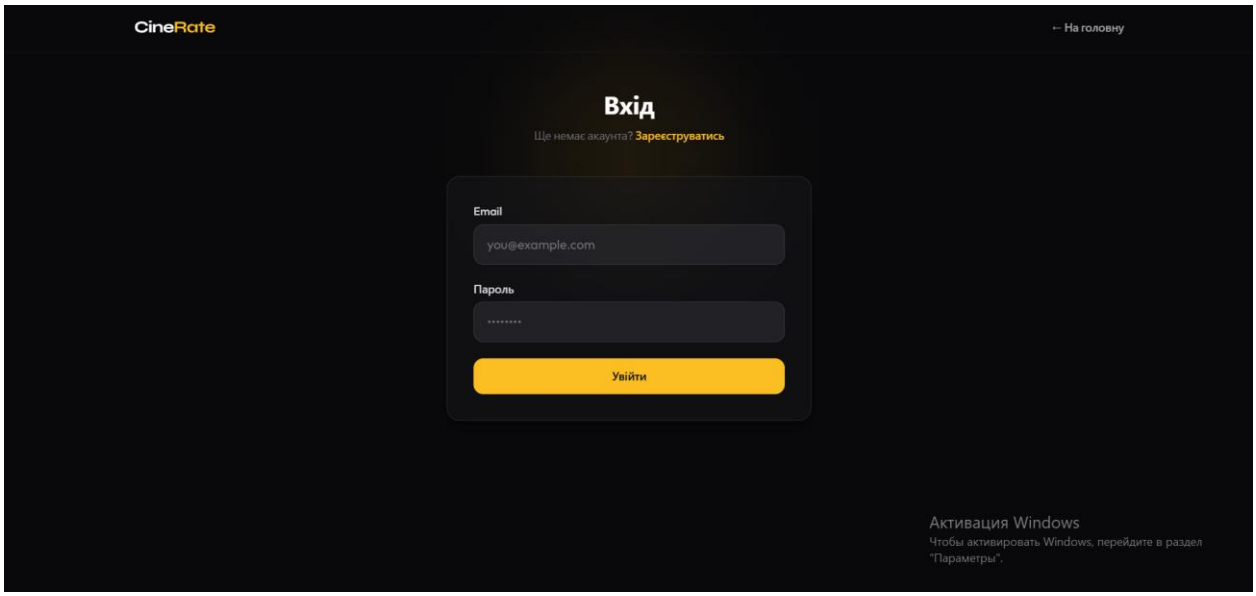


Рисунок 3.7 — Екран входу

3.3.2 Модуль каталогу, пошуку та рекомендацій фільмів

3.3.2.1 Алгоритм формування стрічки фільмів

Модуль формує вміст головного екрана залежно від параметрів пошуку та стану автентифікації користувача. Якщо користувач

авторизований, то йому показуються фільми на основі тих, які він високо оцінив, а також додається 30 % найпопулярніших фільмів у відповідних жанрах. Такий підхід дозволяє поєднати персоналізовані рекомендації з актуальними популярними стрічками, що підвищує зацікавленість користувача та допомагає відкривати новий контент.

Якщо користувач не авторизований, йому показуються фільми, які спільнота високо оцінила та які можна вважати такими, що сподобаються більшості користувачів.

На рис. 3.8 показано алгоритм формування стрічки фільмів та рекомендацій.

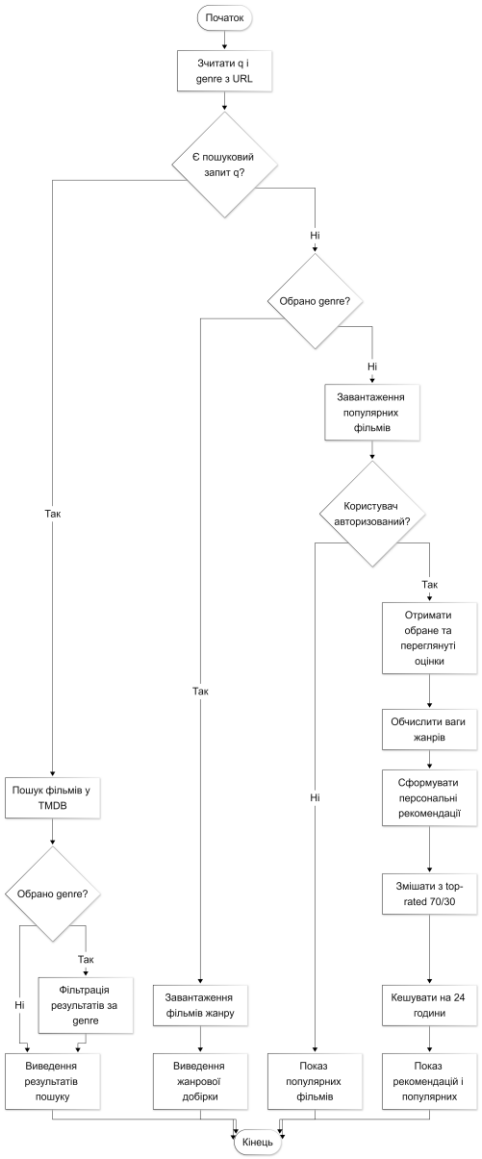


Рисунок 3.8 — Алгоритм формування стрічки фільмів та рекомендацій

3.3.2.2 Функціональність модуля

Модуль каталогу, пошуку та рекомендацій забезпечує такі функції.

Пошук і фільтрація:

- пошук фільмів за назвою;
- фільтрація за жанром;
- комбінація пошуку та жанру одночасно;
- відображення станів завантаження й помилок.

Каталогізація контенту:

- показ популярних фільмів;
- показ топ-фільмів за рейтингом спільноти;
- відображення карток фільмів із постером, назвою та рейтингом.

Персоналізація:

- генерація рекомендацій за вподобаннями користувача;
- виключення вже переглянутих фільмів із рекомендацій;
- локальне кешування рекомендацій.

3.3.2.3 Реалізація та використані технології

Модуль реалізовано за допомогою таких технологій:

- React + React Router — побудова сторінок і маршрутизація;
- TMDB API — джерело даних про фільми, жанри та рейтинги;
- Fetch API — HTTP-запити до зовнішнього API;
- localStorage — кешування рекомендацій і параметрів сесії.

Основні класи та файли модуля:

- src/pages/Home.jsx — головна логіка завантаження каталогу та рекомендацій;
- src/components/SearchBar.jsx — пошук і фільтр за жанром;
- src/components/MovieCard.jsx — картка фільму;
- src/api/movies.js — методи роботи з TMDB API.

На рис. 3.9 зображена головна сторінка з рекомендованими фільмами та пошуком.

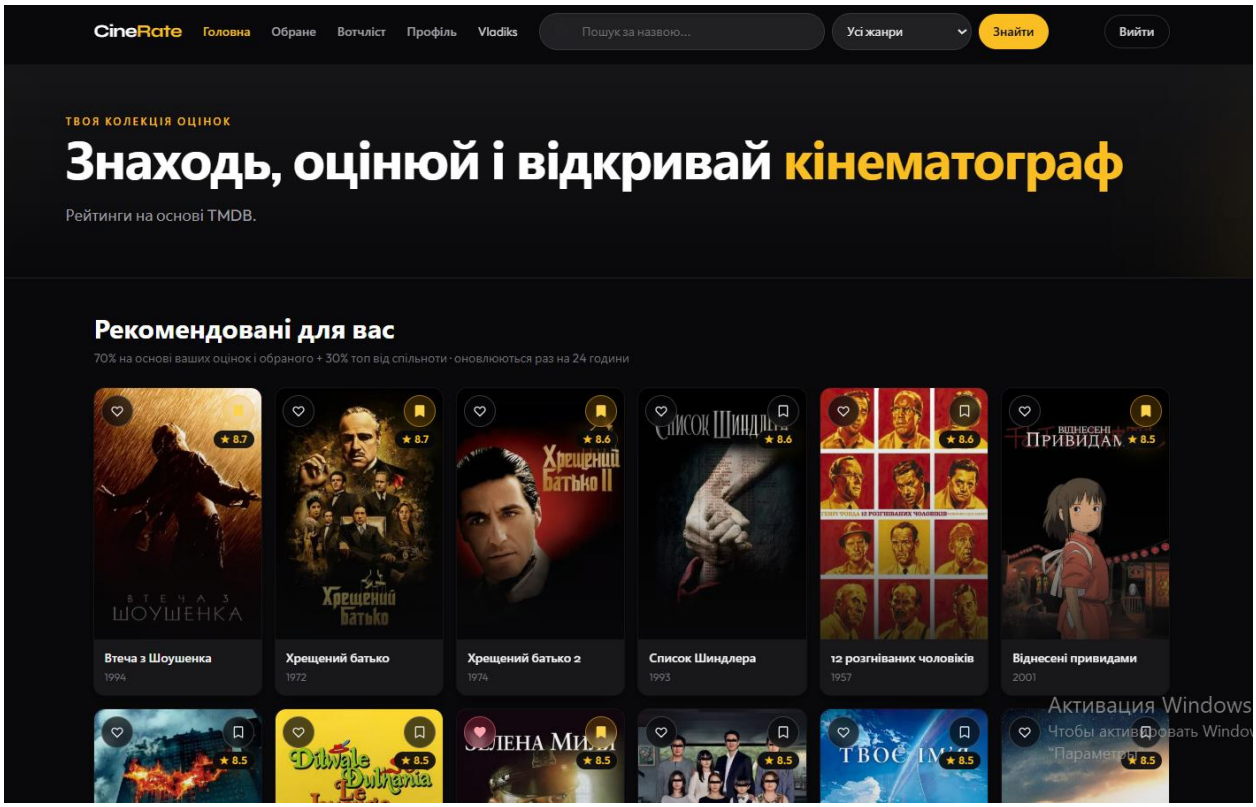


Рисунок 3.9 — Головна сторінка з рекомендованими фільмами та пошуком

3.3.3 Модуль персональних списків (обране та вотчліст)

3.3.3.1 Алгоритм додавання фільму до обраного/вотчліста

Алгоритм взаємодії з персональними списками реалізований через кнопки в картці фільму та на сторінці деталей:

- користувач натискає кнопку «Обране» або «Вотчліст»;
- перевіряється авторизація користувача;
- якщо користувач не авторизований, виконується перехід на сторінку входу;
- якщо авторизований, надсилається запит POST /favorites/toggle або POST /watchlist/toggle;
- сервер додає/видаляє фільм зі списку;

– клієнт оновлює локальний стан списків без перезавантаження сторінки.

Алгоритм додавання фільму до персонального списку зображено на рис. 3.10.

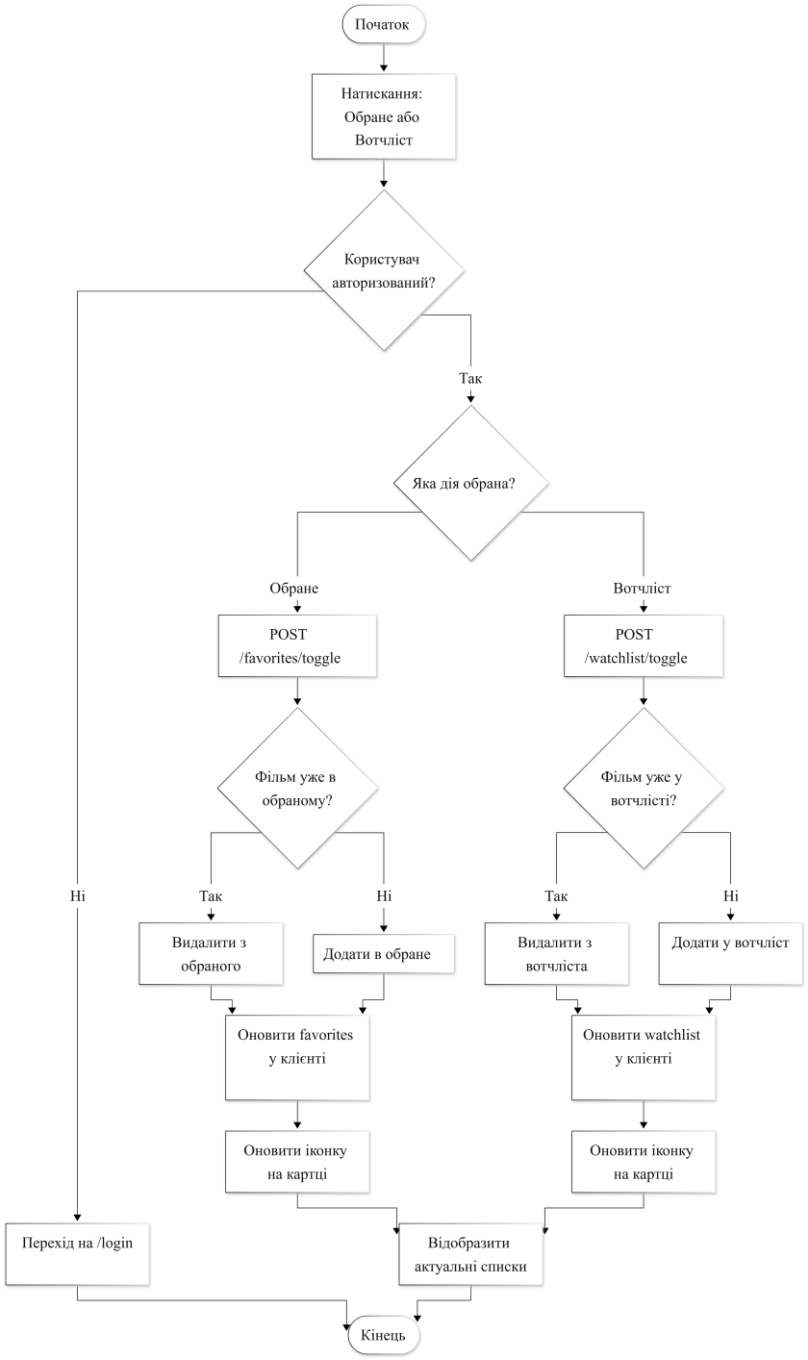


Рисунок 3.10 — Алгоритм додавання фільму до персонального списку

3.3.3.2 Функціональність модуля

Модуль персональних списків забезпечує такі функції.

Управління обраним:

- додавання фільмів в обране;
- видалення фільмів з обраного;
- перегляд повного списку обраного.

Управління вотчлістом:

- додавання фільмів до списку «Подивитися»;
- видалення фільмів зі списку;
- перегляд сторінки вотчліста.

Синхронізація і доступ:

- синхронізація списків із сервером для авторизованих користувачів;
- доступ до списків з будь-якого пристрою після входу в акаунт;
- захист маршрутів списків від неавторизованого доступу.

3.3.3.3 Реалізація та використані технології

Модуль реалізовано за допомогою таких технологій:

- React Context API — централізоване керування станом списків;
- Axios — взаємодія з backend API;
- Express + MongoDB — збереження списків користувача;
- JWT — авторизація запитів до захищених endpoint [12].

Основні класи та файли модуля:

- src/context/AuthContext.jsx — методи toggleFavorite, toggleWatchlist;
- src/pages/Favorites.jsx — екран обраного;
- src/pages/Watchlist.jsx — екран вотчліста;
- src/components/MovieCard.jsx — UI-керування списками;
- backend/Controllers/UserController.js — серверна логіка списків;

– backend/index.js — маршрути /favorites і /watchlist.

На рис. 3.11 зображено сторінку з обраними фільмами.

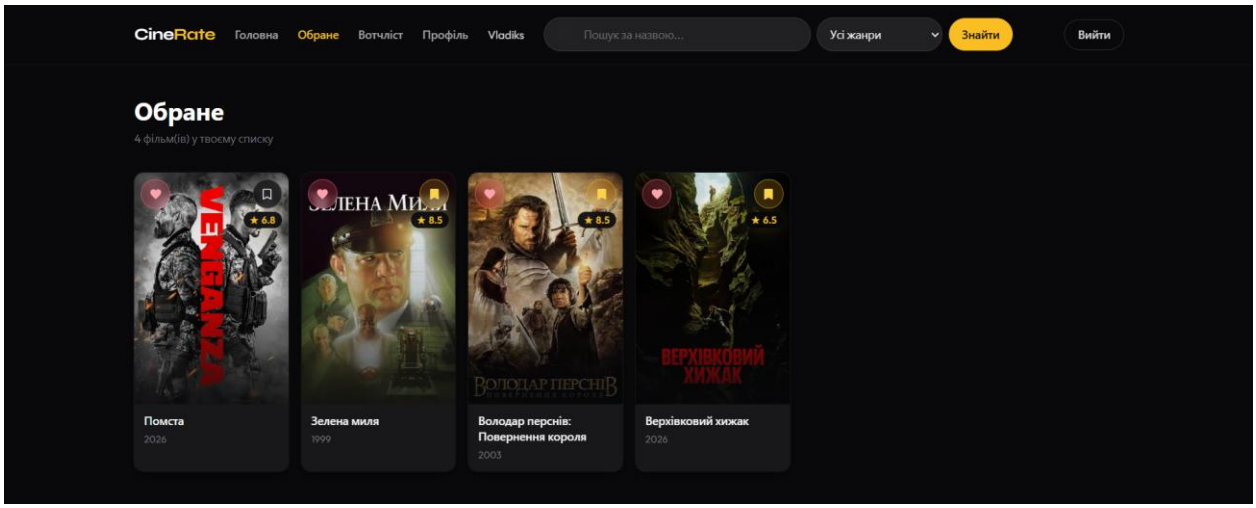


Рисунок 3.11 — Сторінка з обраними фільмами

На рис. 3.12 зображено сторінку з фільмами які користувач планує подивитись в майбутньому.

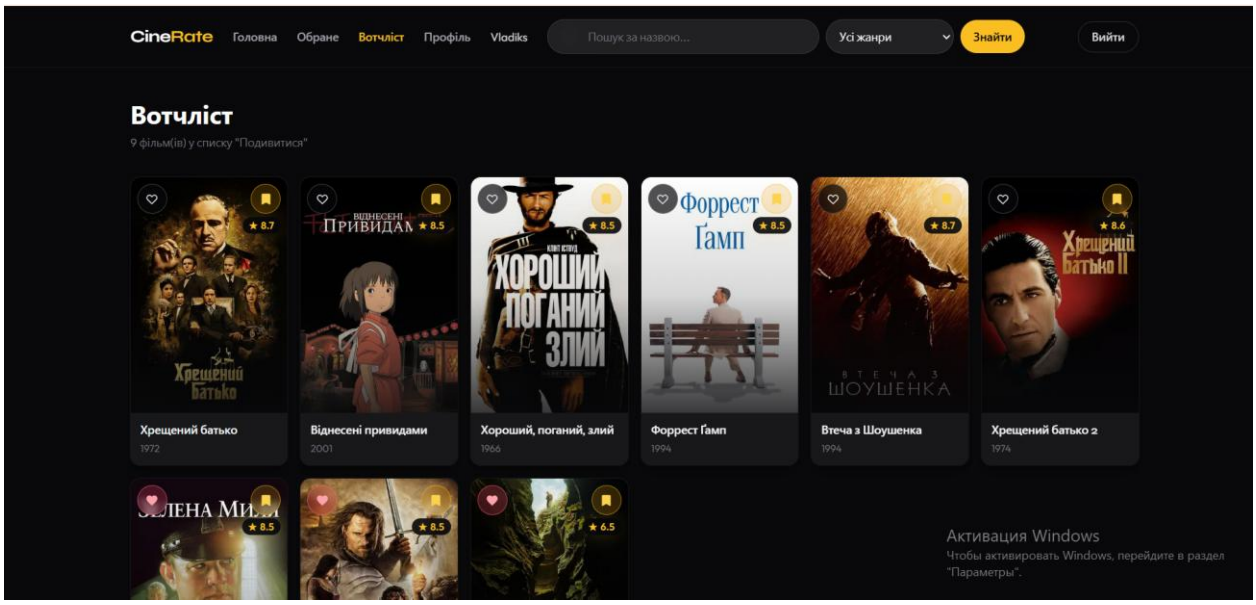


Рисунок 3.12 — Сторінка вотчліст

3.3.4 Модуль відгуків, оцінок

3.3.4.1 Функціональність модуля

Модуль забезпечує такі функції.

Робота з відгуками:

- додавання власної оцінки фільму;
- додавання та редагування текстового коментаря;
- позначення фільму як переглянутого.

Показники спільноти:

- відображення середнього рейтингу фільму;
- відображення кількості оцінок і коментарів;
- показ стрічки коментарів користувачів.

Профіль користувача:

- список переглянутих фільмів;
- персональна статистика (кількість оцінених, кількість коментарів, середня оцінка);
- редагування власних оцінок і коментарів із профілю.

3.3.4.2 Реалізація та використані технології

Модуль реалізовано за допомогою таких технологій:

- React — форми, стани та візуалізація статистики профілю;
- Node.js/Express — API для відгуків та агрегованих даних;
- MongoDB/Mongoose — зберігання відгуків, коментарів і зв’язку з користувачем;
- JWT — доступ до персональних операцій із відгуками.

Основні класи та файли модуля:

- src/pages/MovieDetails.jsx — форма відгуку та блок коментарів;
- src/pages/Profile.jsx — статистика й редагування власних відгуків;
- src/api/reviews.js — запит публічних даних спільноти;

– backend/Controllers/ReviewController.js — логіка getMyReview, upsert, communityData;

– backend/models/Review.js — модель відгуку.

На рис. 3.13 зображено форму для відгуків.

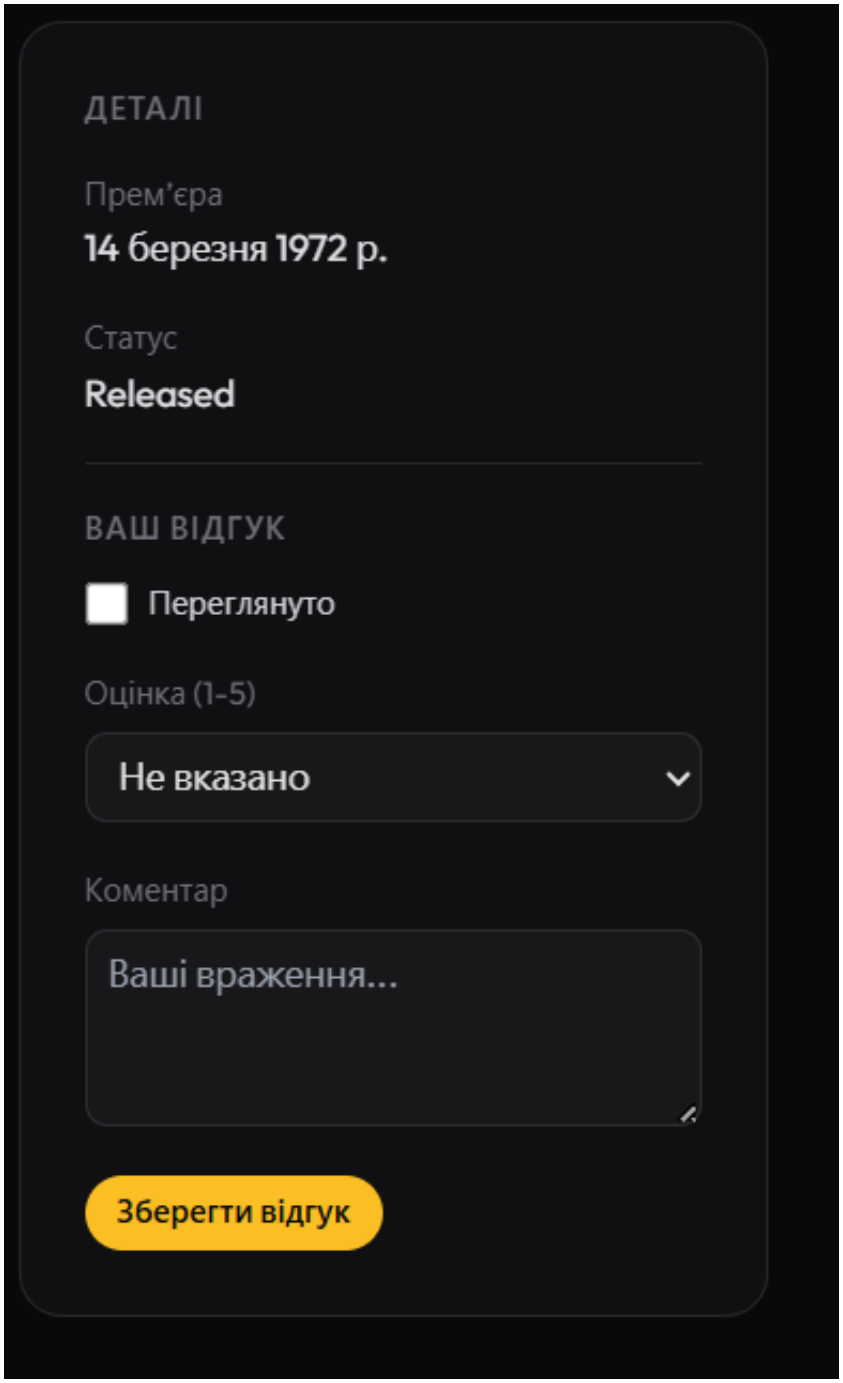


Рисунок 3.13 — Форма для відгуків

На рис. 3.14 профіль користувача зі статистикою його активності.

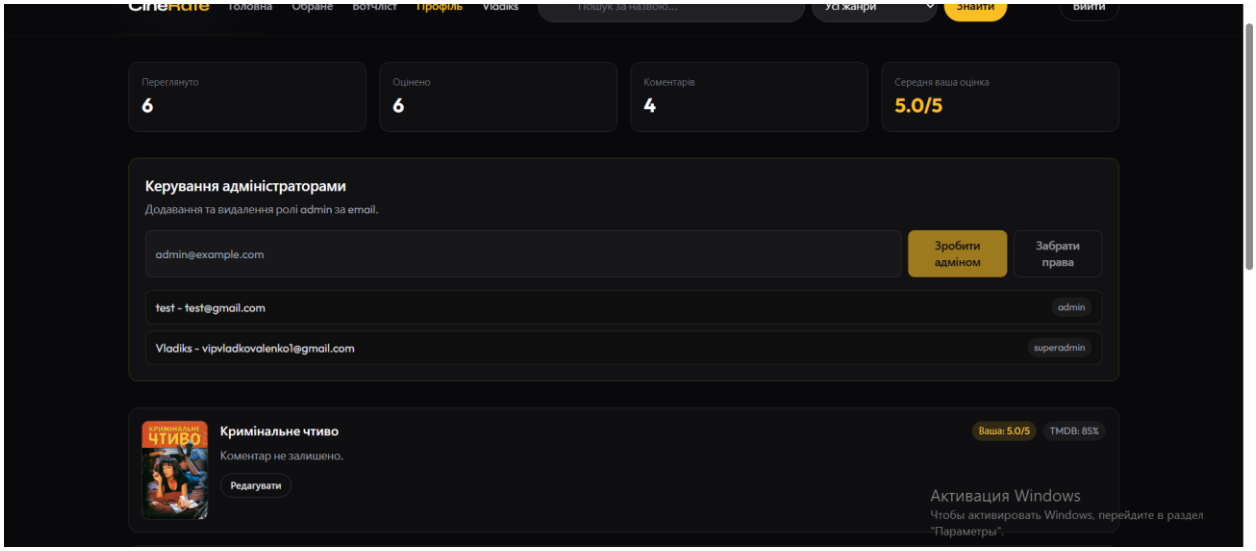


Рисунок 3.14 — Профіль користувача зі статистикою

3.4 Проєктування бази даних

3.4.1 Структура бази даних

База даних застосунку CineRate побудована на документо-орієнтованій моделі та реалізована в MongoDB з використанням Mongoose. Зберігання даних виконується на серверній частині застосунку, а клієнт взаємодіє з базою через REST API [13].

Дані про фільми (назви, постери, жанри, рейтинги TMDB) не дублюються повністю в локальній БД застосунку, а завантажуються з зовнішнього джерела TMDB API.

Основними сутностями бази даних є користувачі та відгуки до фільмів. Колекція користувачів містить облікові дані, роль, а також персональні списки фільмів (обране та вотчліст). Колекція відгуків містить оцінки, коментарі та ознаку перегляду фільму.

В табл. 3.1 показано концептуальну модель бази даних.

Таблиця 3.1 – Опис концептуальної моделі бази даних

№	Сутність	Опис	Атрибут	Тип даних	Опис
1	2	3	4	5	6
1	Users	Користувачі	_id	ObjectId	Унікальний ідентифікатор користувача
			name	String	Ім’я користувача
			email	String	Email користувача
			passwordHash	String	Хеш пароля
			favorites	Array	Список обраних фільмів
			watchlist	Array	Список фільмів для перегляду
			role	String	Роль користувача (user, admin, superadmin)
			createdAt	Date	Час створення запису
			updatedAt	Date	Час останнього оновлення
2	Reviews	Відгуки	_id	ObjectId	Унікальний ідентифікатор відгуку
			user	ObjectId	ID користувача (FK
			movieId	Number	ID фільму
			isWatched	Boolean	Ознака, що фільм переглянуто

Кінець таблиці 3.1.

1	2	3	4	5	6
2	Reviews	Відгуки	rating	Number	Оцінка користувача (1..5)
			comment	String	Текстовий коментар
			createdAt	Date	Час створення відгуку
			updatedAt	Date	Час останнього оновлення

3.4.2 Взаємозв'язки між таблицями

Взаємозв’язки між сутностями реалізовано таким чином:

- Users – Reviews (один до багатьох);
- Reviews.user – Users._id (багато до одного).

Один користувач може залишити багато відгуків, але кожен відгук належить лише одному користувачу.

Для кожної пари користувач + фільм зберігається один унікальний відгук.

Схему взаємозв’язків між таблицями зображено на рис. 3.15.

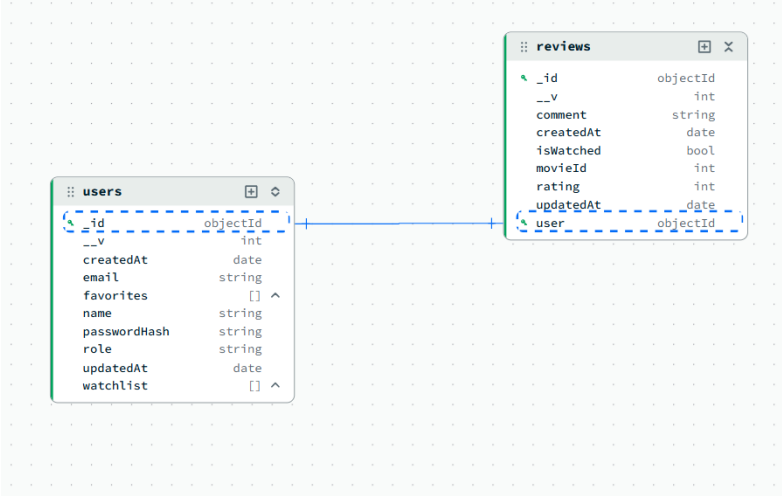


Рисунок 3.15 — Схема взаємозв'язків між таблицями

3.4.3 Індексація та оптимізація

Для оптимізації запитів до бази даних використовуються такі індекси:

- унікальний індекс на Users.email для швидкого пошуку користувача під час авторизації;
- складений унікальний індекс на Reviews(user, movieId) для запобігання дублюванню відгуків;
- індексація за _id у колекціях Users і Reviews (стандартна для MongoDB).

Ці індекси забезпечують коректність даних, прискорюють автентифікацію користувачів, отримання відгуків та роботу персональних модулів застосунку.

3.5 Висновки за розділом 3

У третьому розділі виконано проєктування та описано реалізацію вебзастосунку. Визначено структуру клієнтської та серверної частин, опрацьовано основні алгоритми роботи системи, а також обґрунтовано вибір використаних технологій.

У межах розділу розглянуто ключові функціональні модулі застосунку: модуль реєстрації та авторизації користувачів, модуль каталогу та пошуку фільмів, модуль персональних списків (обране та вотчліст), а також модуль відгуків, оцінювання та профілю користувача. Для кожного модуля описано призначення, функціональні можливості та послідовність виконання основних операцій.

Архітектура застосунку реалізована за клієнт-серверним підходом із розподілом відповідальності між інтерфейсом користувача (React) та серверною логікою (Node.js/Express). Така побудова забезпечує масштабованість, зручність супроводу та можливість подальшого розширення функціоналу.

Окрему увагу приділено проєктуванню бази даних. Сформовано структуру колекцій Users і Reviews, визначено їх атрибути, зв’язки та правила цілісності. Запропоновано індексацію, що підвищує продуктивність під час авторизації, обробки відгуків і виконання типових запитів.

Отже, результати третього розділу підтверджують, що розроблена структура системи є цілісною, технологічно обґрунтованою та придатною для практичного використання. Реалізовані модулі покривають основні сценарії роботи користувача та створюють основу для подальшого розвитку застосунку.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Опис застосування програми

Вебзастосунок призначений для пошуку фільмів, формування персональних добірок та взаємодії користувачів через оцінки і коментарі.

Функції програми:

- автентифікація користувача (реєстрація, вхід, вихід);
- пошук фільмів за назвою;
- фільтрація фільмів за жанром;
- перегляд детальної інформації про фільм;
- додавання/видалення фільмів в обране;
- додавання/видалення фільмів у вотчліст;
- додавання та редагування власного відгуку (оцінка, коментар, статус перегляду);
- перегляд коментарів спільноти до фільму;
- перегляд статистики користувача у профілі.

Мінімальні вимоги до середовища розробки/запуску:

- Node.js (рекомендовано LTS 18+);
- npm;
- доступ до MongoDB (локально або хмарно);
- доступ до мережі Інтернет для отримання даних з TMDB API

4.2 Характеристики програми

Програмним продуктом є вебзастосунок CineRate, створений для пошуку, перегляду та персоналізованого відбору фільмів. Система забезпечує зручний доступ до каталогу фільмів, отримання детальної інформації про вибрані позиції, а також формування індивідуальних списків користувача.

Застосунок надає можливість виконувати пошук за назвою, фільтрацію за жанрами, додавати фільми в обране та вотчліст, залишати

власні оцінки й коментарі, а також переглядати реакцію спільноти. Для авторизованих користувачів реалізовано персональні рекомендації на основі історії взаємодії з контентом.

Програмна система підтримує збереження користувацьких даних, швидке оновлення інтерфейсу без перезавантаження сторінки та стабільну роботу в сучасних браузерах. Використання клієнт-серверної архітектури забезпечує масштабованість, зручність супроводу та можливість подальшого розширення функціоналу.

4.3 Інструкція з експлуатації програми

Для запуску проєкту на локальному комп’ютері потрібно виконати такі кроки:

- клонувати репозиторій у локальну директорію;
- встановити залежності frontend (у корені проєкту): npm install;
- встановити залежності backend (у папці backend): npm install;
- запустити backend (у папці backend): npm run start;
- запустити frontend (у корені проєкту): npm run dev;
- відкрити застосунок у браузері за адресою:http://localhost:5173/.

4.4 Тестування та результати роботи програмного забезпечення

На рис. 4.1 демонструється відображення вебзастосунку у браузері Chrome.

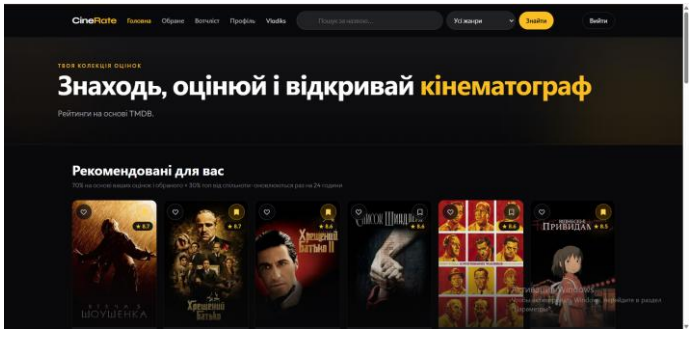


Рисунок 4.1 – Відображення у браузері Google Chrome

На рис. 4.2 демонструється відображення вебзастосунку у браузері Microsoft Edge.

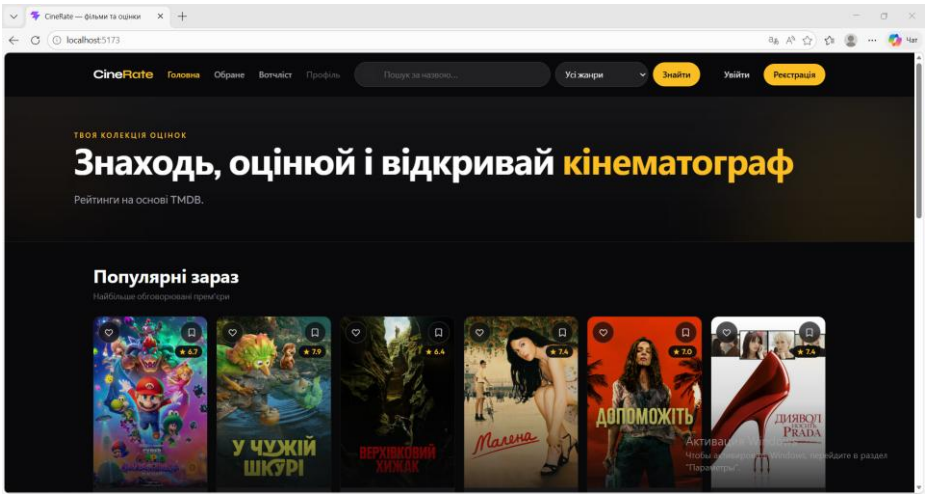


Рисунок 4.2 – Відображення у браузері Microsoft Edge

В табл. 4.1 наведено опис тест-кейсу функції авторизації.

Таблиця 4.1 – Опис тест-кейсу функції авторизації

Мета	Перевірити роботу функції авторизації користувача		
Крок	Опис	Очікуваний результат	Фактичний результат
1	2	3	4
1	Відправлення форми без введених даних	Спрацьовує валідація полів, відображається повідомлення про помилку	Валідація спрацювала, повідомлення про помилку відображено
2	Відправлення форми з некоректними даними	Система відхиляє вхід, з'являється повідомлення про помилку	Вхід відхилено, повідомлення про помилку відображено

Кінець таблиці 4.1

1	2	3	4
3	Відправлення форми з коректними даними	Виконується успішний вхід і переадресація на цільову сторінку	Вхід виконано, переадресація відбулася

В табл. 4.2 наведено опис тест-кейсу додавання фільму в обране.

Таблиця 4.2 – Опис тест-кейсу додавання фільму в обране

Мета	Перевірити роботу функції авторизації користувача		
Крок	Опис	Очікуваний результат	Фактичний результат
1	Натискання кнопки «В обране» неавторизованим користувачем	Система перенаправляє користувача на сторінку входу	Виконано перехід на сторінку входу
2	Натискання кнопки «В обране» авторизованим користувачем	Фільм додається до списку обраного, іконка змінює стан	Фільм додано, стан іконки оновлено
3	Повторне натискання кнопки «В обране»	Фільм видаляється з обраного, іконка повертається у початковий стан	Фільм видалено, стан іконки оновлено

В табл. 4.3 наведено опис тест-кейсу збереження відгуку.

Таблиця 4.3 – Опис тест-кейсу збереження відгуку

Мета	Перевірити роботу функції авторизації користувача		
Крок	Опис	Очікуваний результат	Фактичний результат
1	Введення невалідних даних (некоректна оцінка або надто довгий коментар)	Відображається повідомлення про помилку валідації, дані не зберігаються	Помилка валідації відображена, запис не збережено
2	Введення коректної оцінки та коментаря, натискання «Зберегти»	Дані відгуку зберігаються в базі даних	Відгук успішно збережено
3	Повторне відкриття Сторінки фільму	Відображаються оновлені оцінка/коментар і актуальна статистика	Дані відображаються коректно

4.5 Висновки за розділом 4

У четвертому розділі було розглянуто питання експлуатації, тестування та практичного використання розробленого вебзастосунку CineRate. Описано основні можливості системи, зокрема функції пошуку фільмів, авторизації користувачів, формування персональних списків, додавання оцінок і коментарів, а також перегляду рекомендацій та статистики профілю. Визначено мінімальні вимоги до програмного

середовища та наведено покрокову інструкцію для локального запуску клієнтської та серверної частин застосунку.

У межах тестування було перевірено роботу ключових функціональних модулів системи. Результати тест-кейсів підтвердили коректність реалізації механізмів авторизації, додавання фільмів до списку обраного та збереження користувацьких відгуків. Усі очікувані результати збіглися з фактичними, що свідчить про стабільність роботи застосунку та правильність взаємодії між клієнтською частиною, сервером і базою даних.

Проведене тестування також показало, що вебзастосунок коректно відображається у сучасних браузерах, забезпечує зручну взаємодію користувача з інтерфейсом і підтримує основні сценарії використання без критичних помилок. Отримані результати підтверджують працездатність програмного продукту та можливість його подальшого розвитку й практичного використання у сфері онлайн-сервісів для роботи з кіноконтентом.

ВИСНОВКИ

У дипломній роботі вирішено актуальне завдання розробки сучасного вебзастосунку для пошуку, персоналізації та оцінювання фільмів. Результатом роботи став програмний продукт CineRate, який поєднує зручний користувацький інтерфейс, стабільну серверну частину та функціональність, орієнтовану на реальні сценарії використання.

У процесі виконання роботи проведено аналіз предметної області та сучасних підходів до створення вебзастосунків. На основі цього обрано технологічний стек, що забезпечує баланс між швидкістю розробки, підтримуваністю та масштабованістю: React і Vite для клієнтської частини, Node.js і Express для серверної логіки, MongoDB і Mongoose для зберігання даних.

Спроєктовано та реалізовано клієнт-серверну архітектуру застосунку з розподілом відповідальності між модулями. Реалізовано автентифікацію та авторизацію користувачів, перегляд каталогу фільмів, пошук і фільтрацію за жанрами, роботу з персональними списками (обране, вотчліст), модуль відгуків і оцінювання, а також профіль користувача зі статистичними показниками. Для адміністративних ролей впроваджено функції модерації коментарів.

Окрему увагу приділено проєктуванню бази даних: визначено структуру колекцій, зв’язки між сутностями та механізми забезпечення цілісності даних. Використання індексів для критичних полів дало змогу оптимізувати типові операції авторизації, доступу до відгуків і обробки персональних даних користувача.

Проведене тестування підтвердило коректність реалізації ключових функціональних сценаріїв: вхід до системи, робота з персональними списками, збереження та оновлення відгуків. Отримані результати засвідчили стабільну роботу застосунку в межах поставлених вимог та готовність програмного рішення до практичного використання.

Отже, мету дипломної роботи досягнуто: створено функціональний вебзастосунок, який автоматизує процес взаємодії користувача з кіноконтентом, забезпечує персоналізацію досвіду перегляду та формує основу для подальшого розвитку.

Перспективи подальшого вдосконалення полягають у розширенні рекомендаційного модуля, інтеграції додаткових джерел даних, впровадженні розширеної аналітики поведінки користувачів і підвищенні рівня автоматизованого тестування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Letterboxd Official Website. URL: https://letterboxd.com (date of access: 20.04.2026).

2. IMDb Official Website. URL: https://www.imdb.com (date of access: 20.04.2026).

3. Trakt Official Website. URL: https://trakt.tv (date of access: 20.04.2026).

4. TMDB API Documentation. URL: https://developer.themoviedb.org/docs/getting-started (date of access: 20.04.2026).

5. JavaScript. MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (date of access: 20.04.2026).

6. React Documentation. URL: https://react.dev (date of access: 21.04.2026).

7. Node.js Documentation. URL: https://nodejs.org/en/docs (date of access: 21.04.2026).

8. Express.js Documentation. URL: https://expressjs.com (date of access: 21.04.2026).

9. Vite Documentation. URL: https://vitejs.dev/guide (date of access: 21.04.2026).

10. MongoDB Documentation. URL: https://www.mongodb.com/docs (date of access: 21.04.2026).

11. Mongoose Documentation. URL: https://mongoosejs.com/docs (date of access: 21.04.2026).

12. JWT Introduction – Auth0. URL: https://auth0.com/docs/secure/tokens/json-web-tokens (date of access: 21.04.2026).

13. MDN Web Docs – REST API. URL: https://developer.mozilla.org/en-US/docs/Glossary/REST (date of access: 21.04.2026).

ДОДАТОК А

Текст програми

A.1 Файл AuthContext.jsx

Відновлення сесії, login/register/logout:

```
useEffect(() => {
    const token = readToken();
    if (!token) {
      setAuthReady(true);
      return;
    }
    request("/auth/me", {
      headers: {
        Authorization: `Bearer ${token}`,
      },
    })
      .then(async (userData) => {
        setUser({ ...userData, token });
        const favoritesData = await request("/favorites", {
          headers: {
            Authorization: `Bearer ${token}`,
          },
        });
        setFavorites(favoritesData.favorites ?? []);
        const watchlistData = await request("/watchlist", {
          headers: {
            Authorization: `Bearer ${token}`,
          },
        });
        setWatchlist(watchlistData.watchlist ?? []);
      })
      .catch(() => {
        localStorage.removeItem(STORAGE_TOKEN);
        setUser(null);
        setFavorites([]);
        setWatchlist([]);
        setReviewsByMovieId({});
      })
      .finally(() => {
        setAuthReady(true);
      });
  }, []);
  const register = useCallback(async ({ name, email,
password }) => {
    const data = await request("/auth/register", {
      method: "post",
      data: {
        name: name.trim(),
        email: email.trim().toLowerCase(),
        password,
      },
    });
    localStorage.setItem(STORAGE_TOKEN, data.token);
```

```
    setUser(data);
    setFavorites(data.favorites ?? []);
    setWatchlist(data.watchlist ?? []);
    setReviewsByMovieId({});
  }, []);
  const login = useCallback(async ({ email, password }) =>
{
    const data = await request("/auth/login", {
      method: "post",
      data: {
        email: email.trim().toLowerCase(),
        password,
      },
    });
    localStorage.setItem(STORAGE_TOKEN, data.token);
    setUser(data);
    setFavorites(data.favorites ?? []);
    setWatchlist(data.watchlist ?? []);
    setReviewsByMovieId({});
  }, []);
  const logout = useCallback(() => {
    localStorage.removeItem(STORAGE_TOKEN);
    setUser(null);
    setFavorites([]);
    setWatchlist([]);
    setReviewsByMovieId({});
  }, []);
```

A.2 Файл Home.jsx

У цьому файлі реалізовано основна логіка завантаження, пошуку, рекомендацій:

```
useEffect(() => {
    let cancelled = false;
    setLoading(true);
    setError(null);
    const load = async () => {
      try {
        if (query) {
          const data = await searchMovies(query);
          const filteredByGenre = selectedGenreId
            ? data.filter((movie) => (movie.genre_ids ??
[]).includes(selectedGenreId))
            : data;
          if (!cancelled) {
            setPopularMovies(filteredByGenre);
            setRecommendedMovies([]);
            setRecommendationSource("");
          }
          return;
```

```
        }
        if (selectedGenreId) {
          const          genreTop          =          await
discoverMoviesByGenres([selectedGenreId]);
          if (!cancelled) {
            setPopularMovies(genreTop);
            setRecommendedMovies([]);
            setRecommendationSource("");
          }
          return;
        }
        const popular = await getPopularMovies();
        if (!authReady || !isAuthenticated) {
          if (!cancelled) {
            setPopularMovies(popular);
            setRecommendedMovies([]);
            setRecommendationSource("");
          }
          return;
        }
        const                    cacheKey                    =
`cinerate_recommendations_${user?._id   ??   user?.email   ??
"user"}`;
        try {
          const cachedRaw = localStorage.getItem(cacheKey);
          if (cachedRaw) {
            const cached = JSON.parse(cachedRaw);
            const isFresh =
              cached?.createdAt &&
              Date.now()   -   Number(cached.createdAt)   <
RECOMMENDATION_CACHE_TTL_MS;
            if (isFresh && Array.isArray(cached?.movies)) {
              if (!cancelled) {
                setPopularMovies(popular);
                setRecommendedMovies(cached.movies);
                setRecommendationSource(
                  `${cached.source   ??   "Рекомендації"}   ·
оновлюються раз на 24 години`
                );
              }
              return;
            }
          }
        } catch {
          // ignore invalid cache
        }
  const mergedGenreIds = getWeightedGenreIds(
          favorites ?? [],
          watched ?? [],
          likedMovies.filter(Boolean)
        );
        let recommended = [];
        let source = "";
```

```
        if (mergedGenreIds.length > 0) {
          const discovered = await
discoverMoviesByGenres(mergedGenreIds);
          if (discovered.length > 0) {
            recommended = discovered;
            source = "На основі вашого обраного та високих
оцінок";
          }
        }
        const topRated = await getTopRatedMovies();
        const watchedIds = new Set(watched.map((item) =>
Number(item.movieId)));
        const excludedIds = new Set([...watchedIds]);
        const personalPool = recommended.filter(
          (movie) => !excludedIds.has(Number(movie.id))
        );
        const topRatedPool = topRated.filter(
          (movie) => !excludedIds.has(Number(movie.id))
        );
        const mixedRecommendations = [];
        const usedIds = new Set();
        const maxItems = 48;
        const personalTarget = mergedGenreIds.length > 0 ?
Math.ceil(maxItems * 0.7) : 0;
        const pushUnique = (pool, limit) => {
          for (const movie of pool) {
            if (mixedRecommendations.length >= maxItems ||
limit <= 0) break;
            const movieId = Number(movie.id);
            if (usedIds.has(movieId)) continue;
            mixedRecommendations.push(movie);
            usedIds.add(movieId);
            limit -= 1;
          }
          return limit;
        };
        let personalLeft = pushUnique(personalPool,
personalTarget);
        let globalLeft = pushUnique(topRatedPool, maxItems
- mixedRecommendations.length);
        // If one pool is short, fill the rest from the
other.
        if (personalLeft > 0 && mixedRecommendations.length
< maxItems) {
          pushUnique(topRatedPool, maxItems -
mixedRecommendations.length);
        }
        if (globalLeft > 0 && mixedRecommendations.length <
maxItems) {
          pushUnique(personalPool, maxItems -
mixedRecommendations.length);
        }
```

```
        const              filteredRecommended              =
mixedRecommendations.slice(0, maxItems);
        source =
          mergedGenreIds.length > 0
            ? "70% на основі ваших оцінок і обраного + 30%
топ від спільноти"
            : "Найкращі фільми, які любить спільнота";
        try {
          localStorage.setItem(
            cacheKey,
            JSON.stringify({
              createdAt: Date.now(),
              source,
              movies: filteredRecommended,
            })
          );
        } catch {
          // ignore storage errors
        }
        if (!cancelled) {
          setPopularMovies(popular);
          setRecommendedMovies(filteredRecommended);
          setRecommendationSource(`${source}  · оновлюються
раз на 24 години`);
        }
```

A.3 Файл MovieDetails.jsx

Цей шматок коду відповідає за збереження відгуків та їх видалення:

```
const handleSaveReview = async (event) => {
    event.preventDefault();
    if (!isAuthenticated) {
      navigate("/login", { state: { from: location } });
      return;
    }
    setReviewSaving(true);
    setReviewError(null);
    setReviewSuccess("");
    try {
      const savedReview = await saveMyReview(id, {
        isWatched: !!review?.isWatched,
        rating:   review?.rating   ===   ""   ?   null   :
Number(review.rating),
        comment: review?.comment ?? "",
      });
      setReview({
        isWatched: savedReview?.isWatched ?? false,
        rating: savedReview?.rating ?? "",
        comment: savedReview?.comment ?? "",
      });
```

```
      const communityData = await
getMovieCommunityData(id);
      setCommunity({
        averageRating: communityData?.averageRating ??
null,
        ratingsCount: communityData?.ratingsCount ?? 0,
        commentsCount: communityData?.commentsCount ?? 0,
        comments: communityData?.comments ?? [],
      });
      setReviewSuccess("Ваш відгук збережено.");
    } catch (err) {
      setReviewError(err?.message ?? "Не вдалося зберегти
відгук.");
    } finally {
      setReviewSaving(false);
    }
  };
  const handleDeleteComment = async (reviewId) => {
    setDeleteCommentError("");
    setDeletingCommentId(String(reviewId));
    try {
      await deleteCommentByReviewId(reviewId);
      const communityData = await
getMovieCommunityData(id);
      setCommunity({
        averageRating: communityData?.averageRating ??
null,
        ratingsCount: communityData?.ratingsCount ?? 0,
        commentsCount: communityData?.commentsCount ?? 0,
        comments: communityData?.comments ?? [],
      });
    } catch (err) {
      setDeleteCommentError(err?.message ?? "Не вдалося
видалити коментар.");
    } finally {
      setDeletingCommentId("");
    }
  };
```

A.4 Файл backend/Controllers/UserController.js

Цей шматок коду містить авторизацію та реєстрацію:

```
export const register = async (req, res) => {
  try {
  const errors = validationResult(req);
  if (!errors.isEmpty()) {
    return res.status(400).json({ errors: errors.array()
});
  }
  const salt = await bcrypt.genSalt(10);
  const hash = await bcrypt.hash(req.body.password, salt);
```

```
  const doc = new User({
    name: req.body.name,
    email: req.body.email,
    passwordHash: hash,
    favorites: [],
    watchlist: [],
  });
  const user = await doc.save();

  const token = jwt.sign({
    _id: user._id,
  },  'secret123',
  {
    expiresIn: '30d',
  });
  res.json({ ...mapUser(user), token });
} catch (err) {
  console.error('Error during registration:', err);
  res.status(500).json({ message: 'Server error' });
} }
export const login = async (req, res) => {
  try {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array()
});
    }
    const user = await User.findOne({ email: req.body.email
});
    if (!user) {
      return  res.status(404).json({  message:  'User  not
found' });
    }
    const           isValidPass           =           await
bcrypt.compare(req.body.password, user._doc.passwordHash);
    if (!isValidPass) {
      return res.status(400).json({ message: 'Invalid login
or password' });
    }
    const token = jwt.sign({
      _id: user._id,
    }, 'secret123', {
      expiresIn: '30d',
    });
    res.json({ ...mapUser(user), token });
  } catch (err) {
    console.error('Error during login:', err);
    res.status(500).json({ message: 'Server error' });
  } }
```

A.5 Файл backend/Controllers/UserController.js

Цей шматок коду містить робу зі списками обране та вотчліст:

```
export const toggleFavorite = async (req, res) => {
  try {
    const movie = req.body;
    const movieId = Number(movie?.id);
    if (!movieId) {
      return res.status(400).json({ message: 'Movie id is
required' });
    }
    const user = await User.findById(req.userId);
    if (!user) {
      return res.status(404).json({ message: 'User not
found' });
    }
    const currentFavorites = Array.isArray(user.favorites)
? user.favorites : [];
    const exists = currentFavorites.some((item) =>
Number(item?.id) === movieId);
    const favorites = exists
      ? currentFavorites.filter((item) => Number(item?.id)
!== movieId)
      : [...currentFavorites, movie];
    user.favorites = favorites;
    await user.save();
    return res.json({ favorites, isFavorite: !exists });
  } catch (err) {
    console.error('Error toggling favorite:', err);
    return res.status(500).json({ message: 'Server error'
});
  }
};
export const getWatchlist = async (req, res) => {
  try {
    const user = await User.findById(req.userId);
    if (!user) {
      return res.status(404).json({ message: 'User not
found' });
    }
    const watchlist = Array.isArray(user.watchlist) ?
user.watchlist : [];
    return res.json({ watchlist });
  } catch (err) {
    console.error('Error fetching watchlist:', err);
    return res.status(500).json({ message: 'Server error'
});
  }
};
export const toggleWatchlist = async (req, res) => {
  try {
```

```
    const movie = req.body;
    const movieId = Number(movie?.id);
    if (!movieId) {
      return res.status(400).json({ message: 'Movie id is
required' });
    }
```

A.6 Файл backend/Controllers/ReviewController.js

Цей файл відповідає за роботу з відгуками:

```
export const upsertMyReviewForMovie = async (req, res) => {
  try {
    const movieId = normalizeMovieId(req.params.movieId);
    if (!movieId) {
      return res.status(400).json({ message: "Некоректний
id фільму." });
    }
    const { isWatched, rating, comment } = req.body ?? {};
    const update = {};
    if (typeof isWatched === "boolean") {
      update.isWatched = isWatched;
    }
    if (rating === null || rating === undefined || rating
=== "") {
      update.rating = undefined;
    } else {
      const normalizedRating = Number(rating);
      if (!Number.isFinite(normalizedRating) ||
normalizedRating < 1 || normalizedRating > 5) {
        return res.status(400).json({ message: "Оцінка має
бути від 1 до 5." });
      }
      update.rating = normalizedRating;
    }
    if (comment === null || comment === undefined) {
      update.comment = "";
    } else if (typeof comment === "string") {
      const trimmed = comment.trim();
      if (trimmed.length > 1000) {
        return res.status(400).json({ message: "Коментар не
може бути довшим за 1000 символів." });
      }
      update.comment = trimmed;
    } else {
      return res.status(400).json({ message: "Некоректний
формат коментаря." });
    }
    const review = await Review.findOneAndUpdate(
      { user: req.userId, movieId },
```

```
      { $set: update, $setOnInsert: { user: req.userId,
movieId } },
      { new: true, upsert: true, runValidators: true }
    ).lean();
    if (review?.isWatched === true) {
      await User.findByIdAndUpdate(req.userId, {
        $pull: {
          watchlist: { id: movieId },
        },
      });
    }
    return res.json({ review });
  } catch (err) {
    console.error("Error saving review:", err);
    return res.status(500).json({ message: "Server error"
});
  }
};
export const getMovieCommunityData = async (req, res) => {
  try {
    const movieId = normalizeMovieId(req.params.movieId);
    if (!movieId) {
      return res.status(400).json({ message: "Некоректний
id фільму." });
    }
    const reviews = await Review.find({ movieId })
      .populate("user", "name email")
      .sort({ updatedAt: -1 })
      .lean();
    const ratings = reviews
      .map((item) => item.rating)
      .filter((value) => value !== null && value !==
undefined);
    const averageRating =
      ratings.length > 0
        ? ratings.reduce((sum, value) => sum +
Number(value), 0) / ratings.length
        : null;
    const comments = reviews
      .filter((item) => typeof item.comment === "string" &&
item.comment.trim().length > 0)
      .map((item) => ({
        _id: item._id,
        comment: item.comment,
        rating: item.rating ?? null,
        updatedAt: item.updatedAt,
        user: {
          name: item.user?.name ?? "Користувач",
          email: item.user?.email ?? "",
        },
      }));
    return res.json({
      averageRating,
```

```
      ratingsCount: ratings.length,
      commentsCount: comments.length,
      comments,
    });
  } catch (err) {
    console.error("Error fetching movie community data:",
err);
    return res.status(500).json({ message: "Server error"
});
  }
};
```

A.7 Файл backend/models/User.js

Цей файл містить модель користувача:

```
import mongoose from "mongoose";
const UserSchema = new mongoose.Schema(
  {
    name: { type: String, required: true },
    email: { type: String, required: true, unique: true },
    passwordHash: { type: String, required: true },
    favorites: { type: Array, default: [] },
    watchlist: { type: Array, default: [] },
    role: { type: String, default: 'user' },
  },
  {
    timestamps: true
  }
);
export default mongoose.model('User', UserSchema);
```

A.8 Файл backend/models/Review.js

Цей файл містить модель відгуку:

```
import mongoose from "mongoose";
const ReviewSchema = new mongoose.Schema(
  {
    user: {
      type: mongoose.Schema.Types.ObjectId,
      ref: "User",
      required: true,
    },
    movieId: {
      type: Number,
      required: true,
    },
    isWatched: {
      type: Boolean,
      default: false,
```

```
    },
    rating: {
      type: Number,
      min: 1,
      max: 5,
    },
    comment: {
      type: String,
      maxlength: 1000,
      default: "",
    },
  },
  {
    timestamps: true,
  }
);
ReviewSchema.index({ user: 1, movieId: 1 }, { unique: true
});
export default mongoose.model('Review', ReviewSchema);
```

A.9 Файл SearchBar.jsx

Цей файл реалізує пошук фільмів за назвою та фільтрацію за жанром через URL-параметри:

```
import { useEffect, useState } from "react";
import { useNavigate, useSearchParams, useLocation } from
"react-router-dom";
import { getMovieGenres } from "../api/movies";
export default function SearchBar({ className = "" }) {
  const navigate = useNavigate();
  const location = useLocation();
  const [searchParams] = useSearchParams();
  const qFromUrl = searchParams.get("q") ?? "";
  const genreFromUrl = searchParams.get("genre") ?? "";
  const [value, setValue] = useState(qFromUrl);
  const [selectedGenre, setSelectedGenre] =
useState(genreFromUrl);
  const [genres, setGenres] = useState([]);
  useEffect(() => {
    if (location.pathname === "/") {
      setValue(qFromUrl);
      setSelectedGenre(genreFromUrl);
    }
  }, [qFromUrl, genreFromUrl, location.pathname]);
  useEffect(() => {
    let cancelled = false;
    const loadGenres = async () => {
      try {
        const data = await getMovieGenres();
        if (!cancelled) setGenres(data);
```

```
      } catch {
        if (!cancelled) setGenres([]);
      }
    };
    loadGenres();
    return () => {
      cancelled = true;
    };
  }, []);
  const handleSubmit = (e) => {
    e.preventDefault();
    const q = value.trim();
    const params = new URLSearchParams();
    if (q) params.set("q", q);
    if (selectedGenre) params.set("genre", selectedGenre);
    const queryString = params.toString();
    if (queryString) {
      navigate(`/?${queryString}`);
    } else {
      navigate("/");
    }
  };
  return (
    <form
      onSubmit={handleSubmit}
      className={`flex w-full items-center gap-2
${className}`}
      role="search"
    >
      <div className="relative min-w-0 flex-1">
        <label htmlFor="nav-search" className="sr-only">
          Пошук фільмів
        </label>
        <input
          id="nav-search"
          type="search"
          autoComplete="off"
          placeholder="Пошук за назвою..."
          value={value}
          onChange={(e) => setValue(e.target.value)}
          className="w-full rounded-full border border-
white/10 bg-zinc-800/80 py-2.5 pl-12 pr-4 text-sm text-
zinc-100 placeholder:text-zinc-500 shadow-inner backdrop-
blur-md transition focus:border-amber-400/40 focus:outline-
none focus:ring-2 focus:ring-amber-400/20"
        />
      </div>
      <label htmlFor="genre-filter" className="sr-only">
        Обрати жанр
      </label>
      <select
        id="genre-filter"
        value={selectedGenre}
```

```
        onChange={(e) => setSelectedGenre(e.target.value)}
        className="max-w-[140px] rounded-full border
border-white/10 bg-zinc-800/80 px-3 py-2.5 text-sm text-
zinc-100 shadow-inner backdrop-blur-md transition
focus:border-amber-400/40 focus:outline-none focus:ring-2
focus:ring-amber-400/20 sm:max-w-[200px]"
      >
        <option value="">Усі жанри</option>
        {genres.map((genre) => (
          <option key={genre.id} value={genre.id}>
            {genre.name}
          </option>
        ))}
      </select>
      <button
        type="submit"
        className="shrink-0 rounded-full bg-amber-400 px-4
py-2.5 text-sm font-semibold text-zinc-950 transition
hover:bg-amber-500 focus:outline-none focus:ring-2
focus:ring-amber-400/50"
      >
        Знайти
      </button>
    </form>
  );
}
```

A.10 Файл MovieCard.jsx

Цей файл містить картку фільму та обробку дій “в обране” і “у вотчліст”:

```
 import { Link, useLocation, useNavigate } from "react-
router-dom";
import { getImageUrl } from "../api/movies";
import { useAuth } from "../context/AuthContext";
import FavoriteHeartIcon from "./FavoriteHeartIcon";
import WatchlistBookmarkIcon from
"./WatchlistBookmarkIcon";
function formatRating(value) {
  if (value == null) return "—";
  return Number(value).toFixed(1);
}
export default function MovieCard({ movie }) {
  const {
    isAuthenticated,
    isFavorite,
    toggleFavorite,
    isInWatchlist,
    toggleWatchlist,
  } = useAuth();
```

```
  const navigate = useNavigate();
  const location = useLocation();
  const poster = getImageUrl(movie.poster_path, "w342");
  const year = movie.release_date
    ? new Date(movie.release_date).getFullYear()
    : null;
  const favorite = isFavorite(movie.id);
  const inWatchlist = isInWatchlist(movie.id);
  const handleFavoriteClick = async (e) => {
    e.preventDefault();
    e.stopPropagation();
    if (!isAuthenticated) {
      navigate("/login", { state: { from: location } });
      return;
    }
    try {
      await toggleFavorite(movie);
    } catch {
      // Keep UI stable if request fails.
    }
  };
  const handleWatchlistClick = async (e) => {
    e.preventDefault();
    e.stopPropagation();
    if (!isAuthenticated) {
      navigate("/login", { state: { from: location } });
      return;
    }
    try {
      await toggleWatchlist(movie);
    } catch {
      // Keep UI stable if request fails.
    }
  };
  return (
    <div  className="group  relative  block  overflow-hidden
rounded-xl  bg-zinc-900  shadow-lg  shadow-black/40  ring-1
ring-white/5  transition  duration-300  hover:-translate-y-1
hover:shadow-2xl  hover:shadow-black/50  hover:ring-amber-
400/20">
      <button
        type="button"
        onClick={handleFavoriteClick}
        className={`absolute left-2 top-2 z-20 inline-flex
h-9  w-9  items-center  justify-center  rounded-full  border
backdrop-blur-sm transition ${
          favorite
            ? "border-rose-400/50 bg-rose-500/25 text-rose-
300"
            : "border-white/20 bg-zinc-950/70 text-zinc-200
hover:border-rose-400/50 hover:text-rose-300"
        }`}
```

```
        aria-label={favorite ? "Прибрати з обраного" :
"Додати в обране"}
        title={favorite ? "Прибрати з обраного" : "Додати в
обране"}
      >
        <FavoriteHeartIcon filled={favorite} className="h-4
w-4" />
      </button>
      <button
        type="button"
        onClick={handleWatchlistClick}
        className={`absolute right-2 top-2 z-20 inline-flex
h-9 w-9 items-center justify-center rounded-full border
backdrop-blur-sm transition ${
          inWatchlist
            ? "border-amber-400/50 bg-amber-500/20 text-
amber-300"
            : "border-white/20 bg-zinc-950/70 text-zinc-200
hover:border-amber-400/50 hover:text-amber-300"
        }`}
        aria-label={
          inWatchlist ? "Прибрати з вотчліста" : "Додати у
вотчліст"
        }
        title={inWatchlist ? "Прибрати з вотчліста" :
"Додати у вотчліст"}
      >
        <WatchlistBookmarkIcon         filled={inWatchlist}
className="h-4 w-4" />
      </button>
      <Link to={`/movie/${movie.id}`} className="block">
        <div  className="relative  aspect-[2/3]  overflow-
hidden bg-zinc-800">
          {poster ? (
            <img
              src={poster}
              alt=""
              className="h-full     w-full     object-cover
transition duration-500 group-hover:scale-105"
            />
          ) : (
            <div   className="flex   h-full   items-center
justify-center bg-gradient-to-br from-zinc-800 to-zinc-900
p-4 text-center text-sm text-zinc-500">
              Немає постера
            </div>
          )}
          <div  className="absolute  right-2  top-12  flex
items-center gap-1 rounded-full bg-zinc-950/90 px-2 py-0.5
text-xs font-semibold text-amber-400">
            <span aria-hidden>★</span>
            {formatRating(movie.vote_average)}
          </div>
```

```
        </div>
        <div className="p-3">
          <h3 className="line-clamp-2 font-display text-sm
font-semibold leading-snug text-zinc-100 group-hover:text-
amber-400">
            {movie.title}
          </h3>
          {year && <p className="mt-1 text-xs text-zinc-
500">{year}</p>}
        </div>
      </Link>
    </div>
  );
}
```

A.11 Файл Profile.jsx

Цей файл реалізує профіль користувача, статистику переглядів і керування ролями адміністраторів:

```
import { useEffect, useMemo, useState } from "react";
import { Link, Navigate, useLocation } from "react-router-
dom";
import    {    getImageUrl,    getMovieSummary    }    from
"../api/movies";
import { useAuth } from "../context/AuthContext";
export default function Profile() {
  const location = useLocation();
  const {
    user,
    isAuthenticated,
    authReady,
    getWatchedReviews,
    saveMyReview,
    getAdminUsers,
    grantAdminRoleByEmail,
    revokeAdminRoleByEmail,
  } = useAuth();
  const [items, setItems] = useState([]);
  const [loading, setLoading] = useState(true);
  const [adminEmail, setAdminEmail] = useState("");
  const [admins, setAdmins] = useState([]);
  const   [adminActionLoading,   setAdminActionLoading]   =
useState(false);
  useEffect(() => {
    let cancelled = false;
    const load = async () => {
      setLoading(true);
      try {
        const reviews = await getWatchedReviews();
        const merged = await Promise.all(
```

```
          reviews.map(async (review) => {
            try {
              const          movie          =          await
getMovieSummary(review.movieId);
              return { review, movie };
            } catch {
              return { review, movie: null };
            }
          })
        );
        if (!cancelled) setItems(merged);
      } finally {
        if (!cancelled) setLoading(false);
      }
    };
    if (isAuthenticated) load();
    return () => {
      cancelled = true;
    };
  }, [isAuthenticated, getWatchedReviews]);
  const stats = useMemo(() => {
    const watchedCount = items.length;
    const     withRating     =     items.filter((item)     =>
item.review?.rating != null);
    const withComment = items.filter(
      (item) => typeof item.review?.comment === "string" &&
item.review.comment.trim().length > 0
    );
    const average =
      withRating.length > 0
        ? (
            withRating.reduce((sum,    item)    =>    sum    +
Number(item.review.rating), 0) /
            withRating.length
          ).toFixed(1)
        : "—";
    return {
      watchedCount,
      withRating: withRating.length,
      withComment: withComment.length,
      average,
    };
  }, [items]);
  if      (!authReady)      return      <main><p>Перевіряємо
сесію...</p></main>;
  if   (!isAuthenticated)   return   <Navigate   to="/login"
state={{ from: location }} replace />;
  const refreshAdmins = async () => {
    const list = await getAdminUsers();
    setAdmins(list);
  };
  const handleGrantAdmin = async () => {
    setAdminActionLoading(true);
```

```
    try {
      await grantAdminRoleByEmail(adminEmail);
      await refreshAdmins();
      setAdminEmail("");
    } finally {
      setAdminActionLoading(false);
    }
  };
  const handleRevokeAdmin = async () => {
    setAdminActionLoading(true);
    try {
      await revokeAdminRoleByEmail(adminEmail);
      await refreshAdmins();
      setAdminEmail("");
    } finally {
      setAdminActionLoading(false);
    }
  };
  return (
    <main className="mx-auto max-w-7xl px-4 py-10 sm:px-6
lg:px-8">
      <h1 className="font-display text-2xl font-bold text-
white sm:text-3xl">Профіль</h1>
      <p   className="mt-1   text-sm   text-zinc-500">Ваші
переглянуті фільми, оцінки та коментарі</p>
      <div  className="grid  gap-4  sm:grid-cols-2  lg:grid-
cols-4">
        <div  className="rounded-xl  border  border-white/10
bg-zinc-900/50 p-4">
          <p          className="text-xs          text-zinc-
500">Переглянуто</p>
          <p  className="mt-1  text-2xl  font-bold  text-
white">{stats.watchedCount}</p>
        </div>
        <div  className="rounded-xl  border  border-white/10
bg-zinc-900/50 p-4">
          <p className="text-xs text-zinc-500">Оцінено</p>
          <p  className="mt-1  text-2xl  font-bold  text-
white">{stats.withRating}</p>
        </div>
        <div  className="rounded-xl  border  border-white/10
bg-zinc-900/50 p-4">
          <p          className="text-xs          text-zinc-
500">Коментарів</p>
          <p  className="mt-1  text-2xl  font-bold  text-
white">{stats.withComment}</p>
        </div>
        <div  className="rounded-xl  border  border-white/10
bg-zinc-900/50 p-4">
          <p className="text-xs text-zinc-500">Середня ваша
оцінка</p>
          <p className="mt-1 text-2xl font-bold text-amber-
400">
```

```
            {stats.average    ===    "—"    ?    "—"    :
`${stats.average}/5`}
          </p>
        </div>
      </div>
    </main>
  );
}
```

A.12 Файл backend/index.js

Цей файл містить ініціалізацію сервера та маршрути API:

```
import express from 'express';
import mongoose from 'mongoose';
import cors from 'cors';
import  {  registerValidation,  loginValidation  }  from
'./validations/auth.js';
import checkAuth from './utils/CheckAuth.js';
import {
  register,
  login,
  getMe,
  getFavorites,
  toggleFavorite,
  getWatchlist,
  toggleWatchlist,
  getAdminUsers,
  grantAdminRoleByEmail,
  revokeAdminRoleByEmail,
} from './Controllers/UserController.js';
import {
  getMyReviewForMovie,
  upsertMyReviewForMovie,
  getWatchedReviews,
  getMovieCommunityData,
  deleteCommentByReviewId,
} from "./Controllers/ReviewController.js";
mongoose.connect('mongodb+srv://vipvladkovalenko908:HWD6VJr
a0jV9Ip3t@cluster0.id4qdbz.mongodb.net/movie-
app?retryWrites=true&w=majority&appName=Cluster0'
  ).then(() => {
    console.log('Connected to MongoDB');
  }).catch((err) => {
    console.error('Error connecting to MongoDB:', err);
  });
const app = express();
app.use(cors());
app.use(express.json());
app.post('/auth/register', registerValidation, register);
app.post('/auth/login', loginValidation, login);
app.get('/auth/me', checkAuth, getMe);
```

```
app.get('/favorites', checkAuth, getFavorites);
app.post('/favorites/toggle', checkAuth, toggleFavorite);
app.get('/watchlist', checkAuth, getWatchlist);
app.post('/watchlist/toggle', checkAuth, toggleWatchlist);
app.get("/admin/users", checkAuth, getAdminUsers);
app.post("/admin/grant", checkAuth, grantAdminRoleByEmail);
app.post("/admin/revoke",                          checkAuth,
revokeAdminRoleByEmail);
app.get("/reviews/watched", checkAuth, getWatchedReviews);
app.get("/reviews/:movieId",                       checkAuth,
getMyReviewForMovie);
app.put("/reviews/:movieId",                       checkAuth,
upsertMyReviewForMovie);
app.get("/reviews/movie/:movieId/public",
getMovieCommunityData);
app.delete("/reviews/:reviewId/comment",           checkAuth,
deleteCommentByReviewId);
app.listen(3000, (err) => {
  if (err) {
    return console.log('Error starting server:', err);
  } else {
    console.log('Server is running on port 3000');
  }
});
```

A.13 Файл backend/utils/CheckAuth.js

У цьому файлі реалізовано middleware перевірки JWT-токена:

```
import jwt from 'jsonwebtoken';
export default (req, res, next) => {
    const    token    =    (req.headers.authorization    ||
'').replace(/Bearer\s?/, '');
    if (token)
    {
        try {
            const decoded = jwt.verify(token, 'secret123');
            req.userId = decoded._id;
            next();
        } catch (err) {
            return res.status(403).json({ message: 'Invalid
token' });
        }
    }
    else {
        return  res.status(403).json({  message:  'No  token
provided' });
    }
}
```

ДОДАТОК Б

Слайди презентації

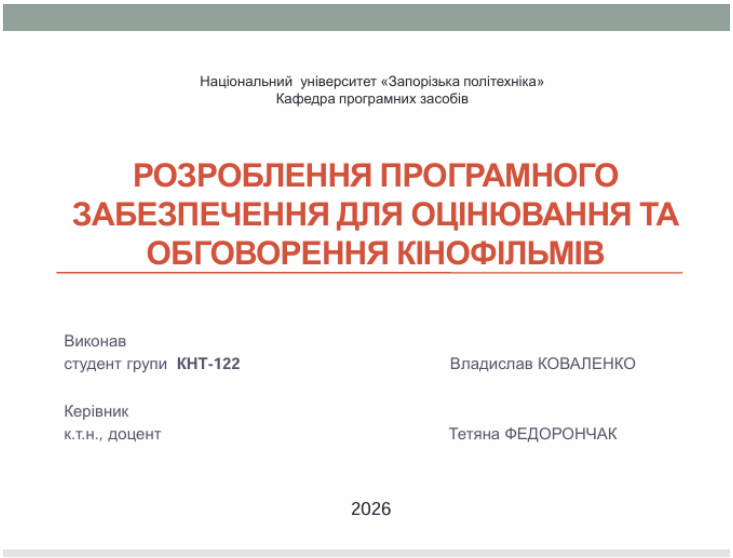


Рисунок Б.1 – Слайд №1

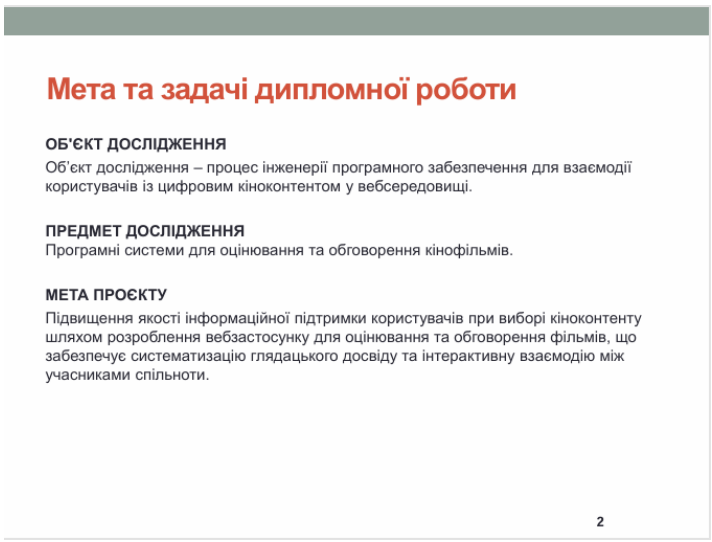


Рисунок Б.2 – Слайд №2

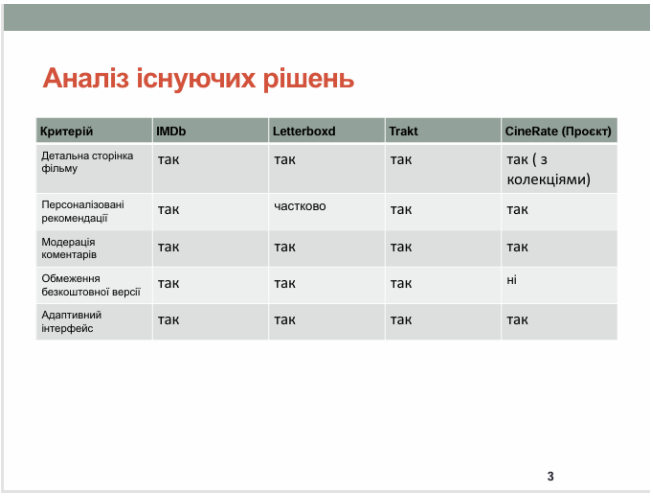


Рисунок Б.3 – Слайд №3

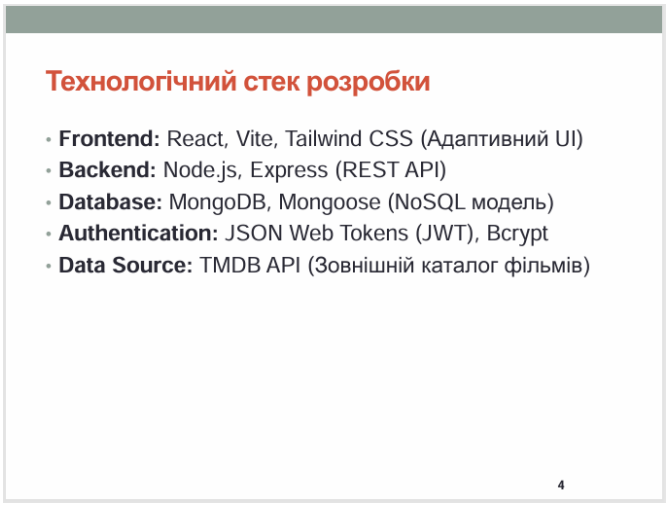
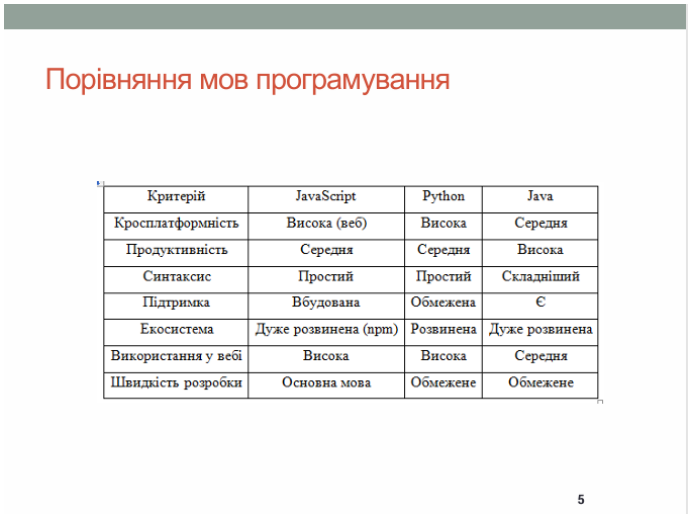


Рисунок Б.4 – Слайд №4



Критерій	JavaScript	Python	Java
Кросплатформність	Висока (веб)	Висока	Середня
Продуктивність	Середня	Середня	Висока
Синтаксис	Простий	Простий	Складніший
Підтримка	Вбудована	Обмежена	Є
Екосистема	Дуже розвинена (npm)	Розвинена	Дуже розвинена
Використання у вебі	Висока	Висока	Середня
Швидкість розробки	Основна мова	Обмежене	Обмежене

Рисунок Б.5 – Слайд №5

Критерій	MongoDB	PostgreSQL
Тип бази даних	NoSQL (документоорієнтована)	Реляційна
Структура даних	Гнучка (JSON/BSON)	Строга (таблиці)
Продуктивність	Висока для CRUD-операцій	Висока для складних запитів
Масштабованість	Висока (горизонтальна)	Висока (вертикальна + горизонтальна)
Робота зі зв'язками	Обмежена	Повноцінна (JOIN)
Простота розробки	Висока	Середня
Інтеграція з Node.js	Дуже зручна	Зручна
Контроль цілісності	Низький	Високий

Рисунок Б.6 – Слайд №6

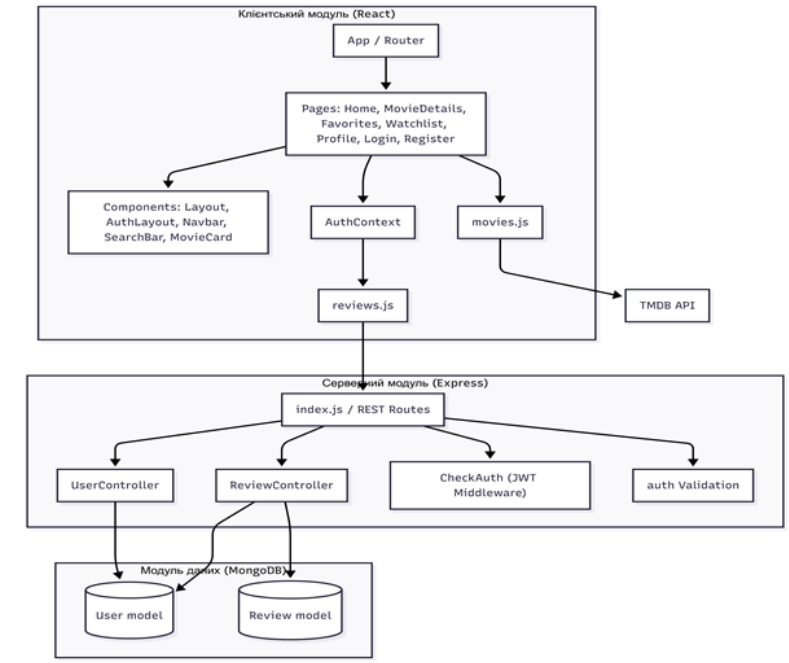
Порівняння середовищ розробки

Критерій	Visual Studio Code	WebStorm	Visual Studio
Підтримка JavaScript	Повна	Повна	Часткова
Швидкість роботи	Висока	Середня	Низька
Споживання ресурсів	Низьке	Середнє	Високе
Плагіни	Дуже багато	Багато	Обмежено
Інтеграція з Git	Вбудована	Вбудована	Вбудована
Вартість	Безкоштовний	Платний	Частково платний
Кросплатформність	Так	Так	Обмежена

7

Рисунок Б.7 – Слайд №7

Архітектура застосунку



8

Рисунок Б.8 – Слайд №8

Рисунок Б.9 – Слайд №9

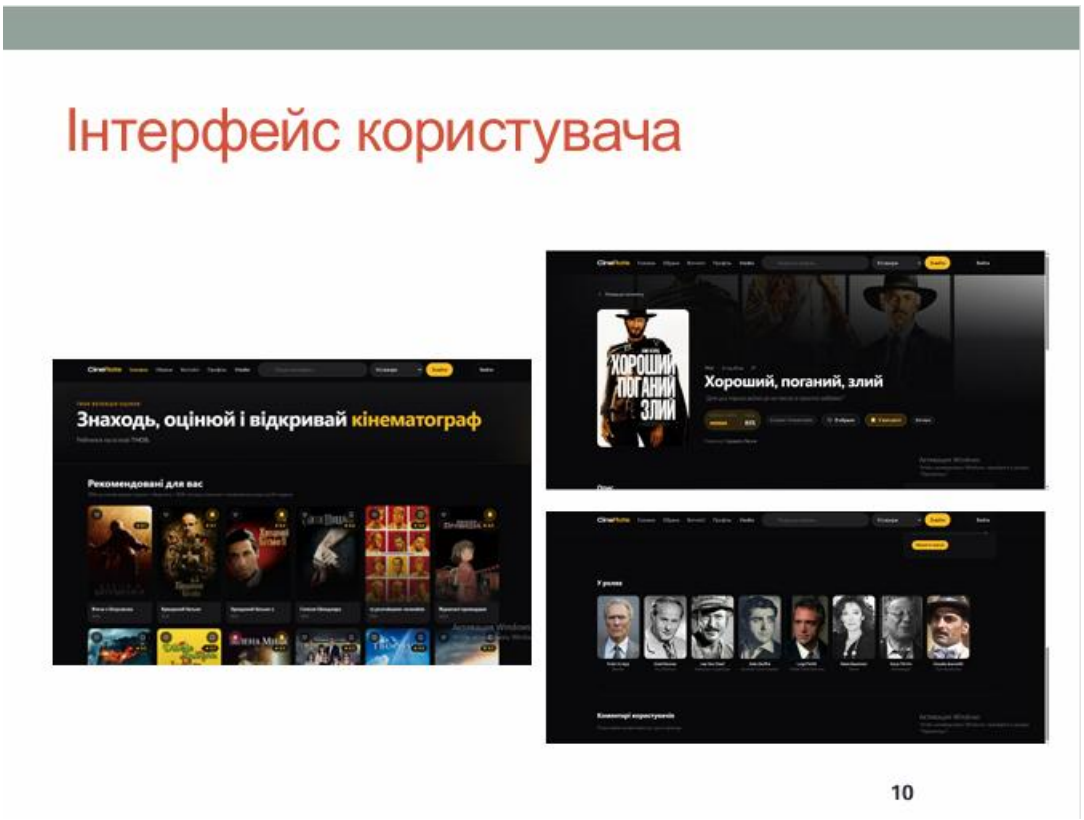


Рисунок Б.10 – Слайд №10

Логіка персоналізованих рекомендацій

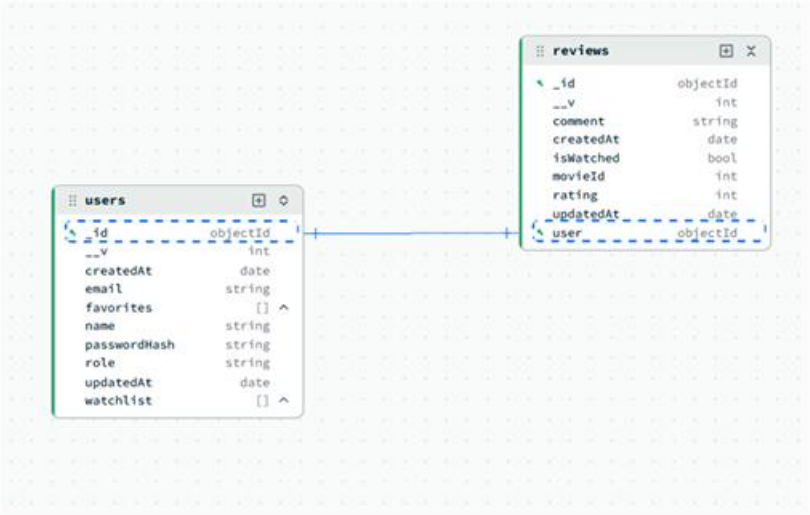
АЛГОРИТМ ФОРМУВАННЯ РЕКОМЕНДАЦІЙ:

1. Аналіз жанрів із списку **"Обране"**.
2. Врахування фільмів з високими оцінками (4-5 зірок).
3. Виключення вже **переглянутих** фільмів із видачі.
4. Домішування популярних трендів спільноти CineRate.
5. Кешування результатів у **localStorage** на 24 години.

11

Рисунок Б.11 – Слайд №11

Структура бази даних (MONGODB)



12

Рисунок Б.12 – Слайд №12

Висновки

1. Проведено аналіз сучасних вебзастосунків для пошуку та оцінювання фільмів.
2. Спроектовано архітектуру та базу даних.
3. Реалізовано вебзастосунок CineRate з рекомендаціями, коментарями, оцінками та адаптивним інтерфейсом.
4. Інтегровано JWT-автентифікацію та TMDB API.
5. Проведено тестування основних функціональних модулів.
6. Мету роботи досягнуто: створено вебзастосунок для оцінювання та обговорення кінофільмів.

13

Рисунок Б.13 – Слайд №13