ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня вищої освіти бакалавр

на тему: «**Розробка інтерактивного вебсервісу для пошуку, сортування та рекомендацій фільмів з використанням React**»

Виконав: здобувач вищої освіти
за освітньою програмою
Інформаційні управляючі системи
спеціальності 126 Інформаційні
системи та технології
ступеня вищої освіти бакалавр
групи 126ІСТ_бд_31[1]
Бойко Євгеній Віталійович
Керівник: Поночовний Юрій
Леонідович
Рецензент: Стрюк Олексій Юрійович

Полтава – 2026 року

ПОЛТАВСЬКИЙ ДЕРЖАВНИЙ АГРАРНИЙ УНІВЕРСИТЕТ
Навчально-науковий інститут економіки, управління, права та інформаційних технологій
Кафедра інформаційних систем та технологій

Освітня програма Інформаційні управляючі системи
Спеціальність 126 Інформаційні системи та технології
Рівень вищої освіти перший (бакалаврський)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Юрій УТКІН
«10» червня 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ ЗДОБУВАЧА ВИЩОЇ ОСВІТИ

Бойко Євгенію Віталійовичу

1. Тема роботи:
«Розробка інтерактивного вебсервісу для пошуку, сортування та рекомендацій фільмів з використанням React»,
керівник роботи д.т.н., професор, професор кафедри інформаційних систем та технологій Поночовний Юрій Леонідович.
Затверджено наказом закладу вищої освіти від «10» квітня 2026 року № 384 ст

2. Строк подання здобувачем вищої освіти роботи «08» червня 2026 року

3. Вихідні дані до роботи: стандарти проєктування та розробки програмного забезпечення, дані офіційних сайтів https://react.dev/ (бібліотека React), https://redux.js.org/ (бібліотека Redux), https://developer.themoviedb.org/docs/getting-started (The Movie Database API), https://github.com/remix-run/react-router/tree/v3/docs (React Router v3), https://styled-components.com/docs (styled-components), https://docs.github.com/en/pages (GitHub Pages), середовище розробки Visual Studio Code, платформа Node.js.

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити):

Розділ 1. Теоретичні основи розробки інтерактивних вебсервісів для роботи з медіаконтентом
Розділ 2. Проєктування інтерактивного вебсервісу для роботи з фільмами
Розділ 3. Практична реалізація та оцінка ефективності вебсервісу

5. Перелік графічного матеріалу: схеми, рисунки, графіки, діаграми за темою та об’єктом дослідження.

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання отримав
Оцінювання економічної ефективності результатів дослідження	Загребельна І. Л., к. е. н., доцент, доцент кафедри економіки та публічного управління	01.04.2026 р.	29.05.2026 р.

7. Дата видачі завдання «10» червня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Вибір і затвердження теми роботи	02.06.2025 р. – 04.06.2025	
2	Складання і затвердження розгорнутого плану та завдання на кваліфікаційну роботу	05.06.2025 р. – 10.06.2025 р.	
3	Опрацювання джерел інформації	11.06.2025 р. – 15.06.2025 р.	
4	Збір, вивчення і обробка інформації, необхідної для виконання роботи	16.06.2025 р. – 20.06.2025 р.	
5	Виконання теоретико-методологічного розділу роботи	08.09.2025 р. – 01.11.2025 р.	
6	Виконання дослідницько-аналітичного розділу роботи	19.01.2026 р. – 14.02.2026 р.	
7	Виконання проєктно-рекомендаційного розділу роботи	16.02.2026 р. – 31.03.2026 р.	
8	Розрахунок економічної ефективності результатів дослідження	01.04.2026 р. – 29.05.2026 р.	
9	Оформлення тексту роботи	01.06.2026 р. – 06.06.2026р.	
10	Попередній захист роботи на кафедрі	08.06.2026 р.	
11	Доопрацювання роботи з урахуванням зауважень і пропозицій	09.06.2026 р. – 10.06.2026 р.	
12	Нормоконтроль	11.06.2026 р. – 15.06.2026 р.	
13	Захист кваліфікаційної роботи	16.06.2026 р. – 18.06.2026 р.	

Здобувач вищої освіти **Євгеній БОЙКО**

Керівник роботи **Юрій ПОНОЧОВНИЙ**

ЗМІСТ

ВСТУП

Актуальність теми кваліфікаційної роботи полягає у зростанні популярності онлайн-платформ для перегляду та пошуку кіноконтенту, що зумовлює необхідність створення сучасних веб-сервісів із зручним та інтуїтивно зрозумілим інтерфейсом. Щодня користувачі стикаються з великою кількістю фільмів різних жанрів, років випуску та рейтингів, що ускладнює процес вибору контенту для перегляду. У зв'язку з цим особливої важливості набувають веб-сервіси, які дозволяють швидко знаходити необхідні фільми, виконувати їх сортування за різними критеріями та отримувати рекомендації відповідно до вподобань користувача. Розвиток сучасних вебтехнологій, зокрема бібліотеки React, відкриває широкі можливості для створення швидких, адаптивних і функціональних вебдодатків. Використання компонентного підходу, висока продуктивність та зручність підтримки програмного коду роблять React одним із найпоширеніших інструментів фронтенд-розробки. Тому дослідження можливостей створення інтерактивного кіно-сервісу на основі React є актуальним та практично значущим завданням.

Метою кваліфікаційної роботи є узагальнення теоретичних відомостей та практична реалізація інтерактивного веб-сервісу для пошуку, сортування та рекомендацій фільмів із використанням бібліотеки React.

Об'єктом дослідження є процес організації та забезпечення взаємодії користувача з інформаційними веб-системами для пошуку та аналізу кіноконтенту.

Предметом дослідження є методи, засоби та технології розробки інтерактивних веб-додатків для роботи з інформацією про фільми, а також особливості застосування бібліотеки React для створення сучасних користувацьких інтерфейсів.

Методи дослідження включають аналіз наукової та технічної літератури, порівняльний аналіз існуючих кіно-сервісів, методи проєктування

інформаційних систем, сучасні підходи до веброзробки, програмну реалізацію функціональних модулів та тестування готового програмного продукту.

Інформаційну базу становлять наукові праці з веброзробки та проєктування інформаційних систем, документація React, матеріали спеціалізованих вебресурсів, результати аналізу сучасних кіно-сервісів та власні дослідження автора.

Практична значущість роботи полягає у створенні інтерактивного веб-сервісу, який забезпечує ефективний пошук фільмів, їх сортування за визначеними параметрами та формування рекомендацій для користувачів. Розроблений веб-додаток може бути використаний як навчальний проєкт, основа для подальшого вдосконалення або як готове рішення для інформаційних ресурсів кінематографічної тематики.

Структура та обсяг кваліфікаційної роботи. Пояснювальна записка складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі розглянуто теоретичні аспекти та сучасні технології розробки веб-додатків. У другому розділі наведено аналіз предметної області, проєктування структури системи та обґрунтування вибору програмних засобів. Третій розділ присвячений реалізації інтерактивного кіно-сервісу засобами React, опису його функціональних можливостей та тестуванню. У висновках наведено результати проведеного дослідження та оцінено ефективність розробленого програмного продукту. Загальний обсяг текстової частини кваліфікаційної роботи складає 47 сторінок формату А4. Вона містить 17 рисунків і 12 таблиць.

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ РОЗРОБКИ ІНТЕРАКТИВНИХ ВЕБСЕРВІСІВ ДЛЯ РОБОТИ З МЕДІАКОНТЕНТОМ

1.1 Аналіз предметної області та огляд існуючих сервісів пошуку і рекомендацій кінофільмів

Стрімкий розвиток цифрових технологій та зростання обсягів онлайн-контенту зумовили появу нового класу вебсервісів – платформ для пошуку, каталогізації та рекомендацій кінофільмів. Сучасний користувач щодня стикається з тисячами одиниць кіноконтенту різних жанрів, мов і років випуску, що ускладнює процес вибору та орієнтації в інформаційному просторі [1]. Саме тому кіносервіси, які забезпечують ефективний пошук і персоналізовані рекомендації, набули особливої практичної цінності.

Предметна область даної роботи охоплює процеси взаємодії користувача з інформаційними системами кінематографічної тематики: пошук фільмів за різними параметрами, їх сортування, фільтрація та отримання рекомендацій на основі вподобань. Об'єктом аналізу є як функціональна складова подібних систем, так і технологічні підходи до їх реалізації [2].

Серед існуючих платформ виділяють кілька категорій. Перша – великі комерційні агрегатори, такі як IMDb та Rotten Tomatoes, які акумулюють мільйони записів про фільми, серіали, акторів і надають широкі можливості пошуку. Друга категорія – стримінгові сервіси (Netflix, Amazon Prime), у яких пошук і рекомендації є допоміжними інструментами для просування власного контенту [3]. Третя – відкриті бази даних, зокрема The Movie Database (TMDb), що надають публічний API для побудови власних застосунків.

Ключовою проблемою більшості існуючих рішень є недостатній рівень персоналізації та перевантаженість інтерфейсу. Дослідження в галузі UX підтверджують, що користувачі залишають сторінку впродовж перших 10-15 секунд, якщо не знаходять потрібну інформацію швидко [5]. Це висуває

жорсткі вимоги до інтерфейсу кіносервісу: мінімалістичний дизайн, швидкий відгук системи та інтуїтивна навігація.

Порівняльний аналіз наявних платформ за ключовими характеристиками наведено у таблиці 1.1.

Таблиця 1.1 – Порівняльний аналіз існуючих кіносервісів

Платформа	Відкритий API	Рекомендації	Пошук за жанром	Безкоштовний доступ
IMDb	Обмежений	Так	Так	Частково
TMDb	Так	Так	Так	Так
Rotten Tomatoes	Ні	Ні	Обмежено	Так
Netflix	Ні	Так	Так	Ні

Як видно з таблиці 1.1, TMDb є єдиною платформою, яка поєднує повністю відкритий API, підтримку рекомендацій і безкоштовний доступ, що робить її оптимальним вибором для розробки власного вебсервісу [4].

З функціонального погляду сучасний кіносервіс має забезпечувати щонайменше три ключові можливості. Перша – повнотекстовий пошук з підтримкою фільтрації за жанром, роком випуску та рейтингом. Друга – багатокритеріальне сортування результатів. Третя – рекомендація фільмів на основі переглянутого або обраного контенту [6]. Ефективність рекомендаційної системи значною мірою визначається метрикою точності, яка розраховується як:

$$P = \frac{|R \cap U|}{|R|} \tag{1.1}$$

де R – множина рекомендованих фільмів,

U – множина фільмів, що справді зацікавили користувача.

Розробка власного вебсервісу на відміну від використання готових рішень дозволяє гнучко налаштовувати функціонал, адаптувати інтерфейс під конкретну аудиторію та інтегрувати необхідні зовнішні джерела даних [7]. Загальну концепцію архітектури подібного сервісу ілюструє рисунок 1.1.

Рисунок 1.1 – Концептуальна архітектура вебсервісу для пошуку фільмів

Як показано на рисунку 1.1, сервіс функціонує за трирівневою моделлю: клієнтський інтерфейс взаємодіє з логікою обробки даних, яка, своєю чергою, звертається до зовнішніх джерел. Такий підхід забезпечує незалежність рівнів та спрощує подальше масштабування [8].

Таким чином, аналіз предметної області підтверджує актуальність розробки інтерактивного вебсервісу для пошуку та рекомендацій фільмів. Існуючі рішення або є надто складними в адаптації, або мають обмежений API-доступ. Розробка власного застосунку з відкритим стеком технологій на базі TMDb API дозволяє усунути ці недоліки та забезпечити користувачам зручний і функціональний інструмент.

1.2 Сучасні технології фронтенд-розробки та обґрунтування вибору React як основного інструменту

Фронтенд-розробка є ключовою складовою створення сучасних вебзастосунків, оскільки саме клієнтська частина безпосередньо визначає якість взаємодії користувача з системою. Технологічна база фронтенду пройшла

значний шлях від статичних HTML-сторінок до складних односторінкових застосунків (SPA – Single Page Application), здатних забезпечувати досвід, порівнянний із нативними десктопними програмами [9].

Базовими технологіями фронтенду залишаються HTML, CSS та JavaScript. HTML визначає структуру контенту, CSS відповідає за його візуальне оформлення, а JavaScript забезпечує інтерактивність та динамічну поведінку інтерфейсу. Сучасний стандарт ES2022+ суттєво розширив можливості мови, додавши підтримку асинхронного програмування, модульної системи та нових синтаксичних конструкцій [10]. Надбудовою над JavaScript є TypeScript – мова зі статичною типізацією, яка підвищує надійність коду у великих проєктах.

На базі JavaScript сформувалася потужна екосистема UI-фреймворків і бібліотек. Три найпоширеніші рішення – React, Angular і Vue – суттєво відрізняються за архітектурою, філософією та сферою застосування [11]. Порівняльну характеристику цих технологій наведено у таблиці 1.2.

Таблиця 1.2 – Порівняльна характеристика React, Angular і Vue

Критерій	React	Angular	Vue
Тип	Бібліотека	Повний фреймворк	Прогресивний фреймворк
Мова	JavaScript / JSX	TypeScript	JavaScript / TypeScript
Крива навчання	Середня	Висока	Низька
Продуктивність	Висока	Висока	Висока
Розмір bundle	Малий	Великий	Малий
Екосистема	Дуже велика	Велика	Середня
Підтримка	Meta (Facebook)	Google	Спільнота

Як свідчать дані таблиці 1.2, React вирізняється оптимальним балансом між гнучкістю, продуктивністю та розміром екосистеми. Angular є потужним, проте надлишковим для проєктів середнього масштабу, а Vue, попри простоту, має меншу спільноту та ринкову присутність [12].

React – це бібліотека JavaScript з відкритим кодом, розроблена компанією Meta у 2013 році. Її ключова концепція – компонентний підхід, за якого інтерфейс поділяється на незалежні, повторно використовувані блоки. Кожен

компонент інкапсулює власну логіку, стан і розмітку, що значно спрощує підтримку та масштабування застосунку [13].

Однією з головних технічних переваг React є використання віртуального DOM (Virtual DOM). Замість безпосереднього оновлення реального дерева документа при кожній зміні даних, React спочатку виконує порівняння (diffing) нової та попередньої версій віртуального DOM, після чого вносить лише мінімально необхідні зміни до реального DOM. Це забезпечує суттєве підвищення швидкості рендерингу, що особливо важливо при роботі з великими списками даних, як-от каталог фільмів [14].

Продуктивність рендерингу React-компонента можна оцінити через час реакції інтерфейсу T_r, який визначається як:

$$T_r = T_{diff} + T_{patch}, \tag{1.2}$$

де T_{diff} – час порівняння віртуальних дерев DOM,

T_{patch} – час застосування змін до реального DOM.

Мінімізація обох складових є метою оптимізації будь-якого React-застосунку. Загальну схему роботи механізму Virtual DOM у React наведено на рисунку 1.2.

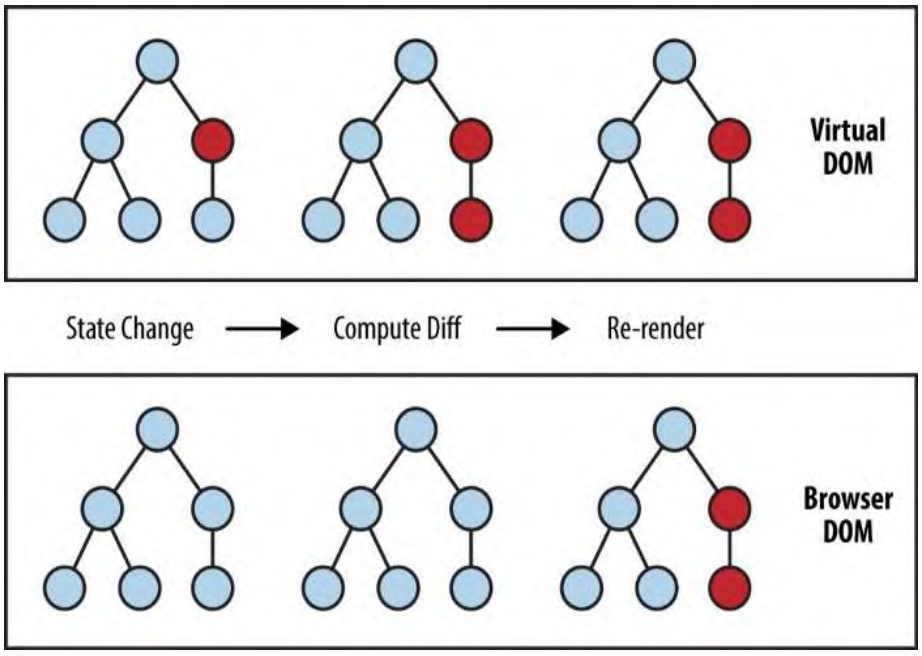

Рисунок 1.2 – Принцип роботи Virtual DOM у React

Як ілюструє рисунок 1.2, механізм diffing дозволяє ізолювати зміни та уникнути повного перерендерингу сторінки, що є критичним для застосунків із динамічними списками та фільтрами [14].

Крім архітектурних переваг, React має розвинену екосистему допоміжних бібліотек. Для маршрутизації між сторінками використовується React Router, для управління глобальним станом – Redux або Zustand, для стилізації – CSS-модулі, Tailwind CSS або React-Bootstrap. Hooks API, введений у версії 16.8, дозволяє використовувати стан та інші можливості React у функціональних компонентах без написання класів, що спростило код і підвищило його читабельність [15].

Згідно зі щорічним опитуванням розробників Stack Overflow Developer Survey 2024, React залишається найбільш уживаною фронтенд-бібліотекою вже кілька років поспіль – її використовують понад 39% опитаних веброзробників [16]. Це підтверджує зрілість технології та наявність великої кількості готових рішень, що прискорює розробку.

Для реалізації інтерактивного кіносервісу React є обґрунтованим вибором: компонентна архітектура природно відповідає структурі каталогу фільмів (картки, списки, фільтри, модальні вікна), Virtual DOM забезпечує швидке оновлення результатів пошуку без перезавантаження сторінки, а широка екосистема скорочує час розробки. Поєднання React із бібліотекою Redux для управління глобальним станом утворює технологічну основу, достатню для реалізації всіх запланованих функціональних модулів сервісу.

1.3 Принципи управління станом вебдодатків та архітектура Redux

У міру зростання складності сучасних вебзастосунків одним із ключових архітектурних завдань стає ефективне управління станом (State Management). Стан застосунку – це сукупність усіх даних, які визначають поточне відображення інтерфейсу: результати пошуку, активні фільтри, обрані елементи, дані користувача тощо. У невеликих проєктах достатньо вбудованих механізмів

React – хуків useState та useContext. Однак зі збільшенням кількості компонентів і глибини їх вкладеності передача даних через props перетворюється на так зване «prop drilling» – громіздку і важко підтримувану практику [17].

Для вирішення цієї проблеми у 2015 році компанія Meta представила патерн Flux, а невдовзі на його основі Ден Абрамов розробив бібліотеку Redux, яка стала де-факто стандартом управління глобальним станом у React-застосунках [18]. Redux реалізує принцип єдиного джерела істини (Single Source of Truth): весь стан застосунку зберігається в одному централізованому об'єкті – Store. Будь-яка зміна стану відбувається виключно через чітко визначений механізм: компонент надсилає Action (об'єкт із полем type), який обробляється чистою функцією – Reducer, що повертає новий стан без мутації попереднього [19]. Архітектуру потоку даних у Redux наведено на рисунку 1.3.

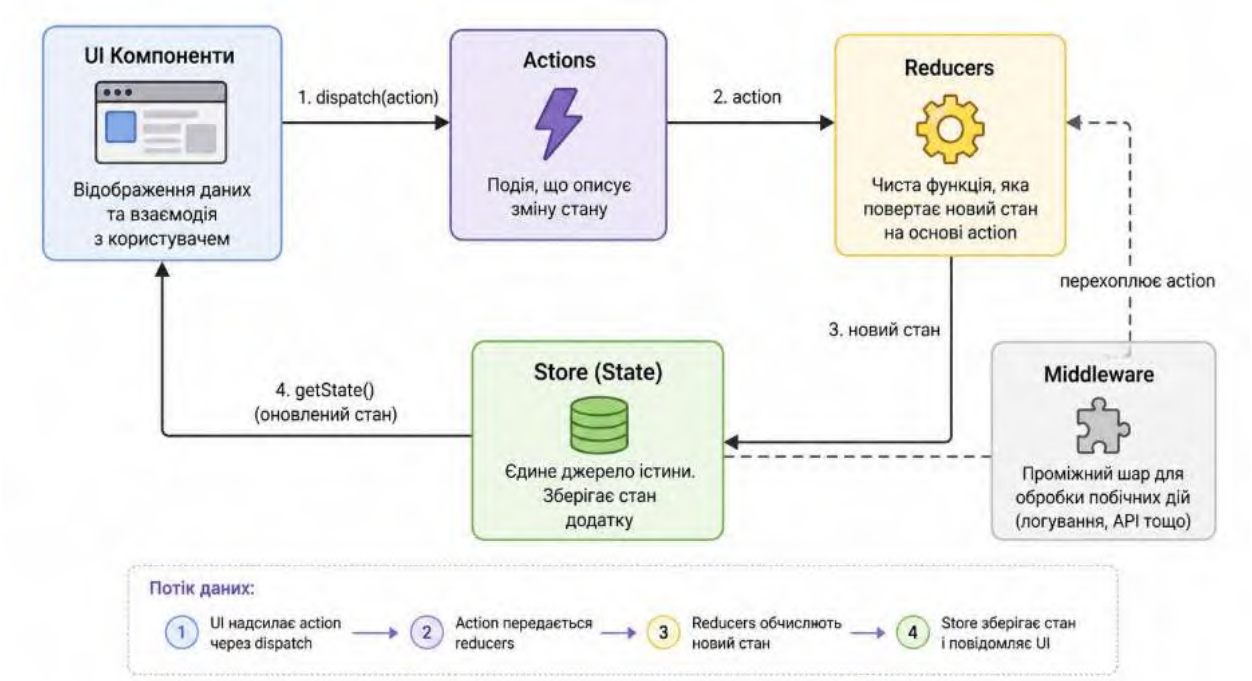

Рисунок 1.3 – Архітектура управління станом за допомогою Redux

Як показано на рисунку 1.3, потік даних у Redux є суворо односпрямованим, що забезпечує передбачуваність поведінки застосунку та суттєво спрощує налагодження. Кожна зміна стану є відтворюваною та може бути відстежена за допомогою Redux DevTools [17].

Основні принципи Redux формулюються через три аксіоми. По-перше, єдине сховище – весь стан застосунку міститься в одному Store. По-друге, стан є незмінним – безпосередня мутація стану заборонена, натомість Reducer повертає новий об'єкт. По-третє, зміни описуються чистими функціями – Reducer отримує поточний стан і Action, повертаючи наступний стан [18].

Сучасна версія бібліотеки – Redux Toolkit (RTK) – суттєво спрощує написання Redux-логіки, усуваючи надлишковий шаблонний код (boilerplate). RTK надає утиліти createSlice, createAsyncThunk та configureStore, які автоматизують типові операції. Зокрема, createAsyncThunk дозволяє елегантно обробляти асинхронні запити до API з автоматичною генерацією станів pending, fulfilled та rejected [19].

Для кіносервісу Redux є обґрунтованим вибором, оскільки застосунок оперує декількома взаємозалежними зрізами стану одночасно: список фільмів, параметри пошуку, активні фільтри, деталі обраного фільму та рекомендації. Зберігання цих даних у єдиному Store дозволяє будь-якому компоненту отримати доступ до потрібних даних без передачі props через проміжні рівні [20]. Порівняння підходів до управління станом у React-застосунках наведено у таблиці 1.3.

Таблиця 1.3 – Порівняння підходів до управління станом у React

Підхід	Масштабованість	Складність налаштування	Відладка	Підходить для кіносервісу
useState / useContext	Низька	Мінімальна	Складна	Частково
Redux (RTK)	Висока	Середня	Відмінна	Так
Zustand	Середня	Низька	Добра	Так

З таблиці 1.3 видно, що Redux Toolkit забезпечує найкращий баланс між масштабованістю та якістю інструментів налагодження, що є критичним для застосунку з динамічними списками та асинхронними запитами до TMDb API.

Важливою характеристикою Redux є передбачуваність переходів між станами. Якщо позначити стан застосунку як S, Action як A, а Reducer як чисту функцію f, то перехід стану описується як:

$$S_{n+1} = f(S_n, A) \tag{1.3}$$

Ця властивість, відображена у формулі (1.3), гарантує, що при однакових вхідних даних Reducer завжди повертає ідентичний результат, що є основою детермінованості та тестованості застосунку [18].

Таким чином, використання Redux Toolkit у поєднанні з React створює надійну архітектурну основу для кіносервісу: централізований стан спрощує синхронізацію між компонентами пошуку, фільтрації та рекомендацій, а інструменти налагодження прискорюють виявлення та усунення помилок на етапі розробки.

1.4 Алгоритмічні підходи до реалізації пошуку, сортування та рекомендацій контенту

Ефективна обробка великих масивів даних є однією з ключових вимог до сучасних інформаційних вебсервісів. Для кіносервісу це набуває особливого значення, оскільки бази даних фільмів налічують сотні тисяч записів, а користувач очікує миттєвої реакції системи на будь-який запит. Три функціональні підсистеми – пошук, сортування та рекомендації – утворюють ядро такого сервісу і визначають якість користувацького досвіду в цілому [21].

Пошук у контексті кіносервісу може здійснюватися за різними атрибутами: назвою фільму, жанром, роком випуску, країною виробництва, іменем актора чи режисера. Найпростішим підходом є лінійний пошук, який передбачає послідовне порівняння кожного запису з пошуковим запитом. Його часова складність становить O(n), де n – кількість елементів у колекції, що є прийнятним лише для невеликих наборів даних [22]. У промислових системах

застосовуються методи індексування та повнотекстового пошуку. Однак у вебзастосунках на базі зовнішнього API, зокрема TMDb, пошук делегується серверній стороні: клієнт надсилає HTTP-запит із параметрами, а сервер повертає відфільтрований результат, що знімає навантаження з браузера та забезпечує швидкий відгук [4].

Важливим елементом пошукового інтерфейсу є реалізація відкладеного пошуку (debouncing) – техніки, за якої запит до API надсилається не після кожного введеного символу, а лише після певної паузи в друкуванні. Це дозволяє суттєво скоротити кількість мережевих запитів і знизити навантаження на API [21].

Сортування є невід'ємною функцією будь-якого каталогу. Користувач кіносервісу може впорядковувати результати за рейтингом, датою виходу, популярністю або назвою. У JavaScript для сортування масивів використовується вбудований метод Array.prototype.sort(), який у сучасних рушіях реалізований на основі алгоритму TimSort – гібриду сортування злиттям та сортування вставками із середньою часовою складністю $O(n \log n)$ [22]. Багатокритеріальне сортування реалізується через компаратор, який послідовно порівнює кілька полів об'єкта. Порівняльну характеристику основних алгоритмів сортування наведено у таблиці 1.4.

Таблиця 1.4 – Порівняння алгоритмів сортування за часовою складністю

Алгоритм	Найкращий випадок	Середній випадок	Найгірший випадок	Стабільний
Бульбашкове	O(n)	O(n²)	O(n²)	Так
Швидке	O(n log n)	O(n log n)	O(n²)	Ні
Злиттям	O(n log n)	O(n log n)	O(n log n)	Так
TimSort	O(n)	O(n log n)	O(n log n)	Так

Як видно з таблиці 1.4, TimSort є оптимальним вибором для клієнтського сортування завдяки стабільності та ефективності на частково впорядкованих даних, що є типовим для відповідей API.

Рекомендаційні системи є найбільш складною з трьох підсистем із точки зору алгоритмічної реалізації. Існує три основні підходи до формування рекомендацій [6]. Контентна фільтрація (Content-Based Filtering) базується на

аналізі характеристик контенту: якщо користувач переглянув фільм певного жанру з високим рейтингом, система рекомендує схожі за атрибутами позиції. Колаборативна фільтрація (Collaborative Filtering) використовує поведінку великої кількості користувачів: фільми, які подобаються користувачам зі схожими вподобаннями, рекомендуються цільовому користувачу. Гібридні системи поєднують обидва підходи та демонструють найвищу точність, однак потребують значних обчислювальних ресурсів і великих обсягів накопичених даних [2].

Для кіносервісу, що розробляється, найбільш доцільною є контентна модель рекомендацій на основі атрибутів фільму. Міра подібності між двома фільмами обчислюється за допомогою косинусної подібності між векторами їхніх ознак:

$$\cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{\left|\vec{A}\right| \cdot \left|\vec{B}\right|}, \tag{1.4}$$

де $\vec{A}$ та $\vec{B}$ – вектори ознак двох фільмів (жанр, рейтинг, популярність),

$\left|\vec{A}\right|$ та $\left|\vec{B}\right|$ – їх евклідові норми.

Чим ближче значення формули (1.4) до 1, тим більш схожими є фільми [6].

Загальну схему взаємодії підсистем пошуку, сортування та рекомендацій у межах кіносервісу наведено на рисунку 1.4.

Рисунок 1.4 – Схема взаємодії підсистем пошуку, сортування та рекомендацій

Як ілюструє рисунок 1.4, усі три підсистеми працюють узгоджено: результати пошуку через API передаються на клієнт, де сортуються та доповнюються рекомендаціями на основі обраного фільму. Такий підхід дозволяє мінімізувати кількість звернень до зовнішнього API та забезпечити швидкий відгук інтерфейсу [21].

Таким чином, грамотний вибір алгоритмів для кожної з трьох підсистем є запорукою якісної роботи кіносервісу. Делегування пошуку на сторону TMDb API з використанням debouncing, клієнтське сортування на базі TimSort та контентна рекомендаційна модель із косинусною подібністю утворюють збалансовану архітектуру, що поєднує продуктивність і простоту реалізації.

РОЗДІЛ 2
ПРОЄКТУВАННЯ ІНТЕРАКТИВНОГО ВЕБСЕРВІСУ ДЛЯ РОБОТИ З ФІЛЬМАМИ

2.1 Формування функціональних та нефункціональних вимог до системи

Розробка будь-якого програмного продукту розпочинається з чіткого визначення вимог – формального опису того, що система має виконувати та в яких умовах. Вимоги традиційно поділяють на функціональні, що описують конкретну поведінку системи, та нефункціональні, що визначають якісні характеристики її роботи [23]. Правильне формування вимог на початковому етапі проєктування дозволяє уникнути переробок на пізніших стадіях і забезпечує відповідність готового продукту очікуванням користувачів.

Розроблюваний вебсервіс «MovieBox» орієнтований на широке коло користувачів, які прагнуть зручно знаходити інформацію про фільми, переглядати деталі про акторський склад і трейлери, а також отримувати суміжний контент. Визначення вимог ґрунтується на аналізі типових сценаріїв використання подібних платформ та можливостей обраного зовнішнього джерела даних – TMDb API [4].

Функціональні вимоги описують конкретні дії, які система має забезпечувати користувачу. На основі аналізу предметної області та структури реалізованого застосунку сформовано такий перелік функціональних вимог.

Перша вимога – відображення каталогу фільмів. Застосунок має завантажувати та відображати список актуальних фільмів з TMDb API одразу після відкриття головної сторінки. Кожна картка фільму повинна містити постер, назву, дату виходу та середній рейтинг.

Друга вимога – пошук фільмів за назвою. Система має забезпечувати повнотекстовий пошук із автопідказками в реальному часі: під час введення тексту у поле пошуку відображається випадний список релевантних результатів

із постером і роком виходу фільму. Підтверджений пошуковий запит формує окрему сторінку результатів.

Третя вимога – перегляд детальної інформації про фільм. При переході на сторінку фільму користувач має бачити розширені відомості: постер, опис, жанри, рейтинг, а також список акторів та вбудовані трейлери з YouTube.

Четверта вимога – перегляд профілю актора. Клацання на картці актора має відкривати сторінку з його фотографією, біографічними даними та списком фільмів за його участю.

П'ята вимога – навігація між розділами. Система має забезпечувати маршрутизацію між головною сторінкою, сторінкою результатів пошуку, сторінкою деталей фільму та профілем актора без перезавантаження браузера.

Шоста вимога – обробка помилок та станів завантаження. При очікуванні відповіді від API система має відображати індикатор завантаження, а у разі відсутності результатів – відповідне повідомлення користувачу.

Нефункціональні вимоги визначають якісні характеристики роботи системи. Вимоги до розроблюваного сервісу систематизовано у таблиці 2.1.

Таблиця 2.1 – Нефункціональні вимоги до вебсервісу MovieBox

Категорія	Вимога	Показник
Продуктивність	Час відгуку інтерфейсу	Не більше 300 мс
Продуктивність	Час завантаження сторінки	Не більше 3 с
Надійність	Обробка помилок API	Обов'язкова для всіх запитів
Сумісність	Підтримувані браузери	Chrome, Firefox, Edge (актуальні версії)
Адаптивність	Підтримка мобільних пристроїв	Коректне відображення від 320 px
Підтримуваність	Модульність коду	Компонентна архітектура React
Безпека	Передача API-ключа	Лише через HTTPS

З таблиці 2.1 видно, що нефункціональні вимоги охоплюють усі ключові аспекти якості системи. Вимога щодо часу відгуку в 300 мс відповідає психологічному порогу сприйняття миттєвої реакції, визначеному в дослідженнях UX [5]. Адаптивність забезпечується використанням сіткової

системи React-Bootstrap, яка автоматично перебудовує макет залежно від ширини екрана. Сценарії використання системи наведено на рисунку 2.1.

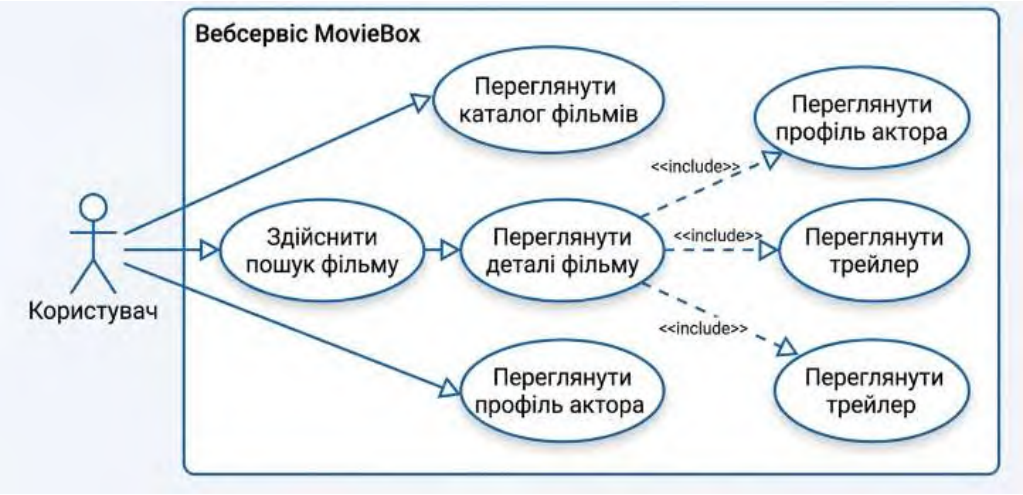

Рисунок 2.1 – Діаграма варіантів використання вебсервісу MovieBox

Як показано на рисунку 2.1, центральним сценарієм є перегляд деталей фільму, який включає перегляд трейлерів та перехід до профілю актора як залежні варіанти використання. Така структура відображає реальну навігаційну логіку застосунку, де сторінка фільму є вузловою точкою взаємодії [24].

Сформовані вимоги є основою для всіх наступних етапів проєктування – визначення архітектури, вибору технологічного стеку та проєктування компонентної структури. Дотримання функціональних вимог гарантує повноту реалізованого функціоналу, тоді як виконання нефункціональних вимог забезпечує якість і зручність кінцевого продукту для користувача [23].

2.2 Архітектурне проєктування вебсервісу та вибір технологічного стеку

Архітектурне проєктування є етапом, на якому визначається загальна структура системи, принципи взаємодії її складових та набір технологій, що застосовуватимуться при реалізації. Від правильності прийнятих на цьому етапі

рішень залежить масштабованість, підтримуваність та продуктивність готового продукту [25]. Для вебсервісу MovieBox обрано архітектуру односторінкового застосунку (SPA) із клієнтською маршрутизацією та централізованим управлінням станом.

Застосунок побудовано за трирівневою логічною моделлю. Перший рівень – рівень представлення – реалізовано засобами React і включає всі UI-компоненти, що відповідають за відображення даних користувачу. Другий рівень – рівень управління станом – реалізовано за допомогою Redux із middleware redux-thunk, який забезпечує обробку асинхронних операцій. Третій рівень – рівень даних – представлений зовнішнім TMDb API, до якого застосунок звертається через стандартний Fetch API браузера [4]. Такий поділ забезпечує незалежність шарів і спрощує тестування та модифікацію кожного з них окремо.

Загальну архітектуру вебсервісу MovieBox наведено на рисунку 2.2.

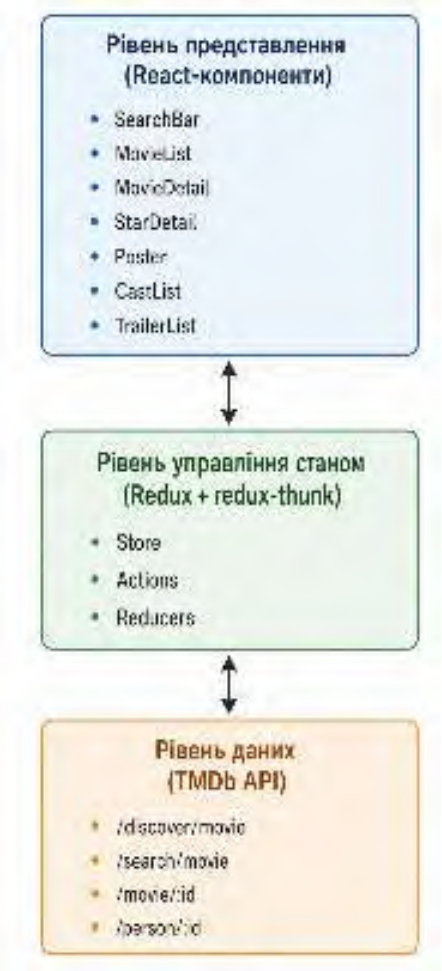

Рисунок 2.2 – Архітектура вебсервісу MovieBox

Як видно з рисунку 2.2, усі запити до TMDb API ініціюються виключно через Redux Actions, що гарантує єдиний контрольований канал отримання даних і унеможливлює прямі виклики API з UI-компонентів [17].

Вибір конкретних бібліотек та інструментів технологічного стеку здійснювався на основі відповідності функціональним вимогам, зрілості технології та якості документації. Повний технологічний стек проєкту наведено у таблиці 2.2.

Таблиця 2.2 – Технологічний стек вебсервісу MovieBox

Технологія	Версія	Призначення
React	15.4.1	Побудова UI-компонентів
Redux	3.6.0	Централізоване управління станом
redux-thunk	2.2.0	Асинхронні Action-creators
react-router	3.0.0	Клієнтська маршрутизація
react-router-redux	4.0.7	Синхронізація роутера зі Store
react-bootstrap	0.30.7	Адаптивна сіткова система та UI
react-autosuggest	9.0.0	Автопідказки у полі пошуку
styled-components	2.4.0	Компонентна стилізація
redux-devtools-extension	2.13.0	Налагодження стану застосунку
TMDb API	v3	Джерело кінематографічних даних

Як свідчать дані таблиці 2.2, стек є цілісним і кожна бібліотека виконує чітко визначену роль. React обрано як основу для побудови інтерфейсу завдяки компонентній архітектурі та механізму Virtual DOM, що забезпечує ефективне оновлення списків фільмів [12]. Redux із middleware redux-thunk забезпечує управління глобальним станом і дозволяє елегантно обробляти асинхронні запити до API: кожен запит породжує три послідовні Actions – початок завантаження, успішне отримання даних або помилку [17].

Маршрутизацію між сторінками реалізовано за допомогою react-router із використанням хеш-навігації (hashHistory), що дозволяє розгортати застосунок на статичному хостингу без необхідності налаштування серверної переадресації. Синхронізацію стану роутера із Redux Store забезпечує бібліотека react-router-redux [25]. В застосунку визначено чотири маршрути: головна сторінка з

каталогом (/), результати пошуку (/search/:keyword), сторінка деталей фільму (/movie/:id) та профіль актора (/star/:id).

Для адаптивної верстки використовується react-bootstrap, сіткова система якого базується на 12-колонковій моделі Bootstrap. Компонент MovieList застосовує значення xs={6} sm={4} md={3}, що забезпечує відображення від двох карток на мобільних пристроях до чотирьох на широких екранах [26]. Компонент react-autosuggest реалізує логіку автопідказок у пошуковому рядку: при кожній зміні введеного тексту формується запит до ендпоінту /search/movie TMDb API, а отримані результати відображаються у випадному списку безпосередньо під полем введення.

Збирання та запуск проєкту здійснюються через react-scripts – інструмент, що абстрагує конфігурацію Webpack і Babel, надаючи зручний інтерфейс команд start, build та test. Для розгортання застосунку на GitHub Pages використовується пакет gh-pages, який публікує зібрану версію застосунку у гілку gh-pages репозиторію [25].

Таким чином, обраний технологічний стек повністю відповідає сформованим функціональним та нефункціональним вимогам. Поєднання React, Redux і TMDb API утворює архітектурно обґрунтовану основу для реалізації всіх запланованих функціональних модулів вебсервісу MovieBox.

2.3 Проєктування інтерфейсу користувача та структури компонентів

Проєктування інтерфейсу користувача є одним із ключових етапів розробки вебзастосунку, оскільки саме від якості UI залежить зручність використання системи та рівень задоволеності кінцевого користувача. У React-застосунках проєктування інтерфейсу нерозривно пов'язане з визначенням компонентної структури: кожен елемент візуального інтерфейсу відповідає конкретному компоненту або їх ієрархії [13]. Для вебсервісу MovieBox визначено чіткий поділ на контейнерні компоненти, які взаємодіють зі Store, та

презентаційні компоненти, що відповідають виключно за відображення отриманих даних.

Верхнім рівнем у ієрархії компонентів є кореневий компонент App, який рендериться для будь-якого маршруту та містить навігаційну панель SearchBar. Нижче розміщуються контейнерні компоненти, що змінюються залежно від активного маршруту: MovieContainer для головної сторінки та результатів пошуку, MovieDetail для сторінки деталей фільму, StarDetail для профілю актора та DisplayMsg для відображення помилок. Кожен контейнер підключений до Redux Store через функцію connect() і самостійно ініціює необхідні запити до TMDb API при монтуванні [17]. Ієрархію компонентів вебсервісу MovieBox наведено на рисунку 2.3.

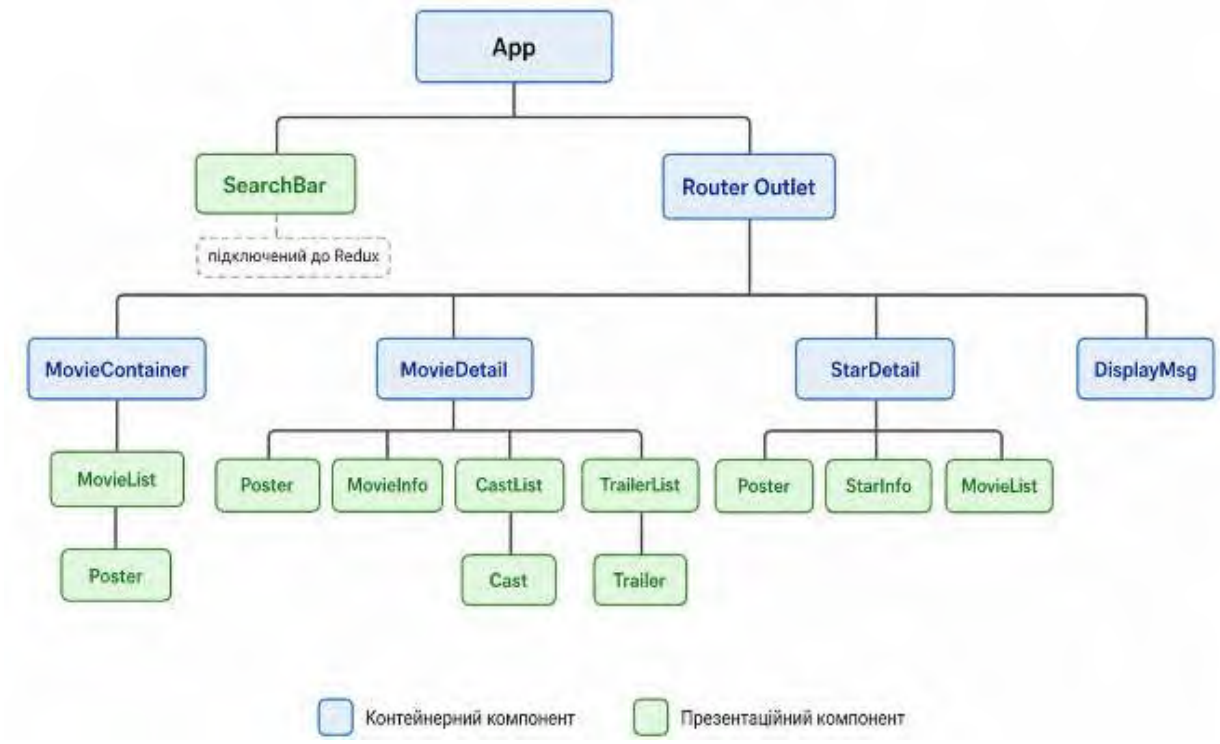

Рисунок 2.3 – Ієрархія компонентів вебсервісу MovieBox

Як видно з рисунку 2.3, презентаційні компоненти утворюють нижній рівень ієрархії та є повністю залежними від props, що передаються батьківськими контейнерами. Такий поділ відповідає принципу єдиної відповідальності та суттєво спрощує повторне використання компонентів – зокрема, Poster і

MovieList застосовуються одночасно на сторінці каталогу та в профілі актора [13]. Перелік усіх компонентів застосунку з описом їх призначення наведено у таблиці 2.3.

Таблиця 2.3 – Компоненти вебсервісу MovieBox та їх призначення

Компонент	Тип	Призначення
App	Контейнер	Кореневий компонент, обгортка маршрутизатора
SearchBar	Контейнер	Навігаційна панель з полем пошуку та автопідказками
MovieContainer	Контейнер	Завантаження та відображення каталогу або результатів пошуку
MovieDetail	Контейнер	Завантаження та відображення деталей фільму
StarDetail	Контейнер	Завантаження та відображення профілю актора
MovieList	Презентаційний	Адаптивна сітка карток фільмів
Poster	Презентаційний	Постер фільму або актора з hover-ефектом
MovieInfo	Презентаційний	Назва, рейтинг, рік, опис фільму
CastList	Презентаційний	Список карток акторів фільму
Cast	Презентаційний	Картка окремого актора з фото та ім'ям
TrailerList	Презентаційний	Список вбудованих YouTube-трейлерів
Trailer	Презентаційний	Окремий вбудований трейлер через iframe
StarInfo	Презентаційний	Ім'я, стать, дата народження та біографія актора
DisplayMsg	Презентаційний	Повідомлення про відсутність результатів або помилку

З таблиці 2.3 видно, що застосунок налічує 15 компонентів, з яких 5 є контейнерними та 10 – суто презентаційними, що свідчить про дотримання принципу поділу відповідальності [27].

Навігаційна панель SearchBar реалізована на базі компонента Navbar з React-Bootstrap зі стилем inverse (темний фон), що забезпечує контрастність і легку помітність. У лівій частині панелі розміщено логотип сервісу та логотип TMDb, у правій – поле пошуку з автопідказками на базі бібліотеки react-autosuggest. При введенні тексту виконується запит до ендпоінту /search/movie TMDb API, а результати відображаються у випадному списку з мініатюрою постера та роком виходу фільму. Вибір підказки з клавіатури або мишею одразу перенаправляє користувача на сторінку деталей відповідного фільму [21].

Компонент Poster реалізує hover-ефект засобами бібліотеки styled-components: при наведенні курсора на картку фільму у нижній частині постера з'являється накладка з назвою, рейтингом та роком виходу. Сітка карток у компоненті MovieList будується на основі адаптивної системи Bootstrap із значеннями xs={6} sm={4} md={3}, що забезпечує відображення двох колонок на мобільних пристроях, трьох – на планшетах і чотирьох – на широких екранах [26].

Сторінка деталей фільму (MovieDetail) компонується за двоколонковою схемою: ліва колонка (ширина md=4) містить постер, права (md=8) – інформаційний блок з назвою, рейтингом, кількістю голосів, роком та описом, а також список до п'яти акторів. Нижче розміщується рядок із трейлерами у вигляді вбудованих iframe-елементів з YouTube, кількість яких обмежена константою TRAILER_MAX_NUM=3. Аналогічну двоколонкову компоновку використовує сторінка профілю актора (StarDetail), де під основним блоком відображається розділ «Known For» із чотирма фільмами за його участю [27].

Таким чином, компонентна структура вебсервісу MovieBox є логічно організованою і відповідає сучасним практикам проєктування React-застосунків. Чіткий поділ на контейнерні та презентаційні компоненти, адаптивна сіткова верстка та продуманий UX пошукової панелі формують зручний і функціональний інтерфейс, що відповідає всім сформованим вимогам.

2.4 Інтеграція зовнішнього API TMDb як джерела кінематографічних даних

Вибір надійного та структурованого зовнішнього джерела даних є критичним рішенням при розробці будь-якого інформаційного вебсервісу. Для MovieBox джерелом кінематографічних даних обрано The Movie Database API (TMDb API) версії 3 – відкритий REST-сервіс, який надає доступ до бази даних, що налічує понад мільйон фільмів, серіалів та персон [4]. Перевагами TMDb є

безкоштовний рівень доступу, висока стабільність, детальна документація, підтримка зображень різних розмірів і щоденне оновлення бази даних.

Взаємодія з TMDb API реалізована через стандартний браузерний Fetch API з використанням middleware redux-thunk, що дозволяє виконувати асинхронні запити безпосередньо в Action-creators Redux. Кожен запит до API породжує послідовність трьох Actions: початок завантаження (встановлює прапор isFetching: true), успішне отримання даних або помилку. Така схема гарантує, що інтерфейс завжди відображає актуальний стан завантаження і користувач не залишається без зворотного зв'язку [17].

Автентифікація запитів здійснюється за допомогою API-ключа, який передається як query-параметр api_key у кожному HTTP-запиті. Базові URL-адреси та ключ винесено в окремий модуль констант const.js, що спрощує їх централізовану зміну та убезпечує від дублювання в коді [28]. У застосунку використовується сім різних ендпоінтів TMDb API. Їх перелік із описом та прикладами сформованих URL наведено у таблиці 2.4.

Таблиця 2.4 – Ендпоінти TMDb API, що використовуються у вебсервісі MovieBox

Ендпоінт	Метод	Призначення
/3/discover/movie	GET	Каталог популярних фільмів
/3/discover/movie?with_cast={id}	GET	Фільми за участю актора
/3/search/movie?query={keyword}	GET	Пошук фільмів за назвою
/3/movie/{id}	GET	Детальна інформація про фільм
/3/movie/{id}/casts	GET	Акторський склад фільму
/3/movie/{id}/videos	GET	Трейлери та відеоматеріали
/3/person/{id}	GET	Профіль актора або персони

Як видно з таблиці 2.4, усі запити виконуються методом GET, що відповідає принципам REST-архітектури для операцій читання даних [28]. Відповіді API повертаються у форматі JSON та обробляються за допомогою ланцюжка Promise-методів: .then(response => response.json()) для десеріалізації, після чого витягується потрібне поле (наприклад, json.results для списків або весь об'єкт для деталей).

Для роботи із зображеннями TMDb надає окремий CDN-сервер з базовою URL-адресою https://image.tmdb.org/t/p/. Розмір зображення задається як частина шляху перед іменем файлу. В застосунку визначено чотири константи розміру: w45 для мініатюр у автопідказках пошуку, w150 для малих зображень, w342 для карток каталогу та w500 для середніх зображень у профілях акторів. Така гнучкість дозволяє завантажувати зображення оптимального розміру для кожного контексту відображення і зменшує трафік [4].

Схему потоку даних від TMDb API до інтерфейсу користувача наведено на рисунку 2.4.

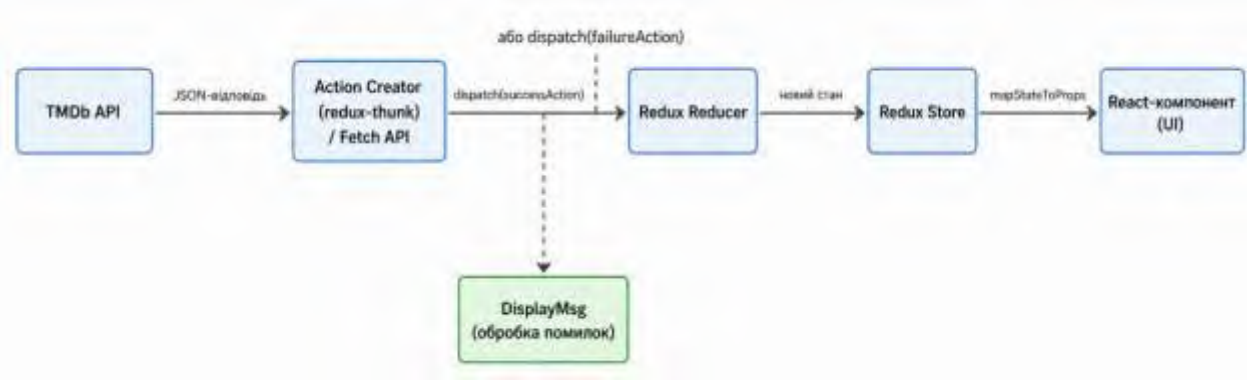

Рисунок 2.4 – Схема потоку даних від TMDb API до інтерфейсу MovieBox

Як ілюструє рисунок 2.4, отримані від TMDb дані не передаються безпосередньо до компонентів, а проходять через Redux Store, що є єдиним джерелом істини для всього застосунку. Функція mapStateToProps підписує компонент на відповідний зріз стану та автоматично ініціює повторний рендеринг при його зміні [17].

Окремої уваги заслуговує обробка відеоматеріалів. Ендпоінт /videos повертає всі відеоматеріали фільму, включаючи тизери, кліпи та інтерв'ю з різних платформ. В Action Creator fetchTrailerList реалізована фільтрація: зберігаються лише відео з полем site === 'YouTube', оскільки саме YouTube-трейлери відтворюються через стандартний iframe з базовою URL https://www.youtube.com/embed/. Кількість відображуваних трейлерів обмежена

константою TRAILER_MAX_NUM = 3, що запобігає перевантаженню сторінки детального перегляду [29].

Таким чином, інтеграція з TMDb API реалізована як повноцінний асинхронний шар застосунку, що забезпечує отримання даних для всіх функціональних модулів – каталогу, пошуку, деталей фільму, профілю актора та трейлерів. Централізована обробка запитів через Redux Actions гарантує передбачуваність потоку даних і спрощує підтримку та розширення функціоналу у майбутньому.

РОЗДІЛ 3
ПРАКТИЧНА РЕАЛІЗАЦІЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ВЕБСЕРВІСУ

3.1 Розробка ключових функціональних модулів пошуку, сортування та рекомендацій

Практична реалізація вебсервісу MovieBox передбачає розробку трьох взаємопов'язаних функціональних модулів: пошуку фільмів, впорядкування результатів та рекомендацій суміжного контенту. Кожен модуль реалізовано у відповідності до архітектурних рішень, визначених на етапі проєктування, із дотриманням принципу розподілу відповідальності між компонентами та Redux-шаром [17].

Модуль пошуку є центральним елементом користувацького інтерфейсу і реалізовано у контейнерному компоненті SearchBar. Схему роботи модуля пошуку наведено на рисунку 3.1.

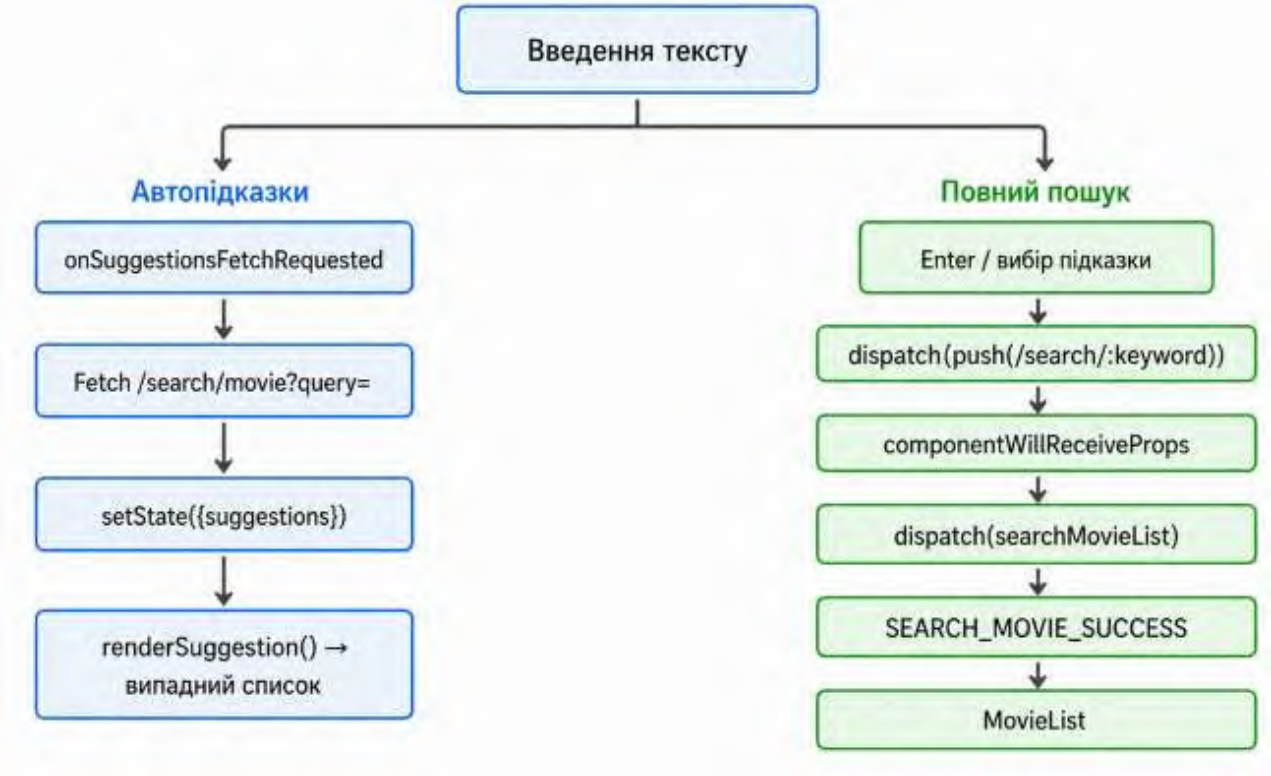

Рисунок 3.1 – Схема роботи модуля пошуку вебсервісу MovieBox

Як видно з рисунку 3.1, обидва сценарії пошуку ведуть до відображення результатів у компоненті MovieList, проте відрізняються механізмом: автопідказки використовують локальний стан компонента, тоді як повний пошук проходить через Redux Store і залишає слід у системі управління станом [17].

Пошук функціонує у двох режимах. Перший – інтерактивні автопідказки в реальному часі: при кожній зміні значення у полі введення викликається метод onSuggestionsFetchRequested, який формує запит до ендпоінту /search/movie?query={keyword} TMDb API та оновлює локальний стан компонента масивом підказок. Кожен елемент підказки містить ідентифікатор, назву, рік виходу та шлях до мініатюри постера розміром w45. Якщо постер відсутній, замість нього відображається логотип сервісу [4]. Другий режим – повний пошук: натискання Enter або вибір підказки з клавіатури ініціює навігацію на маршрут /search/:keyword, при переході на який компонент MovieContainer через метод componentWillReceiveProps диспатчить Action searchMovieList(keyword). Відповідний Action Creator надсилає запит до TMDb API та при успіху диспатчить SEARCH_MOVIE_SUCCESS, що оновлює зріз movieList у Redux Store [28].

Модуль впорядкування результатів реалізовано на рівні серверного API. Ендпоінт /discover/movie TMDb за замовчуванням повертає фільми, відсортовані за показником популярності popularity.desc, а ендпоінт /search/movie – за релевантністю запиту. Таке рішення є архітектурно виправданим для клієнтського SPA-застосунку: делегування сортування серверній стороні унеможливлює завантаження всього масиву даних на клієнт і знижує обчислювальне навантаження на браузер [29]. Результати в обох випадках надходять вже впорядкованими і безпосередньо передаються до компонента відображення.

Модуль рекомендацій реалізовано за принципом контентної фільтрації на основі акторського складу. Логіка рекомендацій вбудована в сценарій навігації по акторах: на сторінці деталей фільму користувач бачить акторський склад і може перейти до профілю будь-якого актора. При монтуванні компонента

StarDetail одночасно диспатчаться два Action Creators: fetchStarDetail(id) для отримання біографічних даних актора та fetchMovieList(id) з параметром id. Якщо Action Creator fetchMovieList отримує аргумент, URL формується як URL_LIST + '?api_key=...&with_cast=' + id, що звертається до ендпоінту /discover/movie?with_cast={actorId} і повертає фільми за участю цього актора. У компоненті StarDetail для відображення у блоці «Known For» береться лише зріз movies.slice(0, 4) – чотири найпопулярніші фільми актора [6].

Перелік Redux Action Types усіх реалізованих модулів та їх призначення наведено у таблиці 3.1.

Таблиця 3.1 – Redux Action Types функціональних модулів MovieBox

Action Type	Модуль	Призначення
SEARCH_MOVIE	Пошук	Ініціювання пошукового запиту
SEARCH_MOVIE_SUCCESS	Пошук	Успішне отримання результатів пошуку
SEARCH_MOVIE_FAILURE	Пошук	Помилка виконання пошуку
FETCH_MOVIES	Каталог / Рекомендації	Ініціювання запиту каталогу або фільмів актора
FETCH_MOVIES_SUCCESS	Каталог / Рекомендації	Успішне отримання списку фільмів
FETCH_MOVIES_FAILURE	Каталог / Рекомендації	Помилка завантаження списку
FETCH_CASTS_SUCCESS	Рекомендації	Отримання акторського складу фільму
FETCH_TRAILERS_SUCCESS	Медіа	Отримання відфільтрованих YouTube-трейлерів

З таблиці 3.1 видно, що модулі каталогу і рекомендацій розділяють спільний зріз стану movieList у Redux Store: цей зріз наповнюється як при завантаженні головної сторінки (fetchMovieList() без параметра), так і при переході на профіль актора (fetchMovieList(actorId) з параметром). Такий підхід дозволяє повторно використовувати компонент MovieList у різних контекстах без дублювання логіки відображення [13].

Таким чином, усі три функціональні модулі реалізовані в єдиному архітектурному стилі: асинхронні запити через redux-thunk, чіткий поділ між Action Types для різних станів завантаження та повторне використання компонентів представлення. Взаємодія модулів пошуку і рекомендацій формує цілісний користувацький сценарій – від знаходження фільму до відкриття суміжного контенту через акторський склад.

3.2 Реалізація адаптивного інтерфейсу та налаштування глобального стану додатку

Реалізація вебсервісу MovieBox ґрунтується на двох взаємопов'язаних аспектах: організації файлової структури проєкту та налаштуванні глобального стану Redux Store як єдиного центру управління даними. Файлова структура проєкту організована за функціональним принципом – кожна директорія групує файли за їх роллю в архітектурі [27]. Структуру директорії src/ наведено на рисунку 3.2.

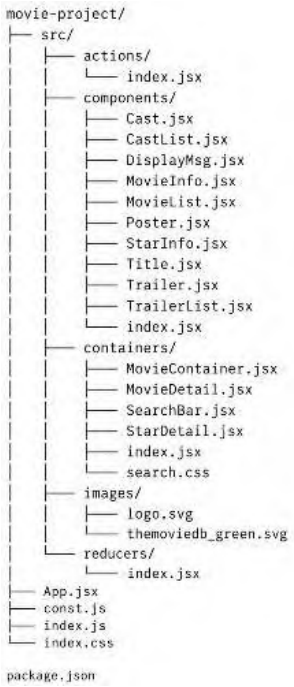

Рисунок 3.2 – Файлова структура проєкту MovieBox

Як видно з рисунку 3.2, директорія src/ містить п'ять функціональних піддиректорій. Директорія actions містить єдиний файл index.jsx з усіма Action Types і Action Creators. Директорія reducers аналогічно зосереджена в одному файлі index.jsx, що об'єднує всі редьюсери. Директорії containers і components розділяють компоненти за принципом підключення до Store. Конфігураційні дані – URL-адреси ендпоінтів, API-ключ та числові константи – зосереджені у файлі const.js [4].

Налаштування глобального стану виконується у файлі index.js. Store створюється функцією createStore з кореневим редьюсером та middleware-ланцюжком, обгорнутим у composeWithDevTools для інтеграції з Redux DevTools Extension. Ключовий фрагмент налаштування наведено на рис. 3.3.

```
const routeMiddleware = routerMiddleware(hashHistory);
let store = createStore(
 movieApp,
 composeWithDevTools(
  applyMiddleware(thunkMiddleware, routeMiddleware)
 )
);
const history = syncHistoryWithStore(hashHistory, store);

ReactDOM.render(
 <Provider store={store}>
  <Router history={history}>
   <Route path="/" component={App}>
    <IndexRoute component={MovieContainer} />
    <Route path="/movie/:id"    component={MovieDetail} />
    <Route path="/star/:id"     component={StarDetail} />
    <Route path="/search/:keyword" component={MovieContainer} />
    <Route path="*"            component={DisplayMsg} />
   </Route>
  </Router>
 </Provider>,
 document.getElementById('root')
);
```

Рисунок 3.3 – Налаштування Redux Store та маршрутизатора у файлі index.js

Як видно з рис. 3.3, thunkMiddleware забезпечує підтримку асинхронних Action Creators, а routeMiddleware синхронізує навігацію зі Store. Компонент Provider передає Store через контекст React усьому дереву компонентів, усуваючи потребу в ручній передачі Store через props [17]. Маршрутизація оголошена декларативно: компонент App є постійною оболонкою для всіх

маршрутів, тоді як вкладені маршрути підставляють відповідний контейнер через this.props.children. Підстановчий маршрут * перехоплює будь-який некоректний URL і відображає DisplayMsg [25]. Кореневий редьюсер movieApp формується функцією combineReducers, що об'єднує сім незалежних зрізів стану. Структуру Redux Store наведено у таблиці 3.2.

Таблиця 3.2 – Структура Redux Store вебсервісу MovieBox

Зріз (slice)	Поля	Призначення
movieList	isFetching, items[], error	Список фільмів для каталогу, пошуку та рекомендацій
movieDetail	isFetching, item{}, error	Детальна інформація про поточний фільм
castList	isFetching, items[], error	Акторський склад поточного фільму
trailerList	isFetching, items[], error	YouTube-трейлери поточного фільму
starDetail	isFetching, item{}, error	Профіль актора або персони
input	рядок	Поточний стан пошукового запиту
routing	об'єкт	Синхронізація react-router-redux зі Store

Як видно з таблиці 3.2, зрізи movieList, castList та trailerList мають ідентичну структуру полів з прапором isFetching, що дозволяє відображати індикатор завантаження при очікуванні відповіді API. Показовим прикладом організації редьюсера є movieList, наведений на рис. 3.4.

```
const movieList = (state = defaultStateList, action) => {
 switch (action.type) {
  case FETCH_MOVIES:
  case SEARCH_MOVIE:
   return { ...state, isFetching: true };
  case FETCH_MOVIES_SUCCESS:
  case SEARCH_MOVIE_SUCCESS:
   return { ...state, isFetching: false, items: action.data };
  case FETCH_MOVIES_FAILURE:
  case SEARCH_MOVIE_FAILURE:
   return { ...state, isFetching: false, error: action.data };
  default:
   return state;
 }
};
```

Рисунок 3.4 – Редьюсер movieList у файлі reducers/index.jsx

Рисунок 3.4 ілюструє кілька ключових принципів Redux. По-перше, редьюсер є чистою функцією – при однакових вхідних аргументах завжди повертає ідентичний результат. По-друге, стан оновлюється через оператор spread {...state} без мутації поточного об'єкта. По-третє, Actions FETCH_MOVIES і SEARCH_MOVIE обробляються спільно – обидва переводять зріз у стан завантаження, що дозволяє повторно використовувати компонент MovieList як для каталогу, так і для результатів пошуку [17].

Адаптивний інтерфейс реалізовано засобами Bootstrap-сітки через компоненти Grid, Row, Col. На сторінці деталей фільму постер займає колонки xs={12} sm={6} md={4}, а інформаційний блок – xs={12} sm={6} md={8}: на мобільному екрані елементи розміщуються вертикально, на ширших – поряд у пропорції 1:2. Стилізацію ефекту наведення на картку фільму реалізовано засобами styled-components – шаблонних рядків CSS з динамічними значеннями з props, що не потребують глобальних CSS-класів [31].

Таким чином, грамотна організація файлової структури, налаштування Redux Store із сімома спеціалізованими зрізами та адаптивна верстка на базі Bootstrap утворюють цілісну технічну основу вебсервісу MovieBox.

3.3 Тестування вебсервісу та розгортання на хостингу

Тестування вебсервісу MovieBox проводилося у двох форматах. Перший – автоматизоване тестування засобами react-scripts test, що запускає середовище Jest із конфігурацією --env=jsdom для емуляції браузерного DOM. Другий – функціональне ручне тестування, у ході якого перевірялась відповідність роботи всіх модулів сформованим вимогам у реальному браузерному середовищі. Саме функціональне тестування є основним для клієнтських SPA-застосунків, що інтенсивно взаємодіють із зовнішнім API [31].

Тестування розпочиналося з перевірки головної сторінки – завантаження каталогу популярних фільмів. При відкритті застосунку компонент

MovieContainer автоматично диспатчить fetchMovieList(), отримуючи від TMDb API список фільмів, упорядкованих за популярністю. Результат відображається у вигляді адаптивної сітки карток із постерами. Знімок екрана головної сторінки наведено на рисунку 3.5.

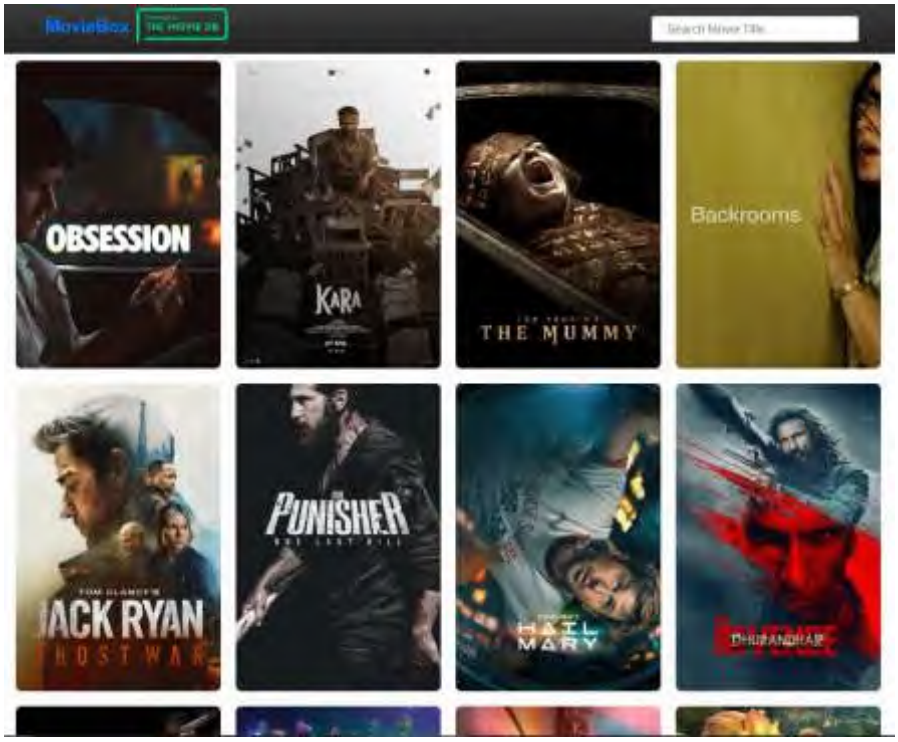

Рисунок 3.5 – Головна сторінка вебсервісу MovieBox з каталогом фільмів

Перевірка модуля пошуку включала введення тексту у поле навігаційної панелі. При кожному натисканні клавіші відображається випадний список автопідказок із мініатюрою постера та роком виходу фільму. Вибір підказки клавішею Enter або кліком миші перенаправляє до сторінки деталей відповідного фільму, а натискання Enter у полі без вибору підказки – до сторінки результатів пошуку. Знімок екрана роботи автопідказок наведено на рисунку 3.6.

Тестування сторінки деталей фільму (/movie/:id) передбачало перевірку одночасного завантаження трьох блоків даних: інформації про фільм, акторського складу та трейлерів. Під час очікування відповідей API відображається текст loading.... Після успішного завантаження сторінка відображає двоколонковий макет із постером ліворуч та блоком із назвою,

рейтингом, кількістю голосів, роком, описом і п'ятьма картками акторів праворуч, а нижче – до трьох вбудованих YouTube-трейлерів. Знімок екрана наведено на рисунку 3.7.

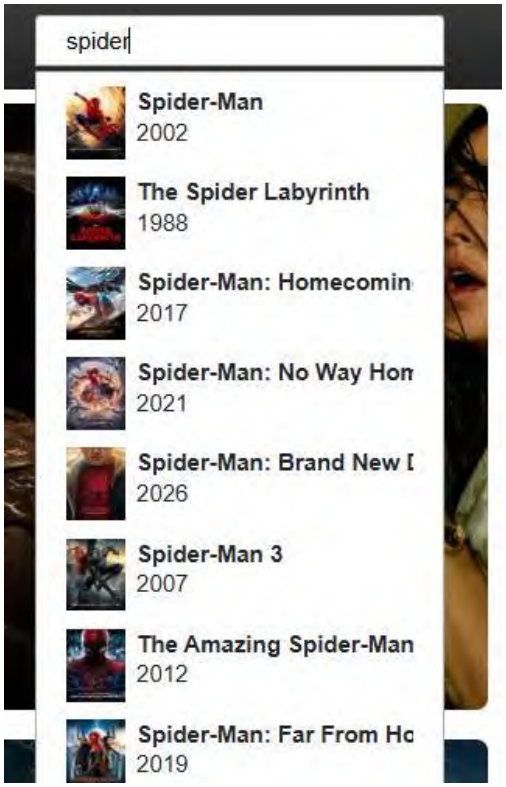

Рисунок 3.6 – Робота модуля пошуку з автопідказками

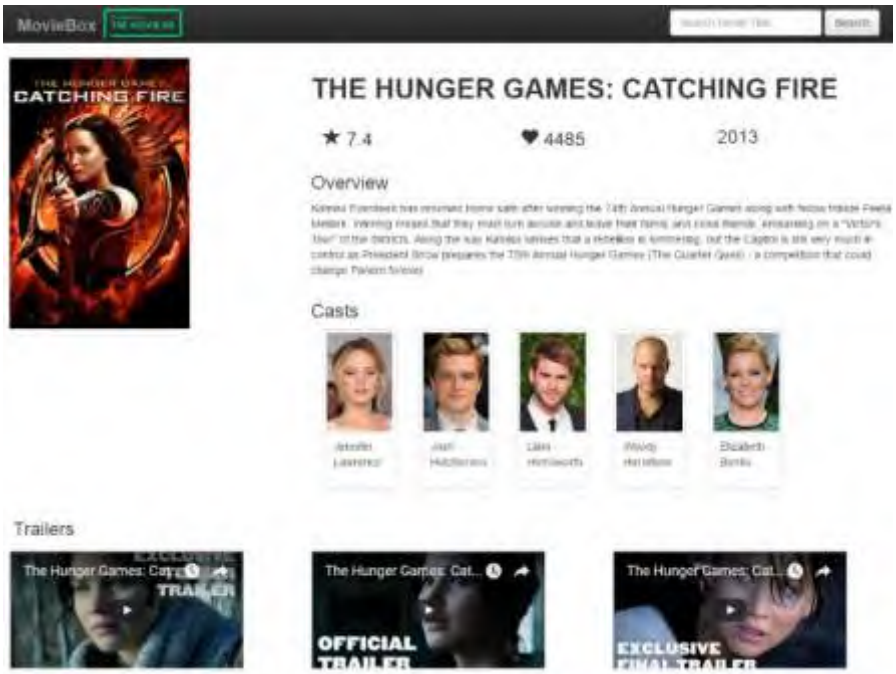

Рисунок 3.7 – Сторінка деталей фільму

Тестування профілю актора (/star/:id) здійснювалося переходом із картки акторського складу на сторінці фільму. Перевірялось коректне відображення фотографії, біографічних даних (ім'я, стать, дата народження, біографія) та секції «Known For» з чотирма фільмами за участю актора, отриманими через ендпоінт /discover/movie?with_cast={id}. Знімок екрана профілю актора наведено на рисунку 3.8.

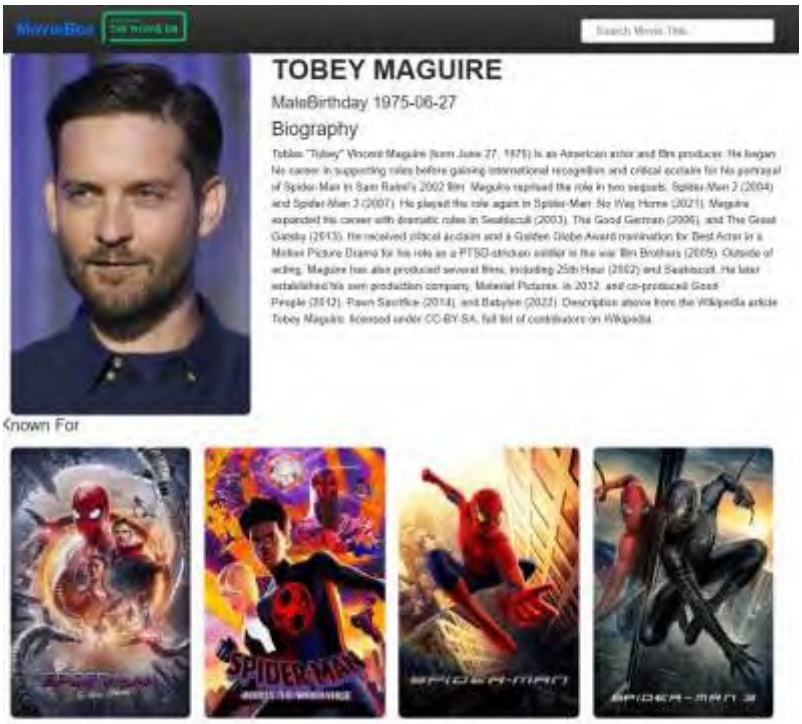

Рисунок 3.8 – Сторінка профілю актора з розділом Known For

Зведені результати перевірки функціональних вимог до системи наведено у таблиці 3.3. Усі десять перевірених функціональних вимог успішно пройдені, що підтверджує коректність реалізації всіх модулів застосунку.

Розгортання застосунку на хостингу здійснюється за допомогою пакета gh-pages на платформі GitHub Pages. У файлі package.json визначено два скрипти розгортання: predeploy виконує збірку проєкту командою npm run build, формуючи оптимізований пакет у директорії build/, після чого deploy публікує вміст цієї директорії у гілку gh-pages репозиторію командою gh-pages -d build. Поле homepage у package.json вказує фінальну адресу застосунку. Використання

hashHistory у маршрутизаторі дозволяє коректно розгорнути SPA на статичному хостингу без необхідності серверної конфігурації переадресації [32].

Таблиця 3.3 – Результати тестування функціональних вимог вебсервісу MovieBox

№	Функціональна вимога	Метод перевірки	Результат
1	Відображення каталогу фільмів	Відкриття головної сторінки	Пройдено
2	Автопідказки при пошуку	Введення тексту у поле пошуку	Пройдено
3	Повний пошук за назвою	Натискання Enter у полі пошуку	Пройдено
4	Перегляд деталей фільму	Перехід на /movie/:id	Пройдено
5	Відображення акторського складу	Перевірка сторінки деталей	Пройдено
6	Перегляд YouTube-трейлерів	Відтворення вбудованого відео	Пройдено
7	Профіль актора	Перехід на /star/:id	Пройдено
8	Рекомендації у розділі Known For	Перевірка профілю актора	Пройдено
9	Обробка некоректного URL	Перехід на /random	Пройдено
10	Адаптивне відображення	Зміна ширини вікна браузера	Пройдено

Таким чином, функціональне тестування підтвердило коректну роботу всіх модулів вебсервісу MovieBox, а автоматизоване розгортання через GitHub Pages забезпечує доступність застосунку за стабільною публічною URL-адресою без потреби у власній серверній інфраструктурі.

3.4 Техніко-економічне обґрунтування ефективності розробленої системи

Техніко-економічне обґрунтування дозволяє оцінити доцільність розробки вебсервісу MovieBox порівняно з альтернативними рішеннями та визначити реальну вартість створеного програмного продукту. Розрахунок охоплює трудомісткість розробки, пряму вартість із урахуванням обов'язкових нарахувань та накладних витрат, а також економічний ефект від використання власної розробки замість замовлення аналогічного продукту в аутсорсинговій компанії [33].

Першим кроком є визначення трудомісткості розробки – сумарного обсягу витраченого робочого часу у людино-годинах за кожним етапом. Розподіл трудомісткості між етапами розробки вебсервісу MovieBox наведено у таблиці 3.4.

Таблиця 3.4 – Трудомісткість розробки вебсервісу MovieBox за етапами

Етап розробки	Трудомісткість, год
Аналіз вимог та проєктування архітектури	24
Налаштування середовища та базової структури проєкту	8
Розробка Redux Store, Actions та Reducers	16
Реалізація модуля пошуку та автопідказок	14
Реалізація каталогу фільмів та адаптивного інтерфейсу	18
Реалізація сторінки деталей фільму	14
Реалізація профілю актора та модуля рекомендацій	12
Інтеграція TMDb API та обробка помилок	10
Тестування та розгортання на GitHub Pages	10

Як видно з таблиці 3.4, найбільш трудомісткими є етапи проєктування та реалізації адаптивного інтерфейсу, що відповідає практиці розробки клієнтських React-застосунків із централізованим управлінням станом.

Для розрахунку вартості розробки визначається годинна ставка розробника. Згідно із зарплатним звітом DOU 2024, медіанна заробітна плата junior frontend-розробника в Україні становить 40 000 грн/місяць [33]. При стандартному місячному фонді робочого часу 168 годин (21 робочий день по 8 годин) годинна ставка становить:

$$C_{\text{год}} = \frac{\text{ЗП}_{\text{міс}}}{T_{\text{міс}}} = \frac{40\,000}{168} \approx 238 \text{ грн/год}$$

Загальна вартість розробки $C_{\text{розр}}$визначається з урахуванням основної заробітної плати, єдиного соціального внеску (ЄСВ, коефіцієнт $K_{\text{ЄСВ}} = 0{,}22$) та накладних витрат (коефіцієнт $K_{\text{НВ}} = 0{,}25$):

$$C_{\text{розр}} = C_{\text{год}} \times T_{\text{розр}} \times (1 + K_{\text{ЄСВ}} + K_{\text{НВ}}) = 29\,988 \times 1{,}47 \approx 44100 \text{ грн.}$$

Таким чином, загальна вартість розробки вебсервісу MovieBox власними силами становить приблизно 44100 грн.

Для обґрунтування економічної ефективності розроблену систему порівняно з двома альтернативами. Перша – замовлення аналогічного застосунку в аутсорсинговій IT-компанії: вартість розробки кастомного вебзастосунку порівнянної складності (React SPA з API-інтеграцією, адаптивним UI та багатосторінковою навігацією) на українському ринку становить від 100000 до 160000 грн, зі середнім значенням близько 120000 грн [34]. Друга – використання готових комерційних SaaS-рішень для каталогізації фільмів: вартість річної підписки становить від 15000 до 30000 грн, проте такі рішення не надають повного контролю над функціоналом і не дозволяють адаптувати інтерфейс під конкретні потреби. Порівняльний аналіз вартості трьох підходів наведено на рисунку 3.9.

Рисунок 3.9 – Порівняння вартості реалізації вебсервісу різними підходами

Як видно з рисунку 3.9, власна розробка є найбільш економічно вигідним варіантом серед повноцінних рішень, що забезпечують повний контроль над кодом і функціоналом. Економічний ефект від вибору власної розробки замість аутсорсингу становить:

$$E = C_{\text{альт}} - C_{\text{розр}} = 120000 - 44100 = 75900 \text{ грн.}$$

Коефіцієнт економічної ефективності розробки:

$$k_e = \frac{E}{C_{\text{розр}}} = \frac{75\,900}{44\,100} \approx 1{,}72.$$

Значення $k_e > 1$ підтверджує, що на кожну вкладену гривню розробки отримується 1,72 грн економічного ефекту порівняно із замовленням аналогічного рішення [33].

Окремою статтею економії є нульова вартість експлуатації: TMDb API надає безкоштовний доступ для некомерційних проєктів, а розгортання на GitHub Pages не потребує витрат на хостинг. Таким чином, після одноразових витрат на розробку сервіс функціонує без жодних recurring-витрат, що суттєво відрізняє його від комерційних SaaS-альтернатив.

Техніко-економічне обґрунтування підтверджує доцільність розробки вебсервісу MovieBox: вартість власної реалізації у 2,7 рази нижча за вартість замовлення аналогічного рішення в аутсорсинговій компанії, а отриманий програмний продукт повністю задовольняє усі функціональні та нефункціональні вимоги.

ВИСНОВКИ

У кваліфікаційній роботі проведено комплексне дослідження та виконано практичну розробку інтерактивного вебсервісу для пошуку, сортування та рекомендацій фільмів з використанням бібліотеки React.

За результатами виконаної роботи можна зробити такі висновки.

У першому розділі проведено аналіз предметної області та огляд існуючих кіносервісів. Встановлено, що більшість комерційних платформ або мають обмежений API-доступ, або є надлишково складними для адаптації. Обґрунтовано вибір TMDb як єдиної платформи з повністю відкритим API та безкоштовним доступом. Досліджено технологічну базу сучасного фронтенд-розробки та обґрунтовано перевагу бібліотеки React над Angular і Vue для проєктів середнього масштабу, зокрема завдяки механізму Virtual DOM, компонентній архітектурі та зрілій екосистемі. Детально розглянуто принципи централізованого управління станом на базі Redux та алгоритмічні підходи до реалізації пошуку (делегування серверній стороні з підтримкою debouncing), сортування (TimSort на клієнтській стороні) та рекомендацій (контентна фільтрація на основі косинусної подібності).

У другому розділі сформовано повний перелік функціональних та нефункціональних вимог до системи, побудовано діаграму варіантів використання та обрано технологічний стек. До складу стеку увійшли React 15, Redux 3.6 із middleware redux-thunk, react-router v3, react-bootstrap, react-autosuggest та styled-components. Спроєктовано ієрархію з 15 компонентів із чітким розподілом на контейнерні та презентаційні. Описано інтеграцію з TMDb API v3 через сім ендпоінтів, що забезпечують отримання каталогу фільмів, результатів пошуку, деталей фільму, акторського складу, трейлерів та профілів акторів.

У третьому розділі реалізовано всі заплановані функціональні модулі вебсервісу MovieBox. Модуль пошуку реалізовано у двох режимах – автопідказки в реальному часі та повний пошук із маршрутизацією. Модуль

рекомендацій реалізовано на основі акторського складу через ендпоінт /discover/movie?with_cast={id}, що відображає чотири найпопулярніші фільми за участю обраного актора. Налаштовано Redux Store із сімома зрізами стану та маршрутизацію через п'ять маршрутів з підтримкою hashHistory. Проведено функціональне тестування, яке підтвердило відповідність усіх десяти перевірених вимог очікуваним результатам. Розгортання застосунку здійснено на платформі GitHub Pages засобами пакета gh-pages.

Виконано техніко-економічне обґрунтування розробки. Загальна трудомісткість становить 126 людино-годин, а вартість власної розробки – близько 44100 грн, що у 2,7 рази менше вартості замовлення аналогічного рішення в аутсорсинговій компанії. Коефіцієнт економічної ефективності дорівнює 1,72, що підтверджує доцільність самостійної розробки. Додатковою перевагою є нульова вартість експлуатації – TMDb API та GitHub Pages є безкоштовними для некомерційних проєктів.

Практична значущість роботи полягає у створенні повнофункціонального вебсервісу, розгорнутого за публічною URL-адресою, який може слугувати навчальним проєктом із сучасного React-стеку або основою для подальшого вдосконалення.

Напрямками подальшого розвитку є міграція на актуальні версії бібліотек (React 18, Redux Toolkit), додавання системи автентифікації з персональним списком обраного, реалізація розширеної фільтрації за жанрами та роками випуску, впровадження серверного рендерингу (Next.js) для покращення SEO, а також розробка повноцінної рекомендаційної системи на основі колаборативної фільтрації з накопиченням даних про вподобання користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Statista Research Department. Number of digital video viewers worldwide 2024. URL: https://www.statista.com/statistics/1109466/global-digital-video-viewers/

2. Gomez-Uribe C. A., Hunt N. The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Transactions on Management Information Systems.* 2022. Vol. 6, no. 4. P. 1–19. URL: https://doi.org/10.1145/2843948

3. Огляд сучасних стримінгових платформ та їх технологій. URL: https://dou.ua/lenta/articles/streaming-platforms-overview/

4. The Movie Database API Documentation. URL: https://developer.themoviedb.org/docs/getting-started

5. Nielsen J. Response Times: The 3 Important Limits. Nielsen Norman Group. 2024. URL: https://www.nngroup.com/articles/response-times-3-important-limits/

6. Ricci F., Rokach L., Shapira B. Recommender Systems Handbook. 3rd ed. Springer, 2022. 1008 p. URL: https://doi.org/10.1007/978-1-0716-2197-4

7. Що таке SPA і чому React є стандартом галузі. URL: https://dou.ua/lenta/articles/react-spa-standard/

8. Çalışkan A., Doğan O. A Systematic Review of SPA Development Technologies. *Journal of Web Engineering*. 2023. Vol. 22, no. 3. P. 411–445. URL: https://doi.org/10.13052/jwe1540-9589.2234

9. Що таке Single Page Application і навіщо воно потрібне. URL: https://dou.ua/lenta/articles/what-is-spa/

10. ECMAScript 2022 Language Specification. ECMA International, 2022. URL: https://tc39.es/ecma262/

11. Mardan A. React Native in Action. 2nd ed. Manning Publications, 2023. 392 p. URL: https://www.manning.com/books/react-native-in-action

12. Aggarwal S. Modern Web-Development Using ReactJS. *International Journal of Recent Research Aspects*. 2023. Vol. 5, no. 1. P. 133–137. URL: https://doi.org/10.2139/ssrn.3444690

13. React офіційна документація. Thinking in React. URL: https://react.dev/learn/thinking-in-react

14. Fedosejev A. React.js Essentials. Packt Publishing, 2022. 260 p. URL: https://www.packtpub.com/product/reactjs-essentials

15. React Hooks – офіційна документація. URL: https://react.dev/reference/react

16. Stack Overflow Developer Survey 2024. URL: https://survey.stackoverflow.co/2024/technology

17. Redux офіційна документація. URL: https://redux.js.org/introduction/getting-started

18. Abramov D., Clark A. You Might Not Need Redux. Medium. 2023. URL: https://medium.com/@dan_abramov/you-might-not-need-redux-be46360cf367

19. Redux Toolkit офіційна документація. URL: https://redux-toolkit.js.org/introduction/getting-started

20. Wieruch R. The Road to Redux. Leanpub, 2022. 180 p. URL: https://leanpub.com/taming-the-state-in-react

21. Gavin M. Search UX: Best Practices for Designing Search Interfaces. Smashing Magazine. 2023. URL: https://www.smashingmagazine.com/2023/02/search-ux-best-practices/

22. Cormen T. H., Leiserson C. E., Rivest R. L., Stein C. Introduction to Algorithms. 4th ed. MIT Press, 2022. 1312 p. URL: https://mitpress.mit.edu/9780262046305/introduction-to-algorithms/

23. Wiegers K., Beatty J. Software Requirements. 3rd ed. Microsoft Press, 2023. 672 p. URL: https://www.microsoftpressstore.com/store/software-requirements-9780735679665

24. Cockburn A. Writing Effective Use Cases. Addison-Wesley, 2021. 304 p. URL: https://www.pearson.com/en-us/subject-catalog/p/writing-effective-use-cases/P200000009417

25. React Router v3 документація. URL: https://github.com/remix-run/react-router/tree/v3/docs

26. React-Bootstrap документація. URL: https://react-bootstrap-v3.netlify.app/getting-started/introduction/

27. Холоденко О. Патерни проєктування React-компонентів. URL: https://dou.ua/lenta/articles/react-component-patterns/

28. Fielding R. T. Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, UC Irvine. 2000 (repr. 2023). URL: https://ics.uci.edu/~fielding/pubs/dissertation/top.htm

29. MDN Web Docs. Using the Fetch API. Mozilla Developer Network. 2024. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch

30. styled-components. Visual primitives for the component age. URL: https://styled-components.com/docs

31. Oskan E. Testing React Applications with Jest. LogRocket Blog. 2023. URL: https://blog.logrocket.com/testing-react-apps-jest/

32. GitHub Pages документація. URL: https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages

33. Зарплати розробників в Україні. URL: https://dou.ua/lenta/articles/salary-report-2024/

34. Скільки коштує розробка вебсайту в Україні у 2024 році. URL: https://dou.ua/lenta/articles/web-development-cost-2024/