МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп'ютерних наук та кібербезпеки

САМОЙЛІЧ ЮРІЙ ВАСИЛЬОВИЧ

РОЗРОБКА ПОШУКОВОЇ СИСТЕМИ ЗА ТЕМАТИКОЮ КІНЕМАТОГРАФІЇ

Спеціальність: 122 Комп'ютерні науки
Освітньо-професійна програма: Комп'ютерні науки та інформаційні технології
Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
МАМЧИЧ ТЕТЯНА ІВАНІВНА,
кандидат фізико-математичних наук, доцент
кафедри комп'ютерних наук та кібербезпеки

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол No_____
засідання кафедри комп'ютерних наук
та кібербезпеки
від _____ 2025 р.
Завідувач кафедри
(_____) Гришанович Т. О

Луцьк — 2025

ЗМІСТ

ВСТУП

У період інтенсивного розвитку інформаційних технологій онлайн-сервіси набули широкого поширення у щоденному використанні. З поширенням інтернет-платформ, зокрема стрімінгових сервісів для перегляду аудіовізуального контенту, користувачам надається доступ до великого обсягу медіаресурсів. Унаслідок цього виникає завдання забезпечення точного та швидкого пошуку необхідного фільму чи серіалу серед значної кількості матеріалів. Із щорічним зростанням обсягу контенту зростає також потреба у застосуванні інструментів для пошуку, що підтримують фільтрацію за заданими параметрами, зокрема за жанром, акторським складом, роком виходу чи країною виробництва.

Незважаючи на наявність великих баз даних, більшість відеосервісів реалізують обмежений набір функцій для пошуку, що, як правило, зводиться до класифікації за базовими ознаками – такими як жанр або рейтинг. Такий підхід не враховує запити користувачів, які очікують доступу до розширених критеріїв відбору. Зокрема, затребуваними залишаються параметри фільтрації за акторським складом, режисером, роком випуску, популярністю, наявністю субтитрів чи встановленим віковим рейтингом.

У зв’язку з викладеним, розроблення вебресурсу для пошуку фільмів і серіалів, який забезпечує багатокритеріальний відбір контенту, є обґрунтованим завданням. Реалізація подібного сервісу дозволяє покращити користувацьку взаємодію з платформою, що впливає на її функціональність та придатність для застосування в межах цифрових інформаційних систем.

Актуальність теми полягає в тому, що в умовах інтенсивного розвитку цифрових технологій та розширення інтернет-середовища користувачі отримують доступ до великого обсягу інформаційних ресурсів, зокрема медіаконтенту. Онлайн-кінотеатри надають доступ до фільмів і серіалів у режимі реального часу, однак зростання обсягу контенту ускладнює процес його пошуку та систематизації. Це зумовлює потребу в засобах фільтрації на основі множини параметрів, серед

яких жанр, рік випуску, акторський склад, режисер, країна виробництва, рейтинг, а також наявність додаткових характеристик, таких як трейлери або відгуки. Формування інтерфейсу з розширеною системою пошуку дозволяє підвищити точність відбору контенту та полегшити взаємодію з ресурсом. За умов зростання попиту на інтернет-платформи для перегляду відео розроблення програмних рішень, орієнтованих на пошук і рекомендацію контенту, становить частину задач у межах інформаційних систем.

Метою кваліфікаційної роботи є створення вебресурсу, що реалізує функції пошуку фільмів і серіалів за сукупністю заданих критеріїв, таких як жанр, актори, рік випуску, країна виробництва тощо. Сайт забезпечуватиме відображення опису, трейлерів, користувацьких оцінок та інших інформаційних елементів, пов’язаних із вибраним контентом.

Завданням кваліфікаційної роботи є:

- виконати аналіз наявних вебресурсів, призначених для пошуку фільмів і серіалів, з метою виявлення їхніх функціональних можливостей, а також обмежень у реалізації пошукових і навігаційних засобів;
- сформувати концептуальну модель вебсайту, що реалізує пошук контенту за визначеними параметрами, з урахуванням актуальних підходів до побудови вебінтерфейсів;
- реалізувати вебзастосунок із використанням інструментів і засобів розробки, які відповідають сучасним технічним вимогам;
- розробити компонент пошуку, що дозволяє здійснювати фільтрацію за такими характеристиками, як жанр, акторський склад, рейтинг, рік випуску та інші параметри;
- реалізувати відображення стислої інформації про кожну одиницю контенту, включаючи трейлер, рейтинг, відгуки користувачів та інші супровідні дані.

Об’єктом дослідження є процес створення вебсайтів, орієнтованих на пошук фільмів і серіалів, який включає дослідження вимог до функціональних складових

та інтерфейсного представлення, а також розроблення програмної архітектури системи.

Предметом дослідження виступають засоби, підходи та технології реалізації вебресурсів, призначених для пошуку контенту за сукупністю заданих характеристик.

РОЗДІЛ 1.
ТЕОРЕТИЧНІ ОСНОВИ ПОШУКОВИХ АЛГОРИТМІВ В ІНТЕРНЕТ-МЕРЕЖІ ТА ЇХ ЗАСТОСУВАННЯ В ГАЛУЗІ ІНФОРМАЦІЙНОГО ПОШУКУ

У цифровому середовищі інформаційний пошук є однією з базових функцій, які забезпечують користувачам доступ до релевантних даних. Зі зростанням кількості інформаційних ресурсів виникає потреба у створенні алгоритмів, здатних не лише забезпечувати швидкий пошук, але й формувати результати з урахуванням релевантності до запиту. Основою таких систем є пошукові алгоритми, які класифікуються за різними принципами залежно від методу обробки запиту, структури індексації, використання моделей ранжування та оцінки релевантності.

До класу традиційних алгоритмів належать ті, що базуються на булевій моделі, де запити формулюються за допомогою логічних операторів. Векторна модель дозволяє розраховувати ступінь подібності між запитом і документом за допомогою математичних методів. Модель на основі ймовірнісного підходу дозволяє враховувати ймовірність відповідності документа запиту. Також активно застосовуються алгоритми, які використовують машинне навчання, нейронні мережі, семантичний аналіз та інші методи, орієнтовані на покращення точності результатів.

Аналіз пошукових алгоритмів передбачає вивчення їхніх характеристик, зокрема складності, масштабованості, здатності до індексації великих обсягів даних, а також адаптивності до змін у структурі інформації. У наступному підрозділі розглянуто класифікацію основних типів пошукових алгоритмів, яка дозволяє структурувати знання про їх призначення та області застосування.

1.1. Поняття та класифікація пошукових алгоритмів

Пошукові алгоритми являють собою формалізовані процедури, призначені для обробки структурованих або неструктурованих даних з метою знаходження елементів, що відповідають заданим умовам. Вони реалізуються в інформаційно-пошукових системах і використовуються для автоматизованого вилучення релевантної інформації на основі користувацького запиту.

Алгоритми пошуку класифікуються за кількома ознаками. За способом організації даних виділяють алгоритми, які працюють з впорядкованими або невпорядкованими масивами. У випадку невпорядкованих структур зазвичай використовується лінійний пошук, що полягає в послідовному порівнянні елементів масиву з цільовим значенням. У впорядкованих структурах застосовується бінарний пошук, який реалізує поділ масиву на частини, що дозволяє зменшити кількість ітерацій для досягнення результату [4].

У більш складних структурах даних, таких як графи, використовуються алгоритми пошуку в глибину (DFS) та в ширину (BFS), які забезпечують повне або часткове обходження вузлів графа відповідно до визначених правил. Такі методи є основою для реалізації навігаційних систем, аналізу мереж, семантичного пошуку та ін.

Пошукові системи, орієнтовані на великі обсяги інформації, зокрема Google, застосовують комбінацію алгоритмів. Це можуть бути алгоритми індексації, ранжування, зворотного посилання, машинного навчання, а також евристичні методи. Комбінація таких рішень дозволяє здійснювати багаторівневу обробку запитів, включаючи попередню фільтрацію, формування результатів видачі та їх персоналізацію [1].

Критерії вибору пошукового алгоритму залежать від характеристик даних (структура, обсяг, частота оновлення) та функціональних вимог до системи (швидкодія, точність, масштабованість). Надалі розглядається докладна

характеристика найбільш поширених типів пошукових алгоритмів, які застосовуються для вирішення задач інформаційного пошуку.

Лінійний пошук реалізується як ітеративна процедура, що перевіряє кожен елемент послідовно. Вхідними даними є масив або список довільної структури, а також значення, яке потрібно знайти. Алгоритм припиняє роботу після знаходження першого елементу, що відповідає умові, або по завершенню перегляду всіх елементів. Перевагою такого підходу є відсутність вимог до впорядкування даних. Це дозволяє використовувати алгоритм у випадках динамічного оновлення вхідної інформації. Проте зростання розміру масиву прямо пропорційно впливає на час виконання.

Бінарний пошук виконується рекурсивно або ітеративно та потребує попередньо відсортованого масиву. У кожній ітерації визначається середній індекс масиву, після чого порівнюється значення за цим індексом з цільовим елементом. Якщо значення менше за цільове, пошук продовжується в правій частині масиву, інакше – в лівій. Перевагою є логарифмічна складність, що дає змогу скоротити кількість операцій у порівнянні з лінійним пошуком. Це досягається завдяки зменшенню області пошуку у два рази на кожному кроці. Недоліком є потреба у впорядкуванні, яке зазвичай має складність O(n log n), що може бути нераціональним у випадку частої зміни даних [6].

Обидва методи знаходять застосування в залежності від умов використання: для невпорядкованих або невеликих наборів даних застосовується лінійний пошук, для великих впорядкованих – бінарний. Подальший розгляд включає більш складні алгоритми пошуку, що використовуються для роботи з графовими структурами та великими обсягами інформації в розподілених системах.

Схематично алгоритми лінійного та бінарного пошуку наведені на рис. 1.1.

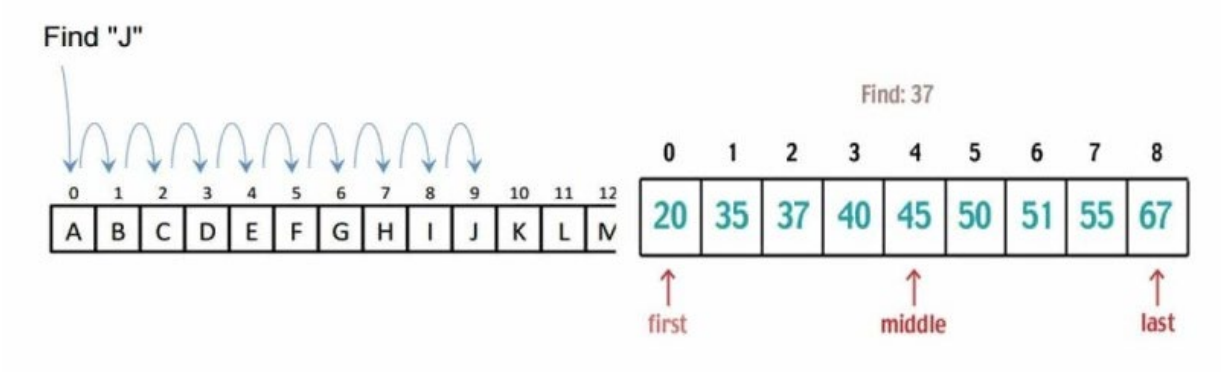

Рисунок 1.1 – Схематичне зображення лінійного та бінарного пошуків

Алгоритм пошуку в глибину (DFS) реалізується за допомогою рекурсії або структури даних типу стек. Починаючи з початкової вершини, алгоритм відвідує сусідні вершини, поки не досягне вершини без невідвіданих сусідів. Після цього він повертається назад до попередньої вершини та продовжує пошук. Такий підхід дозволяє дослідити всі шляхи від джерела до кінця, але може призводити до великої глибини рекурсії або переповнення стека при роботі з глибокими графами. Часова складність алгоритму становить O(V + E), де V – кількість вершин, E – кількість ребер.

Пошук в ширину (BFS) використовує чергу для зберігання вершин, які потрібно відвідати. Алгоритм відвідує всі вершини на поточному рівні до переходу на наступний рівень. Завдяки цій властивості, BFS застосовується для визначення найкоротшого шляху в невагованих графах. Його складність також становить O(V + E). BFS гарантує знаходження рішення з найменшою кількістю переходів у графі, що робить його придатним для задач типу маршрутизації або планування [17].

Обидва алгоритми пошуку – DFS та BFS – не потребують попереднього сортування, однак вимагають спеціального представлення даних у вигляді графів або дерев. Вибір між цими методами визначається характером задачі: DFS використовується для повного обходу або пошуку всіх рішень, BFS – для знаходження оптимальних за кількістю кроків шляхів.

Схематично алгоритми BFS та DFS наведені на рисунку 1.2.

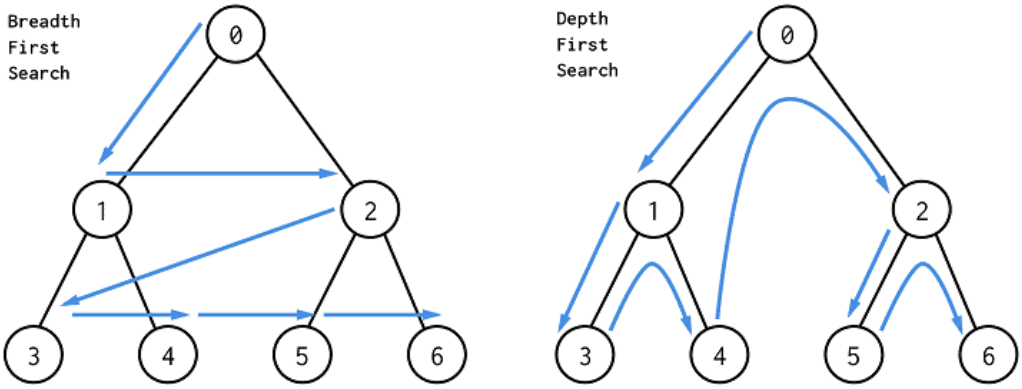

Рисунок 1.2 – Схематичне зображення BFS та DFS

Серед додаткових типів пошукових алгоритмів можна також згадати методи пошуку, засновані на структурування даних, такі як хешування або індексація.

Методи хешування реалізуються шляхом обчислення хеш-функції від ключа, яка визначає індекс зберігання значення в хеш-таблиці. Перевагою такого підходу є постійний час доступу до елемента, якщо уникнути колізій або мінімізувати їх вплив за допомогою методів розв’язання конфліктів, таких як ланцюгове хешування або відкриту адресацію. Недоліком є збільшене використання пам’яті та можливі втрати продуктивності при високій кількості колізій.

Індексація передбачає створення структур, які дозволяють прискорити пошук у великих масивах даних, наприклад, індекси у базах даних. Вони будуються на основі певних полів і дозволяють прискорити доступ до даних, що відповідають запиту. Найчастіше використовуються B-дерева, які дозволяють підтримувати індекси у впорядкованому вигляді з логарифмічною складністю доступу.

1.2. Існуючі підходи до реалізації пошукових алгоритмів в мережі інтернет

У сфері пошуку в Інтернеті застосовуються алгоритми, які забезпечують масштабованість, швидкий доступ до даних та адаптивність до змін вмісту. Відповідно до [12], виділяються чотири типи пошуку: прямий пошук, інвертовані списки, суфіксні дерева та сигнатури.

Прямий пошук реалізується без попередньої індексації. Його суть полягає в порівнянні кожного документа або його фрагментів з пошуковим запитом. Це дає точні результати, але потребує значних ресурсів, особливо при роботі з великими обсягами інформації. В окремих випадках прямий пошук комбінується з попередньо побудованим індексом, після чого використовується, наприклад, алгоритм Бойера-Мура, адаптований до пошуку в текстових масивах.

Інвертований список є основою більшості пошукових систем. Він передбачає створення словника термінів, де кожен термін пов’язаний з переліком позицій у документах. Така структура дозволяє швидко визначити, у яких документах зустрічається слово, та побудувати посилання на його розташування. Позиції можуть зберігатися у вигляді відносних зсувів для зменшення об’єму даних. Додатково застосовуються алгоритми стиснення, наприклад, кодування Хаффмана або LZW, які дозволяють знизити витрати пам’яті при збереженні прийнятного часу доступу.

Суфіксні дерева забезпечують індексацію всіх суфіксів тексту. Вони реалізують більш гнучке та комплексне пошукове дерево, що дозволяє здійснювати пошук за довільними підрядками. Конкретна реалізація, як-от в системі OpenText [2], передбачає оптимізацію структури для забезпечення швидкого доступу до контексту суфіксів. Суфіксні дерева дають можливість виконання запитів з розширеною логікою (наприклад, часткове співпадіння), проте потребують значного об’єму оперативної пам’яті.

Метод сигнатур базується на побудові таблиці хеш-значень (сигнатур) для кожного слова документа. У процесі пошуку зіставляються сигнатури запиту з сигнатурами в таблиці. Це дозволяє уникнути повного аналізу тексту. Результативність залежить від обраної хеш-функції, яка повинна мінімізувати кількість хибнопозитивних спрацювань. Для покращення точності можуть застосовуватись методи булевих векторів та об'єднання кількох хеш-функцій [8].

1.3. Огляд та аналіз аналогічних програмних розробок

Для проведення порівняльного аналізу програмних продуктів, що виконують схожі функції з системами пошуку фільмів і серіалів, зокрема для сервісу JustWatch, можна розглянути два популярних сервіси: Reelgood та JustWatch, де JustWatch буде основним продуктом для порівняння.

Reelgood – це сервіс, який дозволяє шукати контент на понад 20 стрімінгових платформах, таких як Netflix, Hulu, Amazon Prime та інших. Головна сторінка сервісу наведена на рисунку 1.3.

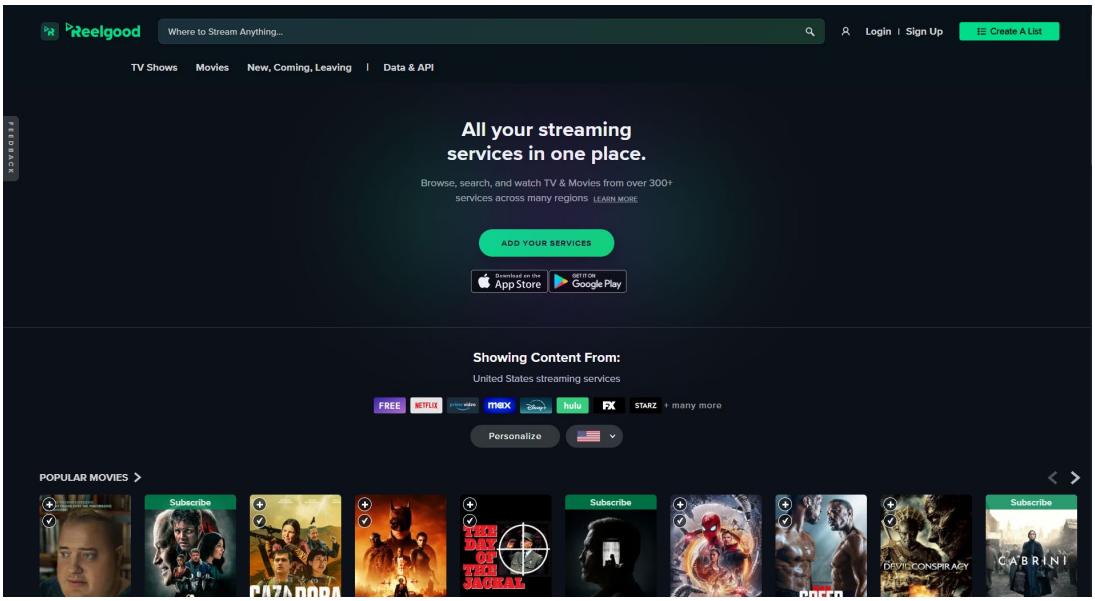

Рисунок 1.3 – Головна сторінка Reelgood

Розробником Reelgood є компанія Reelgood Inc. Система реалізована у вигляді веб-застосунку та мобільних додатків для iOS і Android. Веб-версія використовує JavaScript і React, а мобільні додатки розроблені за допомогою Swift для iOS та Kotlin для Android [24]. Reelgood підтримує систему персоналізованих рекомендацій, яка ґрунтується на історії переглядів користувача, і пропонує функцію створення персональних списків контенту. Крім того, сервіс надає можливість порівняння доступності та популярності контенту на різних сервісах. Цей функціонал можно побачити на рисунку 1.4.

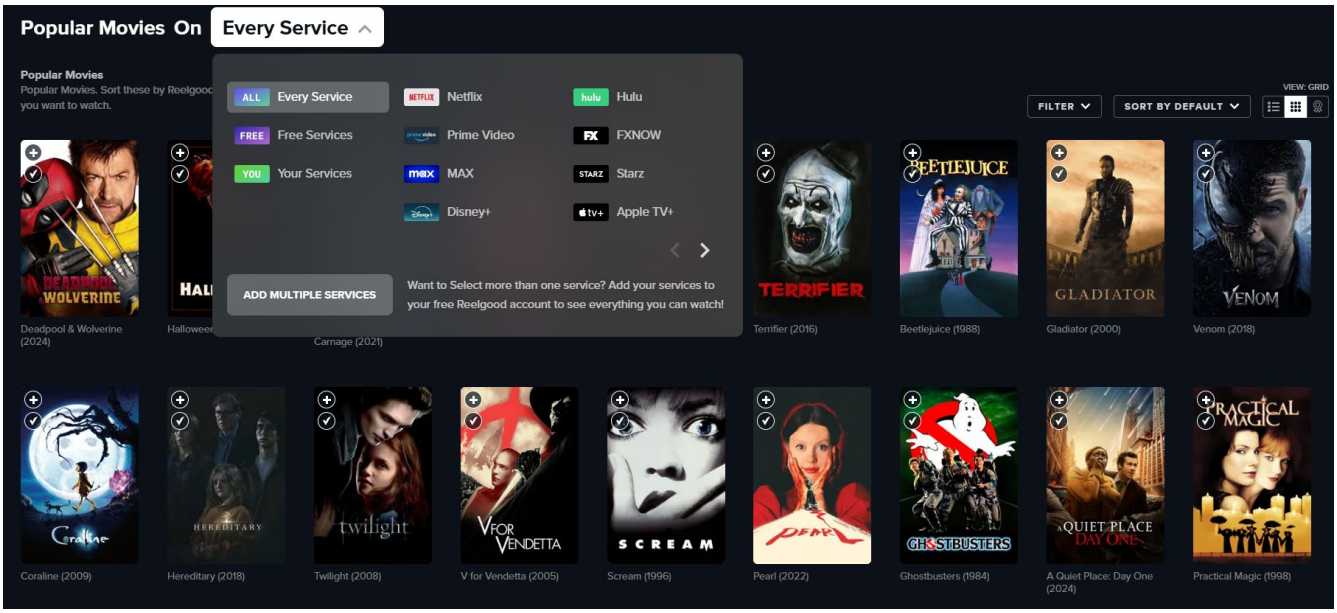

Рисунок 1.4 – Вибір популярних фільмів на різних сервісах в Reelgood

Інтерфейс сервісу Reelgood є досить простим і інтуїтивно зрозумілим, що сприяє зручності використання. Користувачі можуть швидко знаходити потрібні фільми та серіали завдяки структурованому меню та системі фільтрації. Однак цей сервіс має певні обмеження. Одним із них є менш точні рекомендації в порівнянні з іншими сервісами, що використовують складніші алгоритми для персоналізації контенту [21]. Крім того, для доступу до деяких функцій на Reelgood необхідно створювати акаунт, що може бути незручним для користувачів, які хочуть швидко

знайти контент без додаткових кроків. Також варто зазначити, що контент Reelgood може бути більш обмеженим у порівнянні з JustWatch.

JustWatch, зі свого боку, надає більш широку базу даних фільмів та серіалів, доступних на різних стрімінгових платформах, таких як Netflix, Amazon Prime, Hulu, Disney+, HBO, Apple TV та інших. Це дозволяє користувачам здійснювати фільтрацію контенту за різними критеріями, такими як жанр, рік випуску, мова та інші. Крім того, JustWatch має детальну та зручну систему пошуку, що сприяє швидкому знаходженню релевантних фільмів та серіалів. Веб-версія цього сервісу реалізована за допомогою JavaScript, а мобільні додатки для iOS та Android були створені з використанням Swift та Kotlin відповідно.

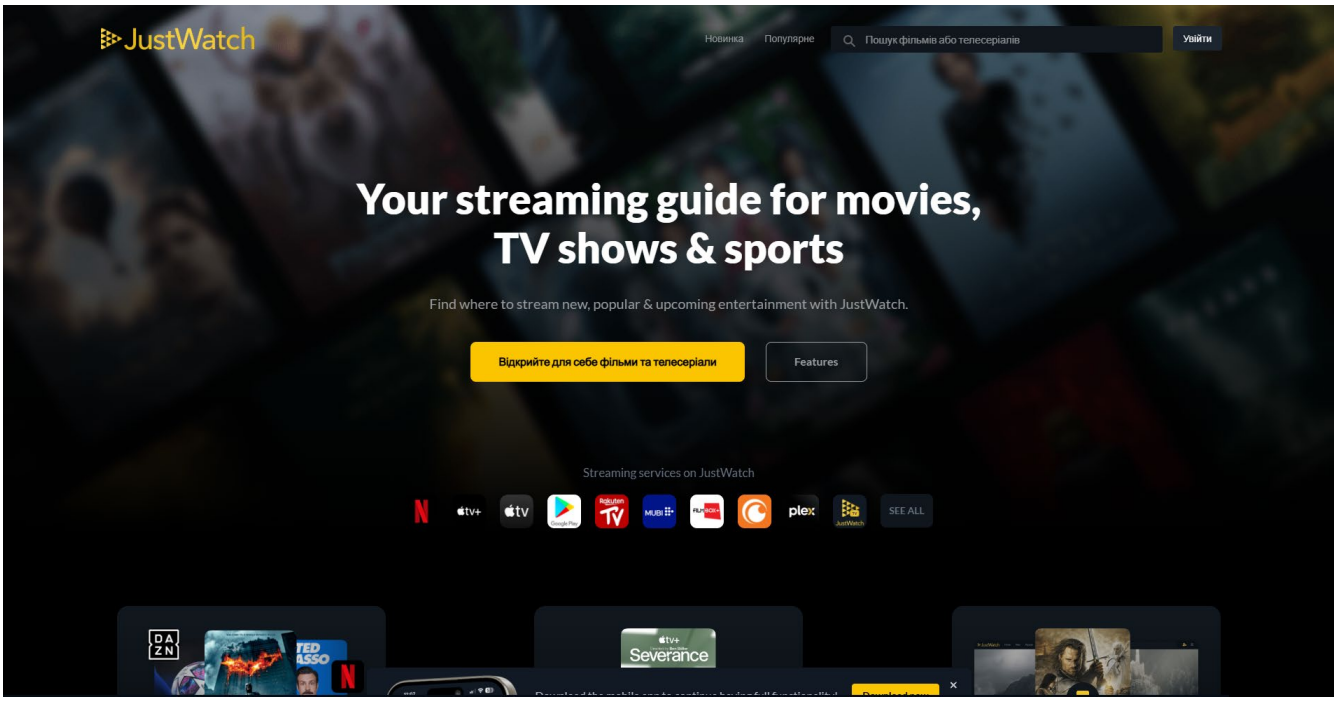

Рисунок 1.5 – Головна сторінка Reelgood JustWatch

JustWatch також надає персоналізовані рекомендації, засновані на історії переглядів користувача, що дозволяє підбирати фільми та серіали відповідно до інтересів. Сервіс допомагає користувачам визначити доступність контенту на різних платформах. Однак, персоналізація рекомендацій в JustWatch може бути

менш точною порівняно з іншими сервісами, які застосовують більш складні алгоритми для аналізу користувацьких уподобань. Крім того, веб-версія сервісу може мати обмежену функціональність порівняно з мобільними додатками, що може обмежувати зручність використання для деяких користувачів [22].

При порівнянні з Reelgood, JustWatch підтримує більшу кількість локальних сервісів для різних країн. JustWatch також забезпечує більшу гнучкість у персоналізації рекомендацій завдяки можливості детальної фільтрації контенту за різними критеріями. Reelgood надає персоналізацію рекомендацій, але в меншому обсязі. Що стосується інтерфейсу, обидва сервіси є зручними, однак JustWatch має більше можливостей для фільтрації та надає зручніший пошук контенту через відображення доступності на різних платформах.

Reelgood виділяється тим, що дозволяє відстежувати контент безпосередньо в мобільному додатку. Це є значною перевагою для користувачів, які хочуть стежити за новинами та змінами у контенті, не переходячи між різними платформами. Якщо основною задачею є створення додатку для пошуку контенту на різних стрімінгових платформах, JustWatch є більш придатним варіантом завдяки своїй здатності відображати доступність контенту на різних сервісах. Водночас, Reelgood є корисним для розробки системи рекомендацій, оскільки підтримує більшу кількість платформ і дозволяє надавати точніші рекомендації на основі користувацьких уподобань.

На основі проведеного аналізу можна сформулювати вимоги до програмного продукту для створення веб-сайту для пошуку фільмів і серіалів. Продукт повинен інтегруватися з API баз даних про фільми, таких як The Movie Database (TMDb) або OMDb API (Open Movie Database), щоб забезпечити доступ до актуальної інформації, включаючи жанри, рейтинги, описи та акторський склад. Це дозволить відображати дані за запитом користувача з можливістю фільтрації за різними параметрами.

Інтерфейс веб-сайту має бути інтуїтивним і адаптивним, щоб підтримувати як десктопні, так і мобільні пристрої. Для цього можна використовувати фреймворки для фронтенду, такі як React або Vue.js, що дозволить реалізувати динамічні компоненти для пошуку і фільтрації контенту. Для бекенд-частини можна застосувати Node.js з Express або Python з Flask чи Django для обробки запитів, підключення до API [9] та передачі обробленої інформації на фронтенд.

Якщо продукт передбачатиме збереження даних користувачів, таких як історія пошуку чи персональні списки, слід передбачити використання бази даних для зберігання цієї інформації. Можна використовувати реляційні бази даних, такі як PostgreSQL або MySQL, або нереляційну MongoDB в залежності від вимог до зберігання даних.

Для досягнення поставленої мети треба забезпечити наявність функцій фільтрації за жанрами, мовою, рейтингом чи роком виходу, щоб користувач міг швидко знайти релевантний контент на основі своїх уподобань.

РОЗДІЛ 2.
РОЗРОБКА ВЕБ-САЙТУ ДЛЯ ПОШУКУ ФІЛЬМІВ ТА СЕРІАЛІВ ЗА ЖАНРАМИ ТА АКТОРАМИ

2.1. Постановка задачі, призначення та вимоги до програмного засобу «Система пошуку фільмів і серіалів»

Метою розробки веб-сайту є створення платформи для пошуку фільмів та серіалів за різними параметрами, такими як назва, жанр та актор. Користувачі зможуть отримувати інформацію про популярні фільми, серіали та їх акторів, переглядати трейлери та ознайомлюватися з детальними відомостями про вибраний контент. Веб-сайт надаватиме лише інформаційні послуги, не передбачаючи необхідність реєстрації чи авторизації, оскільки основне завдання ресурсу – це пошукова платформа для кіно та телебачення.

Призначення програмного засобу полягає в наданні зручного інтерфейсу, що дозволяє користувачам швидко знайти фільм або серіал, який відповідає їхнім запитам. Веб-сайт має забезпечувати не лише функції пошуку, але й можливість ознайомлення з актуальними тенденціями в галузі кіно та телебачення.

Постановка задачі: у рамках кваліфікаційної роботи необхідно розробити веб-додаток для пошуку фільмів та серіалів за назвами, жанрами та іменами акторів. Користувачі повинні мати змогу переглядати популярні фільми та серіали, а також отримувати докладну інформацію про обраний контент, зокрема опис, рік випуску, рейтинг і трейлери.

Серед основних вимог до системи можна виділити такі:

- можливість пошуку фільмів та серіалів за назвами, жанрами та іменами акторів;
- представлення популярних фільмів та серіалів на головній сторінці, що оновлюються в залежності від актуальних рейтингів;

- перегляд трейлерів фільмів та серіалів без необхідності переходити на інші ресурси;
- можливість отримати детальну інформацію про фільм чи серіал, включаючи опис, рік випуску, рейтинг;
- зручний і зрозумілий інтерфейс, що дозволяє користувачам швидко орієнтуватися на сайті, не витрачаючи багато часу на пошук потрібного контенту;
- адаптація сайту для різних пристроїв, зокрема для мобільних телефонів, для забезпечення комфортного користування на всіх платформах.

Оскільки цей сайт надає лише інформацію для ознайомлення та не ставить метою залучення користувачів до авторизації або персоналізації, авторизація на сайті не передбачена. Тому доступ до всіх функцій сайту є відкритим і безкоштовним для всіх користувачів без потреби створювати облікові записи або входити в систему.

2.2. Вибір моделі розробки програмного засобу «Система пошуку фільмів і серіалів»

Для розробки веб-сайту для пошуку фільмів та серіалів було обрано каскадну модель розробки програмного забезпечення (Waterfall Model) [5]. Ця модель передбачає поетапне виконання робіт, де кожен етап базується на результатах попереднього. Каскадна модель є традиційним методом, що забезпечує чітку структуру і дозволяє зосередитись на кожному етапі окремо, знижуючи ймовірність виникнення помилок.

На першому етапі проведено аналіз вимог. Було визначено, що веб-сайт має здійснювати пошук фільмів і серіалів за назвою, жанром та іменами акторів. Також передбачено відображення популярних фільмів та серіалів, перегляд трейлерів та

надання детальної інформації про обраний контент. Цей етап завершено, і отримані результати були використані на наступних етапах розробки.

Наступним етапом є проєктування, на якому вибираються інструменти та технології для розробки веб-сайту. Це включає вибір стеку технологій для фронтенду і бекенду, розробку архітектури системи, створення макетів інтерфейсу та проєктування бази даних. Завданням цього етапу є визначення адаптивного дизайну для забезпечення коректної роботи на мобільних пристроях та створення планів для інтеграції API для отримання даних про фільми та серіали.

На етапі розробки буде реалізовано основний функціонал веб-сайту. Після цього буде проведено етап тестування, на якому перевірятимуться всі функції, зокрема пошук контенту, відображення популярних фільмів та серіалів, перегляд трейлерів, а також адаптивність сайту для мобільних пристроїв. Тестування також включатиме перевірку швидкості пошуку та точності результатів.

Каскадна модель дозволяє чітко відстежувати прогрес кожного етапу, забезпечуючи систематичний підхід до розробки, що знижує ризики на кожному етапі. Оскільки проєкт є навчальним, етапи впровадження та підтримки не передбачені, і завершується розробкою та тестуванням готового продукту.

2.3. Загальний опис проєкту «Система пошуку фільмів і серіалів»

Архітектура системи «Система пошуку фільмів і серіалів» побудована за клієнт-серверною моделлю, де клієнт здійснює взаємодію з сервером, а сервер, в свою чергу, виконує запити до віддаленої бази даних (API) для отримання необхідної інформації. Такий підхід дозволяє розподілити логіку пошуку та обробки даних, зменшуючи навантаження на клієнтську частину і підвищуючи зручність взаємодії з додатком.

Головною метою цієї архітектури є організація взаємодії між користувачем, сервером і віддаленою базою даних. Користувач вводить пошуковий запит, який

відправляється на сервер. Сервер здійснює запит до віддаленого API, отримує великий обсяг даних, виконує їх фільтрацію і передає лише необхідну інформацію клієнту. Це дозволяє мінімізувати кількість даних, які передаються на клієнтську частину, зменшуючи її обробку.

Процес взаємодії між компонентами системи виглядає таким чином: спочатку запит формує клієнт, який передає його на сервер, а сервер, своєю чергою, відправляє запит до віддаленої бази даних. Така схема необхідна для того, щоб клієнт отримував вже оброблені і фільтровані дані, що оптимізує роботу додатку.

Архітектура складається з кількох основних компонентів: клієнт, сервер та віддалена база даних (API). Клієнт відповідає за формування запиту на основі введених користувачем даних і відображення результатів. Сервер обробляє запити, фільтрує отриману інформацію та передає її клієнту. Віддалена база даних містить актуальні дані про фільми, серіали та акторів, які використовуються для пошуку.

Модель взаємодії між компонентами наведено на рисунку 2.1, де зображена схема зв’язків між користувачем, клієнтом, сервером та віддаленою базою даних.

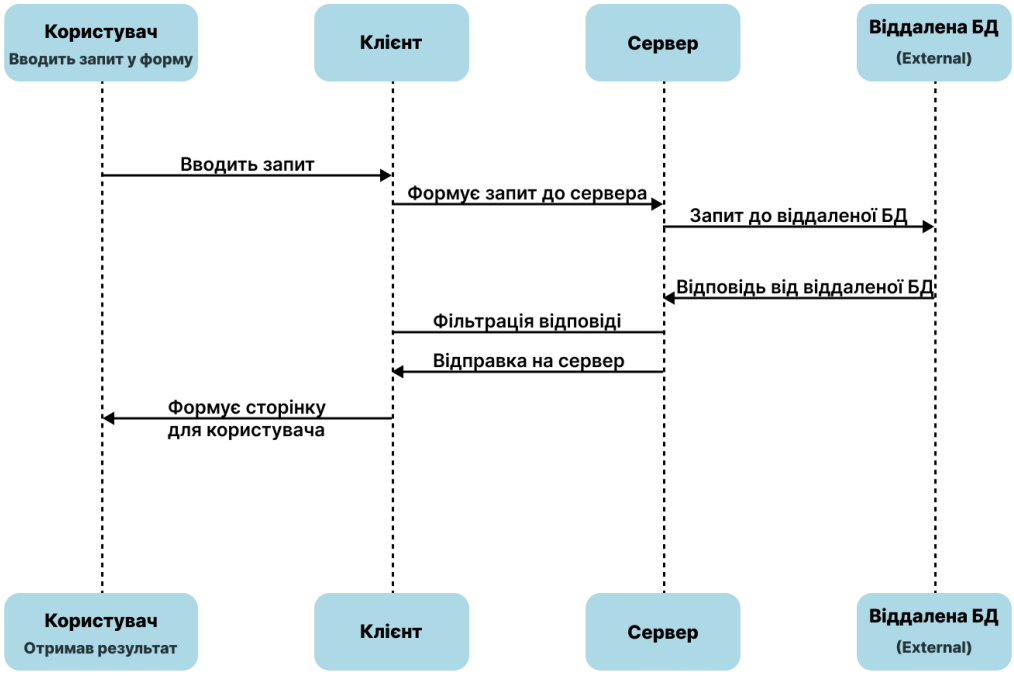

Рисунок 2.1 – Модель взаємодії між компонентами системи

Усі сторінки веб-додатку інтегровані в єдину навігаційну структуру, що забезпечує зручний доступ до різних функціональних частин сайту, дозволяючи користувачам швидко переміщатися між ними.

Схематично карта навігації сайту наведена на рисунку 2.2.

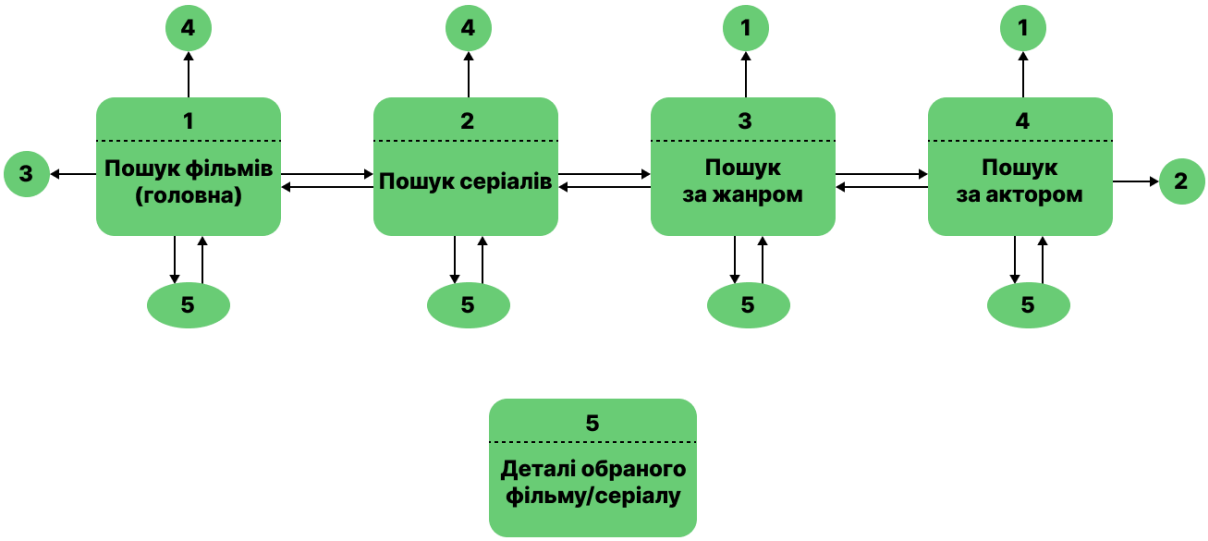

Рисунок 2.2 – Карта навігації

Структура та навігація сайту включають наступні компоненти.

Сторінка пошуку фільмів (головна) – на цій сторінці користувач відразу бачить популярні фільми. До введення запиту відображаються рекомендації та найбільш популярні фільми. Користувач має можливість здійснити пошук за фільмами, серіалами, жанром або актором.

Пошук серіалів – після натискання на посилання або кнопку для пошуку серіалів, користувач потрапляє на окрему сторінку, де доступні фільтри для пошуку серіалів за різними параметрами. Користувач може повернутися на головну сторінку або до інших розділів пошуку.

Пошук за жанром – ця сторінка дозволяє користувачу вибрати або ввести конкретний жанр для пошуку фільмів чи серіалів. Крім того, на сторінці є можливість повернутися на головну або перейти до інших варіантів пошуку.

Пошук за акторами – на цій сторінці користувач може ввести ім'я актора для перегляду списку фільмів або серіалів, у яких він брав участь. Як і на попередніх сторінках, є можливість повернутися на головну сторінку або здійснити інший тип пошуку.

Сторінка деталей фільму або серіалу – ця сторінка надає детальну інформацію про вибраний фільм або серіал, включаючи опис, рік випуску, трейлер та інші дані. На цій сторінці доступна лише кнопка «назад», що дозволяє користувачу повернутися до попередніх результатів пошуку.

Веб-сайт, що розробляється, має на меті забезпечити користувачам доступ до популярних акторів фільмів та серіалів. Оскільки база даних містить акторів різних національностей, для підвищення релевантності результатів пошуку для певної аудиторії необхідно здійснити фільтрацію акторів азіатської групи. Це зумовлено тим, що серед акторів цієї групи можуть зустрічатися менш відомі для англомовної аудиторії. Тому був реалізований механізм фільтрації акторів, що виключає тих, чиї імена мають азіатські символи або характерні риси в написанні.

Математична модель алгоритму пошуку популярних акторів включає два основні етапи: відбір акторів за рейтингом популярності та фільтрацію за національною ознакою. На першому етапі визначається популярність актора, що оцінюється за кількістю голосів, відданих за фільми та серіали, у яких він брав участь. Це дозволяє вибрати найбільш популярних акторів, враховуючи їхню активність серед глядачів. На другому етапі, після визначення найпопулярніших акторів, алгоритм використовує регулярний вираз для перевірки імен акторів та виключення тих, чия стилістика написання імені містить азіатські символи. Це дозволяє сформувати список акторів, найбільш релевантних для англомовної аудиторії.

Крім того, система застосовує модель для визначення популярності фільмів і серіалів, яка враховує як середній рейтинг, так і кількість голосів. Це допомагає

уникнути ситуацій, коли фільми з малою кількістю голосів отримують високий рейтинг. Описаний підхід закріплюється формулою 2.1.

$$R = \frac{(v \times r) + (m \times C)}{v + m} \quad (2.1)$$

де:

R – підсумковий рейтинг фільму,

v – кількість голосів за фільм,

r – середній рейтинг фільму,

m – мінімальна кількість голосів для потрапляння у вибірку,

C – середній рейтинг усіх фільмів у базі.

2.4. Обґрунтування вибору інструментальних засобів розробки «Система пошуку фільмів і серіалів»

Для реалізації клієнтської частини було вибрано фреймворк React. React JS [7] є відкритою бібліотекою JavaScript [3], що використовується для розробки інтерфейсів користувача. Вона була розроблена компанією Facebook [13] і здобула популярність серед розробників по всьому світу. React дозволяє створювати застосунки з високою продуктивністю та масштабованістю. Однією з основних концепцій React JS є компоненти, які є самостійними блоками коду, що відповідають за рендеринг конкретної частини інтерфейсу. Цей фреймворк призначений для розробки інтерфейсів веб-додатків. Основною метою React на клієнтській стороні є створення інтерактивних, динамічних та чутливих інтерфейсів для користувачів. Завдяки цьому можна реалізувати багатофункціональні та інтерактивні застосунки з швидким рендерингом і перемиканням між сторінками. Компонентний підхід дозволяє створювати модульний код, який легко

масштабувати і підтримувати. React забезпечує швидкий рендеринг через використання віртуального DOM [23].

Для серверної частини обрано середовище Node.js у поєднанні з фреймворком Express [25]. Node.js є середовищем виконання JavaScript, заснованим на рушії V8 від Google, що дозволяє виконувати JavaScript поза браузером. Його основною особливістю є подієво-орієнтована, неблокуюча модель введення-виведення (I/O), що дозволяє операціям, які звичайно займають багато часу (наприклад, доступ до баз даних або виконання HTTP-запитів) [18], виконуватись асинхронно, не блокуючи виконання інших частин коду. Це дозволяє Node.js обробляти велику кількість запитів одночасно, без необхідності створювати окремий процес для кожного запиту [11].

Асинхронність у Node.js досягається через застосування колбеків, промісів та механізму async/await. Коли сервер отримує запит, він виконує відповідний код, а довготривалі операції, такі як зчитування даних з бази даних, виконуються у фоновому режимі. Після завершення операції сервер обробляє результат за допомогою "зворотного виклику", не блокуючи обробку інших запитів, що можуть надійти під час виконання попередньої операції.

Фреймворк Express обраний для реалізації серверної логіки через свою простоту, гнучкість та широкий набір функціональних можливостей. Express є легким серверним фреймворком для Node.js, що дозволяє організувати маршрутизацію (визначення реакції сервера на різні HTTP-запити), обробляти запити та відповіді, а також працювати з середовищем, зокрема налаштовувати заголовки, аутентифікацію та логування.

HTTP-запити, що обробляються сервером, є основою для взаємодії між клієнтом та сервером. Клієнт надсилає HTTP-запит (наприклад, GET або POST), який сервер отримує через відповідний маршрут, визначений у Express.

Для організації взаємодії між клієнтською частиною додатку та сервером у розробленому веб-додатку використовується бібліотека Axios. Axios – це

популярна JavaScript-бібліотека [15], що надає зручний спосіб виконання HTTP-запитів у браузері або Node.js. Вона є проміс-орієнтованою, що спрощує роботу з асинхронними операціями та надає гнучкий інтерфейс для налаштування запитів та обробки відповідей.

Основною метою використання Axios є спрощення взаємодії між клієнтською частиною додатку і сервером. Бібліотека дозволяє виконувати стандартні HTTP-запити, такі як GET, POST, PUT, DELETE та інші. Axios надає зручний API для налаштування параметрів запиту та обробки відповідей, що дозволяє зменшити складність цих операцій.

Як джерело інформації про фільми, серіали та акторів було вибрано віддалену базу даних TMDB [16]. Її API надає доступ до актуальних даних та містить детальну документацію для інтеграції. API TMDB реалізовано за принципом REST, що дозволяє здійснювати запити через стандартні HTTP-методи, такі як GET, для отримання різноманітної інформації про контент. Веб-сервіс забезпечує зручний доступ до даних про фільми, серіали, акторів, рейтинги та жанри. Для взаємодії з API необхідний унікальний API-ключ, який дозволяє здійснювати запити та отримувати дані, забезпечуючи захист від несанкціонованого доступу. Використання TMDB [20] забезпечує швидке та зручне отримання необхідної інформації, що робить його підходящим джерелом для інтеграції в веб-додаток.

Оскільки TMDB не надає трейлерів до фільмів і серіалів, для їх пошуку було обрано API YouTube [14]. YouTube API також базується на принципах REST і дозволяє виконувати запити для пошуку відео, зокрема офіційних трейлерів, за допомогою ключових слів. Алгоритм пошуку додає до назви фільму або серіалу запит "official trailer", що дозволяє більш точно знаходити відповідні відео серед великої кількості контенту на YouTube. Пошукова система YouTube API зазвичай надає найпопулярніші відео у відповіді на запит, де перший результат часто є необхідним трейлером. У відповіді від YouTube API включаються метадані відео,

такі як заголовок, опис та кількість переглядів, що допомагає підтвердити релевантність знайденого контенту.

2.5. Етапи реалізації програмного засобу «Система пошуку фільмів і серіалів»

Для початку роботи над проєктом необхідно вибрати каталог для зберігання коду. Далі в цьому каталозі ініціалізується новий проєкт за допомогою команди npm init. Це створює файл package.json, що містить інформацію про залежності та налаштування проєкту.

Після цього можна редагувати або видаляти файли, створені за замовчуванням, залежно від вимог. Наприклад, у новому проєкті створюється файл index.js, який є початковим файлом для реалізації клієнтської частини застосунку.

У файлі index.js відбувається ініціалізація та рендеринг основного компонента React – App. Цей файл виконує підключення React до HTML-документу і рендерить логіку застосунку. Код цього файлу наведено в лістингу 2.1.

Лістинг 2.1 – index.js

```
// Імпортуємо необхідні бібліотеки: React та ReactDOM
import React from 'react';
import ReactDOM from 'react-dom/client';

// Імпортуємо основний компонент додатку
import App from './App';

// Створюємо корінь React додатку, до якого буде прив'язаний наш компонент
const root = ReactDOM.createRoot(document.getElementById('root'));
```

```
// Рендеримо компонент App в корінь документа
root.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>
);
```

У цьому коді спочатку імпортуються бібліотеки React та ReactDOM. React використовується для створення компонентів, тоді як ReactDOM відповідає за взаємодію з DOM, тобто за додавання компонентів у HTML-документ. Далі імпортується компонент App, який є основною частиною застосунку. Створюється корінь додатку за допомогою методу ReactDOM.createRoot(). Цей метод прив'язує додаток до елемента з id="root" в HTML-файлі (цей елемент повинен бути визначений у index.html). Для рендерингу компоненту App використовується метод root.render(). Теги <React.StrictMode> активують режим, який допомагає виявляти потенційні проблеми в застосунку.

Після ініціалізації проєкту файл index.html автоматично створюється в директорії public. Оскільки цей файл містить лише базову структуру HTML-документа без значних змін для нашого додатку, детально його описувати не має необхідності. Ми зосереджуємось на подальших етапах розробки, де буде реалізована логіка функціональності додатку.

Наступним етапом є робота з файлом App.js, який є основним компонентом для організації маршрутизації застосунку. У цьому файлі визначається, які компоненти будуть рендеритися на різних сторінках залежно від вибору користувача, тобто від URL-адреси. Він відповідає за підключення різних частин інтерфейсу до відповідних маршрутів. Код файлу App.js наведено в лістингу 2.2.

Лістинг 2.2 – App.js

```
    // Імпортуємо необхідні компоненти з бібліотеки react-router-dom для маршрутизації
    import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';

    // Імпортуємо компоненти сторінок
    import Home from './pages/Home';
    import HomeTV from './pages/HomeTv';
    import MovieDetails from './pages/MovieDetails';
    import TVDetails from './pages/TVDetails';
    import SearchByGenre from './pages/SearchByGenre';
    import SearchByActors from './pages/SearchByActors';

    function App() {
      return (
        // Обгортаємо всю маршрутизацію в компонент Router
        <Router>
          {/* Визначаємо всі маршрути, які відповідають певним шляхам */}
          <Routes>
            {/* Основна сторінка для фільмів */}
            <Route path="/" element={<Home />} />
            {/* Сторінка з деталями фільму за ID */}
            <Route path="/movie/:id" element={<MovieDetails />} />
            {/* Сторінка для телевізійних шоу */}
            <Route path="/tv" element={<HomeTV />} />
            {/* Сторінка з деталями телевізійного шоу за ID */}
            <Route path="/tv/:id" element={<TVDetails />} />
            {/* Сторінка для пошуку за жанром */}
            <Route  path="/search-by-genre"  element={<SearchByGenre />} />
```

```
            {/* Сторінка для пошуку за актором */}
            <Route path="/search-by-actors" element={<SearchByActors
/>} />
          </Routes>
        </Router>
      );
    }

    export default App;
```

У цьому коді використовується бібліотека react-router-dom для організації маршрутизації в додатку. Компонент Router обгортає всю маршрутизацію, тоді як Routes містить перелік усіх маршрутів, що використовуються в додатку. Кожен маршрут визначає шлях (наприклад, /, /movie/:id) та компонент, який буде відображатися за цим шляхом (наприклад, Home, MovieDetails).

Згідно зі структурою маршрутизації, потрібно організувати структуру папок та файлів у проєкті. Структура має бути такою, щоб кожен з компонентів сторінок, як-от Home, HomeTV, MovieDetails, був розміщений у відповідних файлах у папці pages. Така організація сприяє розділенню коду для кожної сторінки, що полегшує підтримку та масштабування проєкту в майбутньому.

Розглянемо реалізацію головної сторінки Home.js, яка є основною сторінкою додатку. Інші сторінки будуть мати схожу структуру з головною, але з урахуванням специфічних запитів до API для кожної конкретної сторінки. Тобто структура кожної сторінки базуватиметься на загальних підходах, описаних у компоненті Home.js, а основні відмінності будуть полягати в налаштуванні конкретних запитів та обробці отриманих даних.

Спочатку налаштовується структура сторінки шляхом імпорту необхідних компонентів і бібліотек, що забезпечують можливість виконання запитів і роботи з даними. Початковий код для файлу Home.js наведено на лістингу 2.3.

Лістинг 2.3 – Початковий код для файлу Home.js

```
    import React, { useEffect, useState } from 'react'; // Імпортуємо React та хуки useEffect і useState для роботи зі станом і виконанням побічних ефектів
    import axios from 'axios'; // Імпортуємо axios для виконання HTTP-запитів
    import { Link } from 'react-router-dom'; // Імпортуємо Link для створення навігаційних посилань між сторінками
    import Header from '../Header'; // Імпортуємо компонент Header для відображення заголовка сторінки
    import '../styles/Home.css'; // Імпортуємо CSS-стилі для цієї сторінки

    const Home = () => {
      // Тіло компонента
    };

    export default Home;
```

Далі реалізується функціональність для налаштування стани для зберігання даних, а також створення функції для виконання запитів до сервера з метою отримання фільмів. Це забезпечує динамічне завантаження фільмів на сторінку. Стани зберігають список фільмів та пошуковий запит користувача. Запити до сервера надсилаються залежно від наявності активного пошукового запиту. Код, який реалізує цю функціональність, наведено в лістингу 2.4.

Лістинг 2.4 – Налаштування станів та запитів сторінки Home.js

```
      // Стани для зберігання фільмів та пошукового запиту
      const [movies, setMovies] = useState([]);
      const [searchQuery, setSearchQuery] = useState('');
```

```
        // Функція для отримання фільмів з сервера
        const fetchMovies = async (query = '') => {
          try {
            // Якщо є запит на пошук, формуємо відповідний URL для
запиту
            const endpoint = query
              ?                    `http://localhost:5000/api/search-
movies?query=${query}`  // Запит для пошуку
              : 'http://localhost:5000/api/popular-movies';   // Запит
для отримання популярних фільмів
            // Виконуємо запит
            const response = await axios.get(endpoint);
            // Якщо є запит, оновлюємо фільми відповідно до результатів
пошуку
            setMovies(query ? response.data : response.data.results);
          } catch (error) {
            console.error('Error loading movies', error);  // Обробка
помилок при запиті
          }
        };

        // Викликаємо fetchMovies для отримання фільмів при першому
рендері
        useEffect(() => {
          fetchMovies();
        }, []);

        // Обробка введення в поле пошуку
        const handleSearch = (e) => {
          const value = e.target.value;
          setSearchQuery(value);  // Оновлюємо стан з пошуковим запитом
          fetchMovies(value);  // Виконуємо запит для пошуку фільмів
        };
```

У цьому коді використано хук useState для створення двох станів: movies, який зберігає дані фільмів, та searchQuery, що фіксує пошуковий запит користувача. По-друге, функція fetchMovies виконує запити до сервера, в залежності від наявності пошукового запиту. Якщо запит наявний, він передається до API для пошуку фільмів, а у разі його відсутності – отримуються популярні фільми.

Застосування хука useEffect забезпечує виклик функції fetchMovies під час першого рендеру компонента, що дозволяє автоматично завантажити популярні фільми без додаткових дій з боку користувача. Введення користувача обробляється через функцію handleSearch, яка оновлює стан searchQuery і викликає fetchMovies для пошуку фільмів за новим запитом.

Наступним етапом є визначення того, що компонент Home має повертати після налаштування станів і запитів. Після отримання даних про фільми, ці дані потрібно відобразити на сторінці. У головному компоненті розміщуються такі елементи, як шапка, поле для пошуку фільмів та сітка карток фільмів. Кожна картка містить постер фільму, його заголовок і є посиланням на сторінку з детальнішою інформацією про фільм. Код для відображення цих елементів наведено в лістингу 2.5.

Лістинг 2.5 – Відображення елементів сторінки Home.js

```
    return (
      <div className="home-container">
        <Header /> {/* шапка */}
        <div className="content">
          <h2 className="header-title">Movies</h2>
          <input
            type="text"
            className="search-input"
            placeholder="Search movies..."
            value={searchQuery}
```

```
              onChange={handleSearch}
            />
            <div className="movies-grid">
              {movies.map((movie) => (
                <Link    to={`/movie/${movie.id}`}    key={movie.id}
className="movie-card">
                  <img

src={`https://image.tmdb.org/t/p/w500${movie.poster_path}`}
                    alt={movie.title}
                    className="movie-poster-card"
                  />
                  <h3 className="movie-title-card">
                    {movie.title}
                  </h3>
                </Link>
              ))}
            </div>
          </div>
          <Footer />
        </div>
      );
```

У даному коді використовується контейнер home-container, в якому розміщуються всі основні елементи сторінки. Спочатку додається компонент Header, який відповідає за верхню частину сторінки. Далі йде блок контенту, що містить заголовок "Movies", поле для введення запиту на пошук і сітку фільмів.

Поле для пошуку реалізоване через елемент <input>, де користувач може вводити запит. Кожен введений символ оновлює стан пошукового запиту, що спричиняє новий запит до API і завантаження відповідних фільмів. Запити виконуються через функцію handleSearch, яка була описана раніше.

Однією з основних частин є відображення фільмів. Для кожного елемента масиву movies створюється картка з постером і заголовком. Для отримання зображень постерів фільмів використовуються URL-адреси стандартного формату https://image.tmdb.org/t/p/w500, які дозволяють отримати постери фільмів з TMDB у розмірі 500 пікселів по ширині.

На поточному етапі сторінка може повертати помилку, оскільки запити до сервера та обробка відповідей не реалізовані. Сторінка Home.js робить два запити: один – для пошуку фільмів за назвою через http://localhost:5000/api/search-movies?query=, а інший – для отримання популярних фільмів через http://localhost:5000/api/popular-movies. Далі розглянемо реалізацію цієї взаємодії на сервері.

Для початку необхідно створити файл server.js, в якому налаштовується сервер на основі Node.js, використовуючи Express, Axios для виконання запитів до TMDB, а також CORS для дозволу на перехресні запити. Початкове налаштування сервера наведено на лістингу 2.6.

Лістинг 2.6 – Початкове налаштування сервера

```
    const express = require('express');  // Імпортуємо Express для створення сервера
    const axios = require('axios');  // Імпортуємо Axios для виконання запитів
    const cors = require('cors');  // Імпортуємо CORS для дозволу перехресних запитів

    const app = express();  // Ініціалізуємо додаток Express
    const port = 5000;  // Встановлюємо порт для сервера

    app.use(cors());  // Дозволяємо перехресні запити
```

Тепер розглянемо реалізацію маршруту /api/popular-movies, який надає список популярних фільмів. Цей маршрут використовує API TMDB для отримання актуальної інформації про популярні фільми. Код цього маршруту наведено на лістингу 2.7.

Лістинг 2.7 – Маршрут /api/popular-movies на сервері server.js

```
    app.get('/api/popular-movies', async (req, res) => {
      try {
        // Виконуємо запит до API TMDB для отримання популярних фільмів
        const response = await axios.get(`${TMDB_API_URL}/movie/popular?api_key=${TMDB_API_KEY}&language=en-EN&page=1`);

        // Відправляємо дані, отримані від API, клієнту
        res.json(response.data);
      } catch (error) {
        // Якщо сталася помилка, виводимо її в консоль і відправляємо помилку клієнту
        console.error('Помилка при отриманні популярних фільмів:', error.message);
        res.status(500).send('Помилка при отриманні популярних фільмів');
      }
    });
```

Маршрут /api/popular-movies обробляє GET-запит, що дозволяє клієнту отримувати список популярних фільмів. Запит до TMDB API здійснюється за допомогою методу axios.get(), де передаються такі параметри: api_key=${TMDB_API_KEY} (унікальний ключ для доступу до TMDB, який

отримується після реєстрації на сайті TMDB і створення проекту), language=en-EN (параметр, що вказує на отримання даних англійською мовою), та page=1 (параметр, що обмежує запит лише першою сторінкою результатів). Для пагінації доступні інші параметри, однак на цьому етапі обмежуємося лише першою сторінкою.

При успішному виконанні запиту відповідь від TMDB API містить дані про фільми, такі як назви, описи, рейтинги тощо. Ці дані повертаються клієнту у форматі JSON через метод res.json(response.data).

У разі виникнення помилки при запиті до TMDB API вона перехоплюється за допомогою блоку catch. У разі помилки на сервері клієнт отримує статус 500 і повідомлення про помилку.

Далі розглянемо реалізацію другого маршруту /api/search-movies, який обробляє запити на пошук фільмів за їхньою назвою. Користувач вводить пошуковий запит, і цей маршрут повертає результати з TMDB, відсортовані за кількістю голосів. Код для реалізації цього маршруту наведено на лістингу 2.8.

Лістинг 2.8 – Маршрут /api/ search-movies на сервері server.js

```
    app.get('/api/search-movies', async (req, res) => {
      const query = req.query.query;  // Отримуємо параметр 'query' з запиту
      try {
        // Виконуємо запит до API TMDB для пошуку фільмів за назвою
        const response = await axios.get(`${TMDB_API_URL}/search/movie?api_key=${TMDB_API_KEY}&query=${query}`);

        // Сортуємо отримані фільми за кількістю голосів, від найбільш голосованих до найменш голосованих
        const sortedMovies = response.data.results.sort((a, b) => b.vote_count - a.vote_count);
```

```
        // Відправляємо відсортовані фільми як JSON відповідь
        res.json(sortedMovies);
      } catch (error) {
        // Якщо сталася помилка, відправляємо клієнту повідомлення
про помилку
        res.status(500).send('Помилка при пошуку фільму');
      }
    });
```

Маршрут /api/search-movies обробляє запити на пошук фільмів за їх назвою. Клієнт надсилає запит, передаючи параметр query, що містить назву фільму для пошуку. Параметр отримується через req.query.query, що дозволяє здійснювати динамічний пошук фільмів за різними назвами, залежно від значення цього параметра.

Для виконання запиту до TMDB API використовуємо метод axios.get(), формуючи URL, який включає параметри api_key (унікальний ключ для доступу до API) і query (назва фільму, який шукається). Після отримання даних виконується сортування фільмів за кількістю голосів. Для цього застосовуємо метод sort(), що упорядковує масив фільмів у порядку спадання голосів, таким чином, на початку списку знаходяться фільми з найбільшими оцінками.

Після сортування даних вони відправляються клієнту у форматі JSON через метод res.json(). У разі помилки під час запиту до TMDB API, вона перехоплюється блоком catch. У такому випадку сервер повертає відповідь зі статусом 500 та повідомленням про помилку.

Завершенням реалізації є створення сторінки для відображення деталей обраного фільму чи серіалу. Ця сторінка, названа MovieDetails.js, має структуру, подібну до Home.js, але з іншими стилями, описаними в додатку А. Вона також включає функціональність для отримання та відображення трейлерів. Для цього використовуємо спеціальний URL для відео на YouTube, який генерується на основі

відповіді від API YouTube. Процес формування цього URL описано в методі setTrailerUrl, наведеному на лістингу 2.9.

Лістинг 2.9 – Отримання трейлеру на клієнті

```
       if (response.data?.title) {
         const trailerResponse = await
axios.get(`http://localhost:5000/api/search-
trailer?title=${response.data.title} official trailer`);
         if (trailerResponse.data?.videoId) {

setTrailerUrl(`https://www.youtube.com/embed/${trailerResponse.data.v
ideoId}`);
         }
       }
```

Цей URL надає можливість вбудовувати трейлер фільму безпосередньо на сторінці, що значно покращує взаємодію користувача з додатком. Використання API для отримання трейлерів дозволяє динамічно завантажувати відео трейлери для кожного обраного фільму, що підвищує інтерес користувачів та зручність взаємодії з додатком.

Тепер розглянемо реалізацію маршруту на сервері для пошуку трейлерів фільмів. Маршрут /api/search-trailer обробляє GET-запит, який дозволяє отримати трейлер за назвою фільму. Клієнт передає параметр title, що містить назву фільму, і сервер виконує запит до API YouTube для пошуку відео, яке містить фразу "official trailer" разом з назвою фільму. Після цього отримуємо ID відео і відправляємо його назад клієнту. Код обробки запиту наведений на лістингу 2.10.

Лістинг 2.10 – Отримання трейлеру на сервері

```
    app.get('/api/search-trailer', async (req, res) => {
```

```
      const movieTitle = req.query.title;
      try {
        const response = await
axios.get(`${YT_API_URL}?key=${YT_API_KEY}&q=${movieTitle} official
trailer&part=snippet&type=video`);
        if (response.data.items.length > 0) {
          const videoId = response.data.items[0].id.videoId;
          res.json({ videoId });
        } else {
          res.status(404).send('Трейлер не знайдено');
        }
      } catch (error) {
        res.status(500).send('Помилка при пошуку трейлера');
      }
    });
```

Цей код виконує наступні дії:

- він отримує назву фільму через req.query.title, що дозволяє динамічно шукати трейлер по назві;
- використовує метод axios.get(), щоб зробити запит до YouTube API, де вказуються параметри: API-ключ (YT_API_KEY), запит на пошук фільму та тип результатів video;
- якщо відповідь містить результати, сервер бере ID відео з першого елемента в списку і відправляє його клієнту у вигляді об'єкта { videoId };
- якщо трейлер не знайдено, сервер відправляє відповідь з помилкою 404: "Трейлер не знайдено";
- якщо сталася помилка при виконанні запиту до YouTube API, сервер відправляє статус 500 і повідомлення про помилку.

2.5.1 Інструкція користувачу

Програма є веб-застосунком для пошуку та відображення фільмів і серіалів за допомогою відкритих API (TMDB для фільмів та YouTube для трейлерів). Назва програми: «MovieFinder». Це додаток, що дозволяє користувачам здійснювати пошук фільмів і серіалів, переглядати популярні фільми та їх деталі, а також дивитися трейлери через YouTube.

Ця програма дозволяє користувачам:

- шукати фільми за назвою;
- переглядати популярні фільми та серіали;
- отримувати деталі про фільм чи серіал, зокрема трейлери;
- використовувати пошук за жанром або актором.

Для запуску програми необхідно виконати наступні вимоги:

- Тип ЕОМ: ПК або ноутбук, що підтримує сучасні браузери.
- Операційна система: Windows, macOS, Linux.
- Обсяг оперативної пам’яті: Мінімум 4 ГБ оперативної пам'яті.
- Програмне забезпечення: Для запуску програми потрібен браузер, що підтримує JavaScript та React (наприклад, Google Chrome, Mozilla Firefox, Safari). Крім того, потрібно мати Node.js для запуску локального серверу та npm для управління залежностями.
- Периферійні пристрої: Мишка, клавіатура. Для перегляду відео – підтримка відео в браузері.

У програмі можуть виникати кілька типів помилок. Ось деякі з них:

- Помилка з’єднання з сервером: Якщо програма не може підключитися до серверу або API (наприклад, при відсутності інтернет-з’єднання), відображається повідомлення "Помилка з’єднання з сервером. Перевірте своє підключення до Інтернету."

- Помилка пошуку фільму: Якщо при запиті на сервер сталася помилка, користувач отримає повідомлення "Помилка при пошуку фільму. Спробуйте ще раз пізніше."
- Не знайдено трейлера: Якщо для обраного фільму не знайдено трейлера на YouTube, користувач побачить повідомлення "Трейлер не знайдено."

Користувач взаємодіє з програмою через веб-інтерфейс. При першому відкритті сторінки відображаються популярні фільми. Для пошуку фільмів за назвою користувач вводить назву фільму в поле пошуку. Програма запитує API, отримує список фільмів і відображає їх на екрані. Для перегляду деталей фільму або серіалу, користувач може клікнути на картку фільму, після чого відобразиться більше інформації, включаючи трейлер, якщо він є. Для більш точного пошуку можна використовувати додатковий пошук (за жанром або актором).

2.6. Організація тестування та налагодження програмного засобу «Система пошуку фільмів і серіалів»

Налагодження програмного забезпечення здійснювалось паралельно з процесом розробки за допомогою логування та дебагінгу. Це сприяло швидкому виявленню та усуненню помилок на різних етапах створення програми. Логування використовувалось для моніторингу запитів до API та перевірки отриманих результатів. Для більш детального аналізу та тестування функціональності сервера були створені модульні тести, що дозволили перевірити правильність роботи основних маршрутів API.

Для тестування серверної частини було реалізовано тестування за допомогою фреймворків Mocha і Chai [19]. Цей тест перевіряє два основні маршрути API:

- /api/popular-movies – для отримання популярних фільмів;
- /api/search-movies?query=<назва> – для пошуку фільмів за назвою;

Для інтеграційного тестування застосовувалась бібліотека Supertest, яка дозволяє виконувати запити до сервера та перевіряти відповіді на ці запити. Текст тесту наведено на лістингу 2.11.

Лістинг 2.11 – Тест searchMovies.test.js

```
import request from 'supertest';
import { expect as _expect } from 'chai';
const expect = _expect;

describe('Movie API', () => {
  it('should return a list of popular movies', async () => {
    const response = await request('http://localhost:5000')
      .get('/api/popular-movies')
      .expect(200);


expect(response.body).to.have.property('results').that.is.an('array')
;
  });

  it('should search for movies by title', async () => {
    const query = 'Inception';
    const response = await request('http://localhost:5000')
      .get(`/api/search-movies?query=${query}`)
      .expect(200);

    expect(response.body).to.be.an('array');
    expect(response.body[0].title).to.include(query);
  });
});
```

У результаті виконання цього тесту було отримано результати, які наведено на рисунку 2.3.

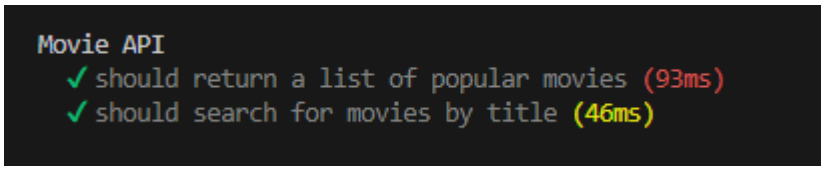

Рисунок 2.3 – Результат тестування запитів

Як видно з результатів, обидва тести були успішно пройдені, що свідчить про правильність роботи основних методів API для отримання популярних фільмів і пошуку фільмів за назвою.

Результатом розробки, налаштування та тестування запитів є система пошуку фільмів і серіалів, яка дає можливість користувачам отримувати інформацію про популярні фільми, здійснювати пошук фільмів за назвою, а також переглядати деталі обраного фільму, включаючи трейлер.

Після завершення реалізації та тестування програми була створена початкова сторінка з популярними фільмами. Ця сторінка є основним інтерфейсом для користувача та відображає список фільмів, які мають найвищу популярність. Всі дані на сторінці отримуються динамічно за допомогою API TMDB. Початкову сторінку з популярними фільмами можна побачити на рисунку 2.4.

На цій сторінці реалізовано поле для пошуку фільму за назвою, яке є основною функцією системи. Це поле дає змогу користувачу вводити назву фільму, а програма динамічно уточнює результати пошуку в реальному часі.

Результат пошуку фільму відображено на рисунках 2.5 та 2.6. На них видно, як програма показує фільми, що відповідають введеній назві, з постерами, заголовками та можливістю переходу до детальної сторінки вибраного фільму.

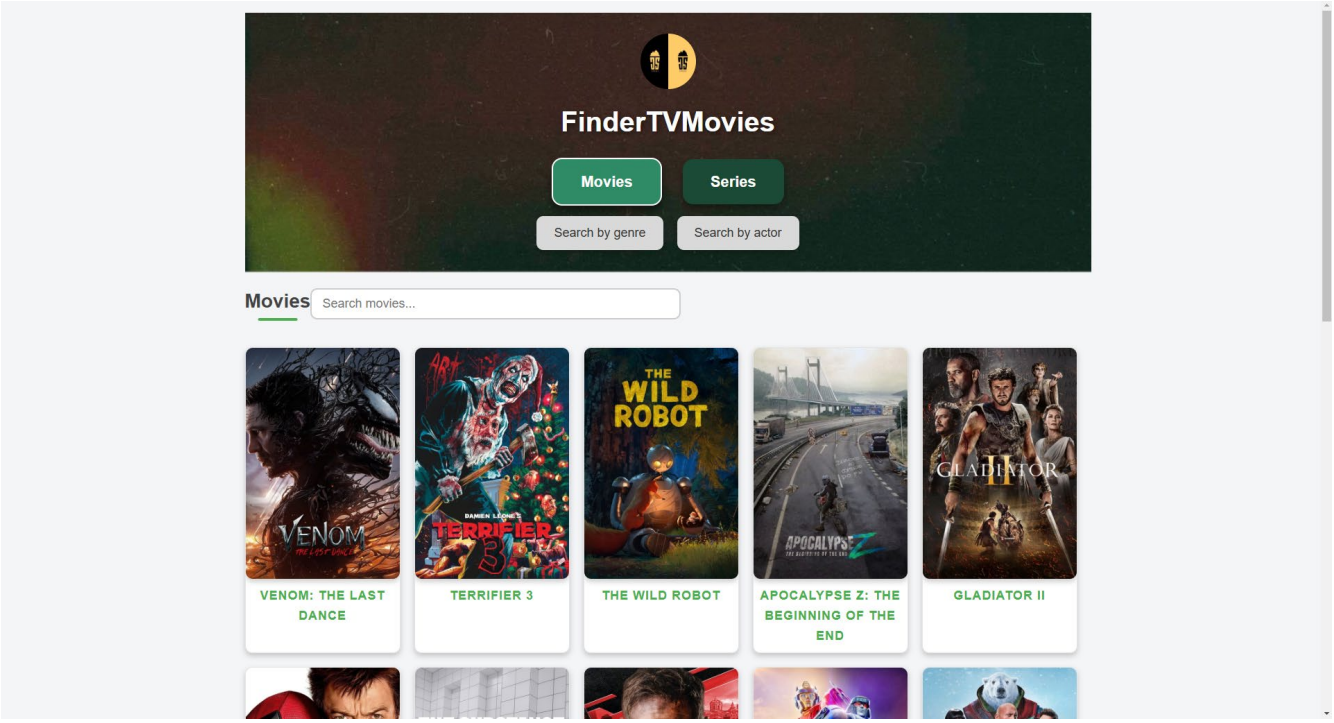

Рисунок 2.4 – Початкова сторінка сайту

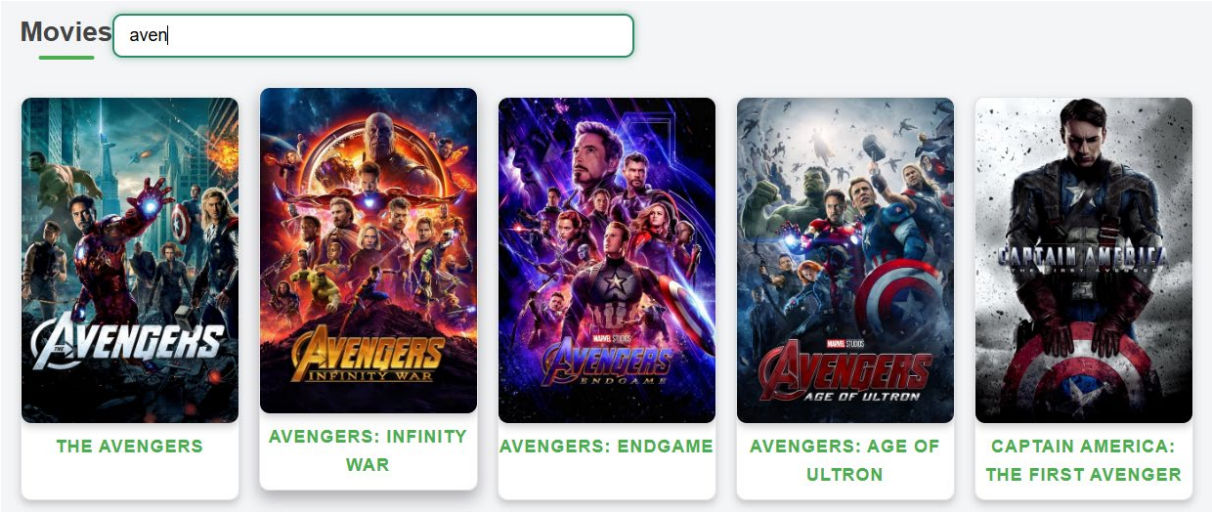

Рисунок 2.5 – Результат пошуку фільму

При кожному введенні нового символу система надсилає запит до серверної частини через маршрут /api/search-movies, що дозволяє відображати відповідні

фільми на основі введеного запиту. Такий підхід забезпечує інтерактивність користувацького досвіду, зменшуючи час на пошук і спрощуючи навігацію.

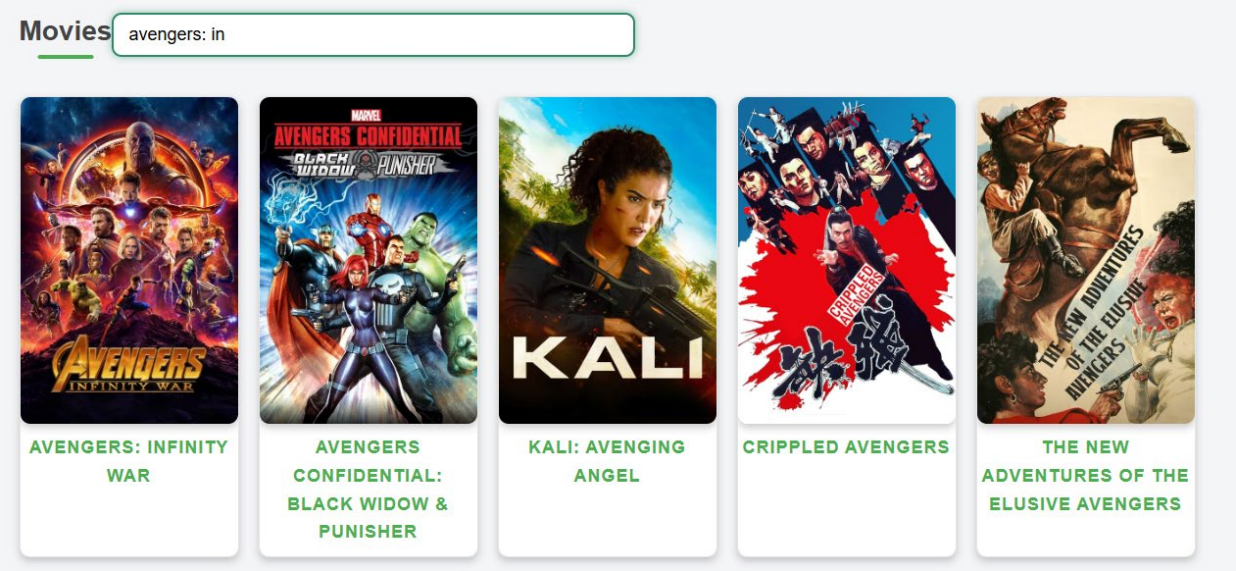

Рисунок 2.6 – Результат пошуку фільму з уточненням

Аналогічний функціонал реалізовано й для пошуку серіалів, рисунки 2.7 та 2.8.

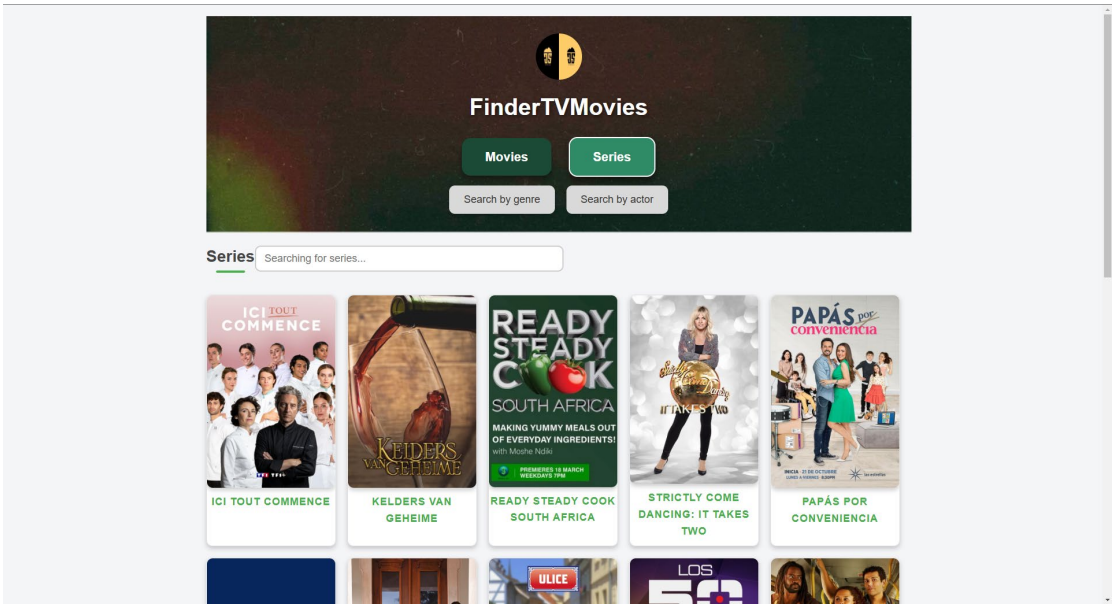

Рисунок 2.7 – Сторінка пошуку серіалів

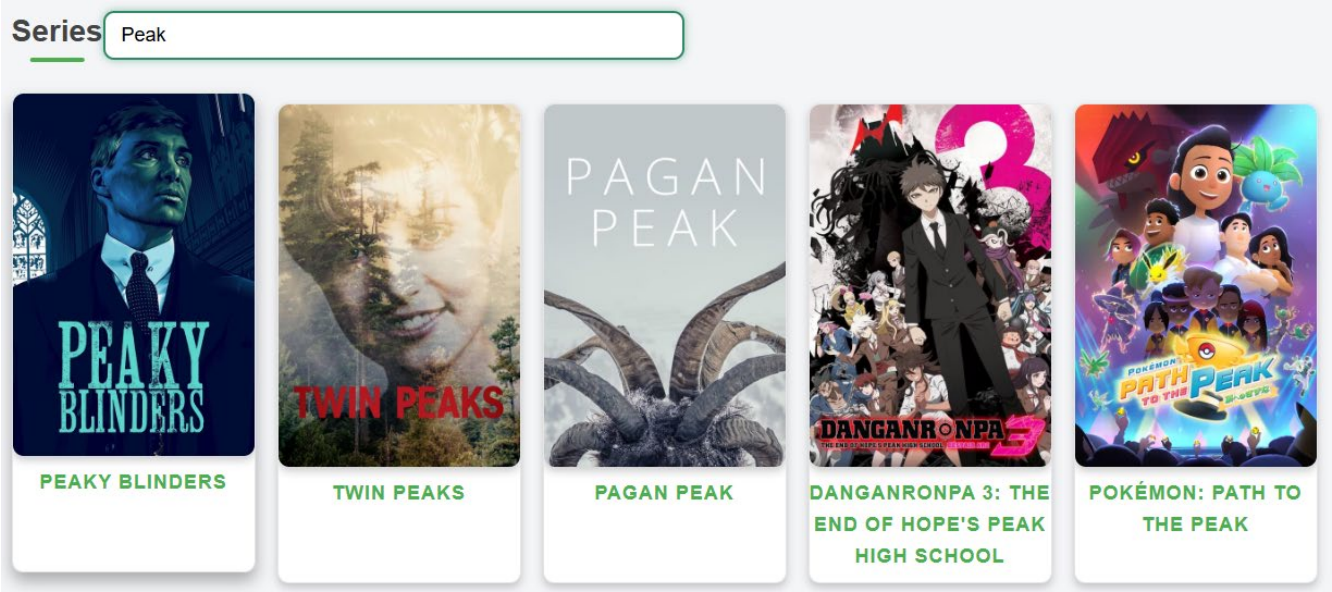

Рисунок 2.8 – Результат пошуку серіалів

Також реалізовано функціональність для пошуку фільмів за жанрами. На сайті є кнопка "Search by Genre", яка відкриває інтерфейс для вибору жанру.

Користувач може вибрати жанр із динамічно сформованого списку, що отримується за допомогою запиту до TMDB API через маршрут /api/genres. Після вибору жанру система надсилає запит до маршруту /api/movies-by-genre, передаючи обраний жанр як параметр. У результаті на екрані з'являється список популярних фільмів, що відповідають вибраному жанру, як показано на рисунку 2.9.

Крім того, система дозволяє здійснювати пошук фільмів за актором, що дає можливість знайти фільми, в яких бере участь певний актор. Ця функція реалізована через інтерфейс, де користувач може обрати актора зі списку або ввести його ім’я вручну. У будь-якому випадку сервер виконує пошук фільмів через маршрут /api/movies-by-actor і повертає відсортований список робіт актора, починаючи з найпопулярніших, як показано на рисунку 2.10.

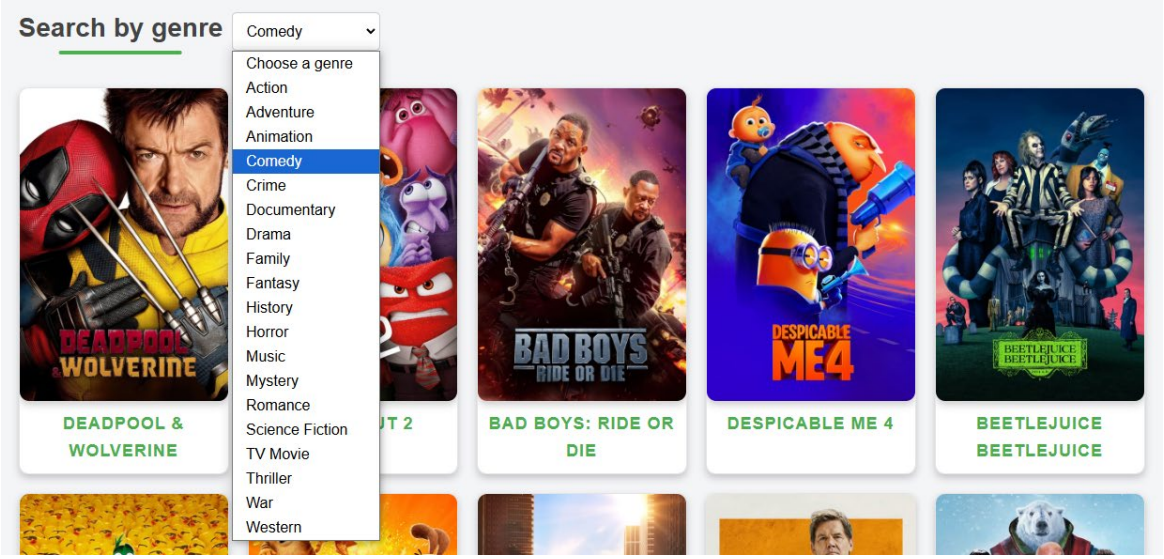

Рисунок 2.9 – Результат пошуку фільмів за жанром

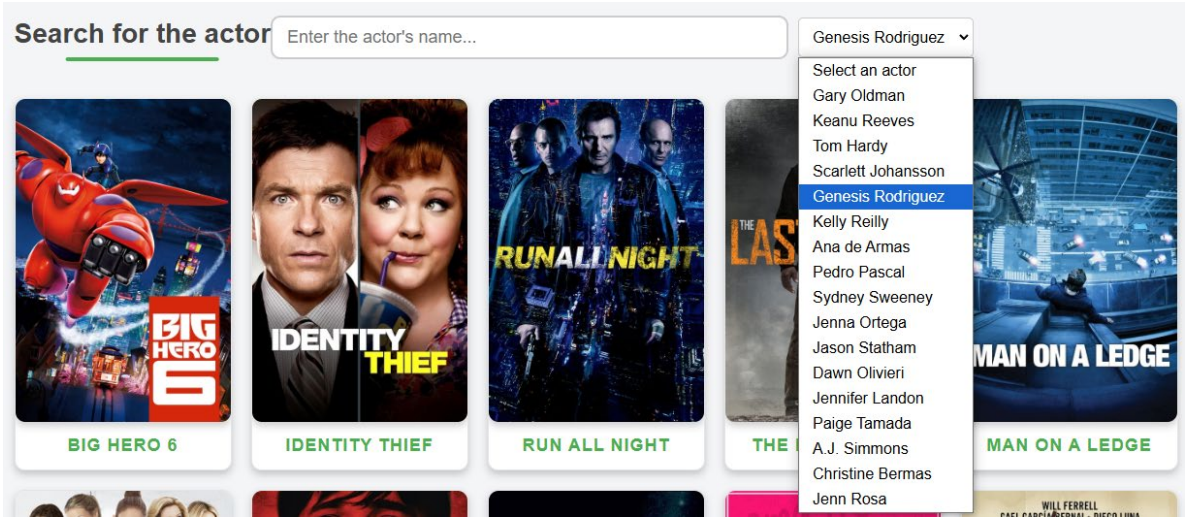

Рисунок 2.10 – Результат пошуку фільмів за актором

На цьому етапі було проведено тестування функціоналу надання детальної інформації про обраний фільм або серіал. Під час виконання пошуку користувач отримує результати у вигляді карток, кожна з яких містить зображення, назву та основні дані, що дозволяють швидко зорієнтуватися в результатах пошуку.

Після натискання на вибрану картку система автоматично перенаправляє користувача на сторінку з детальною інформацією про обраний фільм чи серіал. На

рисунку 2.11 відображено вигляд сторінки з деталями про фільм, де представлено всі основні елементи. Рисунок 2.12 демонструє приклад сторінки з інформацією про серіал, яка містить відповідні дані.

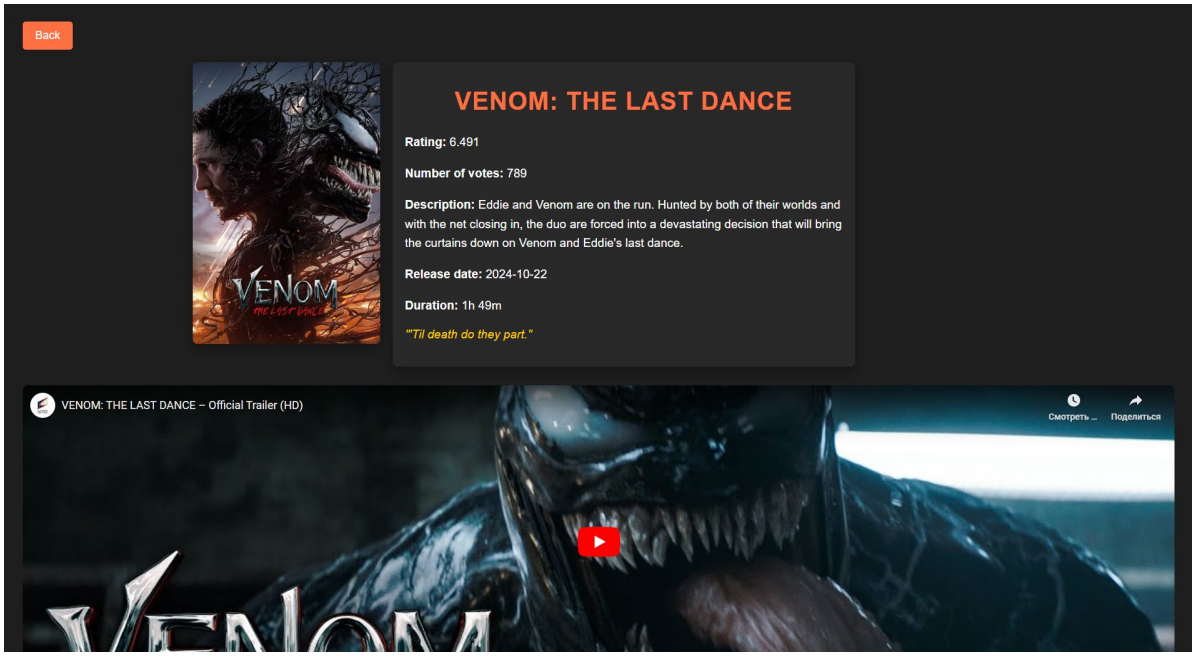

Рисунок 2.11 – Сторінка з додатковою інформацією про фільм

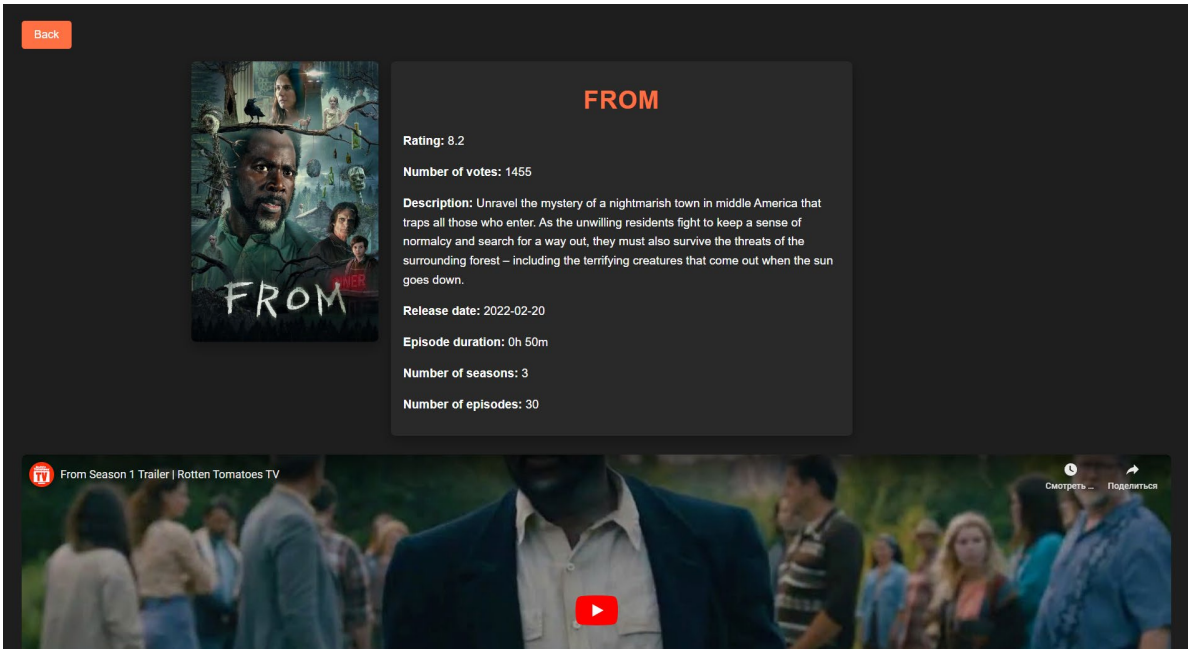

Рисунок 2.12 – Сторінка з додатковою інформацією про серіал

2.7. Рекомендації по використанню та впровадженню програмного засобу «Система пошуку фільмів і серіалів»

Програмне забезпечення «Система пошуку фільмів і серіалів» може бути застосоване як для особистого, так і для комерційного використання. Воно є корисним для користувачів, які шукають інформацію про фільми та серіали, а також для операторів медіа-платформ, що прагнуть інтегрувати можливість пошуку популярного контенту через API TMDB. Завдяки функціоналу пошуку за жанрами, акторами та назвами, система охоплює різноманітні запити користувачів.

Програмний засіб реалізовано як веб-додаток, що вимагає доступу до Інтернету та підтримки актуальних веб-браузерів. Рекомендовані версії браузерів: Google Chrome (версія 90 і вище), Mozilla Firefox (версія 88 і вище), Safari (версія 13 і вище). Для належної роботи сервера необхідно використовувати Node.js версії 14.0 або новішої, а також наявність підключення до мережі для здійснення запитів до API TMDB та YouTube.

Для клієнтської частини достатньо пристрою з мінімальними характеристиками, такими як:

- Процесор: 1 ГГц або швидший;
- Оперативна пам'ять: 2 ГБ;
- Дисплей: роздільна здатність 1280×720 пікселів.

Для серверної частини рекомендується:

- Процесор: 2 ГГц або швидший;
- Оперативна пам'ять: 4 ГБ;
- Накопичувач: 20 ГБ вільного простору.

Для запуску клієнтської частини необхідно зайти в кореневу директорію проєкту, де знаходиться файл package.json, виконати команду npm install для встановлення всіх залежностей, а потім запустити клієнт за допомогою команди npm start. Серверна частина запускається окремо, для цього в іншій консолі

потрібно з кореневої директорії перейти до папки src і виконати команду node server.js. Інтеграція між клієнтською та серверною частинами потребує налаштування базового URL сервера у файлі конфігурації клієнта. Для повноцінної роботи додатку необхідно отримати API-ключі для TMDB та YouTube і редагувати їх у файлі server.js.

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено веб-додаток для пошуку фільмів і серіалів, який дає змогу користувачам знаходити контент за різними параметрами, зокрема за назвою, акторами та жанром. Веб-сайт пропонує зручний інтерфейс, що дозволяє переглядати популярні фільми, здійснювати пошук за конкретними критеріями, а також отримувати детальну інформацію про фільми, серіали та їхні трейлери.

Для розробки додатку використано сучасні веб-технології. Клієнтська частина реалізована на основі React, що забезпечує створення динамічних та інтерактивних компонентів для пошуку та перегляду фільмів. Серверна частина побудована на Node.js, що гарантує стабільну обробку запитів і взаємодію з API для отримання даних про фільми з TMDB та трейлерів з YouTube.

Додаток інтегрується з API TMDB для отримання інформації про фільми, серіали та їхні параметри, такі як жанр, актори, рейтинг та інші. Для отримання трейлерів використовується API YouTube. Запити до обох API здійснюються через серверну частину додатку.

Тестування функціональності додатка проводилося за допомогою фреймворків Mocha та Supertest. У процесі тестування перевірялась робота запитів на отримання списку популярних фільмів та пошуку фільмів за назвою. Результати тестів підтвердили, що система коректно обробляє запити та надає правильні відповіді на них. Тест на перевірку отримання списку фільмів показав, що дані від API TMDB повертаються у правильному форматі, а пошук за назвою фільму здійснюється коректно.

Незважаючи на досягнуті результати, є кілька обмежень, що не дозволяють зробити додаток повністю функціональним у всіх аспектах. Наприклад, не вдалося реалізувати пошук за роком випуску чи країною виробництва, оскільки TMDB API не підтримує одночасний пошук за кількома параметрами, такими як актори, назва

фільму та країна. Крім того, пошук за роком чи країною не має значного практичного застосування, оскільки більшість користувачів надають перевагу пошуку за жанром або конкретними акторами, а не за параметрами на кшталт року чи країни.

Попри наявні обмеження, проект демонструє добрі результати в пошуку фільмів та серіалів. Для подальшого розвитку додатка можливі кілька напрямів удосконалення. Одним із таких є інтеграція комерційних API, які надають розширені можливості, зокрема детальніші метадані та додаткові функції для пошуку. Також доцільно впровадити комбінований пошук за кількома критеріями одночасно, що дозволить здійснювати точніший пошук фільмів за різними параметрами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алгоритми і структури даних [Електронний ресурс] // ПНПУ. – 2012. – Режим доступу до ресурсу: http://elcat.pnpu.edu.ua/docs/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B8%20%D1%96%20%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B8%20%D0%B4%D0%B0%D0%BD%D0%B8%D1%85/lab2_search.html..

2. Барабуха М. Методи оптимізації використання пам’яті для узагальнених суфіксних масивів [Електронний ресурс] / Марія Барабуха // НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ» Факультет інформатики Кафедра мережних технологій факультету інформатики. – 2023. – Режим доступу до ресурсу: https://ekmair.ukma.edu.ua/server/api/core/bitstreams/453b1917-5186-4726-9091-5e77521d3820/content.

3. Вступ до JavaScript [Електронний ресурс] // JavaScript. – 2024. – Режим доступу до ресурсу: https://uk.javascript.info/intro.

4. Караванова Т.П. Інформатика: Методи побудови алгоритмів та їх аналіз. Необчисл. алгоритми: Навч. посіб. для 9-10 кл. із поглибл. вивч. інформатики. – К.:Генеза, 2007. – 216 с.: іл. – Бібліогр.: с. 212.

5. Каскадна модель [Електронний ресурс] – Режим доступу до ресурсу: https://qalight.ua/baza-znaniy/kaskadna-model-waterfall-model/.

6. Порівняння лінійного і бінарного пошуку [Електронний ресурс] // JavaRush. – 2023. – Режим доступу до ресурсу: https://javarush.com/ua/quests/lectures/ua.javarush.python.core.lecture.level16.lecture03.

7. Сапранков С. Навіщо і як вивчати React.js [Електронний ресурс] / Станіслав Сапранков // RobotDreams. – 2024. – Режим доступу до ресурсу: https://robotdreams.cc/uk/blog/237-zachem-i-kak-uchit-react-js.

8. Тесленко О. Метод оцінки антропогенного впливу в програмно-конфігурованих мережах [Електронний ресурс] / Тесленко О. // КПІ ім. Ігоря Сікорського. – 2022. – Режим доступу до ресурсу: https://conf.ztu.edu.ua/wp-content/uploads/2021/05/113-4.pdf.

9. Що таке API: простими словами про складне [Електронний ресурс] // HostIQ. – 2021. – Режим доступу до ресурсу: https://hostiq.ua/blog/ukr/what-is-api/.

10. Що таке NodeJS [Електронний ресурс] // NodeJS. – 2024. – Режим доступу до ресурсу: https://nodejs.tutorial.in.ua/what-is-nodejs/#:~:text=Node.,%D0%9E%D1%81%D0%BD%D0%BE%D0%B2%D0%BD%D1%96%20%D0%BE%D1%81%D0%BE%D0%B1%D0%BB%D0%B8%D0%B2%D0%BE%D1%81%D1%82%D1%96%20Node..

11. Що таке React JS і для чого він потрібен? Джерело: https://dan-it.com.ua/uk/blog/chto-takoe-react-js-i-dlja-chego-on-nuzhen/ © Dan-it.com.ua [Електронний ресурс] // Dan-IT. – 2024. – Режим доступу до ресурсу: https://dan-it.com.ua/uk/blog/chto-takoe-react-js-i-dlja-chego-on-nuzhen/.

12. Яцишин В. Алгоритми пошукових систем [Електронний ресурс] / Яцишин В // VI Всеукраїнська студентська науково - технічна конференція. – 2022. – Режим доступу до ресурсу: https://elartu.tntu.edu.ua/bitstream/123456789/9676/2/Conf_2013v1_Iatsishin_V_P-Alhoritmi_poshukovikh_sistem_128.pdf.

13. Abramov D. Introducing react.dev [Електронний ресурс] / Dan Abramov // React. – 2023. – Режим доступу до ресурсу: https://react.dev/blog/2023/03/16/introducing-react-dev.

14. API Reference [Електронний ресурс]. – 2024. – Режим доступу до ресурсу: https://developers.google.com/youtube/v3/docs.

15. Axios API [Електронний ресурс] // Axios. – 2024. – Режим доступу до ресурсу: https://axios-http.com/uk/docs/api_intro.

16. Barros F. The Netflix Recommender System [Електронний ресурс] / Fábio Barros // Academia. – 2015. – Режим доступу до ресурсу: https://www.academia.edu/33391850/ .

17. Chen S. BFS vs DFS – Difference Between Them [Електронний ресурс] / Sarah Chen // Guru. – 2024. – Режим доступу до ресурсу: https://www.guru99.com/difference-between-bfs-and-dfs.html.

18. Deep D. HTTP Requests [Електронний ресурс] / Daksha Deep // CodeCademy. – 2024. – Режим доступу до ресурсу: https://www.codecademy.com/article/http-requests.

19. Felice S. Unit testing for NodeJS using Mocha and Chai [Електронний ресурс] / Sandra Felice // BrowserStack. – 2023. – Режим доступу до ресурсу: https://www.browserstack.com/guide/unit-testing-for-nodejs-using-mocha-and-chai.

20. Get started with the basics of the TMDB API [Електронний ресурс] // TMDB. – 2024. – Режим доступу до ресурсу: https://developer.themoviedb.org/docs/getting-started.

21. Hadji-Vasilev A. How to Use Reelgood in 2024 [Електронний ресурс] / Andrej Hadji-Vasilev // CloudWars. – 2024. – Режим доступу до ресурсу: https://www.cloudwards.net/how-to-use-reelgood/.

22. Newman J. These two underdog apps have solved streaming TV’s biggest headache [Електронний ресурс] / Jared Newman // FastCompany. – 2020. – Режим доступу до ресурсу: https://www.fastcompany.com/90526983/these-two-underdog-apps-have-solved-streaming-tvs-biggest-headache.

23. React-dom [Електронний ресурс] // NPM. – 2024. – Режим доступу до ресурсу: https://www.npmjs.com/package/react-dom.

24. Reelgood [Електронний ресурс] // Public APIs. – 2024. – Режим доступу до ресурсу: https://publicapis.io/reel-good-media-api.

25. Sharma A. Express JS Tutorial [Електронний ресурс] / Anubhav Sharma // Simplilearn. – 2023. – Режим доступу до ресурсу:

https://www.simplilearn.com/tutorials/nodejs-tutorial/what-is-express-js#:~:text=StackExplore%20Program-,What%20Is%20Express%20JS%3F,helps%20manage%20servers%20and%20routes.

Анотація

Самойліч Ю.В. Розробка пошукової системи за тематикою кінематографії. Рукопис

Кваліфікаційна робота на здобуття освітнього ступеня “бакалавр” за спеціальністю 122 Комп’ютерні науки Волинський національний університет імені Лесі Українки, Луцьк,2025 р.

Робота присвячена розробці вебресурсу для пошуку фільмів та серіалів за заданими критеріями, такими як жанр, акторський склад, рік випуску та країна виробництва.

У роботі виконано аналіз сучасних підходів до побудови пошукових систем, розглянуто принципи роботи існуючих рішень у сфері інформаційного пошуку.

Розроблено вебзастосунок, що інтегрується з віддаленими базами даних, зокрема The Movie Database (TMDB) та YouTube API, що забезпечує користувачам можливість зручного та швидкого пошуку, перегляду трейлерів та ознайомлення з інформацією про обраний контент.

Реалізація інтерфейсу здійснена з використанням фреймворку React, серверної частини — на базі Node.js та Express. Запропонований підхід дозволяє покращити користувацький досвід та оптимізувати процес пошуку аудіовізуального контенту.

Ключові слова: пошукова система, вебзастосунок, фільми і серіали, фільтрація контенту, React, The Movie Database, стрімінгові сервіси