Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет комп'ютерної інженерії та управління
(повна назва)

Кафедра комп'ютерних інтелектуальних технологій та систем
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)
Інтелектуальна система рекомендацій для вибору фільмів на основі аналізу уподобань
(тема)

Виконав:
здобувач IV року навчання,
групи КІУКІ-21-10
Артем РУДЕНКО
(власне ім'я, прізвище)

Спеціальність
123 – Комп'ютерна інженерія
(код і повна назва спеціальності)
Тип програми освітньо-професійна
Освітня програма
Комп'ютерна Інженерія
(повна назва освітньої програми)

Керівник ас. каф. КІТС Андрій ТАТАРНИКОВ
(посада, власне ім'я, прізвище)

Допускається до захисту

Завідувач кафедри _____ Олег РУДЕНКО
(підпис) (власне ім'я, прізвище)

2025 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління

Кафедра Комп'ютерних інтелектуальних технологій та систем

Рівень вищої освіти перший (бакалаврський)

Спеціальність 123 Комп'ютерна інженерія
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Комп'ютерна інженерія
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«____»_____ 20 25 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

здобувачеві Руденку Артему Андрійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Інтелектуальна система рекомендацій для вибору фільмів на основі аналізу уподобань

затверджена наказом по університету від " 21 " травня 2025 р. № 399 Ст

2. Термін подання студентом роботи до екзаменаційної комісії

3. Вхідні дані до роботи

1. Середовище розробки Jupyter Notebook
2. Документація мови програмування Python
3. Документація бібліотеки Scikit-learn
4. Середовище розробки Visual Studio Code.
5. Документація бібліотеки ReactJS.
6. База даних фільмів The Movie Data Base

4. Перелік питань, що потрібно опрацювати в роботі

1. Аналіз предметної області.
2. Аналіз використовуваних технологій.
3. Програмна реалізація інтерфейсу та використання натренованих моделей
4. Інструкція користувача.
5. Висновки.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) _____
Слайд – презентація – 16 слайдів

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Строк / терміни виконання етапів роботи	Примітка
1	Аналіз проблеми та огляд існуючих рішень	26.05.2025-28.05.2025	Виконано
2	Вибір технологій розробки,	29.05.2025-31.05.2025	Виконано
	інструментальних засобів		
3	Створення тренувальних наборів даних	01.06.2025-02.06.2025	Виконано
4	Тренування моделей	03.06.2025-07.06.2025	Виконано
5	Розробка серверної та клієнтської частин	08.06.2024-10.06.2025	Виконано
6	Тестування застосунку	11.06.2025-12.06.2025	Виконано
7	Оформлення матеріалів кваліфікаційної роботи	13.06.2025-16.06.2025	Виконано
8	Подання кваліфікаційної роботи керівникові та	17.06.2025-18.06.2025	Виконано
	її попередній захист		
9	Подання роботи на рецензування	19.06.2025-20.06.2025	Виконано

Дата видачі завдання 26 травня 2025 р.

Здобувач _____
(підпис)

Керівник роботи _____ ас. Андрій ТАТАРНИКОВ
(підпис) (посада, власне ім'я, прізвище)

РЕФЕРАТ

Пояснювальна записка курсового проєкту: 72 с., 21 рис., 2 дод., 30 джерел.

РЕКОМЕНДАЦІЙНА СИСТЕМА, РЕКОМЕНДАЦІЇ ФІЛЬМІВ, PYTHON, KNN, ITEM-BASED, CONTENT-BASED, KNN, SVD, FLASK, TMDB API, SCIKIT-LEARN, PANDAS, JAVASCRIPT, REACT.

Метою кваліфікаційної роботи є створення застосунку, який аналізуватиме вподобання користувачів, історію переглядів фільмів та пропонуватиме рекомендації на основі алгоритмів машинного навчання. Користувачі зможуть авторизуватися та переглядати індивідуальні рекомендації.

У ході виконання кваліфікаційної роботи було створено застосунок, який аналізує оцінені та переглянуті користувачем фільми, рекомендуючи на основі цих даних схожі кінострічки. Проведено порівняння методів рекомендації із визначенням переваг та недоліків кожного з них.

Підготовка набору даних була виконана використовуючи мову Python та представлений сайтом TMDB власний API. Тренування моделей та збір даних було виконано у середовищі Jupyter Notebook. Серверна частина була написана на Python та фреймворку Flask API. Це було зроблено для кращої взаємодії з моделями. Клієнтська частина створена завдяки використанню бібліотеки ReactJS, яка забезпечує використання HTML та CSS разом з мовою Javascript.

ABSTRACT

Course project: 72 pages, 21 figures, 2 appendices, 30 sources.

RECOMMENDATION SYSTEM, FILM RECOMMENDATIONS, PYTHON, KNN, ITEM-BASED, CONTENT-BASED, KNN, SVD, FLASK, TMDB API, SCIKIT-LEARN, PANDAS, JAVASCRIPT, REACT.

The aim of the qualification work is to create an application that will analyze user preferences, movie viewing history, and offer recommendations based on machine learning algorithms. Users will be able to log in and view individualized recommendations.

In the course of the qualification work, an application was created that analyzes the movies rated and watched by the user, recommending similar movies based on this data. A comparison of recommendation methods was made to determine the advantages and disadvantages of each.

The dataset was prepared using Python and the TMDB's own API. Model training and data collection were performed in the Jupyter Notebook environment. The server side was written in Python and the Flask API framework. This was done for better interaction with the models. The client part was created using the ReactJS library, which provides the use of HTML and CSS along with the Javascript language.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

DOM – (Document Object Model)

VS Code – Visual Studio Code

Jupyter – Jupyter Notebook

API – Application Programming Interface

TMDB – The Movie Data Base

CSS – Cascading Style Sheets

HTML – Hyper Text Mark-up Language

UI – User Interface

JS – JavaScript

JSX – JavaScript XML

Front-end – Client-Side

Back-end – Server-Side

KNN – K-Nearest Neigbors

SVD – Singular Value Decomposition

TF – term frequency

IDF – inverse document frequency

ПЗ – Програмне забезпечення

ШІ – Штучний інтелект

CURL – Client Uniform Resource Locator

ВСТУП

В сучасних умовах розвитку технологій, індустрія кіно дуже швидко розвивається, випускаючи багато нових фільмів різних жанрів кожного місяця. Пошук фільмів за рейтингами або за окремими жанрами вже не відповідає сучасним критеріям, так як відсутнє врахування уподобань користувачів. Кожен з глядачів має власні смаки щодо фільмів, і може мати бажання побачити свого улюбленого актора в фільмах, або щоб у сюжеті були цікаві ключові деталі, які були переглянуті раніше. Саме тому з’являється необхідність в персоналізованих рекомендаційних системах, які будуть враховувати ці деталі та надавати такі рекомендації, які користувач дійсно хоче.

Існуючі рекомендаційні системи відрізняються різними підходами, які застосовуються для аналізу даних. До найчастіше використовуваних можна віднести колаборативну фільтрацію, яка працює завдяки врахуванню вподобань інших користувачів, та контентну фільтрацію, яка базується на аналізі характеристик фільмів. В свою чергу використання гібридних підходів дозволяє поєднати переваги колаборативного та контентного підходів, щоб досягти підвищення рівня точності результатів.

Проте, рекомендаційні системи мають велику кількість обмеженнь, серед яких є проблема «холодного старту», яка виникає через обмежену кількість початкових даних, через що відповідність вподобань буде на низькому рівні.

Отже, розробка інтелектуальної системи рекомендації фільмів, задача якої враховувати вподобання користувачів, контекст перегляду та інші аспекти, є актуальною задачею, вирішення якої допоможе покращити користувацький досвід та відкриє нові можливості для індустрії кінематографу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд сучасного стану індустрії кінематографу

Сучасний кінематограф розвивається дуже швидкими темпами та стрімко розширює масштаби свого виробництва. За останнє десятиліття, кількість щорічно випущених фільмів зросла в декілька разів. Згідно з даними актуальної статистики, кожного року у світ виходять щонайменше 10 тисяч повнометражних фільмів, при цьому не враховуючи списки короткометражних стрічок та велику кількість серіалів.

Завдяки цифровій революції змінились не тільки методи створення фільмів, а і методи їх розповсюдження та перегляду. Кінотеатри почали поступатися онлайн платформам та стримінговим сервісам, таким як Netflix, Amazon Prime Video, HBO Max та Apple TV+. Інші цифрові платформи не тільки надають можливість перегляду будь-якого фільму із багатотисячної бібліотеки, але і фінансують створення власних кіно-франшиз, тим самим, створюючи серйозну конкуренцію традиційним кіностудіям.

Через те, що ці компанії також створюють власні фільми, які в свою чергу розраховані як на аудиторію власних платформ так і на аудиторію інших країн та культур пропорційно розширює представництво різних країн та культур у глобальному кінопросторі. Через це вибір фільмів стає ще більш різноманітним, а для користувачів процесс вибору з кожним роком стає все дедалі важчим.

1.2 Проблеми пошуку фільмів за користувацькими вподобаннями

Швидке зростання кількості доступних фільмів створює для глядачів серйозну проблему інформаційного перевантаження. Користувачі багатьох стримінгових платформ стикаються з явищем, яке називається «парадоксом

вибору», коли маючи велику кількість опцій, замість спрощення отримання необхідного результату, користувач стикається з ускладненням процесу прийняття рішення в сторону того чи іншого фільму, через що користувач може завершити пошук, так і не отримавши необхідний результат.

Серед основних проблем, з якими стикається користувач виділяються:

– надмірна кількість контенту – користувач сервісів може витрачати більше двадцяти хвилин на вибір фільму для перегляду;

– обмеженість звичайних систем – стандартна класифікація за жанрами є надто узагальненою і не відображає різноманітність стилів кіно;

– суб'єктивність оцінок та рейтингів – загальні рейтинги та оцінки критиків не завжди відповідають індивідуальним вподобанням глядача;

– динамічність уподобань – смаки користувачів змінюються з часом та залежать від контексту, що ускладнює створення статичних профілів.

Ці проблеми підкреслюють необхідність створення рекомендаційних систем, які здатні враховувати не лише явні вподобання користувача такі як жанри та смаки, але і приховані, такі як контекст, емоційну складову та баланс між знайомим і новим контентом.

1.3 Існуючі методи та підходи до створення рекомендаційних систем

Існуючі рішення пошуку представляють собою алгоритмічні системи, які використовують той, чи інший підхід у вирішенні задачі надання рекомендацій користувачу. Серед них особливої уваги заслуговують такі методи фільтрації як колаборативний, контентий та гібридний.

1.3.1 Колаборативна фільтрація

Фільтрація за допомогою цього методу пошуку інформації, виконується на основі рекомендацій інших користувачів, використовуючи для цього інформацію про схожі уподобання, базуючись на поведінці та

взаємодії із запропонованими елементами [1]. Даний підхід найчастіше реалізується на основі використання User-Based або Item-Based підходів фільтрації (рисунок 1.1).

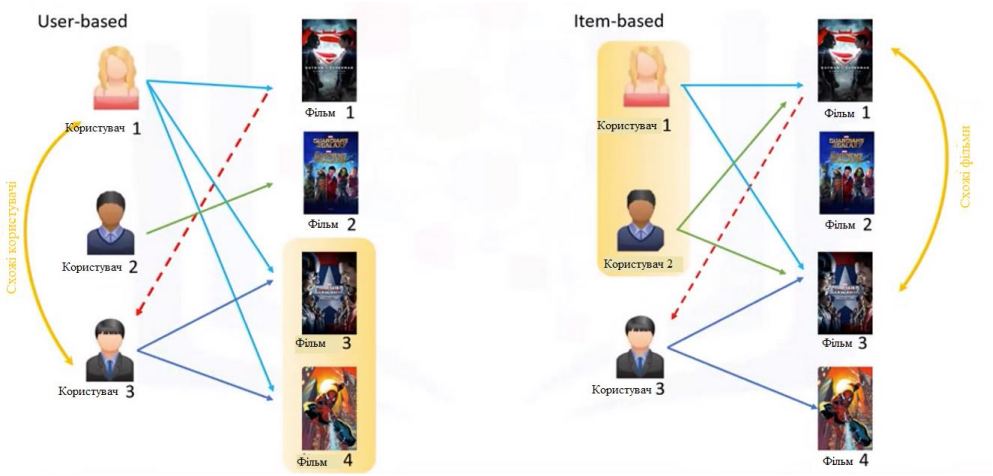

Рисунок 1.1 – User-Based та Item-Based підходи

При використанні User-Based, система робить визначення користувачів зі схожими вподобаннями, аналізує їхню історію та рекомендує фільми, які користувач ще не дивився. Але завдяки тому, що інші користувачі з такими ж уподобаннями дивилися фільм, система робить висновок що запропонована кінострічка потенційно сподобається користувачу.

Item-Based підхід знаходить подібність між самими фільмами, базуючись на оцінках користувачів, інформації про фільм, використовуючи дані про користувацькі оцінки, коментарі та інше [2].

1.3.2 Контента фільтрація

На відміну від колаборативної фільтрації, що базується на колективному досвіді користувачів, контентна фільтрація формує рекомендації, аналізуючи характеристики самих фільмів та індивідуальні дані користувача, зосереджуючись на їхній схожості.Контентна фільтрація

використовує властивості об'єктів для відбору, що відповідають запиту користувача. Цей метод часто враховує особливості інших об'єктів, до яких користувач виявляє інтерес.

Рисунок 1.2 – Content-Based підхід

При фільтруванні на основі вмісту враховуються насамперед такі характеристики фільму, як жанри, опис, акторський склад та особливості, визначені ключовими словами та тегами (рисунок 1.2). Додатково можуть враховуватися уподобання користувачів, які отримуються на основі переглядів та оцінок [3].

1.3.3 Гібридні підходи

Гібридна рекомендаційна система ефективно поєднує контентну фільтрацію і фільтрацію на основі контенту для підвищення якості результатів фільтрації. Використовуючи сильні сторони обох підходів, та гнучку систему налаштувань, система може пом'якшити обмеження, притаманні кожному з використаних методів, що дозволяє досягти підвищення рівня задоволення користувачів та рівня персоналізованого досвіду [4]. Наприклад, коли дані про користувача є обмеженими, система може компенсувати це аналізом характеристик фільму, таких як жанр, актори або опис. У той же час, для користувачів із багатою історією переглядів система може робити висновки на основі схожості з іншими глядачами [5].

1.4 Аналіз існуючих рекомендаційних платформ та їх обмежень

Великі гравці індустрії, такі як Amazon, HBO, Disney+, Netflix, та інші, активно впроваджують алгоритми персоналізованих рекомендацій з метою підвищення якості взаємодії з користувачами. Основною метою таких систем є швидке надання релевантного контенту, зменшення часу на пошук і збільшення часу взаємодії з платформою.

Водночас існують спеціалізовані сервіси, орієнтовані саме на фільми, серед них є IMDb, Letterboxd, TMDB. Вони використовують як прості системи оцінювання і рецензій, так і складніші алгоритми рекомендацій. Деякі сервіси дозволяють створювати персональні списки, відстежувати переглянуті фільми, брати участь у спільнотах з обговореннями. Однак більшість із них все ще залежать від базових механізмів сортування за популярністю або жанром.

1.4.1 Комерційні системи (Netflix, Amazon Prime)

Комерційні платформи розповсюдження фільмів першими почали впроваджувати рекомендаційні системи у сфері цифрового контенту. Алгоритми рекомендацій, які вони використовують, мають вирішальне значення для утримання аудиторії, так як вони дозволяють швидко підібрати схожі фільми чи серіали, не навантажуючи користувача необхідністю введення додаткових параметрів пошуку.

Netflix, зокрема, широко відомий своєю системою рекомендацій, що поєднує колаборативну фільтрацію, аналіз поведінки користувачів, та A/B тестування. Система враховує не лише оцінки користувачів та переглянуті фільми, а також враховує час перебування на сторінці фільму, день тижня, тип пристрою, та інші деталі поведінки певного користувача. Тим самим, Netflix максимально персоналізує головну сторінку, підбираючи не лише рекомендації, а і навіть підписи до фільмів та постери [6].

Але, незважаючи на це, платформа не надає пояснень своїх механізмів і дуже часто просто намагається рекомендувати лише популярний контент, тим самим, зменшуючи різноманіття вибору.

Amazon Prime Video також активно використовує рекомендаційні алгоритми, зокрема на основі контентної фільтрації та глибокого аналізу поведінки користувача в межах усієї екосистеми Amazon. Сервіс враховує не лише дані про перегляди, але й історію покупок, пошукові запити та навіть відгуки. Такий міжсервісний підхід дозволяє будувати більш цілісні профілі користувачів і автоматично пропонувати контент, що максимально відповідає їхнім інтересам [7].

Проте, як і у випадку з Netflix, система має обмеження в можливостях ручного налаштування, а самі рекомендації можуть бути надто комерціалізованими – платформа іноді віддає перевагу контенту, який вигідніший для просування, а не для користувача.

1.4.2 Спеціалізовані кінематографічні сервіси (IMDb, Letterboxd)

Спеціалізовані кінематографічні сервіси, такі як IMDb та Letterboxd, надають користувачам багатий функціонал для дослідження та обговорення фільмів, незалежно від їх категорій.

IMDb (Internet Movie Database) – це одна з найстаріших та найвідоміших онлайн-баз даних про фільми, серіали, акторів та інший пов'язаний контент. Користувачі можуть оцінювати фільми за 10-бальною шкалою, на основі якої формується загальний рейтинг кожного фільму на платформі. Варто зауважити, що IMDb не займається генерацією рекомендацій, які враховують індивідуальні вподобання користувачів. Натомість платформа обмежується демонстрацією списків популярних стрічок, сформованих на основі колективних оцінок усіх користувачів.

Letterboxd – це соціальна платформа для кінолюбителів, яка дозволяє користувачам відстежувати переглянуті фільми, писати рецензії, створювати

списки та взаємодіяти з іншими учасниками спільноти. Хоча Letterboxd не має вбудованої системи рекомендацій, кілька сторонніх проєктів намагалися запровадити такі функції, використовуючи дані з платформи.

Як приклад можна навести ситуації, коли деякі з цих проєктів збирають рейтинги користувачів і застосовують алгоритми колаборативної фільтрації, для надання персоналізованих рекомендацій. Водночас такі платформи не є офіційними, що в свою чергу означає необхідність додаткових зусиль для інтеграції з обліковим записом користувача на Letterboxd [8].

1.5 Виклики та проблеми рекомендаційних систем

Незважаючи на істотний розвиток у галузі рекомендаційних технологій, існують певні труднощі, які відчутно впливають на точність та результативність сформованих рекомендацій.

1.5.1 «Холодний старт» моделей

«Холодний старт» є однією із найбільш поширених проблем в рекомендаційних системах, який виникає при недостатній кількості інформації про користувача або нову кінострічку, для якої треба зробити якісну рекомендацію. У випадку користувача, система не має достатньої кількості інформації про оцінки та історію переглядів, що викликає складності у визначенні вподобань користувача. Якщо розглядати новий фільм, то первинної інформації буде недостатньо для того, щоб внести його у список рекомендацій. Така ситуація виникає через відсутність оцінок користувачів, і потрібен час для формування статистики.

Вирішити цю проблему можна залученням контекстної інформації, такої як жанри, акторський склад, опис та ключові слова. У гібридних системах комбінуються різні джерела даних, що дає змогу зменшити вплив «холодного старту» та покращити точність рекомендацій на ранніх етапах.

1.5.2 Врахування контексту та змінних вподобань

Важливо зазначити, що смаки та уподобання користувачів ніколи не будуть постійними та вузько направленими. Вони змінюються з часом, під впливом настрою, ситуації, віку, або ж поточного соціального оточення.

Врахування цих факторів дозволяє створювати більш персоналізований та точний користувацький досвід. Проте реалізація подібних систем потребує збору додаткових даних та складніших моделей, що ускладнює як розробку, так і підтримку таких рішень.

1.6 Постанова задачі дослідження та розробки

Проєкт представляє собою веб-застосунок, який дозволяє користувачам зареєструватися, шукати фільми, додавати їх до історії перегляду або списку «переглянути пізніше», оцінювати, та отримувати автоматизовані рекомендації на основі цих даних. Крім цього, система дозволяє надсилати звіти про помилки, що сприяє покращенню якості сервісу.

Головні функції застосунку, які повинні бути реалізовані:

- реєстрація та аутентифікація користувачів;
- аналіз історії переглядів та оцінок користувача;
- генерація персоналізованих рекомендацій;
- фільтрація рекомендацій за вказаними параметрами;
- відображення інформації про певний фільм;
- аналіз наданих користувачу рекомендацій;
- надання користувачу статистики щодо його переглянутих фільмів.

Варто зазначити, що перед створенням основного застосунку, який буде вміщувати в собі серверну та клієнтську частини, необхідно створити набір даних та натренувати на ньому моделі, завдяки яким і буде працювати головний метод надання рекомендацій.

2 АНАЛІЗ ВИКОРИСТОВУВАНИХ ТЕХНОЛОГІЙ

2.1 Огляд засобів розробки

Для реалізації поставленої задачі необхідно звернутися до засобів тренування моделей та веб-розробки. В цьому розділі розглядається використання сучасних методів створення ШІ моделей та їх подальша реалізація в веб-застосунку, зокрема Python, Jupyter Notebook, JavaScript, Visual Studio Code, React, Flask та TMDB API, які відіграють важливу роль у процесі розробки.

Python широко використовується для аналізу і обробки даних завдяки простому синтаксису та наявності потужних бібліотек, таких як pandas, numpy і scikit-learn. Для зручної роботи з кодом, графіками й описами в одному середовищі ефективним інструментом є Jupyter Notebook, який забезпечує інтерактивність та гнучкість при проведенні досліджень і експериментів. У проєктах, що пов’язані з фільмами, доцільно інтегрувати API The Movie Database (TMDB), який відкриває доступ до великої кількості актуальної інформації про фільми, акторів і рейтинги, що значно розширює можливості аналізу і побудови рекомендаційних моделей.

Для створення серверної частини застосунку часто використовується мікрофреймворк Flask, реалізований на Python. Flask забезпечує гнучкість, мінімалізм та велику кількість можливостей з розробки різноманітних застосунків з необхідною функціональністю без зайвих ускладнень. Visual Studio Code (VS Code) є популярним редактором коду, який підтримує інтеграцію з Flask та надає зручні інструменти для налагодження, автодоповнення коду та управління проєктом. Використання VS Code спрощує процес розробки та підвищує продуктивність.

На стороні клієнта JavaScript залишається ключовою мовою для створення інтерактивних вебінтерфейсів. Зокрема, бібліотека React дозволяє

будувати компонентні структури інтерфейсу, полегшуючи підтримку та удосконалення коду. React став одним із найпопулярніших інструментів для створення сучасних односторінкових веб-застосунків.

2.2 Створення та тренування моделей

Моделі машинного навчання є ключовим аспектом у досягненні мети цього проєкту. Вони будуть шукати фільми за назвою, аналізувати переглянуті та оцінені користувачем фільми, рекомендуючи на їх основі нові. Для їх створення необхідно обрати середовище та мову програмування, а також створити набір даних, який буде використовуватись як для тренування, так і для бази даних фільмів в цілому.

2.2.1 Мова програмування Python

Python – це універсальна мова комп'ютерного програмування, яку можна використовувати практично для будь-чого. Від веб-розробки та автоматизації до аналізу даних і тестування програмного забезпечення – ця мова універсальна та легка в користуванні, так як написання коду та його читання настільки інтуітивно зрозуміле, як читання англійської мови.

Крім того, Python має величезну бібліотеку інструментів, які скорочують час розробки. Наприклад, Python надає легкий доступ до багатьох бібліотек та інших інструментів розробки, які дозволяють швидко виконати математичний аналіз, будувати графіки та запити.

Python за останні роки зайняв своє місце в авангарді виокристання технологій ШІ. На це є кілька причин, серед яких ефективний, точний синтаксис, низька точка входу та проста інтеграція. Однак велика кількість бібліотек з відкритим вихідним кодом, таких як scikit-learn та tensorflow, показали переваги Python, та зробили його основним інструментом для машинного навчання та штучного інтелекту [9].

2.2.2 Середовище розробки Jupyter Notebook

Jupyter Notebook – це веб-застосунок, що дозволяє створювати та обмінюватися інтерактивними документами, які можуть містити програмний код, текстові пояснення, математичні формули, графіки та інші візуальні елементи. Його активно застосовують у сфері створення програм, опрацювання інформації, машинному навчанні та інших сферах.

Його архітектура включає два ключові елементи: ядро (kernel), що виконує код, та інтерфейс користувача, який забезпечує взаємодію з ядром і відображення результатів [10].

Jupyter Notebook ідеально підходить для машинного навчання, так як дозволяє запускати код окремо, ділянку за ділянкою, дозволяючи поступово завантажувати дані, перетреновувати моделі та тестувати їх. Якщо б створення моделі відбувалося б у звичайному коді, то програмі довелось би постійно завантажувати дані та знову тренувати модель, коли користувачу треба лише протестувати її точність.

Завдяки цьому Jupyter Notebook є ідеальним вибором, щоб натренувати моделі, оскільки зменшується час навчання, дозволяючи робити це частинами та зберігаючи проміжний результат.

2.2.3 Створення набору даних з TMDB API

The Movie Database (TMDB) API – це потужний інструмент, що надає розробникам доступ до великого обсягу даних про фільми, телесеріали, акторів та інший контент. TMDB є спільнотою, яка постійно оновлює та розширює свою базу даних новими фільмами, забезпечуючи актуальність та точність наданої інформації.

TMDB API дозволяє отримувати детальні відомості про фільми, включаючи описи, жанри, дати виходу, рейтинги користувачів, а також інформацію про акторський склад та знімальну групу. Крім того, API надає

доступ до зображень, таких як постери та кадри з фільмів, що є корисним для створення візуально привабливого застосунку.

Використання TMDB API є безкоштовним для некомерційних та навчальних проєктів, однак для отримання ключа доступу необхідно зареєструватися на сайті та дотримуватися умов використання, включаючи обов'язкове зазначення джерела даних [11].

2.2.4 K-Nearest Neighbors

Метод K-Nearest Neighbors (KNN) належить до базових алгоритмів машинного навчання, що застосовується як для класифікаційних, так і для регресійних завдань. Його суть полягає в припущенні, що об'єкти з подібними ознаками, як правило, належать до однакової групи або мають близькі значення. У процесі класифікації новий об'єкт зіставляється з «k» найближчими прикладами з навчальної вибірки, після чого йому автоматично присвоюється той клас, який найчастіше зустрічається серед цих сусідів (рисунок 2.1).

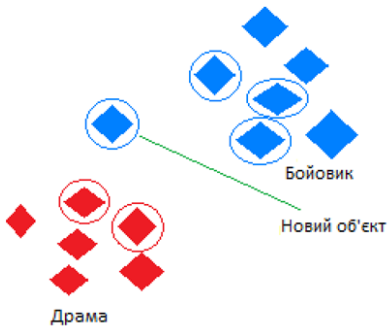

Рисунок 2.1 – Визначення класу для об’єкту

Наприклад, якщо серед 5 найближчих сусідів 3 належать до класу «Бойовик» і 2 до класу «Драма», то новий об'єкт буде автоматично віднесений до «Бойовик».

Для визначення «близькості» між об'єктами використовуються метрики відстані, такі як евклідова або косинусна відстань. Вибір метрики залежить від природи даних та специфіки задачі.

KNN є ненавчальним (non-parametric) методом, не будуючи явної моделі під час навчання. Замість цього, вся навчальна вибірка зберігається, і обчислення відбуваються під час передбачення. Це робить KNN простим у реалізації, але неефективним при роботі з великим обсягом даних [12].

2.2.5 Singular Value Decomposion

Сингулярний розклад (SVD) є базовим інструментом лінійної алгебри, що дає змогу представити довільну матрицю у вигляді трьох окремих компонент: дві ортогональні матриці та одну діагональну. Формально, для матриці M розміром m×n, розклад виглядає так (рисунок 2.2):

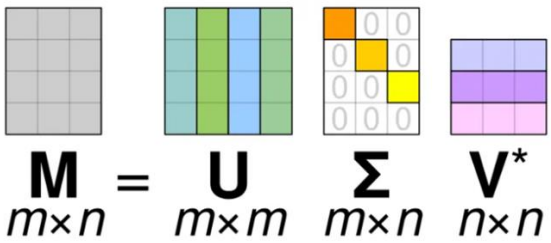

Рисунок 2.2 – Розклад матриці M

Де:

– U – ортогональна матриця розміром m×m, стовпці якої формують базис у просторі рядків початкової матриці;

– Σ – діагональна матриця розміром m×n, де на діагоналі розташовані сингулярні значення;

– V^T – транспонована ортогональна матриця розміром n×n, утворена з векторів, які задають напрямки у просторі стовпців вихідної матриці.

Завдяки цьому розкладу з’являється можливість аналізувати структури даних, зокрема виявляти напрямки найбільшої варіації в даних, що є корисним для задач зменшення розмірності, шумозаглушення та побудови рекомендаційних систем [13].

2.3 Технології серверної частини застосунку

Створення серверної частини не менш важливе, ніж розробка моделей, оскільки вона має забезпечити ефективну взаємодію як з навчальними моделями, клієнтською частиною, а також з базою даних, яка сформована на основі даних, що використовувались під час процесу навчання моделей. Саме тому для створення та тестування серверної частини застосунку також буде використовуватися мова Python.

2.3.1 Фреймворк Flask API

RESTful API (Representational State Transfer) – це поширений архітектурний стиль для створення мережевих застосунків, який базується на стандартних HTTP-методах. Він дає змогу створювати масштабовані веб-сервіси, які не зберігають стан між запитами. [14].

Flask є одним із найпопулярніших фреймворків на Python, який вирізняється простотою, мінімалізмом та гнучкістю у використанні. Завдяки цим якостям Flask активно застосовується при використанні RESTful API, особливо коли потрібна проста у налаштуванні структура проєкту.

Оскільки Flask належить до мікрофреймворків, не накладаються жорсткі вимоги щодо додаткових інструментів або бібліотек, що надає розробнику свободу вибору. Можливість використання розширень для інтеграції роботи з БД, авторизацією та іншими компонентами, дозволяє розробляти як невеликі прототипи програм, так і повноцінні масштабовані веб-застосунки які можуть використовуватися в різних сферах.

2.3.2 Середовище розробки VS Code

Середовище розробки VS Code являє собою редактор тексту в якому доступне підключення модулів, які розширюють функціонал та дозволяють користувачам працювати з різними мовами програмування. Висока продуктивність дозволила отримати популярність серед розробників у всьому світі та стати одним з основних інструментів роботи з кодом.

Редактор забезпечує підтримку численних мов програмування завдяки розвиненій системі розширень. Користувачі можуть легко додавати нові модулі для взаємодії з різними мовами, утилітами та зовнішніми сервісами.

Для розробки на Python існує офіційне розширення, яке інтегрує такі функції, як підсвічування синтаксису, автодоповнення коду (IntelliSense), відлагодження, підтримка юніт-тестування та інтеграція з віртуальними середовищами. Вбудовані функції перевірки коду на помилки дозволяють зменшити час необхідний на написання програми завдяки використанню вбудованих функцій підказок [15].

Вибір VS Code як середовища розробки також пояснений тим, що розробка клієнтської частини буде проводитися там, де і серверна. Тому буде набагато зручніше використовувати один застосунок одразу для двух частин, так як це зменшує час на запуск обох проєктів та додає зручності в переміщенні між ними.

2.4 Розробка клієнтської частини

Протестувати коректність роботи моделей та точність рекомендацій можна і за допомоги запитів до back-end’у, використовуючи cURL запити в консолі або спеціалізовані застосунки, наприклад Postman. Але все ж таки, клієнтська частина необхідна, тому що застосунок розрахований на звичайних користувачів. Також створення front-end’у зробить зручнішим надсилання запитів на клієнтську частину, без посторонніх методів.

2.4.1 Використання JavaScript

JavaScript – це мова програмування високого рівня та динамічної природи, яка на сьогоднішній день є ключовим інструментом для розробки більшості сучасних веб-застосунків. Завдяки використанню JavaScript відкриваються можливості для реалізації складних функцій на веб-сторінках, таких як інтерактивний контент, анімації, мультимедіа та інше. Майже всі веб-сайти у світі використовують JavaScript, завдяки чому забезпечується динамічна поведінка веб-сторінок.

Visual Studio Code забезпечує винятково потужне середовище для розробки на JavaScript. Редактор надає комплексну підтримку цієї мови програмування, що робить процес створення, тестування та налагодження JavaScript-застосунків значно ефективнішим.

Асинхронна модель дозволяє швидко обробляти користувацькі запити та взаємодіяти з базою даних без затримки основного процесу. Це особливо актуально для функціоналу рекомендацій у реальному часі, де потрібна миттєва реакція на дії користувача [16].

2.4.2 Фреймворк React

React виступає як бібліотека, що забезпечує зручне створення гнучких та масштабованих інтерфейсів. За рахунок розділення інтерфейсу на незалежні компоненти, кожен із яких має власний стан та логіку, розробникам вдається досягти високої модульності та зручності супроводу. Зміни в одному компоненті не зачіпають інші, що знижує ризик помилок і полегшує тестування [17].

Ключовою перевагою React є застосування віртуального DOM. Це полегшена версія реального DOM, яка зберігається в оперативній пам’яті та дозволяє React оптимізовано відстежувати зміни. При оновленні стану компонентів система порівнює новий віртуальний DOM з попереднім і

визначає мінімально необхідні зміни для внесення в реальний DOM, що суттєво підвищує продуктивність візуалізації та змешує час очікування відображення інформації на сторінці (рисунок 2.3).

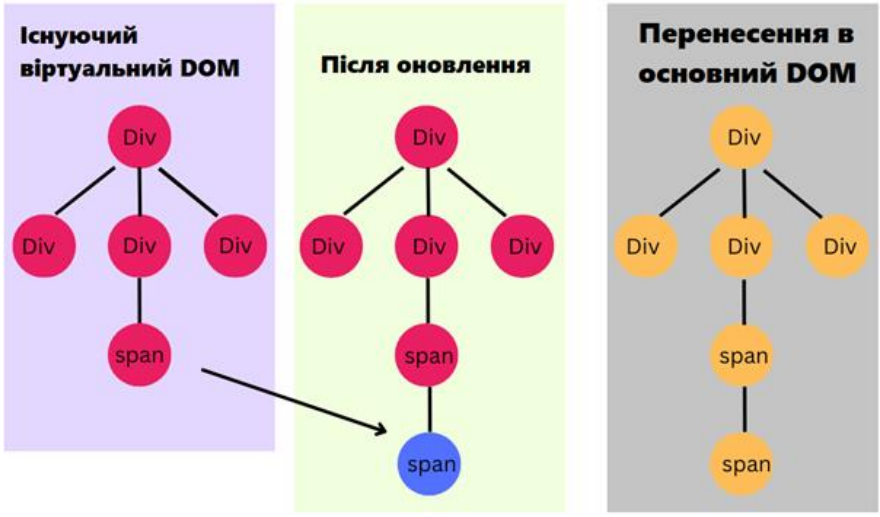

Рисунок 2.3 – Принцип роботи процесу узгодження

У контексті системи рекомендацій фільмів це забезпечує оперативне оновлення списків рекомендацій у відповідь на дії користувача – наприклад, при зміні фільтрів або виставленні оцінки без потреби повного перезавантаження сторінки. React автоматично обробляє зміни стану інтерфейсу, що дозволяє уникнути ручного керування атрибутами, обробниками подій та прямим оновленням DOM, яке було б необхідним у традиційній реалізації [18].

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Створення набору даних

Початковим етапом реалізації проєкту є збір інформації про фільми та оцінки користувачів, їх узагальнення у єдиний формат. Потім, на основі цих датасетів будуть проводитися тренування моделей KNN та SVD. Цей етап розробки буде проводитися в середовищі програмування Jupyter Notebook, який довзоляє багатократні повторення окремих ділянок коду.

3.1.1 Застосування TMDB API

Щоб отримати інформацію про фільми скористаємося бібліотекою request в Python, яка дозволяє робити запити на сайти. В нашому випадку використаємо її для виконання запитів до TMDB API, щоб отримати дані.

Після реєстрації на сайті та отримавши погодження на використання API, отримуємо ключ, який можемо використовувати у Python при запитах до сайту. За виконання запитів відповідає функція make_api_request, призначена для відстеження переміщення вперед по сторінкам з кінострічками та отримання інформації про певний фільм (лістинг 3.1) [19].

Лістинг 3.1 – Функція make_api_request().

```
def make_api_request(endpoint, params=None, use_cache=True):
    if params is None:
        params = {}
    params['api_key'] = API_KEY
    if 'language' not in params:
        params['language'] = 'uk-UA'
    cache_key = f"{endpoint}_{json.dumps(params, sort_keys=True)}"

    if use_cache and cache_key in request_cache:
        return request_cache[cache_key]

    url = f"{BASE_URL}{endpoint}"
```

```
    max_retries = 3
    retry_count = 0

    while retry_count < max_retries:
        try:
            response = requests.get(url, params=params)
            response.raise_for_status()
            data = response.json()

            if use_cache:
                request_cache[cache_key] = data
            return data
```

3.1.2 Збір даних

Для отримання інформації про фільм створимо функцію get_movie_details(), яка буде приймати в себе ідентифікатор фільму та повертати повну інформацію про фільм, виконавши запит за допомоги make_api_request(). Необхідною інформацією для тренувальних наборів є: опис фільму, акторський склад, режисер та ключові слова. Окрім них, корисними будуть тривалість фільму, середня оцінка, бюджет, мова оригіналу, рік виходу у прокат та ключові слова (лістинг 3.2).

Лістинг 3.2 – Функція get_movie_details().

```
def get_movie_details(movie_id):
results = {}
    endpoints = {
        'details': f"/movie/{movie_id}",
        'credits': f"/movie/{movie_id}/credits",
        'keywords': f"/movie/{movie_id}/keywords"
}
if not results.get('details'):
        return None
    movie_data = results['details']
    credits_data = results.get('credits', {})
    keywords_data = results.get('keywords', {})
# Збираємо деталі
details = {
        "movieId": None,
        "tmdbId": movie_id,
        "title": movie_data.get("title", ""),
return details
```

Додамо ще дві функції, створивши певну ієрархію (рисунок 3.1), яка дозволить проводити обробку по всій базі даних за необхідні роки.

Функція process_movie_batch() проводить обробку інформації по всім фільмах на певній сторінці, використовуючи функцію get_movie_details(). Спочатку отримується список ідентифікаторів фільмів, для кожного з яких викликається функція get_movie_details для отримання детальної інформації. Якщо дані успішно отримано, вони додаються до списку результатів, який повертається після завершення обробки всіх фільмів у списку.

Функція collect_movies() виконує загальний збір даних про фільми з усіх сторінок, використовуючи функцію process_movie_batch(). Далі формуються параметри запиту, та надсилаються на сервер. Після отримання результатів сторінки фільмів обробляються в списки, які потім діляться на підсписки. У результаті зібрані дані додаються до спільного списку, а кожному фільму присвоюється унікальний ідентифікатор.

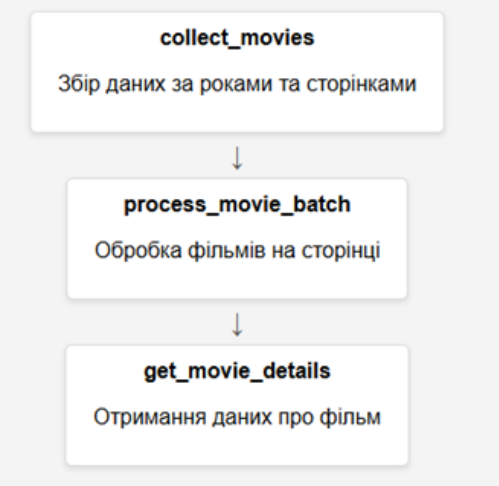

Рисунок 3.1 – Ієрархія функцій

Варто зазначити, що в програмі використовується бібліотека багатопоковості «concurrent futures», яка дозволяє розділити виконання запитів між «працівниками», які значно прискорюють збір даних [20].

Тимчасове вимкнення світла може обнулити збір даних. Тому, було прийнято рішення створювати датасети зі збереженням проміжних результатів за кожен рік, та файл checkpoint.json, який зберігає дані про

поточний рік та сторінку [21]. Код з використанням збереження проміжних результатів продемонстрований на лістингу 3.3.

Лістинг 3.3 – Використання проміжних результатів

```
checkpoint_data = {}
if os.path.exists(checkpoint_file):
             with open(checkpoint_file, 'r', encoding='utf-8') as f:
                checkpoint_data = json.load(f)
                all_movies = checkpoint_data.get('movies', [])
                movie_id_counter = checkpoint_data.get('next_id', 1)
            last_processed_year = checkpoint_data.get('last_year',
            start_year - 1)
                  last_processed_page=
                  checkpoint_data.get('last_page',0)
                if last_processed_page > 0:
                    start_year = last_processed_year + 1
                else:
                    start_year = last_processed_year
            with open(checkpoint_file, 'w', encoding='utf-8') as f:
                json.dump(checkpoint_data, f, ensure_ascii=False)
            if len(all_movies) % 100 == 0 or len(all_movies) > 0:
             temp_df = pd.DataFrame(all_movies)

            cols = temp_df.columns.tolist()
             cols = ['movieId'] + [col for column in columns if col !=
            'movieId']
             temp_df = temp_df[cols]
             temp_df.to_csv(f"movies_data_temp_{year}.csv", index=False)
            print(f"Збережено    проміжні    дані:    {len(all_movies)}
            фільмів")
```

Після обробки, отримуємо набір даних з усіма фільмами за 45 років (1980 – 2025) з усього світу. Датасет налічує 270 тисяч невідфільтрованих елементів. Загальний час збору інформації зайняв 48 годин з перервами при використанні двадцяти потоків у функції collect_movies().

3.1.3 Фільтрація набору даних.

Перед тим, як тренувати модель, необхідно позбавитися від елементів, які будуть створювати шум та зменшувати точність рекомендацій відсутністю в них ключових даних. Спочатку необхідно видалити всі фільми, які не мають в переліку отриманих даних опису фільму англійською мовою (лістинг 3.4). Видалення необхідно для зменшення кількості помилок, які

можуть викликати такі фільми. До того ж англійський опис є у більшої частини отриманих фільмів, і лише не популярні стрічки не містять в собі англійського опису на сайті TMDB.

Лістинг 3.4 – Видалення елементів без опису англійською мовою

```
df_cleaned = df.dropna(subset=["overview_en"]).reset_index(drop=True)
df_cleaned["movieId"] = range(1, len(df_cleaned) + 1)
```

Так як рекомендаційна модель на основі вмісту фільмів використовує ключові слова для визначення схожості, їхня відсутність або мала кількість може вплинути на точность рекомендацій.

Не маючи достатньої кількості ключових слів, система збільшує ваги іншим елементам опису фільму, а саме жанрам, акторам та загальному опису. І вже потім, під час надання рекомендацій, система не дораховується ключових слів та рекомендує фільми за жанром, навіть якщо вони не схожі на ті фільми, які користувач оцінив та переглянув.

Для цього необхідно прибрати фільми, які не мають ключових слів взагалі та фільми, які мають їх менше ніж шість (лістинг 3.5).

Лістинг 3.5 – Видалення фільмів з малою кількістю ключових слів

```
def count_keywords(keyword_str):
    if pd.isna(keyword_str): return 0
    return len(keyword_str.split(" "))
movies2["num_keywords"] = movies2["keywords"].apply(count_keywords)
filtered_movies = movies2[movies2["num_keywords"] >= 6].copy()
```

3.2 Тренування моделей

Після створення та фільтрації набору даних з фільмами перейдемо до тренування моделей, які будуть використані у серверній частині програмної реалізації. Для тренування KNN-моделі будемо використовувати створений набір даних на основі опису фільмів. Нажаль, створення тренувального

набору з оцінками вимагає наявності великої кількості користувачів, які повинні оцінити фільми за їхніми смаковими вподобаннями. Тому, для тренування SVD-моделі буде використовуватися вже готовий набір даних ratings, взятий з набору тренувальних наборів «MovieLens», але перероблений під створений власний набір даних.

3.2.1 Тренування KNN – моделі

Для тренування KNN моделі необхідно імпортувати бібліотеки з набору scikit-learn: TfidfVectorizer та NearestNeighbors. Перша бібліотека необхідна, щоб створити вектори із контенту фільмів, друга – щоб натренувати модель на основі цих векторів. Також імпортується бібліотека Pickle, яка необхідна для збереження даних моделей та TF-IDF матриць у файли формату «.pkl» для зручного відкриття в інших ділянках коду. Бібліотеки pandas та numpy є золотим стандартом у керуванні наборами даних, тому їх також необхідно підключити до проєкту.

Після імпорту бібліотек завантажимо наш набір даних та створимо тимчасову колонку в ньому, яка буде вміщати об’єднаний опис усіх параметрів елементу, а саме: огляд фільму, акторський склад, ключові слова, жанри, та режисерський склад. Так як просто об’єднаний набір таких даних не дасть конкретного результату, необхідно додати ваги до окремих параметрів. Для цього необхідно створити два методи, а саме emphasize_first_actor() та emphasize_first_genre(), які повертають оновлений акторський склад та жанри (лістинг 3.6).

Лістинг 3.6 – Функції збільшення ваг emphasize()

```
def emphasize_first_actor(cast_str):
    actors = cast_str.split(" | ")
    main_actor = actors[0].replace(" ", "_")
    emphasized = f"{main_actor} " * 15
    other_actors = " ".join([a.replace(" ", "_") for a in actors[1:]])
if len(actors) > 1 else ""
```

```
        return emphasized.strip() + " " + other_actors
    def emphasize_first_genre(genre_str):
    main_genre = f"{genres[0]} {genres[0]} {genres[0]} {genres[0]}
{genres[0]} return main_genre.strip()
```

Важливим уточненням, без якого неможливі коректні рекомедації, є перетворення формату імені акторів з «John Smith» на формат «John_Smith». Ця заміна порожнього простору прибирає можливість рекомендацій, основуючись на імені або прізвищі актора.

Після цього створимо конструкцію, яка записує до тимчасової колонки параметри з вагами, утворюючи один набір слів, який потім буде використаний в функції векторизації (лістинг 3.7).

Лістинг 3.7 – Об’єднання параметрів із вагами

```
movies2["content"] = (
    movies2["overview_en"].fillna("").apply(clean_text) + " " +
    movies2["cast"].fillna("").apply(emphasize_first_actor) + " " +
    movies2["keywords"].fillna("").str.repeat(8) + " " +
 movies2["genres"].fillna("").apply(emphasize_first_genre).str.repeat(3+
    movies2["cast"].fillna("").apply(emphasize_first_actor) + " " +
    movies2["keywords"].fillna("").str.repeat(8) + " " +
    (movies2["director"].fillna("")+ " ").str.repeat(3)  +
    movies2["genres"].fillna("").apply(emphasize_first_genre).str.repeat
(3)).str.replace("|", " ")
```

Далі перетворюємо текстовий контент тимчасової колонки на вектори, використовуючи раніше імпортований TfidfVectorizer (лістинг 3.8).

Лістинг 3.8 – Створення TfiidfVectorizer

```
tfidf = TfidfVectorizer(
    stop_words="english",
    max_features=10000,
    ngram_range=(1, 2),
    min_df=2,
    max_df=0.7,
tfidf_matrix = tfidf.fit_transform(movies2["content"])
```

Необхідно зазначити параметри, за якими буде відбуватися векторизація. Спочатку вказується словник, за яким будуть фільтруватися

такі слова як for, of, at, year тощо, так як вони не містять жодної уточнювальної інформації. Вказуємо ngram_range=(1,2), щоб створювати списки. Враховуються лише терміни, що зустрічаються щонайменше у 2 фільмах (min_df=2) і не частіше, ніж у 70% усіх (max_df=0.7) [22].

Тепер, коли всі дані оброблені та готові до використання, необхідно провести тренування моделі, та зберегти у результи саму модель, TF-IDF матрицю та TF-IDF векторизатор (лістинг 3.9).

Лістинг 3.9 – Тренування та збереження файлів

```
nn = NearestNeighbors(n_neighbors=4, metric="cosine", algorithm="brute")
nn.fit(tfidf_matrix)

with open("nearest_neighbors.pkl", "wb") as f:
    pickle.dump(nn, f)
with open("tfidf_matrix.pkl", "wb") as f:
    pickle.dump(tfidf_matrix, f)
with open("tfidf.pkl", "wb") as f:
    pickle.dump(tfidf, f)
```

Тренування моделі займає приблизно 2 хвилини. Після цього готову модель можна використовувати в програмній реалізації.

3.2.2 Тренування Truncated SVD

Перед тренуванням моделі необіхдно завантажити попередньо зібраний набір даних з оцінками користувачів до бази даних. Це необхідно для зменшення обчислювальної складності при повторному тренуванні моделі після додавання оцінок від нових користувачів.

Після цього, отримуємо готову таблицю об’ємом 8 тисяч користувачів та 25 мільйонів оцінок до 44 тисяч фільмів. Тепер необхідно додати до цієї таблиці список оцінок від усіх реальних користувачів. Для цього була створена функція get_combined_ratings_data(), яка в результаті роботи повертає об’єднаний масив даних усіх користувачів. Далі виконується тренування Truncated SVD моделі із кількістю найбільш важливих ознак

(num_factors) для використання у програмі. Спершу отримуються всі наявні оцінки користувачів, автоматично будується відповідність між унікальними користувачами та фільмами, створюється розріджена матриця рейтингу, де рядки відповідають користувачам, а стовпці – опрацьованим кінострічкам. Основна частина тренування моделі відбувається за допомогою методу TruncatedSVD (лістинг 3.10) [23].

Лістинг 3.10 – Тренування Truncated SVD моделі

```
svd = TruncatedSVD(n_components=num_factors, random_state=42)
U = svd.fit_transform(rating_matrix)
sigma = svd.singular_values_
Vt = svd.components_
```

Далі ці компоненти разом з індексами користувачів і фільмів зберігаються у файл svd_model.pkl для подальшого використання в системі рекомендацій.

3.3 Розробка серверних компонентів

Серверна частина системи реалізована за допомогою мови програмування Python. У проєкті застосовано бібліотеку Flask API, яка оптимально відповідає поставленим завданням. Для розробки та тестування застосунку використовувався редактор коду VS Code.

3.3.1 Використання SQLite у створенні бази даних

SQLite є реляційною базою даних, яка функціонує без потреби в окремому сервері та зберігається як звичайний файл на диску. Таблиці в такій базі можна створити за допомогою методу cursor.execute(), вказавши необхідний SQL-запит [24].

На рисунку 3.2 схематично зображений результат створення усіх необхідних таблиць у базі даних та їхні зв’язки між собою.

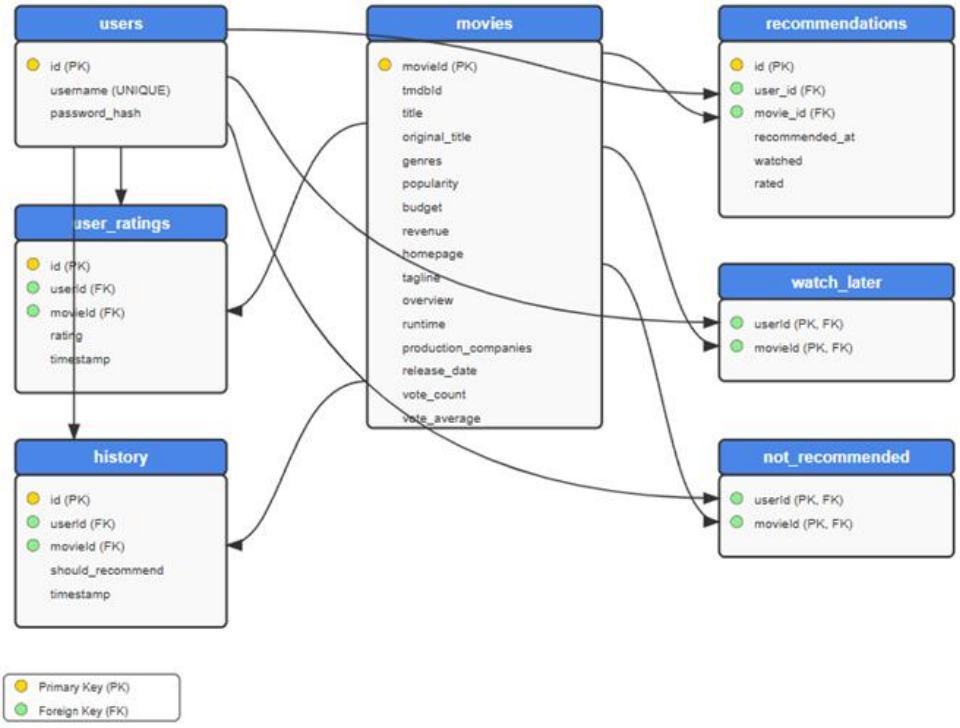

Рисунок 3.2 – Схема створеної бази даних

Для того, щоб застосунок задовольняв необхідні умови для користування ним багатьма людьми, було створено такі таблиці:

– Users – зберігання даних зареєстрованих користувачів. Таблиця містить автоматично створений ідентифікатор користувача, унікальне ім’я користувача та хеш пароля;

– User_ratings – зберігає оцінки, поставлені зареєстрованими користувачами. Таблиця пов’язана з таблицею користувачів і фільмів за зовнішніми ключами;

– Movies – містить повну інформацію про фільми. До неї входять такі поля, як унікальний ідентифікатор фільму, назва, жанр, бюджет, прибуток, рейтинг, мова, дата виходу, список акторів, режисер, ключові слова, опис українською та англійською мовами;

– History – відображає історію взаємодії користувачів із фільмами;

– Recommendations – зберігає дані про фільми, які були рекомендовані конкретному користувачу;

– Watch_later – надає користувачам можливість додавати фільми до списку для перегляду в майбутньому;

– Not_recommended – використовується для зберігання фільмів, які користувач виключив із списку своїх рекомендацій.

Після створення бази даних необхідно завантажити до таблиці movies всю інформацію про фільми із створеного набору даних. Функція load_movies зчитує фільми з CSV-файлу та зберігає їх у таблицю movies бази даних SQLite. В процесі роботи функція обробляє кожний рядок та вставляє дані у відповідні стовпці, ігноруючи дублікати за допомогою обробки винятків IntegrityError.

3.3.2 Початкове завантаження моделей та фільмів

При кожному запуску застосунку повинні завантажуватися натреновані моделі та під’єднуватися фільми із бази даних. Для цього була використана бібліотека pickle, яка дозволяє як запаковувати так і розпаковувати моделі без додаткових методів [25].

Створимо функцію пошуку фільмів за векторизованими назвами українською та англійською мовами (лістинг 3.11).

Лістинг 3.11 – Пошук векторизованих назв фільмів

```
def find_best_matches(movie_title, top_n=10):
    input_vector = vectorizer.transform([movie_title])

    similarities=cosine_similarity(input_vector, title_matrix).flatten()
    best_match_indices = np.argsort(similarities)[-top_n:][::-1]
    results = []

    for idx in best_match_indices:
        similarity_score = float(similarities[idx])
        if similarity_score > 0:
```

3.3.3 Реалізація рекомендацій

Функція find_similar_movies() відповідає за пошук фільмів, які є схожими до заданого фільму використовуючи для цього контентний підхід. У цій функції використовується матриця ознак, сформованої за допомогою TF-IDF (методу зважування важливості слів та фраз у текстах), і попередньої навченої моделі пошуку найближчих сусідів [26]. Розробка функції пошуку фільмів find_similar_movies() продемонстрована на лістингу 3.12.

Лістинг 3.12 – Функція find_similar_movies()

```
def find_similar_movies(movie_id, top_n=20):
    conn = sqlite3.connect("database.db")
   movies = pd.read_sql("SELECT movieId, original_title FROM movies",
conn)
    conn.close()
      try:      movie_idx      =      movies[movies["movieId"]      ==
movie_id].index[0]
distances, indices =
nn.kneighbors(tfidf_matrix[movie_idx], n_neighbors=top_n + 1)
 for                 i,                  rec_movie_id                 in
enumerate(movies.iloc[indices[0][1:]]["movieId"]):
            similarity_score = 1 - distances[0][i+1]
            recommendations.append((rec_movie_id, similarity_score))
        return recommendations
```

Функція для взаємодії з моделлю SVD get_svd_recommendations() формує список рекомендованих фільмів для заданого користувача на основі матриці сингулярного розкладу (SVD). В процесі роботи автоматично перевіряється, чи існує користувач у словнику індексів, і якщо так, то отримується його вектор переваг, який перемножується з матрицею фільмів та формується список передбачених рейтингів (лістинг 3.13).

Лістинг 3.13 – Функція get_svd_recommendations

```
user_vector = np.dot(U[user_index], sigma)
predicted_ratings = np.dot(user_vector, Vt)
all_recommendations = [(index_to_movie[idx], score) for idx, score in
enumerate(predicted_ratings)]
all_recommendations.sort(key=lambda x: x[1], reverse=True)
```

```
filtered_recommendations = [(movie_id, score) for movie_id, score in
all_recommendations if movie_id not in excluded_movies]
top_recommendations = dict(filtered_recommendations[:top_n])
return top_recommendations
```

Після створення окремих функцій, які взаємодіють з моделями, проводиться їхня інтеграція в одну функцію, яка буде використовувати їх відповідно до обраних користувачем методів рекомендації. Для цього на початку функції реалізовано отримання параметрів, за якими функція буде рекомендувати: за усіма схожими фільмами, тільки за фільмами які були оцінені та за схожими прихованими ознаками.

Після ініціалізації реалізовано виклик функцій рекомендацій фільмів (лістинг 3.14), які будуть повертати фільми у словник recommendation_scores. Це словник, у якому зберігається вся проміжна інформація про кожен рекомендований фільм: скільки балів отримано від кожного з методів (KNN, Ratings_KNN, SVD) і скільки разів загалом була рекомендована та чи інша кінострічка.

Лістинг 3.14 – Виклик функцій та повертання балів

```
recommendation_scores = defaultdict(lambda: {"knn": 0, "ratings": 0,
"svd": 0, "count": 0})
    # 2. KNN рекомендації (якщо увімкнено)
    if filters.get('use_knn', True):
        for movie_id in watched_movies:
            similar_movies = find_similar_movies(movie_id, top_n)
   # 3. Item-based рекомендації з оцінок (якщо увімкнено)
    if filters.get('use_ratings', True):
        for movie_id, rating in rated_movies.items():
            similar_movies = find_similar_movies(movie_id, top_n)
   # 4. SVD-рекомендації (якщо увімкнено)
    if filters.get('use_svd_based', False):
        try:
svd_recs = get_svd_recommendations(user_id, excluded_movies, top_n + 1)
```

Фінальне фільтрування оцінок передбачає додавання до них ваг, які були зазначені на старті функції та підсумомування фінального рахунку за всіма трьома методами разом. Якщо якогось джерела немає то ставиться 0. Отримані результати потрібно фільтрувати за параметрами, які були вказані

на старті функції, а саме, за жанром фільмів, оцінками та тривалістю. Всі фільми, які не підходять за цими параметрами, не будуть показані користувачу, якщо він застосував фільтри. Після фільтрації функція повертає результат користувачу.

3.3.4 Створення кінцевих точок у backend

Для повноцінної роботи веб-застосунку необхідно створити кінцеві точки в серверній частині, які будуть викликатися при виборі користувачем того чи іншого пункту або маршруту. Ключовим точкою у рекомендаційній системі є метод GET «/recommendations» (лістинг 3.15), який викликає раніше розроблену функцію рекомендацій та повертає результат у JSON [27].

Лістинг 3.15 – Метод recommendations

```
@app.route("/recommendations", methods=["GET"])
def recommendations():
      user_id = request.args.get("user_id")
    filters = {
  'use_knn': request.args.get('use_knn', 'true').lower() == 'true',
  'use_ratings':   request.args.get('use_ratings',   'true').lower()   ==
'true',
'use_svd_based':  request.args.get('use_svd_based',  'true').lower()  ==
'true',
'categories': request.args.get('categories', '').split(','),
'duration_min': int(request.args.get('duration_min')),
'duration_max': int(request.args.get('duration_max')),
'rating_min': float(request.args.get('rating_min')) ,
'rating_max': float(request.args.get('rating_max')) ,
recommendations     =     get_recommendations(user_id,     top_n=top_n,
filters=filters)

    return jsonify(recommendations)
```

Далі було розроблено інші кінцеві точки, які необхідні для додавання, видалення оцінок, пошуку та керуванню інформацією про фільми:

– /rate_movie [POST] – приймає user_id, movie_id та rating. Зберігає оцінку фільму в базу даних та позначає фільм як переглянутий;

– /get_rated_movies [GET] – повертає всі фільми з оцінками для

переданого user_id. Використовується для відображення історії оцінювання;

– /remove_movie_rating [POST] – видаляє оцінку конкретного фільму для користувача. Очищає запис із таблиці user_ratings;

– /find_movie [POST] – шукає фільми, назва яких частково чи повністю збігається з введеною. Повертає до 10 найкращих збігів;

– /register [POST] – реєструє нового користувача, хешує пароль і зберігає його в базу. Повертає JWT-токен;

– /login [POST] – перевіряє логін і пароль. Якщо все вірно – повертає JWT-токен і дані користувача.

Створені кінцеві точки охоплюють повний цикл взаємодії користувача з системою: від оцінювання та пошуку до управління списками.

3.4 Розробка клієнтської частини застосунку

Frontend застосунку реалізований мовою Javascript використовуючи фреймворк React, який має необхідний функціонал для цього. Під час реалізації клієнтської частини необхідно надати користувачу такі можливості:

- реєструватися та авторизуватися;
- переглядати свої оцінені та переглянуті фільми, статистику;
- отримувати персоналізовані рекомендації;
- шукати фільми за назвами;
- отримувати інформацію про фільми;
- повідомляти адміністрацію про проблеми на сайті.

3.4.1 Реєстрація та авторизація

Щоб користувач міг додавати фільми до своєї колекції, а потім отримувати щодо них рекомендації, його логін повинен бути у базі даних. Для цього потрібно реалізувати реєстрацію на сайті. Спочатку створимо

головний клас User, який буде вміщувати в собі методи register та login (лістинг 3.16).

Лістинг 3.16 – Методи login та register

```
async login(username, password) {
    try {
  const response = await axios.post("/login", { username, password
});
      console.log("Відповідь від сервера:", response.data);
      if (response.data.access_token) {
        this.userId = response.data.user_id;
        this.username = username;
        this.token = response.data.access_token;
async register(username, password) {
    try {
    const response = await axios.post("/register", { username, password
});
      console.log("Відповідь від сервера:", response.data);
      if (response.data.access_token) {
        this.userId = response.data.user_id;
        this.username = username;
        this.token = response.data.access_token;
```

Маючи ці методи реалізуємо сторінки для входу та реєстрації. Компоненти Login та Register відповідають за взаємодію з користувачем через форму. Коли користувач вводить логін або пароль на обох сторінках і натискає кнопку, спрацьовують функції – обробники подій handle, які викликають відповідний метод. Після отримання позитивного результату обидва методи спрямовують користувача на сторінку профілю [28].

3.4.2 Пошук фільмів та отримання інформації

Після того, як користувач зареєструвався на сайті, йому необхідно додати фільми до своєї колекції переглянутих та оцінених. Для цього створена сторінка пошуку фільмів FindMovie.

Після введення користувачем повної назву фільму, ключового слова або ж частини назви, англійською або українською мовами, автоматично спрацьовує асинхронний обробник подій searchMovie (лістинг 3.17).

Виконується перевірка чи введено назву, надсилається POST-запит на сервер, обробляється відповідь, оновлюється стан (знайдені фільми, повідомлення, завантаження постерів). Якщо виникає помилка або фільм не знайдено, користувач отримує відповідне повідомлення.

Лістинг 3.17 – Обробник подій searchMovie

```
const searchMovie = async () => {
    if (!movieTitle.trim()) {
      setMessage("Введіть назву фільму для пошуку");
      return;}
    try {
    const response = await axios.post("/find_movie", { movie_title:
movieTitle});
```

Після того, як користувач знайшов фільм у пошуку, йому необхідно його оцінити, поставити відмітку про перегляд або ж просто переглянути детальну інформацію про цей фільм. Для цього створено окрему сторінку MovieDetails та відповідні методи для обробки подій для забезпечення необхідного функціоналу (лістинг 3.18)

Лістинг 3.18 – Обробники подій сторінки MovieDetails

```
const markAsWatched = async () => {
await axios.post("/add_to_history", { user_id: userId, movie_id: movieId
});
      setStatus(prev => ({ ...prev, watched: true }));};
  const rateMovie = () => {setShowRatingModal(true);};
  const handleRateMovie = async (rating) => {
await axios.post("/rate_movie", { user_id: userId, movie_id: movieId,
rating });
      setStatus(prev => ({ ...prev, rated: true, rating }));};
  const removeRecommendation = async () => {
      await  axios.post(`/not_recommended/${userId}/add`,  {  user_id:
userId, movie_id: movieId });
      setStatus(prev => ({ ...prev, not_recommended: true }));};
  const addToWatchlist = async () => {
 await  axios.post(`/watch_later/${userId}/add`,  {  user_id:  userId,
movie_id: movieId });
    setStatus(prev => ({ ...prev, watch_later: true }));  };
```

markAsWatched надсилає запит, щоб додати фільм до історії переглядів користувача і оновлює стан як "переглянуто". rateMovie лише відкриває

модальне вікно для оцінювання, а handleRateMovie надсилає оцінку на сервер. removeRecommendation додає фільм до списку «не рекомендованих», змінює відповідний статус і повертає користувача назад на попередню сторінку. addToWatchlist дозволяє додати фільм у список «переглянути пізніше» й оновлює локальний стан для відображення цього статусу.

Перед тим як отримати інформацію про фільм використовується функція fetchData, яка збирає інформацію з двох запитів до backend: /movie, який надає інформацію саме про цей фільм та /movie_status – надає інформацію про поточні відмітки, які виставив користувач щодо цього фільму, чи переглянув користувач цю кінострічку, на який бал її було оцінено, тощо (лістинг 3.19).

Лістинг 3.19 – Фунція fetchData

```
    const fetchData = async () => {
        const [movieRes, statusRes] = await Promise.all([
          axios.get(`/movie/${movieId}`),
          axios.get(`/movie_status/${userId}/${movieId}`)]);
        if (movieRes.data.tmdbId) {
      const tmdbRes =
 await
axios.get(`https://api.themoviedb.org/3/movie/${movieRes.data.tmdbId}`,
{
            params: { api_key: "" } });
          setTmdbData(tmdbRes.data); }};
    fetchData();
  },
[movieId, userId]);
```

Реалізовано модальне вікно для оцінювання фільму, яке використовується rateMovie. Користувач може виставити оцінку від 0 до 10 із кроком 0.5, після чого ця оцінка зберігається, передається головному компоненту через onRate, і модальне вікно автоматично закривається [29].

Важливою частиною реалізації візуальної складової про фільми є постери. Отримавши відповідь від TMDB API реалізовано візуальне представлення постеру та фону сторінки (лістинг 3.20).

На сторінку також було додано опис фільму, його середня оцінка,

ключові слова, акторський склад, тривалість, дата виходу жанри та інші деталі [30].

Лістинг 3.20 – Зображення постеру та фону фільму на сторінці

```
<div className="movie-details-movie-backdrop" style={{
        backgroundImage: tmdbData?.backdrop_path ?
       url(https://image.tmdb.org/t/p/original${tmdbData.backdrop_path}`
      }}>


        <div className="movie-details-movie-info-container">
          <div className="movie-details-poster-container">
            {tmdbData?.poster_path ? (
              <img
      src={`https://image.tmdb.org/t/p/w500${tmdbData.poster_path}
      `}


                alt={movie.original_title}
                className="movie-details-movie-poster"/>))} </div>
```

3.4.3 Профіль користувача

Після виставлення оціненки та перегляду фільмів користувач може побачити переглянуті фільми у своєму профілі. Для реалізації сторінки профілю користувача на backend вже були створені таблиці rated_movies, history та watch_later, в яких зберігаються дані про фільми які користувач вже оцінив, переглянув, чи перегляне пізніше. Також на сторінці профілю відображається статистика користувача.

На сторінці Profile було реалізовано функцію renderMovieCarousel для відображення фільмів, яка приймає масив фільмів, назву секції, заголовок і шлях до повної сторінки цієї секції. Якщо фільми ще завантажуються, відображається повідомлення «Завантаження фільмів...». Якщо масив фільмів порожній, функція відображає повідомлення «Список порожній». У випадку, коли фільми наявний в списку, функція відображає тільки 4 фільми одночасно. Це реалізовано за допомогою зрізу масиву, який визначається на основі активного індексу секції (рисунок 3.3).

Список має дві кнопки прокрутки: одну для перемотування вліво та вправо. У правому верхньому куті кожної секції також є кнопка «Переглянути всі», яка веде на окрему сторінку зі списком усіх фільмів у цій категорії за датою додавання.

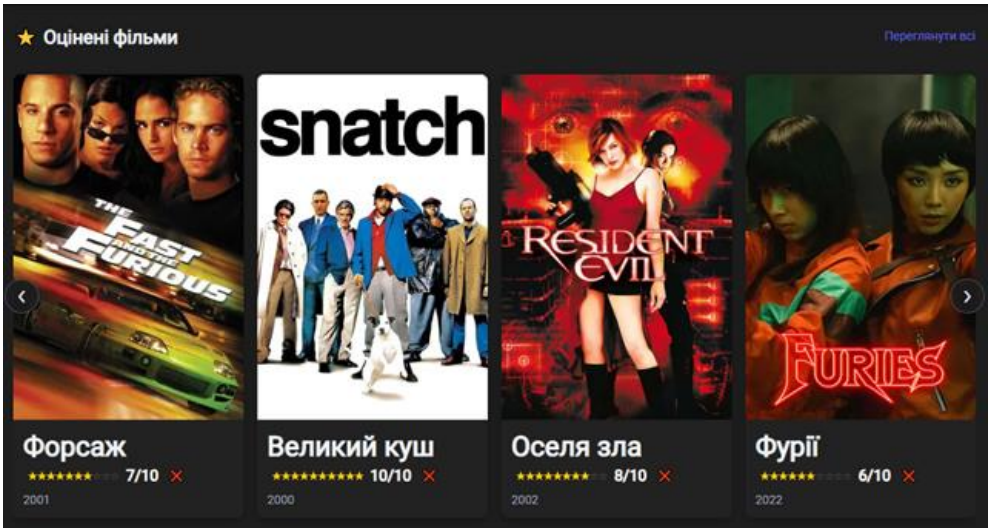

Рисунок 3.3 – Список оцінених фільмів

Сторінка профілю викликає два модальних компоненти RecommendationStats та UserStatsChart для забезпечення відображення статистики. Це статистика по переглянутим фільмам, а саме час перегляду кожного жанру, та статистика рекомендацій, яка відображає, скільки рекомендованих фільмів користувач переглянув, яку оцінку поставив та улюблений жанр.

3.4.4 Сторінка рекомендацій

Після сторення всіх інших сторінок, які необхідні для управління фільмами, перейдемо до побудови сторінки з рекомендаціями. Спочатку відбувається звернення до backend функції get_recommendations(), надсилаючи параметри відповідно вибору користувача на сайті. За

надсилання запиту з параметрами відповідає функція-оброблювач fetchRecommendations (лістинг 3.21)

Лістинг 3.21 – Функція fetchRecommendations

```
const fetchRecommendations = async () => {
    setLoading(true);
    try {const params = {
        user_id: User.userId,
        use_knn: filters.use_knn,
        use_ratings: filters.use_ratings,
        use_svd_based: filters.use_svd_based,
        categories: filters.categories.length
        > 0 ? filters.categories.join(',') : '',
        duration_min: filters.duration_min,
        duration_max: filters.duration_max,
        rating_min: filters.rating_min,
        rating_max: filters.rating_max,
        top_n: 40  };};
```

Було створено елементи вибору та функцію-обробник подій handleInputChange, яка також відповідає за параметри фільтрування отримуючи інформацію про тривалість і середню оцінку (лістинг 3.22).

Лістинг 3.22 – Обробник подій HandleInputChange

```
const handleInputChange = (e) => {
    const { name, value, type, checked } = e.target;
    if (type === 'checkbox') {
      setFilters({ ...filters, [name]: checked });
    } else if (name === 'duration_min') {
      const minVal = parseInt(value);
      const maxVal = parseInt(filters.duration_max);
    } else if (name === 'duration_max') {
      const minVal = parseInt(filters.duration_min);
      const maxVal = parseInt(value);
    } else if (name === 'rating_min') {
      const minVal = parseFloat(value);
      const maxVal = parseFloat(filters.rating_max);
    } else if (name === 'rating_max') {
      const minVal = parseFloat(filters.rating_min);
      const maxVal = parseFloat(value); }  };
```

Компонент також включає функцію очищення збережених даних, яка видаляє всі збережені рекомендації та фільтри з localStorage та очищає стан компонента до значень за замовчуванням.

3.4.5 Зв’язок з адміністрацією та управління

Використовуючи програму, користувачі можуть зіткнутися із різного роду проблемами, які не були передбачені при створенні веб-застосунку. Тому, користувачам була надана можливість повідомляти адміністрацію про помилки в системі або некоректність роботи системи рекомендацій.

Для цього створено модальне вікно ReportBugModal, в основі якого знаходиться кілька змінних стану: title для зберігання заголовка звіту, description для опису проблеми, priority для встановлення пріоритету, isSubmitting для відстеження процесу надсилання форми, error для зберігання повідомлень про помилки та success для відображення успішного надсилання звіту про помилку (лістинг 3.23).

Лістинг 3.23 – Компонент ReportBugModal

```
const ReportBugModal = ({ onClose }) => {

  const handleSubmit = async (e) => {

    setIsSubmitting(true);
      const response = await fetch('/bug-report', {


        body: JSON.stringify({
          title,
          description,
          priority
        })});

      setTimeout(() => {
        onClose();
      }, 2000);
  };
```

Функція handleSubmit використовується для надсилання форми звіту про помилку. Спочатку автоматично перевіряється, чи заповнені обов'язкові поля, після цього очікується відповідь від серверу. Якщо поля не заповнені, або відбувалася помилка встановлюється відповідне повідомлення.

Розроблена окрема сторінка зі списком усіх відправлених користувачем

повідомлень, в якій відображається поточний статус заявки. Функція UserBugReports завантажує дані про помилки з бази даних (лістинг 3.24).

Лістинг 3.24 – Сторінка UserBugReports

```
const UserBugReports = () => {
  useEffect(() => {
const fetchReports = async () => {
        const response = await fetch('/my-bug-reports', {
headers: {'Authorization': `Bearer ${userInstance.getToken()}`}});},[]);
```

AdminDashboard перевіряє, чи має користувач рівень доступу «адміністратор», надсилаючи запит до /admin/users. Якщо доступ дозволено, відображається панель керування для адміністраторів, в іншому випадку користувач перенаправляється на головну сторінку. Також AdminDashboard викликає ці два компоненти на сторінці керування (лістинг 3.25).

Лістинг 3.25 – Сторінка AdminDashboard

```
  return (
      <h1>Адміністративна панель</h1>
      <div className="admin-tabs">
<button
      className={`tab-button ${activeTab === "reports" ? "active" :""}`}
      onClick={() => setActiveTab("reports")}>Звіти про помилки</button>
       <button
       className={`tab-button ${activeTab === "users" ? "active" : ""}`}
      onClick={() => setActiveTab("users")}> Користувачі</button></div>

<div className="tab-content">
        {activeTab === "reports" && <AdminBugReports />}
        {activeTab === "users" && <AdminUsers />}</div></div>);};
```

Перший компонент AdminBugReports містить в собі всі повідомлення від користувачів. Використовуючи функцію fetchReports виконується запит на backend щоб отримати список повідомлень.

Компонент містить функції – обробники handleStatusChange, який змінює статус звіту, відправляючи PUT-запит із новим значенням і токеном авторизації, та handlePriorityChange оновлює пріоритет звіту.

Другий компонент AdminUsers відповідає за сторінку адміністративного керування користувачами. Після завантаження сторінки відбувається запит до серверу з використанням токена авторизації, щоб отримати список користувачів.

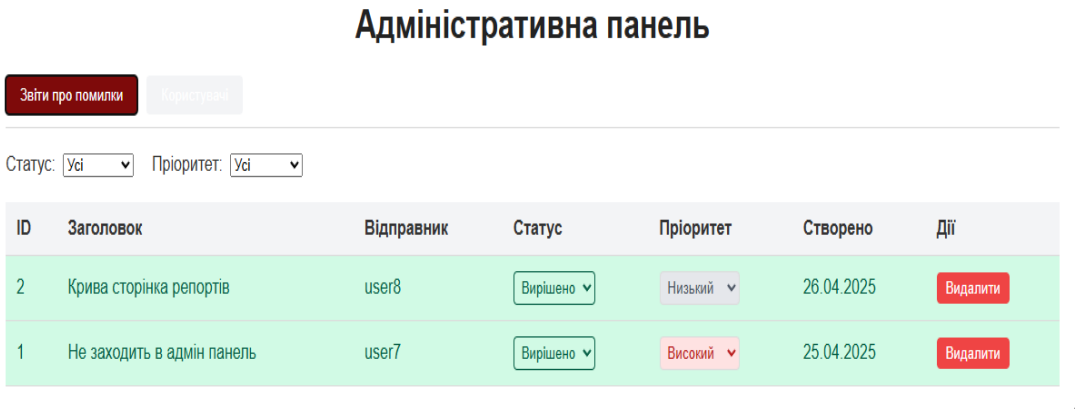

Рисунок 3.4 – Адміністративна панель

Результат створення адміністративної панелі показаний на рисунку 3.4, демонструючи звіти про помилки, які були надіслані від різних користувачів, та можливості адміністратора.

4 ІНСТРУКЦІЯ КОРИСТУВАЧА

4.1 Реєстрація та авторизація

При першому підключенні на сайт, користувач не має можливості шукати фільми та отримувати рекомендації. Для отримання доступу до всіх функцій користувач повинен зареєструватися на сайті (рисунок 4.1).

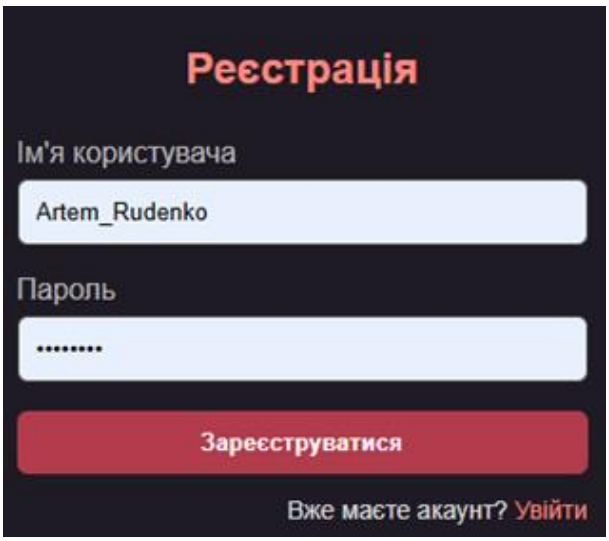

Рисунок 4.1 – Реєстрація користувача

Якщо користувач захоче зареєструватися за іменем яке вже існує в системі, то автоматично отримується повідомлення, що такий користувач вже існує. Якщо ж користувач вже зареєструвався, є можливість авторизуватися, натиснувши кнопку «Увійти». Після цього, сайт перенаправляє користувача на сторінку авторизації, яка відрізняється від сторінки реєстрації тим, що відбувається не додавання користувача, а проста перевірка на його наявність у базі та правильність введеного паролю. Якщо користувач ввів неправильний пароль або ім’я то відображається помилка, що логін або пароль невірні.

4.2 Основні функції

Натиснуваши кнопку «Пошук» у верхній частині веб-застосунку, користувач переходить до пошукової сторінки (рисунок 4.2), яка надає можливість знайти потрібний фільм, щоб переглянути про нього інформацію, або додати його до свого списку рекомендацій.

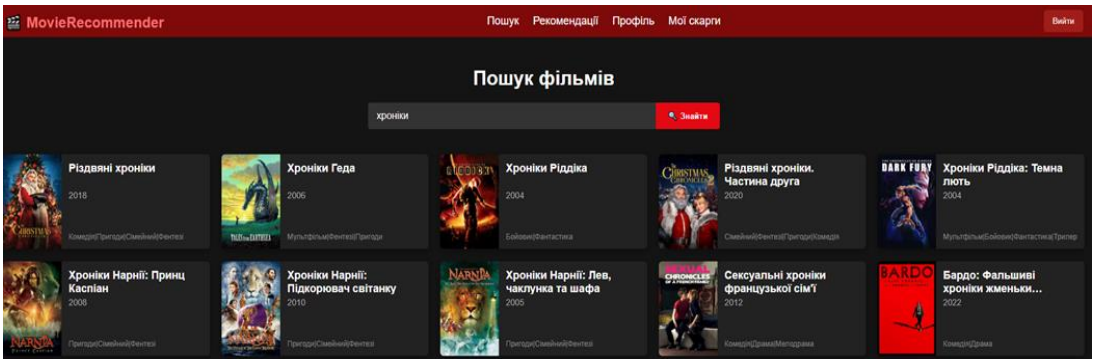

Рисунок 4.2 – Сторінка пошуку

Коли користувач натискає на картку фільму, він перенаправляється на окрему сторінку з детальною інформацією про цей фільм. (рисунок 4.3).

Рисунок 4.3 – Сторінка з інформацією про фільм

На цій сторінці можна позначити фільм, як переглянутий, щоб додати до списку рекомендацій, оцінити фільм (рисунок 4.4) та виключити його зі

списку рекомендацій. Також користувач може відкласти перегляд фільму на інший час, додавши його до окремого списку «Переглянути пізніше». Такий фільм не буде рекомендуватися та приймати участі в створені рекомендацій.

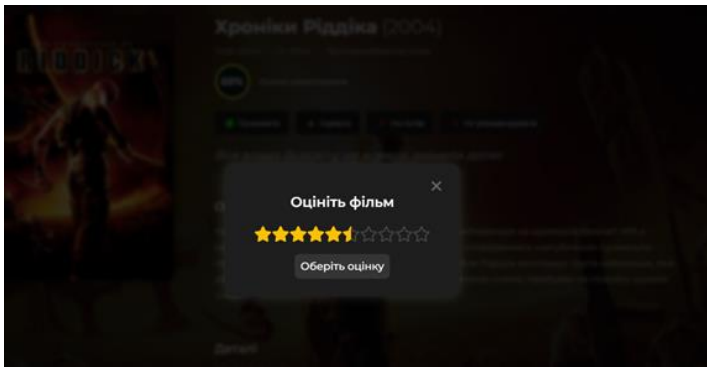

Рисунок 4.4 – Вікно оцінювання фільму

Після того, як користувач додав усі переглянуті фільми до своєї історії, оцінив деякі з них та додав до пункту «переглянути пізніше», можна побачити їх на сторінці профілю (рисунок 4.5). Масштаб відображення у браузері було зменшено для наглядовості.

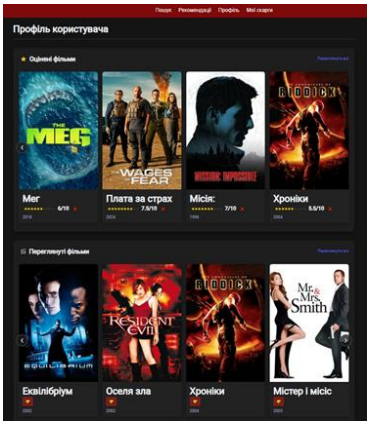

Рисунок 4.5 – Сторінка профілю користувача з переглянутими фільмами

Прогорнувши вниз сторінки, користувач зможе побачити свою статистику щодо переглянутих фільмів (рисунок 4.6), а саме кількість

переглянутих хвилин по жанрам окремо, та рекомендаційну статистику, яка демонструє скільки фільмів, які порекомендувала система, оцінив користувач або було переглянуто.

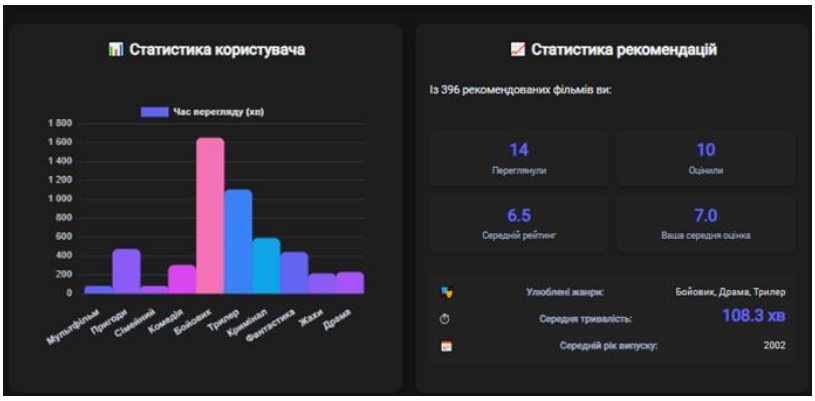

Рисунок 4.6 – Статистика користувача

Також користувач може розгорнути списки з фільмами, до яких була поставлена оцінка, або були переглянуті.

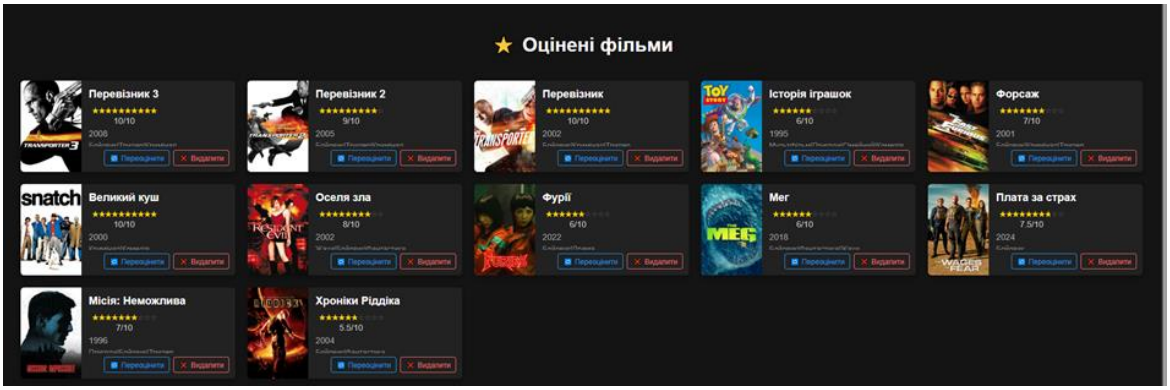

Рисунок 4.7 – Сторінка оцінених фільмів

Натиснувши на кнопку «Переглянути всі» біля відповідного списку з фільмами, користувач перейде на окрему сторінку з усіма фільмами певної категорії (рисунок 4.7). Користувач має можливість змінювати або видаляти оцінку на сторінці з оціненими фільмами.

На сторінці з переглянутими фільмами (рисунок 4.8) користувач може натиснути «не рекомендувати», щоб система не спиралася на цей фільм при побудові персональних рекомендацій.

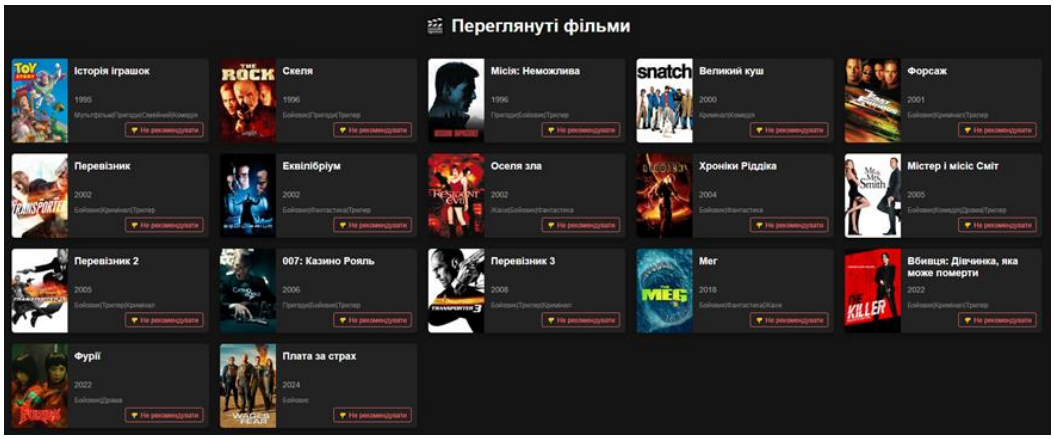

Рисунок 4.8 – Сторінка переглянутих фільмів

Натиснувши на кнопку «рекомендації», користувач переходить до сторінки, де може отримати рекомендації на основі своїх переглянутих та оцінених фільмів (рисунок 4.9).

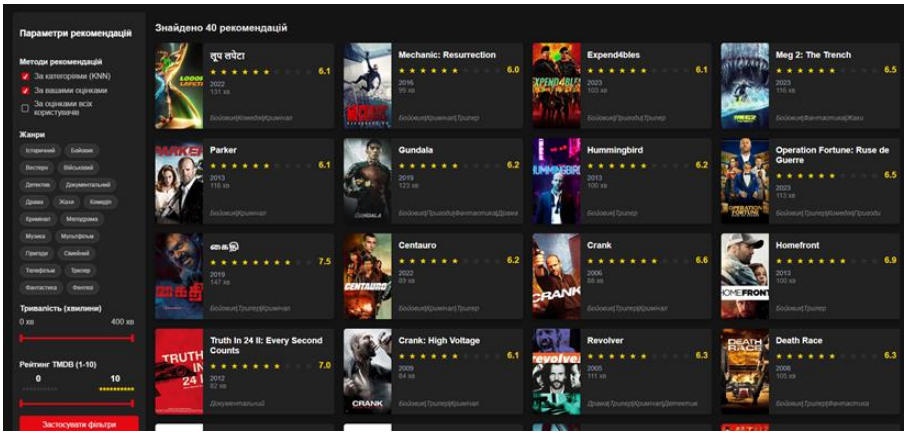

Рисунок 4.9 – Сторінка рекомендацій

Перед користувачем відкривається інтерфейс вибору параметрів, з якими можна взаємодіяти та налаштовувати список рекомендацій.

Обравши в параметрах рекомендацій «За оцінками всіх користувачів», відображаються рекомендації, які спираються на оцінки та досвід від інших користувачів, які вже переглянули ці фільми (рисунок 4.10).

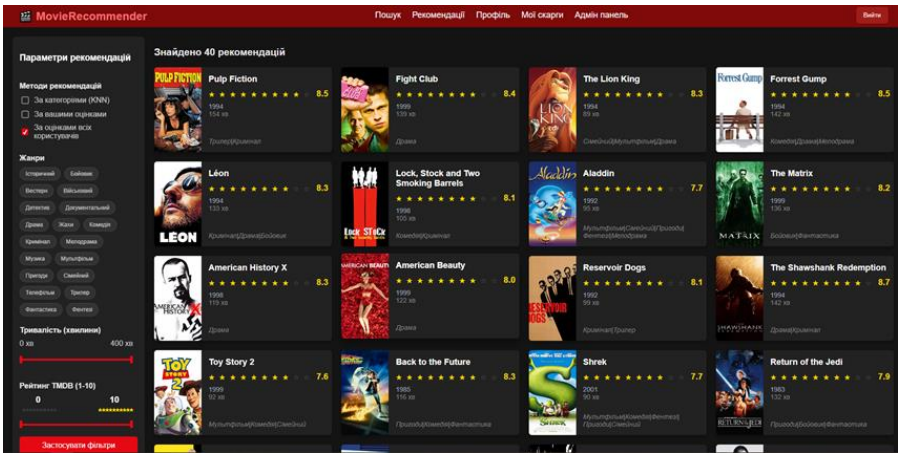

Рисунок 4.10 – Результати рекомендацій від SVD-моделі

Але варто зазначити, що рекомендації на основі SVD-моделі можуть отримувати лише користувачі, які були в базі даних під час тренування моделі. Тому, найкращим рішенням є перетренування моделі, коли активність веб-застосунку мінімальна.

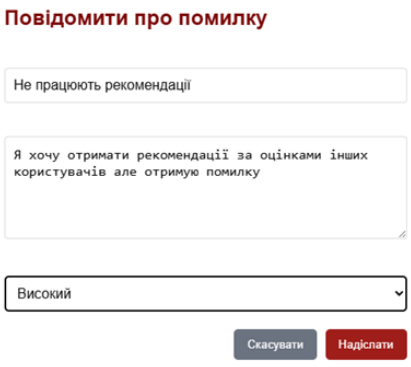

Рисунок 4.11 – Написання повідомлення

Користувач може написати звіт, щодо проблем виявлених в процесі роботи, для цього потрібно натиснути на відповідну кнопку. Після цього

відкриється вікно написання звіту (рисунок 4.11), в якому користувач зможе описати ситуацію. Після цього, адміністратор, може побачити проблему у загальному списку, та відреагувати на неї (рисунок 4.12):

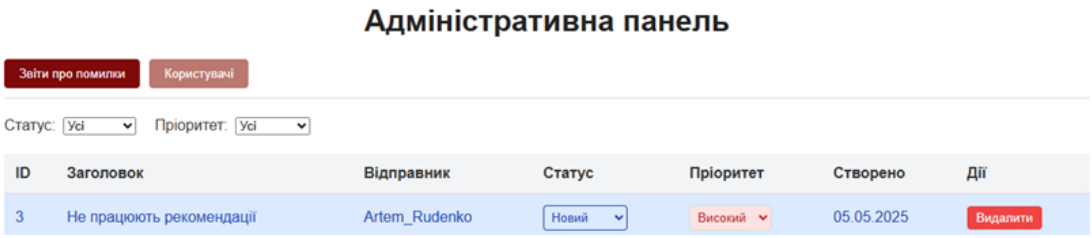

Рисунок 4.12 – Отримання звіту від користувача

Також адмістратор може побачити усіх користувачів, які зареєструвалися на сайті та надавати їм ролі.

Аналіз проблем стандартних підходів дозволив з’ясувати, що більшість популярних систем орієнтуються не на пошук схожих чи цікавих фільмів для конкретного користувача, а на рекомендацію найбільш популярного контенту, що обмежує індивідуалізацію пропозицій.

Розробка алгоритмів рекомендацій та їх інтеграція у веб-застосунок дозволили створити систему, що пропонує користувачам схожі фільми з урахуванням їхніх особистих вподобань та загальних тенденцій серед усіх оцінок. Такий підхід забезпечує більш релевантні та персоналізовані рекомендації для кожного користувача.

Зібрані під час роботи застосунку дані свідчать про значне покращення користувацького досвіду: процес підбору фільмів став простішим, а точність рекомендацій – значно вищою. Завдяки зрозумілому та зручному веб-інтерфейсу користувачі можуть швидко отримувати персоналізовані рекомендації, а також налаштовувати параметри, щоб отримувати результати, які максимально відповідають їхнім смакам та інтересам, що суттєво покращує взаємодію з платформою та задоволення від користування.

ВИСНОВКИ

Кваліфікаційна робота спрямована на реалізацію системи рекомендацій фільмів користувачам, які ще не були переглянуті, базуючись на їхніх вже переглянутих фільмах.

Розглянуто існуючі системи та виявлено обмеження стандартних підходів, зокрема слабку адаптацію до індивідуальних смаків глядача, обмежену точність рекомендацій та нав’язування популярних фільмів заради монетизації.

Розроблений застосунок дозволяє користувачам реєструватися, авторизуватися та отримувати рекомендації, що базуються як на контентних характеристиках фільмів (жанри, опис, акторський склад), так і на методах колаборативної фільтрації, зокрема за допомогою моделей, побудованих із застосуванням Truncated Singular Value Decomposition. Для підвищення якості рекомендацій інтегровано декілька алгоритмів, що використовують різні підходи для оцінки схожості між фільмами. Завдяки цьому система може враховувати як особисті вподобання конкретного користувача, так і загальні тренди серед великої кількості оцінок.

Для покращення користувацього досвіду було реалізовано повноцінний веб-застосунок із клієнтською та серверною частинами який дозволяє користувачу реєструватися, входити в систему та переглядати рекомендації від системи.

Створена система дозволяє максимально точно підбирати фільми, орієнтуючись на інтереси кожного користувача, його попередні перегляди та загальні тенденції у світі кіно. Реалізація застосунку підтверджує актуальність використання гібридних підходів у рекомендаційних системах, а також демонструє потенціал сучасних технологій і методів машинного навчання для створення інтерактивних, зручних та персоналізованих сервісів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Murel J., Kavlakoglu E.. What is collaborative filtering? IBM. URL: www.ibm.com/think/topics/collaborative-filtering (дата звернення: 27.04.2025)

2. Руденко А., Татарников А. Алгоритми машинного навчання для персоналізованих рекомендацій фільмів, VIII Міжнародна студентська наукова конференція. Україна. 2025. С 71–72

3. Ms. Tejashri S. P., Prof. Shivendu B., Content Based Filtering and Collaborative Filtering: A Comparative Study. Journal of Advanced Zoology. India. 2024. vol 45. С 96 – 100.

4. Hybrid Recommendation Systems Overview. Restack.io. URL: www.restack.io/p/recommendation-systems-answer-hybrid-recommendation-systems-cat-ai (дата звернення: 27.05.2025)

5. Руденко А., Татарников А. Гібридні методи рекомендацій фільмів: поєднання колаборативної та контентної фільтрації, VII Всеукраїнська студентська наукова конференція. 2025. С 412–413

6. Dobhal S. Netflix’s Content Recommendations. Product Round. URL: productround.com/netflix-recommendation-engine (дата звернення: 27.05.2025)

7. Minzheong S. A Comparative Study on Over-The-Tops, Netflix & Amazon Prime Video: Based on the Success Factors of Innovation. International Journal of Advanced Smart Convergence. Korea. 2021. Vol.10. С 62-74.

8. Krishnan N. Letterboxd: The future go-to app for film lovers?. Medium. 2024. URL: https://medium.com/%40krishnan357/letterboxd-the-future-go-to-app-for-film-lovers-d9db624d0948 (дата звернення: 28.05.2025)

9. A comprehensive overview of Python. Domo. URL: https://www.domo.com/es/glossary/what-is-python (дата звернення: 29.05.2025)

10. Для чого використовують Jupyter Notebook?. FoxMinded. URL: https://foxminded.ua/jupyter-notebook/ (дата звернення:29.05.2025)

11. API Terms of Use. The Movie Database. URL:

https://www.themoviedb.org/api-terms-of-use (дата звернення: 29.05.2025)

12. Srivastava Tavish. Guide to K-Nearest Neighbors Algorithm in Machine Learning. Analytics Vidhya. 2025. URL: analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering (дата звернення 29.05.2025)

13. Krasniuk A. The Math behind the Singular Value Decomposition. Medium. 2024. URL: https://medium.com/%40krasniuk-ai/the-math-behind-the-singular-value-decomposition-a847abf22fc1 (дата звернення 30.05.2025)

14. Створення RESTful API за допомогою Python і Flask. Sharpcoder blog. URL: uk.sharpcoderblog.com/blog/creating-restful-apis-with-python-and-flask (дата звернення: 30.05.2025)

15. Python in Visual Studio Code. Visual Studio Code. URL: code.visualstudio.com/docs/languages/python (дата звернення: 30.05.2024)

16. What is JavaScript?. Mozilla developer. URL: developer.mozilla.org/enUS/docs/Learn_web_development/Core/Scripting/What_is_JavaScript (дата зверне-ння: 31.05.2025)

17. Колесніков Д. Що таке React JS? Як почати вивчати Реакт? Brainlab. 2025. URL: https://brainlab.com.ua/uk/blog-uk/shho-take-react-js-yak-pochaty-vyvchaty-reakt (дата звернення: 31.05.2025)

18. Prashant Y. The Power of React's Virtual DOM: A Comprehensive Explanation. Synfusion. 2025. URL: www.syncfusion.com/blogs/post/react-virtual-dom (дата звернення: 31.05.2025)

19. How to Extract Data from tmdB using Python. Aionlinecourse. URL: https://www.aionlinecourse.com/blog/how-to-extract-data-from-tmdb-using-python (дата звернення: 01.06.2025)

20. Katiyar S. Introduction to concurrent.futures in Python. Medium. 2024. URL: https://medium.com/@smrati.katiyar/introduction-to-concurrent-futures-in-python-009fe1d4592c (дата звернення:04.06.2025)

21. Ryan. Save and Load JSON Files in Python. Medium. 2024. URL: https://medium.com/%40ryan_forrester_/save-and-load-json-files-in-python-a-complete-guide-49a5760b8b49 (дата звернення: 04.06.2025)

22. Understanding min_df and max_df in scikit CountVectorizer. Geeks for geeks. 2024. URL: https://www.geeksforgeeks.org/understanding-mindf-and-maxdf-in-scikit-countvectorizer/ (дата звернення: 06.06.2025)

23. TruncatedSVD. Scikit-Learn Machine Learning in Python. URL: https://scikitlearn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html (дата звернення: 07.06.2025)

24. SQLite3 – DB-API 2.0 interface for SQLite databases. Python. URL: https://docs.python.org/uk/3.9/library/sqlite3.html (дата звернення: 08.06.2025)

25. Pickle – Python object serialization. Python. URL: https://docs.python.org/uk/3.13/library/pickle.html (дата звернення: 08.06.2025)

26. Chanseok K. TF-IDF and similarity scores. Chan`s Jupyter. 2020. URL:https://goodboychan.github.io/python/datacamp/natural_language_processing/2020/07/17/04-TF-IDF-and-similarity-scores.html (дата звернення: 08.06.2025)

27. Bruno K., Juan Cruz M. Developing RESTful APIs with Python and Flask. Auth0. 2025. URL: https://auth0.com/blog/developing-restful-apis-with-python-and-flask/ (дата звернення 09.06.2025)

28. Ken M.. User registration and login with React and Axios. Open Replay. 2022. URL: https://blog.openreplay.com/user-registration-and-login-with-react-and-axios/ (дата звернення: 09.06.2025)

29. How to create a rating component in ReactJS?. Geeks for geeks. 2024. URL: https://www.geeksforgeeks.org/how-to-create-a-rating-component-in-reactjs/ (дата звернення: 10.06.2025)

30. Yoatam M. How To Fetch and Display Movie List Using TMDB API. Medium. 2024. URL: https://medium.com/@yoatam/how-to-fetch-and-display-movie-list-using-tmdb-api-461ffebed1c5 (дата звернення: 10.06.2025)