

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «магістр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)

на тему:

**«Побудова та проєктування веб-застосунку з
елементами машинного навчання для кінотеатру»**

Виконав студент групи ПЗм-23-1
СВІСТЕЛЬНИК Олександр Сергійович

Керівник роботи:
ЛОКТИКОВА Тамара Миколаївна

Рецензент:
ЄФРЕМОВ Юрій Миколайович

Житомир – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»

Зав. кафедри інженерії
програмного забезпечення
Тетяна ВАКАЛЮК
«01» жовтня 2024 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **СВІСТЕЛЬНИК Олександр Сергійович**

Керівник роботи: **ЛОКТИКОВА Тамара Миколаївна**

Тема роботи: **«Побудова та проєктування веб-застосунку з елементами машинного навчання для кінотеатру»**,

затверджена Наказом закладу вищої освіти від **«01» жовтня 2024 р., №520/с**

Вихідні дані для роботи: Об'єктом дослідження є використання інформаційних технологій для побудови та проєктування вебзастосунку з елементами машинного навчання для кінотеатру.

Предметом дослідження є використання сучасних веборієнтованих технологій розробки для створення вебзастосунку з елементами машинного навчання для кінотеатру.

Керівники / консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Керівник/ консультант	Завдання видав	Завдання прийняв
1	Локтікова Т.М.	10.10.2024р.	10.10.2024р.
2	Локтікова Т.М.	10.10.2024р.	10.10.2024р.
3	Локтікова Т.М.	10.10.2024р.	10.10.2024р.

РЕФЕРАТ

Індивідуальне завдання кваліфікаційної магістерської роботи полягає в дослідженні принципів побудови та проєктування веб-застосунку з елементами машинного навчання для кінотеатру, а саме використання сучасних вебтехнологій для розробки вебзастосунку, який характеризується детальною коректною роботою сповіщень рекомендаційної системи індивідуально для кожного користувача.

Кваліфікаційна магістерська робота на тему «Побудова та проєктування веб-застосунку з елементами машинного навчання для кінотеатру» складається з теоретичного дослідження, вебзастосунку, рекомендаційної системи у вигляді API та пояснювальної записки. Пояснювальна записка до кваліфікаційної магістерської роботи складається зі вступу, трьох розділів, списку використаних джерел з 40 найменувань, 44 рисунки, 11 таблиць та 8 додатків. Повний обсяг роботи становить 81 сторінка.

КЛЮЧОВІ СЛОВА: ВЕБЗАСТОСУНОК, РЕКОМЕНДАЦІЙНА СИСТЕМА, ФІЛЬМ, API, PHP, LARAVEL.

ABSTRACT

The individual task of the qualifying master's thesis is to study the principles of building and designing a web application with machine learning elements for a cinema, namely the use of modern web technologies to develop a web application characterized by detailed correct operation of the recommendation system notifications individually for each user.

The qualifying master's thesis on «Building and Designing a Web Application with Machine Learning Elements for a Cinema» consists of a theoretical study, a web application, a recommendation system in the form of an API, and an explanatory note. The explanatory note to the qualifying master's thesis consists of an introduction, three chapters, a list of 40 references, 44 figures, 11 tables, and 8 appendices. The total volume of the work is 81 pages.

KEYWORDS: WEB APPLICATION, RECOMMENDATION SYSTEM, MOVIE, API, PHP, LARAVEL.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ.....	6
ВСТУП	7
РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБГРУНТУВАННЯ АКТУАЛЬНОСТІ ТЕМИ РОБОТИ.....	9
1.1 Обґрунтування актуальності теми роботи та постановка задачі	9
1.2 Огляд й аналіз існуючих аналогічних вебзастосунків	11
1.3 Вибір й обґрунтування архітектури системи	13
1.4 Обґрунтування вибору інструментальних засобів	16
Висновки до першого розділу.....	18
РОЗДІЛ 2 ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ФІЛЬМІВ.....	20
2.1 Аналіз існуючих підходів.....	20
2.3 Методи і алгоритми інтелектуального аналізу даних для рішення.....	23
2.4 Характеристика джерела даних для проведення аналізу.....	25
2.5 Побудова моделі аналізу і практична реалізація	31
2.6 Аналіз результатів.....	37
Висновки до другого розділу.....	38
РОЗДІЛ 3 СТВОРЕННЯ ВЕБЗАСТОСУНКУ ТА ІНТЕГРАЦІЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ	40
3.1 Проєктування бази даних та обробка даних	40
3.2 Розробка адмін-панелі та API вебзастосунку.....	46
3.3 Інтеграція рекомендаційної системи у вебзастосунок.....	50
Висновки до третього розділу	56
ВИСНОВКИ.....	57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ.....	60
ДОДАТКИ.....	64

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

СКБД – система керування базами даних – набір взаємопов'язаних даних (база даних) і програм для доступу до цих даних.

ER-діаграма – (англ. entity-relationship diagram) – модель даних, яка дозволяє описувати концептуальні схеми за допомогою узагальнених конструкцій блоків.

API – (англ. application programming interface) – спосіб завдяки якому дві або більше комп'ютерні програми можуть спілкуватися між собою. Набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

SPA – (англ. single page application) – вебдодаток, який взаємодіє з користувачем шляхом динамічного переписування поточної веб-сторінки новими даними з веб-сервера замість стандартного методу завантаження нових сторінок.

SVD – (англ. singular-value decomposition) – метод розкладу матриці з дійсними або комплексними числами.

RMSE – (англ. root mean square error) – захід відмінностей між істинними або прогнозованими значеннями, з одного боку, та спостережуваними значеннями, або оцінювач, з іншого.

MAE – (англ. mean absolute error) – міра похибок між парними спостереженнями, що виражають одне й те саме явище.

ВСТУП

Актуальність теми. Розвиток технологій та зміна споживацьких уподобань створюють потребу в інноваціях у сфері розваг та культури. Вебзастосунки стають необхідним інструментом для забезпечення зручної та персоналізованої взаємодії з кінотеатрами.

Впровадження елементів машинного навчання дозволить створити унікальний досвід для кожного користувача. Алгоритми аналізу виборів глядача сприятимуть точним рекомендаціям щодо фільмів, що збільшить ймовірність задоволення від перегляду та залучить нових клієнтів.

Застосунок може полегшити процес покупки квитків, забезпечуючи електронні квитки та інноваційні методи оплати. Це не тільки збільшить ефективність обслуговування клієнтів, але й підвищить рівень безпеки та зручності для глядачів.

Аналіз даних, який забезпечить застосунок, може стати цінним інструментом для управління кінотеатром. Інсайти, отримані з великої кількості даних, допоможуть виробникам рішень оптимізувати роботу кінотеатру, пристосовуючи його до вимог аудиторії та максимізуючи прибуток.

Мета і завдання дослідження. Метою дослідження є побудова та проєктування вебзастосунку з елементами машинного навчання для кінотеатру. Встановлена мета обумовлює наступні завдання:

- постановка задачі;
- огляд й аналіз існуючих аналогічних вебзастосунків;
- вибір й обґрунтування архітектури системи;
- обґрунтування вибору інструментальних засобів;
- визначення варіантів використання та об'єктно-орієнтованої структури системи;
- розробка бази даних системи;
- проєктування та реалізація алгоритмів роботи системи, включно з алгоритмами машинного навчання;

- реалізація вебзастосунку для кінотеатру;
- порядок встановлення та налаштування параметрів системи;
- тестування роботи програмного продукту.

Об’єктом дослідження є використання інформаційних технологій для побудови та проєктування вебзастосунку для кінотеатру.

Предметом дослідження є використання сучасних веборієнтованих технологій розробки для створення веб-застосунку з елементами машинного навчання для кінотеатру.

Наукова новизна роботи полягає у використанні методів машинного навчання для надання персоналізованих рекомендацій користувачам вебзастосунку кінотеатру. Особливістю системи є використання гібридного підходу пошуку рекомендацій для отримання найбільш точних результатів.

Практичне значення полягає у покращенні досвіду користувачів вебзастосунку кінотеатру шляхом надання найбільш релевантних рекомендацій фільмів на основі їх попередніх відвідувань.

Апробація. За темою випускної кваліфікаційної роботи магістра були подані та прийняті до публікації тези: Свістельник О.С., Локтікова Т.М. «Побудова та проєктування веб-застосунку з елементами машинного навчання для кінотеатру». VII Всеукраїнська науково-технічна конференція «Комп’ютерні технології: інновації, проблеми, рішення», 02-03 грудня 2024 р., м. Житомир.

РОЗДІЛ 1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБГРУНТУВАННЯ АКТУАЛЬНОСТІ ТЕМИ РОБОТИ

1.1 Обґрунтування актуальності теми роботи та постановка задачі

В сучасному світі кіноіндустрія займає провідне місце у сферах розваг та відпочинку. Майже в кожному місті є хоча б якийсь кінотеатр, що надає можливість переглядати новинки на великому екрані.

З розвитком Інтернету та мобільних телефонів онлайн послуги поступово витісняють майже всі дії, що вимагають здійснювати певні зусилля, та надають можливість у кілька кліків виконати поставлену задачу. Для прикладу, ще 10 років тому, щоб забронювати квиток у кіно було необхідно знайти номер кінотеатру та через телефонну розмову дізнатися всю інформацію, яка вас цікавить. Нині це можна спростити переходом на відповідний вебсайт та навіть відразу придбати квиток онлайн.

Проте пандемія COVID-19 [1] суттєво вплинула на світ кіноіндустрії на початку 2020-х років, викликаючи зупинку або перенесення виходу багатьох фільмів, закриття кінозалів і кінотеатрів, а також скасування або перенесення фестивалів. Це призвело до значного зниження глобальних касових зборів на мільярди доларів та збільшення популярності стрімінгових платформ.

Найкасовіший фільм 2020 року заробив 503 мільйони доларів у всьому світі. Це стало першим випадком з 2007 року, коли найкасовіший фільм року не заробив більше 1 мільярда доларів.

У 2021 році світові касові збори показали ознаки відновлення, зростаючи на 78% порівняно з 2020 роком. Незважаючи на продовження пандемії, фільм «Людина-павук: Додому шляху нема» [2] швидко став найкасовішим фільмом 2021 року і шостим найкасовішим фільмом усіх часів, перевищивши 1 мільярд доларів заробітку за майже місяць після виходу в прокат.

Велика кількість експертів вважала, що кіноіндустрія ніколи не повернеться на попередній рівень. Але як видно з тенденцій, це не так, люди люблять відвідувати кіно, попри популяризацію стрімінгових сервісів.

Найяскравішим прикладом цього є прокат сиквелу “Аватар” [3], що став 3 у списку найкасовіших фільмів усіх часів.

Тому й було вирішено розробити вебзастосунок для кінотеатру, що виконуватиме поставлені задачі. Тепер користувач матиме можливість без жодної комунікації дізнатися все про фільм та отримати відгуки про нього. Також, важливо зазначити, що застосунок повинен коректно працювати у всіх сучасних браузерях.

На основі вивчення функціонування кінотеатральної індустрії було вирішено розробити вебзастосунок, використовуючи фреймворки Laravel [4] та FastAPI [5], а також бібліотеку React [6].

В даній роботі докладно розглянемо лише реалізацію бекенд частини та адмін-панелі. Проте, також, опишемо принцип побудови фронтенд частини застосунку.

Програмний продукт повинен мати наступні функціональні характеристики:

- адмін-панель у вигляді сайту для керування вмістом застосунку;
- наявність списків фільмів, що зараз у прокаті або невдовзі з’являться;
- наявність сторінки певного фільму;
- керування профілем;
- купівля квитків;
- отримання персоналізованих сповіщень про новинки;
- можливість перегляду придбаних квитків;
- можливість повернення квитка.

Програмний комплекс повинен за допомогою клієнт-серверної архітектури з використанням веб-технологій, що не вимагають додаткового ліцензування.

Серверна частина програмного комплексу має бути розроблена на основі веб-серверу Apache2 [7] та системи керування базами даних MySQL [8]. При побудові застосунку необхідно передбачити, що взаємодія між клієнтської та серверною частинами повинна здійснюватися шляхом звернення до відповідних

ендпоїнтів розробленого приватного API [9]. Клієнтська частина повинна відповідати усім сучасним вимогам до вебзастосунків.

Отримання рекомендації має реалізовуватися за допомогою викликів до API з створеною моделлю.

Розробка системи включає кілька основних етапів. Спершу необхідно створити бекенд частину додатку з використанням фреймворку Laravel, де буде розроблено приватне API для таких сутностей: «Фільм», «Зала», «Сеанс», «Квитки», «Оплата» та «Профіль користувача».

Далі необхідно розробити інтерфейс для користувачів. Потрібно обрати відповідний фреймворк для реалізації інтерфейсу та створити сто за його допомогою. Також важливо реалізувати механізм інформування користувача про процес завантаження вмісту.

Наступним етапом є налаштування взаємодії з API на стороні клієнта та створення роутингу застосунку за допомогою бібліотеки React. Потрібно забезпечити виклик API через відповідну бібліотеку та налаштувати роутинг додатку.

Останній етап включає розробку рекомендаційної системи [10] для фільмів з використанням машинного навчання [11]. Для цього необхідно створити API за допомогою фреймворку FastAPI мови програмування Python [12], яке буде отримувати необхідну інформацію та працювати з оптимальною моделлю.

Результатом виконання цих етапів є створення вебзастосунку, що коректно працює у всіх сучасних браузерах.

1.2 Огляд й аналіз існуючих аналогічних вебзастосунків

В наш час ідеї переповнюють світ, тому створення чогось абсолютно нового стає викликом. У процесі проєктування застосунків важливо вивчати наявні аналоги, щоб краще зрозуміти вимоги користувачів та сформулювати унікальну ціннісну пропозицію для продукту, завдяки якій рішення зможе створити конкуренцію на ринку. Для порівняння були обрані наступні застосунки: Multiplex [13] та Планета Кіно [14].

Multiplex – найбільша мережа багатозальних кінотеатрів України заснована у 2004 році. Має вебзастосунок, завдяки якому можна взаємодіяти з кінотеатром онлайн.

На рис. 1.1 зображено афішу кінотеатру.

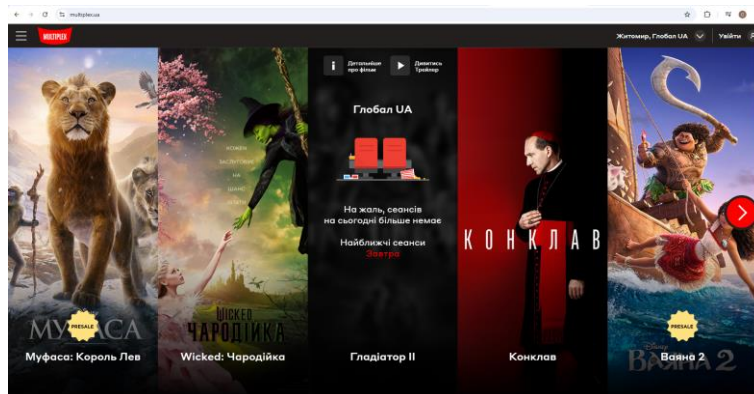


Рисунок. 1.1 - Multiplex (афіша кінотеатру)

Переваги даного вебзастосунку включають можливість перегляду активних та майбутніх фільмів, а також отримання детального опису кожного фільму. Користувачі можуть купувати квитки без необхідності авторизації, а також мають опцію повернення квитків у разі потреби.

До недоліків можна віднести некоректну роботу сторонніх API, проблеми зі сповіщеннями та відсутність персоналізованих рекомендацій для користувачів.

Планета Кіно – мережа кінотеатрів, засновником і власником якої є ТОВ Тріумф Медіа Груп. Перший кінотеатр мережі відкрили у Києві 2008 року. Має вебзастосунок, завдяки якому можна взаємодіяти з кінотеатром онлайн.

На рис. 1.2 зображено афішу кінотеатру.

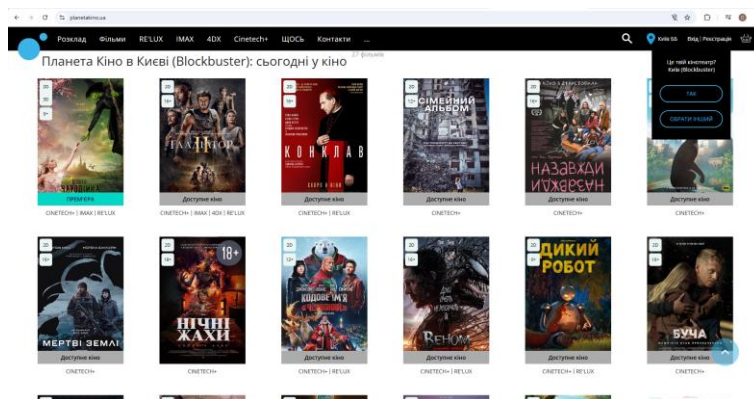


Рисунок. 1.2 - Планета Кіно (афіша кінотеатру)

До переваг можна віднести гарний та інтуїтивний дизайн, можливість перегляду активних та майбутніх фільмів, а також отримання докладного опису кожного фільму.

Недоліками є необхідність обов'язкової авторизації для купівлі квитка, відсутність інтеграції з агрегаторами та відсутність персоналізованих рекомендацій для користувачів.

Організуємо представлення отриманих даних за допомогою табличної структури.

Таблиця 1.1

Характеристика	Multiplex	Планета Кіно
Перелік активних та майбутніх фільмів	+	+
Докладний опис фільму	+	+
Коректна робота сторонніх API	-	-
Коректна робота сповіщень;	-	+
Необов'язкова авторизація для купівлі квитка.	+	-
Наявність персоналізованих рекомендацій	-	-

Грунтуючись на отриманих результатах, можна визначити такі вимоги до застосунку: необхідність перегляду активних та майбутніх фільмів, отримання докладного опису кожного фільму, коректна інтеграція зі сторонніми API, правильна робота сповіщень, можливість купівлі квитка без обов'язкової авторизації та наявність персоналізованих рекомендацій для користувачів.

1.3 Вибір й обґрунтування архітектури системи

В сучасному світі існує декілька варіантів архітектурних підходів для побудови вебзастосунку з використанням React та Laravel. Для взаємодії між клієнтом та сервером розглянемо наступні підходи: RESTful API [15], GraphQL [16].

Розглянемо кожну схему окремо.

RESTful API є популярним підходом для створення взаємодії між клієнтським та серверним програмним забезпеченням. Розглянемо переваги і недоліки RESTful API в контексті використання з React та Laravel.

На рис. 1.3 зображено схему RESTful API.

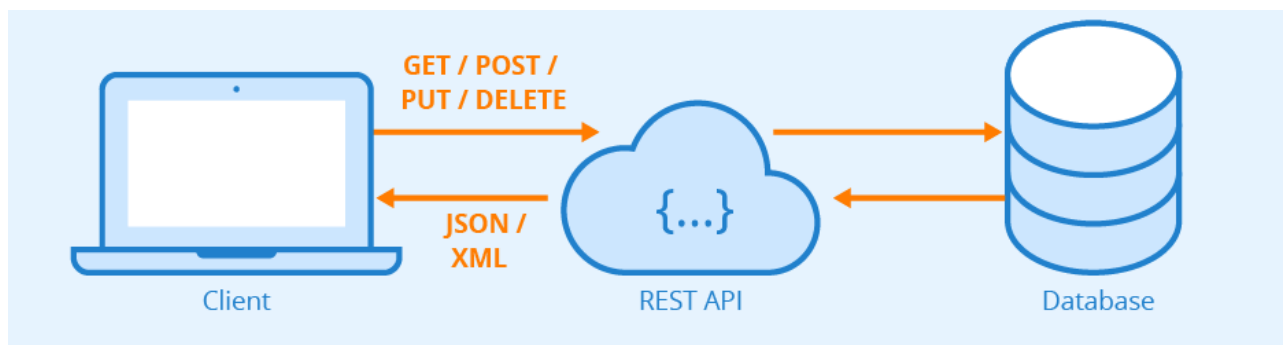


Рисунок. 1.3 - Схема RESTful API

До переваг належать наступне. React ефективно інтегрується з будь-яким RESTful API, що дозволяє створювати зручні та функціональні вебзастосунки незалежно від особливостей серверної частини. RESTful API спрощує взаємодію між фронтендом (React) та бекендом, навіть якщо вони створені на різних мовах, наприклад, PHP, Python або Node.js, що не створює проблем для інтеграції. Laravel дотримується принципів RESTful-архітектури, роблячи API зрозумілим, структурованим та легким у подальшій розробці та підтримці. Завдяки RESTful підходу Laravel API можна легко масштабувати для збільшення кількості користувачів або запитів, забезпечуючи гнучкість системи.

До недоліків належать наступне. Вебзастосунки можуть зазнавати затримок через нестабільне з'єднання, що призводить до помилок та тривалого часу відповіді від сервера. Іноді API повертає більше даних, ніж необхідно на клієнтській стороні, що збільшує навантаження на мережу та сповільнює роботу застосунку. Для коректної роботи API сервер має серіалізувати дані у зручному форматі, а неправильне налаштування серіалізації може призвести до надмірного обсягу інформації у відповіді. Також середовище RESTful API безстанісне, тобто сервер не зберігає інформацію між запитами. Це ускладнює реалізацію задач, де необхідне збереження стану, наприклад, авторизація чи відстеження дій користувача.

GraphQL – це мова запитів для взаємодії з API, яка дозволяє клієнтам отримувати саме ті дані, які їм потрібні. У розробці React та Laravel API GraphQL використовується для забезпечення ефективної комунікації між фронтендом (React) та серверною частиною (Laravel API).

На рис. 1.4 зображено схему GraphQL.

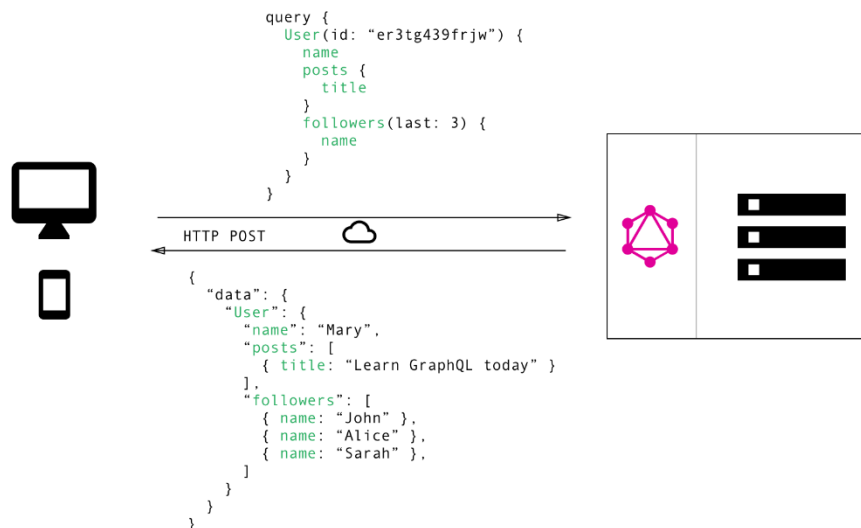


Рисунок. 1.4 - Схема GraphQL

Можна виділити наступні переваги для даної архітектури. За допомогою GraphQL клієнт отримує тільки ті дані, які йому необхідні, без зайвого «шуму», що особливо зручно при роботі з конкретною інформацією. GraphQL дозволяє отримати всю необхідну інформацію через один запит, усуваючи необхідність робити кілька запитів до різних кінцевих точок API. Завдяки можливості запитувати лише потрібні дані, GraphQL зменшує навантаження на сервер та мережу, забезпечуючи максимально ефективне використання ресурсів. Також GraphQL ефективно справляється з великими обсягами даних та складними запитами, дозволяючи легко масштабувати систему відповідно до зростаючих потреб.

До недоліків можна віднести те, що GraphQL може бути складним для розуміння та використання для новачків, особливо при обмежених ресурсах для навчання. GraphQL потребує ретельного налаштування як на стороні клієнта, так і на стороні сервера, що може сповільнити початковий етап розробки. Через природу роботи GraphQL деякі запити можуть бути занадто великими або

складними, що створює навантаження при обробці великої кількості одночасних запитів. Оскільки GraphQL дозволяє створювати дуже гнучкі запити, необхідно їх оптимізувати, щоб уникати проблем з продуктивністю або зайвим навантаженням на сервер.

Проаналізувавши вищеперелічені підходи, було обрано використовувати RESTful API через його універсальність та стандартизований підхід.

1.4 Обґрунтування вибору інструментальних засобів

Розробка програмного продукту, спрямованого на виконання конкретних завдань, включає в себе використання спеціальних інструментів. Для створення бекенду додатку було вирішено користуватися редактором PhpStorm, а для API з рекомендаційною системою за допомогою Python – PyCharm, використовуючи студентську підписку від компанії JetBrains.

PhpStorm та PyCharm [17] – це інтегровані середовища розробки (IDE) від компанії JetBrains. PhpStorm призначений для розробки на мові PHP [18] та пов'язаних технологій, надаючи підтримку PHP, HTML [19], CSS [20], JavaScript [21] та інших відомих мов, а також вбудовану систему управління версіями та інструменти для роботи з базами даних. З іншого боку, PyCharm спеціалізується на розробці застосунків за допомогою мови програмування Python.

Для створення бекенд частини було вибрано мову програмування PHP, яка має широкий вибір фреймворків. З числа популярних варіантів, таких як Symfony [22], Laravel і Yii2 [23], обрано Laravel. Основними перевагами Laravel є можливість легко створювати API без додаткових витрат ресурсів, а також активна спільнота та якісна документація, що відрізняє його від Yii2 та Symfony.

Для клієнтської частини планується використовувати JavaScript-бібліотеку React, яка наразі є одним із найпопулярніших рішень для створення додатків за архітектурою SPA [24]. У порівнянні з конкурентами, такими як Vue [25] та Angular [26], React вирізняється однією з найбільших спільнот розробників. Це забезпечує широку підтримку, доступ до численних багатофункціональних бібліотек та розширень.

MySQL було обрано в якості системи керування базами даних через численні переваги. Серед них – широка підтримка від розробників і користувачів, висока продуктивність для обробки великих обсягів даних, легка інтеграція з іншими технологіями та мовами програмування, а також різноманітні механізми безпеки, такі як рівні доступу, шифрування даних і контроль доступу.

OpenServer [27] дозволяє створювати локальний вебсервер для розробки вебзастосунків, в нашому випадку приватного API, без необхідності публікації на віддаленому сервері. Цей інструмент включає вбудований комплект програм, таких як вебсервер Apache, СКБД MySQL, мову програмування PHP, phpMyAdmin [28] та інші корисні інструменти і розширення.

Для створення рекомендаційної системи фільмів було обрано мову програмування Python завдяки її численным перевагам у сфері розробки алгоритмів машинного навчання та обробки даних. Python має потужні бібліотеки для роботи з даними та машинного навчання, такі як NumPy [29], Pandas [30], Scikit-learn [31] і Surprise [32], які дозволяють швидко реалізовувати та тестувати різні алгоритми рекомендацій. Синтаксис Python зрозумілий та лаконічний, що дає змогу зосередитися на розробці логіки без зайвих складнощів програмування, що особливо важливо для наукових досліджень і прототипування. Python є стандартом у галузі Data Science, забезпечуючи доступ до великої кількості матеріалів, навчальних ресурсів і активної спільноти, яка завжди готова надати підтримку. Крім того, Python легко інтегрується з базами даних, веб-фреймворками, такими як Flask [33] або FastAPI, та хмарними сервісами, що спрощує створення комплексного програмного рішення.

Ці переваги роблять Python найкращим вибором для розробки рекомендаційної системи, що потребує ефективної обробки даних і гнучкої реалізації алгоритмів.

Для взаємодії між моделлю рекомендаційної системи та бекендом було вирішено створити API за допомогою фреймворку FastAPI. Бо FastAPI – оптимальний вибір для проекту, що потребує сучасного, швидкого й

масштабованого API з асинхронною обробкою запитів. Він найкраще підходить для інтеграції рекомендаційної системи, де важливі швидкість відповіді й автоматизація перевірки даних. На відміну від Flask, де асинхронність і продуктивність не є ключовими факторами. Однак для більш складних завдань вимагатиме додаткових розширень, що може ускладнити підтримку.

Для забезпечення ефективної роботи системи, максимальної продуктивності та оптимального використання апаратних ресурсів, були встановлені наступні вимоги:

1. Вимоги до серверного обладнання:

- оперативна пам'ять – 8 GB;
- жорсткий диск – 256 GB;
- операційна система – Linux;
- процесор з частотою від 3 GHz;
- node.js версії 18 і вище.

2. Вимоги до робочої станції користувача:

- підтримка браузера *Google Chrome* версії 45+ або;
- підтримка браузера *Mozilla Firefox* версії 38+ або;
- підтримка браузера *Microsoft Edge* версії 12+ або;
- підтримка браузера *Opera* версії 30+ або;
- підтримка браузера *Safari* версії 9+.

Висновки до першого розділу

У цьому розділі визначена постановка завдання для кваліфікаційної роботи, включаючи формулювання функціональних вимог та визначення основних етапів виконання проєкту. Здійснено аналіз аналогів продукту, що сприяло встановленню критеріїв розробки, таких як перелік активних та майбутніх фільмів, докладний опис фільму, коректна робота сторонніх API, коректна робота сповіщень, необов'язкова авторизація для купівлі квитка та наявність персоналізованих рекомендацій.

Також було проведено аналіз існуючих архітектур, в результаті чого обрано використання схеми RESTful API. Основний вибір технологічного стеку для реалізації включає в себе використання Laravel для розробки бекенду, React для фронтенду та Python для створення рекомендаційної системи.

Вимоги до серверного обладнання та робочої станції користувача визначені та додатково обґрунтовані в тексті.

РОЗДІЛ 2 ДОСЛІДЖЕННЯ ТА РЕАЛІЗАЦІЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ФІЛЬМІВ

2.1 Аналіз існуючих підходів

Метою даного розділу є розробка рекомендаційної системи для фільмів із використанням методів машинного навчання, яка здатна ефективно передбачати вподобання користувачів. Для досягнення цієї мети необхідно провести огляд існуючих підходів до створення рекомендаційних систем, включаючи методи колаборативної фільтрації [34], контент-орієнтовані підходи та гібридні моделі. Потім слід вибрати найбільш ефективний метод або комбінацію методів для розробки рекомендаційної системи. Важливим етапом є збір та підготовка даних про користувачів і фільми, що включає історію переглядів, рейтинги та інші релевантні характеристики.

Додатково необхідно розробити модель машинного навчання для прогнозування вподобань користувачів. Після цього необхідно провести навчання та тестування моделі, оцінити її ефективність за допомогою відповідних метрик, таких як точність, повнота та F1-міра. Завершальним етапом є реалізація рекомендаційної системи та демонстрація її роботи на прикладі для перевірки її ефективності та функціональності.

Рекомендаційні системи можна класифікувати за кількома основними підходами.

Колаборативна фільтрація включає методи на основі користувачів, які використовують інформацію про подібність між користувачами для рекомендації фільмів. Якщо двоє користувачів мають схожі вподобання, система рекомендуватиме їм однакові фільми. Іншим підходом є методи на основі елементів, які аналізують подібність між елементами (фільмами) для рекомендації. Наприклад, якщо користувач оцінив певний фільм, система рекомендуватиме інші схожі фільми.

На рис. 2.1 зображено суть колаборативної фільтрації.

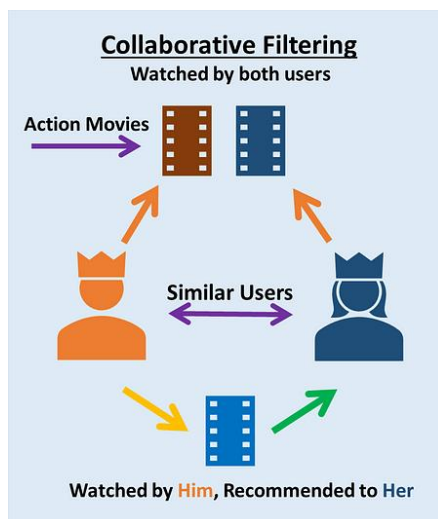


Рисунок. 2.1 - Суть колаборативної фільтрації

Контент-орієнтовані підходи використовують характеристики фільмів, такі як жанр, режисер, актори та інші параметри, для створення профілю користувача. На основі його попередніх вподобань система рекомендує нові фільми, що відповідають цьому профілю.

На рис. 2.2 зображено суть контент-орієнтованого підходу.

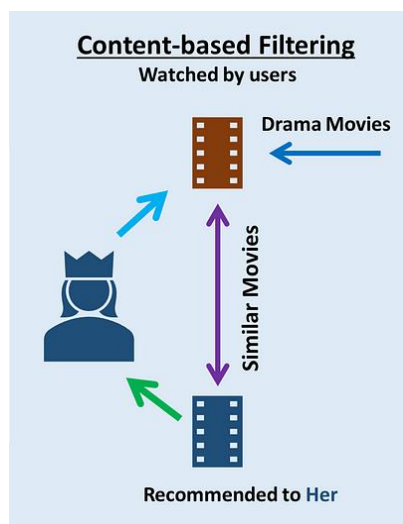


Рисунок. 2.2 - Суть контент-орієнтованого підходу

Гібридні моделі поєднують елементи колаборативної фільтрації та контент-орієнтованих підходів, що дозволяє покращити точність і надійність рекомендацій.

На рис. 2.3 зображено суть гібридної моделі.

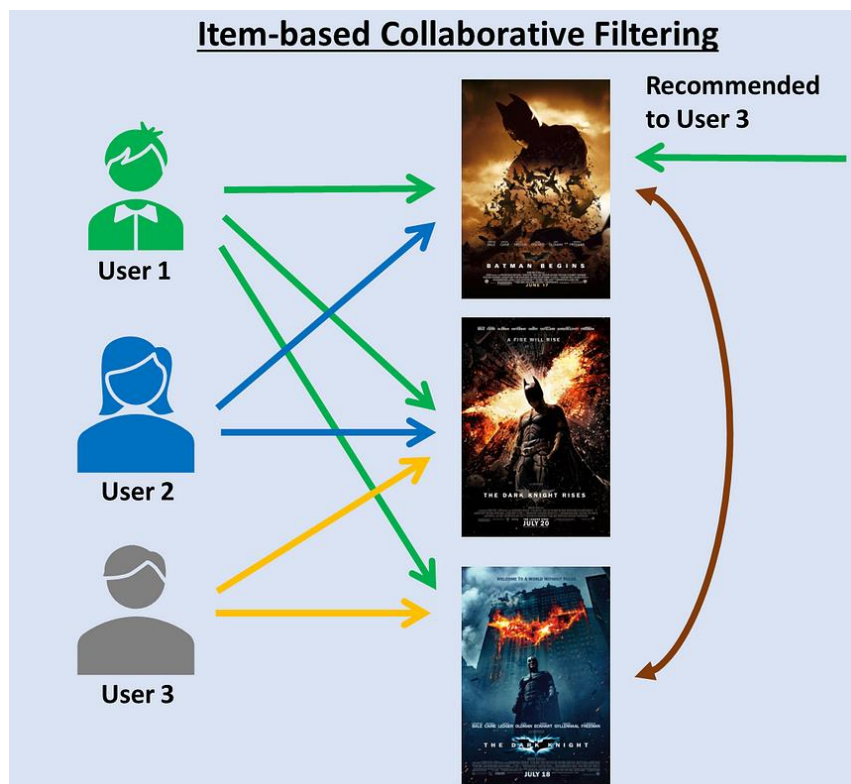


Рисунок. 2.3 - Суть гібридної моделі

2.2 Засоби та методи вирішення задачі

Для реалізації рекомендаційної системи використовуватимуться такі інструменти та методи. У сфері машинного навчання застосовуватимуться алгоритми колаборативної фільтрації, зокрема К-ближніх сусідів і матрична факторизація, а також алгоритми контент-орієнтованих підходів, наприклад методи TF-IDF для аналізу текстових описів фільмів. Для підвищення точності та надійності рекомендацій використовуватимуться гібридні алгоритми, що поєднують обидва підходи.

Мова програмування Python буде основним інструментом розробки завдяки своїм численным бібліотекам. Pandas використовуватиметься для роботи з табличними даними, що забезпечить зручність об'єднання, групування, вибору та фільтрації. Бібліотека NumPy дозволить виконувати ефективні обчислення з масивами даних. Для візуалізації даних будуть використовуватися Seaborn і Matplotlib.pyplot, які забезпечать можливість створення різноманітних графіків. Ast.literal_eval допоможе безпечно обробляти рядки, що містять літеральні

структури даних, а `Collections.Counter` стане у пригоді для підрахунку елементів. Робота з датами та часом здійснюватиметься за допомогою `Datetime`. Для збереження та відновлення об'єктів Python використовуватиметься `Joblib`. Бібліотека `Scikit-learn` стане основним інструментом для реалізації алгоритмів машинного навчання, таких як класифікація, регресія та кластеризація. Для рекомендаційних систем буде використано `Surprise`, що надасть реалізації популярних алгоритмів, таких як SVD [35] і KNN, а також інструменти для оцінки моделей і їх параметрів.

Оцінка ефективності рекомендаційної системи здійснюватиметься за допомогою відповідних метрик. RMSE [36] та MAE [37] дозволять оцінити похибки, а метрики класифікації допоможуть проаналізувати якість рекомендацій.

Застосування цих підходів та методів дозволить розробити ефективну рекомендаційну систему, яка покращить досвід користувачів у виборі фільмів та забезпечить високу точність рекомендацій.

2.3 Методи і алгоритми інтелектуального аналізу даних для рішення

Розробка ефективної системи рекомендацій для фільмів включає застосування різноманітних методів та алгоритмів інтелектуального аналізу даних. Ось основні з них:

Базові моделі рекомендаційних систем включають кілька підходів. Глобальне середнє значення (Global Mean Rating) передбачає обчислення середнього рейтингу всіх фільмів у наборі даних. Хоча цей підхід є простим у реалізації, він не враховує індивідуальних уподобань користувачів. Модель середнього значення користувача (User Mean Rating) обчислює середній рейтинг, який дає кожен користувач, і базує прогноз для будь-якого фільму на середньому рейтингу, виставленому цим користувачем іншим фільмам. Це дозволяє врахувати схильності конкретного користувача. Модель середнього значення елемента (Item Mean Rating), своєю чергою, обчислює середній рейтинг для кожного фільму. Прогноз для будь-якого фільму ґрунтується на середньому

рейтингу, який цей фільм отримав від усіх користувачів, що дозволяє врахувати його популярність.

Середнє значення елемента розраховується за такою формулою:

$$\text{Prediction} = \frac{\text{User Mean Rating} + \text{Item Mean Rating}}{2}$$

Існує колаборативна та контентна фільтрації і гібридний підхід. Тепер докладно розглянемо підходи.

Колаборативна фільтрація включає різні методи для створення рекомендацій. Метод SVD, який є сингулярним розкладанням, розбиває матрицю користувач-фільм на три менші матриці, що дозволяє виявити латентні фактори, які визначають переваги користувачів. Завдяки цьому можна робити більш точні рекомендації, ґрунтуючись на схожих уподобаннях інших користувачів. Метод KNNWithZScore базується на алгоритмі k-ближчих сусідів для визначення схожих користувачів або фільмів із використанням нормалізованих рейтингів за Z-Score, що враховує відмінності в шкалі рейтингів між користувачами і підвищує точність рекомендацій. SVDpp є покращеним варіантом SVD, який додатково враховує неявні переваги користувачів, наприклад, інформацію про перегляди фільмів, що не були оцінені. Це дозволяє працювати з більшим обсягом даних і забезпечує більш точні прогнози.

Контентна фільтрація використовує характеристики фільмів (жанр, режисер, актори тощо) для створення рекомендацій. Для цього використовується косинусна подібність між фільмами, що дозволяє знайти схожі фільми на основі їхніх характеристик.

Гібридний підхід об'єднує колаборативну та контентну фільтрацію для створення точних і персоналізованих рекомендацій. Один із методів – гібридний прогноз рейтингу, який комбінує результати обох підходів і враховує популярність фільмів, формуючи прогноз рейтингу як зважене середнє. Для старих користувачів система враховує їхню історію переглядів і оцінок, створюючи список рекомендацій на основі схожих фільмів і загальних уподобань інших користувачів. Новим користувачам, які ще не мають історії оцінок, рекомендації формуються на основі загальної популярності фільмів та їх

жанрових характеристик. Система може запропонувати найпопулярніші фільми, які мають високі рейтинги і багато переглядів, або популярні фільми в жанрах, які можуть зацікавити користувача.

Застосування цих методів і алгоритмів дозволяє створити потужну і гнучку систему рекомендацій, яка може ефективно працювати як з новими, так і зі старими користувачами, враховуючи їх переваги та характеристики фільмів. Гібридний підхід особливо корисний, оскільки він об'єднує переваги різних методів для досягнення найкращих результатів.

2.4 Характеристика джерела даних для проведення аналізу

У цьому розділі надамо вичерпний огляд набору даних, використаних для побудови системи рекомендацій фільмів. Використано повний набір даних MovieLens, який охоплює багате сховище інформації про фільми, включаючи акторський склад, знімальну групу, рейтинги користувачів тощо. Набір даних було отримано на Kaggle.

На рис. 2.4 зображено завантаження датасету.

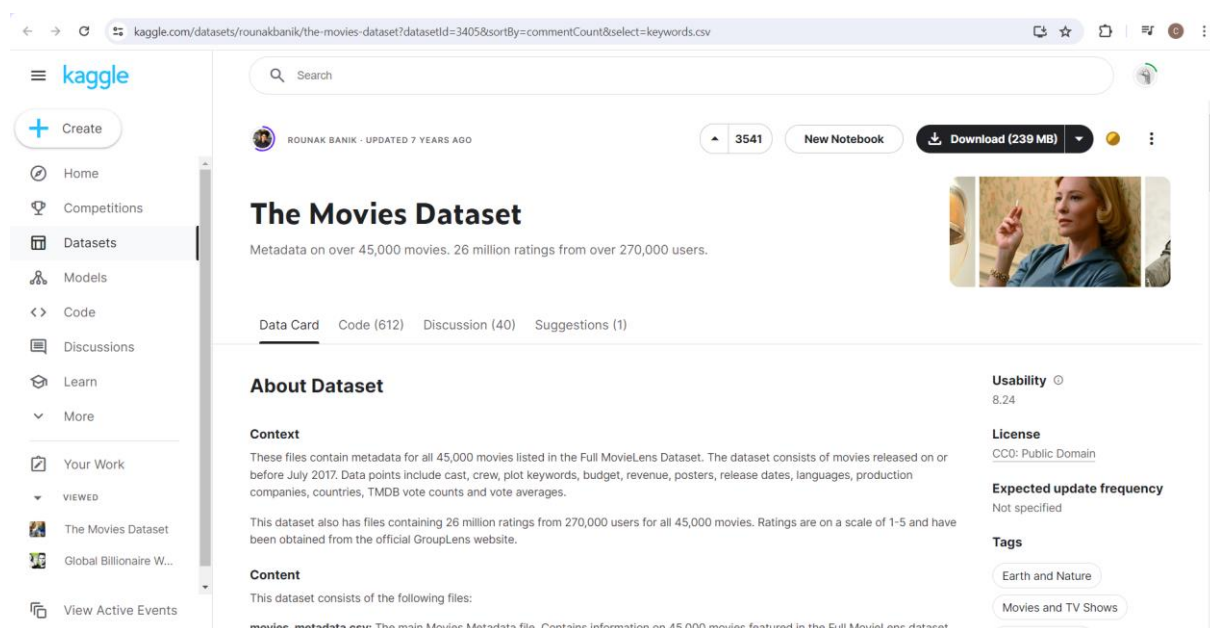


Рисунок. 2.4 - Завантаження датасету

Нижче розглянемо ключові аспекти цього набору даних у табличному поданні:

Таблиця 2.1

Набір даних про рейтинги (ratings_small.csv)

Колонка	Опис
1	2
userId	Унікальний ідентифікатор для кожного користувача, дуже важливий для відстеження індивідуальної рейтингової поведінки
movieId	Унікальний ідентифікатор для кожного фільму, що пов'язує рейтинги з конкретними фільмами
rating	Оцінка, виставлена користувачем фільму, зазвичай за шкалою від 0,5 до 5 або від 1 до 10
timestamp	Фіксує час, коли було виставлено оцінку, що є цінним для створення рекомендацій, чутливих до часу

Цей набір даних слугує основою для побудови надійних систем рекомендацій фільмів, фіксуючи оцінки користувачів для різних фільмів.

Таблиця 2.2

Набір даних про фільми (movie.csv)

Колонка	Опис
id	Унікальний ідентифікатор для кожного фільму
genres	Класифікує кожен фільм за одним або кількома жанрами
title	Офіційна назва фільму
overview	Короткий опис або синопсис фільму
revenue	Фінансові надходження від касових зборів та інших джерел
vote_average	Середня оцінка, виставлена глядачами
vote_count	Загальна кількість оцінок, які отримав фільм

Повна колекція метаданих про фільми, яка включає різноманітні характеристики, починаючи від назви та слогану фільму і закінчуючи його тривалістю та доходами.

Таблиця 2.3

Набір ключових слів (keywords.csv)

Колонка	Опис
1	2
id	Унікальний ідентифікатор, який посилається на набір даних фільмів

Продовження табл. 2.3

1	2
keywords	Набір слів або фраз, що узагальнюють суть фільму, які використовуються для фільтрації та пошуку на основі вмісту

Цей набір даних збагачує фільми списком ключових фраз і слів, що містить перелік слів або фраз, які узагальнюють теми або унікальні особливості кожного фільму. Ці ключові слова можуть слугувати швидкими посиланнями для розуміння основної теми фільмів, елементів сюжету або унікальних особливостей кожного фільму.

Таблиця 2.4

Набір даних про титри (credits.csv)

Колонка	Опис
id	Унікальний ідентифікатор, який можна використовувати для злиття з набором даних про фільми
cast	Перелічує акторів та їхні ролі, корисно для рекомендацій на основі акторів
crew	Вказує персонал, залучений до різних аспектів виробництва, таких як режисура та сценарій, корисний для рекомендацій на основі режисера чи сценариста

Містить детальну інформацію про акторів та знімальну групу, задіяних у кожному фільмі, надаючи детальний погляд на людей, які стоять за виробництвом.

Тепер необхідно обробити дані для подальшої роботи з ними. Для цього зчитуємо та об'єднуємо фрейми даних в один комплексний набір даних. Для цього візьмемо базовий фрейм даних з movies.csv та об'єднуємо з keywords.csv та credits.csv, використовуючи унікальні ідентифікатори фільмів як спільний ключ.

Реалізацію даного кроку зображено на рис. 2.5.

```
movies_df = pd.read_csv(filepath_or_buffer 'movies.csv', low_memory=False)
keywords_df = pd.read_csv('keywords.csv')
credits_df = pd.read_csv('credits.csv')
ratings_df = pd.read_csv('ratings_small.csv')

movies_df = movies_df[~(movies_df['id'].str.contains('-'))]
movies_df['id'] = movies_df['id'].astype('int64')
movies_df = pd.merge(movies_df, keywords_df, on='id', how='left')
movies_df = pd.merge(movies_df, credits_df, on='id', how='left')
```

Рисунок. 2.5 - Зчитання та об'єднання фреймів даних

Після об'єднання очистимо набір даних шляхом видалення непотрібних стовпців та заповнення відсутніх текстових даних словом «Unknown», а числових значень — числом 0. Цей процес гарантує, що дані є чистими та готовими до аналізу.

На рис. 2.6 зображено реалізацію даного процесу.

```
# Видалення стовпців
column_to_drop = ['belongs_to_collection', 'homepage', 'status', 'video']
movies_df = movies_df.drop(column_to_drop, axis=1)

# Заміна відсутніх значень в текстових стовпчиках на 'Unknown'
text_columns = ['original_language', 'tagline']
movies_df[text_columns] = movies_df[text_columns].fillna('')

# Заміна відсутніх значень в числових стовпчиках на '0'
number_columns = ['runtime']
movies_df[number_columns] = movies_df[number_columns].fillna(0)

# Видалення рядків з пропущеними значеннями
movies_df.dropna(inplace=True)
movies_df.head()
```

Рисунок. 2.6 - Очищення набору даних

Далі виконаємо важливі коригування формату даних, щоб зробити набір даних більш придатним для аналізу. Для цього виконаємо перетворення стовпців дати, стандартизацію форматів і забезпечення числової сумісності.

На рис. 2.7 зображено реалізацію даного процесу.

```
# Перетворення стовпця "release_date" у формат часу і виокремлення року у новий стовпець "рік"
# Якщо перетворення не вдається, значення перетворюються на NaT (Not a Timestamp)
movies_df['year'] = pd.to_datetime(movies_df['release_date'], errors='coerce').dt.year

# Перетворення стовпця "year" у цілочисельний тип, при помилці значення перетворюються на NaT у його цілочисельне
# представлення
movies_df['year'] = movies_df['year'].astype(int)

# Перетворення стовпця "release_date" у формат дата-час, щоб стандартизувати його формат
# Якщо перетворення не вдається, значення перетворюються на NaT (Not a Timestamp)
movies_df['release_date'] = pd.to_datetime(movies_df['release_date'], format="%Y-%m-%d", errors='coerce')

# Перетворення стовпця "popularity" у тип float64, щоб забезпечити можливість виконання числових операцій
movies_df['popularity'] = movies_df['popularity'].astype('float64')

# Перетворення стовпця "timestamp" у ratings_df у формат datetime, використовуючи Unix timestamp
# Лямбда-функція перетворює кожен мітку часу в об'єкт datetime
ratings_df['timestamp'] = ratings_df['timestamp'].apply(lambda x: datetime.datetime.fromtimestamp(x))
```

Рисунок. 2.7 - Коригування формату даних

Далі трансформуємо певні стовпці, щоб зробити набір даних більш зручним для аналізу та розробки функцій. Стовпці, «genres», «keywords», «cast» та «crew» містять складні або текстові дані, які можуть не відразу бути корисними в їх поточній формі.

Ці стовпці перетворюються на списки елементів, що полегшує витяг важливої інформації. З стовпця «crew» витягуємо ім'я режисера для кожного фільму, оскільки це може бути важливою особливістю для рекомендацій. Для стовпців «genres» та «keywords» обмежуємо дані до 3 верхніх елементів, якщо їх більше 3 елементів. а кож очищуємо текстові дані, перетворюючи все на нижній регістр і видаляючи пробіли. Це забезпечує одноманітність текстових функцій. стовпець «bag_of_words» створюється шляхом об'єднання відповідних функцій. Це слугуватиме складовою функцією для системи рекомендацій.

На рис. 2.8 зображено реалізацію даного процесу.

```
# Функція для вишування значень зі стовпця у фреймі даних та розширення цільового списку
# цільовим значенням
def extend_list_from_column(df, column_name, target_list, key='name'):
    # Заміна NaN на порожні списки і застосування literal eval
    # Отримання значення за допомогою key і розширення цільового списку
    temp_df = df[column_name].fillna('').apply(literal_eval).apply(
        lambda x: [i[key] for i in x] if isinstance(x, list) else []
    )
    for i in temp_df:
        if i:
            target_list.extend(i)

# Функція для отримання імені режисера зі стовпчика "crew"
def get_director(x):
    return next((i['name'] for i in x if i['job'] == 'Director'), np.nan)

# Функція для отримання списку перших 3 елементів або всього списку, залежно від того, що
# більше
def get_list(x):
    if isinstance(x, list):
        names = [i['name'] for i in x]
        return names[:3] if len(names) > 3 else names
    return []

# Перетворення рядків у нижній регістр та видалення пробілів
def clean_data(x):
    if isinstance(x, list):
        return [str.lower(i.replace(" ", "")) for i in x]
    else:
        if isinstance(x, str):
            return str.lower(x.replace(" ", ""))
        else:
            return ''

# Функція для створення пакету слів з різних стовпців
def create_bag_of_words(x):
    return ' '.join(x['keywords'] + x['cast'] + [x['director']] + x['genres'])

# Ініціалізація порожніх списків для зберігання певних типів даних
spoken_languages_list = []
cast_list = []
crew_list = []
company_list = []
country_list = []
original_language_list = []

# Розширення списку мов оригіналу зі стовпчика "original language"
for i in movies_df['original_language']:
    original_language_list.extend(i.split(' '))

# Використання функції extend_list_from_column для заповнення ініціалізованих списків
extend_list_from_column(movies_df, 'spoken_languages', spoken_languages_list)
extend_list_from_column(movies_df, 'cast', cast_list, key='character')
extend_list_from_column(movies_df, 'crew', crew_list)
extend_list_from_column(movies_df, 'production_companies', company_list)
extend_list_from_column(movies_df, 'production_countries', country_list)

# Заповнення значення NaN і застосування literal eval до певних стовпців
for col in ['crew', 'cast', 'genres', 'keywords', 'production_companies']:
    movies_df[col] = movies_df[col].fillna('').apply(literal_eval)

# Застосування функції get_director для вишування режисерів зі стовпчика "crew"
movies_df['director'] = movies_df['crew'].apply(get_director)

# Застосування функції get_list для відсікання або збереження цілих списків у певних
# стовпцях
for col in ['cast', 'genres', 'keywords', 'production_companies']:
    movies_df[col] = movies_df[col].apply(get_list)

# Застосування функції clean_data для очищення стовпців функцій
features = ['adult', 'cast', 'keywords', 'director', 'genres', 'production_companies']
for feature in features:
    movies_df[feature] = movies_df[feature].apply(clean_data)

# Створення стовпця "bag of words" шляхом конкатенації відповідних стовпців
movies_df['bag_of_words'] = movies_df.apply(create_bag_of_words, axis=1)
```

Рисунок. 2.8 - Трансформація даних

Нарешті можемо дослідити набір даних. Розпочнемо з побудови гістограми на основі стовпця «release_date», Щоб краще зрозуміти розподіл фільмів за роки. Це дасть уявлення про концентрацію випусків фільмів протягом багатьох років.

На рис. 2.9 зображено гістограму на основі стовпця «release_date».

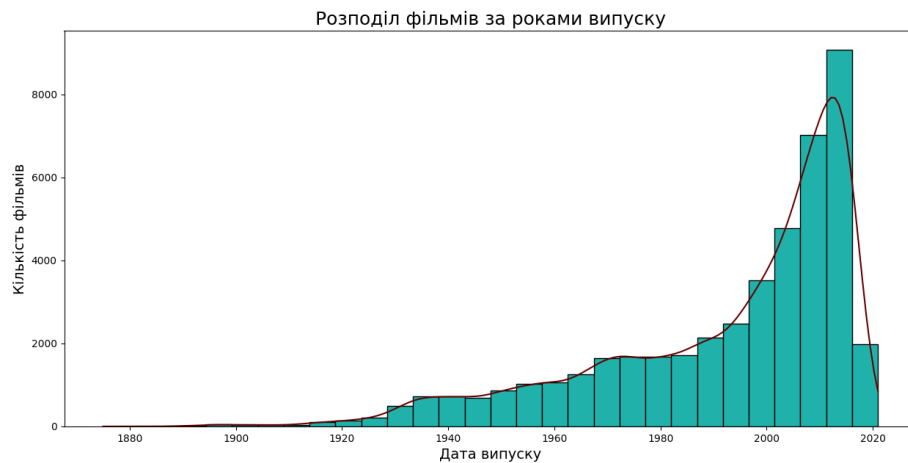


Рисунок. 2.9 - Гістограма на основі стовпця «release_date»

Гістограма показує, що від лише одного фільму в 1874 році до майже 2000 у пікові роки кіноіндустрія демонструє експоненціальний ріст, що підкреслює її незмінну привабливість. Але різке скорочення випуску фільмів у 2020 році, ймовірно, відображає глобальний вплив пандемії COVID-19 на індустрію. До 1985 року індустрія значно зросла, щорічні випуски досягли 373, що вказує на період швидкого розширення.

Далі для прикладу визначимо та візуалізуємо 5 найкращих жанрів фільмів. Для цього використаємо стовпчасту діаграму для відображення 5 найпоширеніших жанрів і кругову діаграму, щоб показати їхній розподіл у відсотках у всіх жанрах.

На рис. 2.10 зображено гістограму на основі стовпця «release_date».

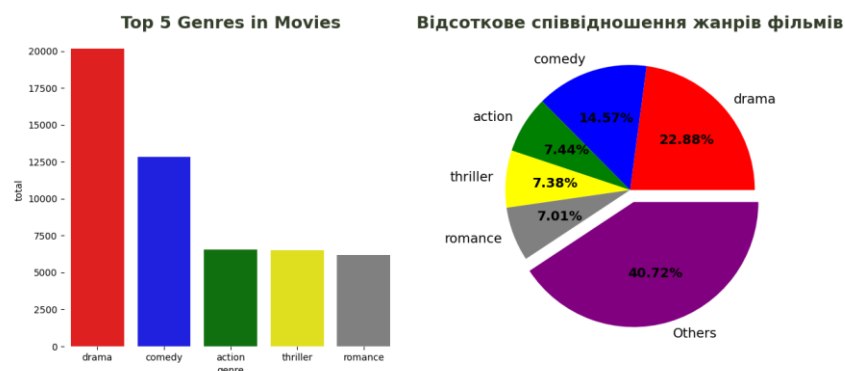


Рисунок. 2.10 - Кращі жанри фільмів за допомогою стовпчастої та кругової діаграм

Загальний підрахунок фільмів з цих даних показав, що драматичні фільми є найпоширенішими, за ними йдуть комедії, які викликають сміх. Трилери дарують хвилювання, романтичні фільми розповідають історії кохання, а бойовики – пригоди. Категорія «Інші» охоплює різні жанри, надаючи широкий вибір фільмів. Ця інформація підкреслює різноманітність вибору для різних уподобань у кіно.

2.5 Побудова моделі аналізу і практична реалізація

Тепер перейдемо до безпосередньої реалізації системи. Щоб оцінити базові моделі, розділимо набір даних на Data Train (75%) і Data Test (25%). Навчальний набір використовуватиметься для навчання та побудови рекомендаційної моделі, тоді як набір для тестування служить незалежним заходом для оцінки ефективності моделі. Це гарантує, що система надає точні та надійні рекомендації щодо фільмів при використанні реальних даних.

У процесі оцінювання середньоквадратична похибка (RMSE) буде ключовою метрикою. RMSE показує, наскільки точно прогнози моделі збігаються з фактичними рейтингами. Чим нижче RMSE, тим краще працює модель.

Для початку побудуємо базові моделі.

Результати її роботи наведено на рис. 2.11, а лістинг в додатку А.

```
RMSE для моделі глобального середнього рейтингу: 1.06
RMSE для моделі середньої оцінки користувача: 0.96
RMSE для моделі середнього значення рейтингу елемента: 1.00
RMSE для моделі середнього рейтингу користувач-елемент: 0.92
RMSE для моделі середньозваженої оцінки: 0.92, with setting the best weight is 0.60
RMSE для моделі з константою: 1.49
RMSE для моделі з демпфуванням середньої оцінки користувача: 0.96
```

Рисунок. 2.11 - Результат роботи базових моделей

Ці базові моделі дають суттєве розуміння процесу рекомендацій. Примітно, що модель середнього рейтингу за елементом користувача та модель середньозваженого рейтингу працюють виключно добре з RMSE 0,92. Це вказує

на те, що поєднання середніх оцінок користувача та елемента призводить до більш точних прогнозів. Досліджуючи ці базові моделі, було отримано цінну інформацію та встановлюємо еталон продуктивності для системи рекомендацій.

Далі побудуємо систему рекомендацій з колаборативною фільтрацією, використовуючи бібліотеку Surprise. Метою є порівняти продуктивність різних алгоритмів колаборативної фільтрації, щоб визначити той, який забезпечує найвищу точність прогнозування, що вимірюється середньоквадратичною помилкою (RMSE). Це такі алгоритми, як SVD (Singular Value Decomposition), KNNWithZScore (K-Nearest Neighbors with Z-Score normalization) та SVDpp (SVD with Power Iteration).

Ці три алгоритми спільної фільтрації будуть оптимізовані за допомогою методу GridSearchCV для точного налаштування їх гіперпараметрів за допомогою 3-кратної перехресної перевірки, яка застосовується для навчання моделі.

Результати реалізації наведено на рисунку 2.12, а лістинг в додатку Б.

```
Найкращий алгоритм: SVD
Найкращі параметри: {'n_factors': 150, 'n_epochs': 35, 'lr_all': 0.012, 'reg_all': 0.15}
Найкраща Оцінка RMSE на Data Train : 0.8970635747677034
```

Рис. 2.12. Результат роботи базових моделей

Було отримано такі оцінки RMSE для кожного алгоритму:

- SVD: 0.897;
- KNNWithZScore: 0.929;
- SVDpp: 0.898.

Примітно, що алгоритм SVD виділяється найнижчим показником RMSE, що вказує на його виняткову точність.

SVD (Singular Value Decomposition) – це метод факторизації матриці, який розкладає матрицю взаємодії користувача з елементом на кілька матриць, враховуючи приховані фактори. Ці приховані фактори забезпечують стисле, але ефективне представлення вихідних даних, що дозволяє SVD робити більш точні прогнози. Було додатково налаштовано за допомогою оптимальних гіперпараметрів, що призвело до найкращого показника RMSE.

Також, обчислимо зважений рейтинг для кожного фільму, щоб покращити систему рекомендацій фільмів. Мета – пропонувати якісні та популярні фільми.

Починаємо з фільтрування фільмів із відсутніми значеннями в `vote_average` або `vote_count`. Далі застосуємо формулу IMDb для розрахунку зваженого рейтингу для кожного фільму. Ця формула враховує середній рейтинг (R), загальний середній (C), кількість голосів (v) і мінімальний поріг голосування (m):

Формула IMDb для розрахунку зваженого рейтингу для кожного фільму має такий вигляд:

$$\text{Weighted Rating} = \frac{R \times v + C \times m}{v + m}.$$

Потім нормалізуємо `popularity` та `weighted_average`, використовуючи масштабування `MinMax`. Після чого кожен фільм отримує оцінку, яка складається з різних факторів: 40% `weighted_average`, 40% `popularity`, 5% `time_decay_factor` і 15% `normalized_revenue`. Тепер класифікуємо фільми на основі їхніх балів у порядку спадання, щоб створити список найпопулярніших фільмів із найвищим рейтингом.

На рис. 2.13 зображено результати виконання, а лістинг коду наведено в додатку В.

[28]:

	original_title	year	revenue	score
0	Minions	2015	1.156731e+09	0.699055
1	Avatar	2009	2.787965e+09	0.584354
2	Wonder Woman	2017	8.205804e+08	0.564407
3	Beauty and the Beast	2017	1.262886e+09	0.551252
4	Big Hero 6	2014	6.521054e+08	0.540299
5	The Dark Knight	2008	1.004558e+09	0.532660
6	Guardians of the Galaxy Vol. 2	2017	8.634161e+08	0.518944
7	Deadpool	2016	7.831130e+08	0.500611
8	Pulp Fiction	1994	2.139288e+08	0.499935
9	Gone Girl	2014	3.693304e+08	0.489802

Рисунок. 2.13 - Результати виконання

Фільтрація на основі вмісту на основі косинусної подібності полягає в аналізі вмісту фільмів для виявлення подібності. Мов йде про ключові слова,

жанри, акторський склад і режисера – усі ті деталі, які роблять фільм таким, яким він є. Модель аналізує ці атрибути, щоб визначити, наскільки схожі два фільми. Якщо вони близькі, вони скорочують рекомендацію.

Потім модель сортує ці обчислені бали та рекомендує 10 найкращих фільмів, найбільш схожих на вибраний користувачем фільм. Основна ідея полягає в тому, що якщо користувачеві подобається певний фільм, він, ймовірно, також оцінить схожі.

На рис. 2.14 зображено рекомендаційні фільми для «Pulp Fiction», а лістинг коду наведено в додатку Г.

```
449          Fresh
22378    Reasonable Doubt
12327          Cleaner
7252    Kill Bill: Vol. 2
777      A Time to Kill
1634      Jackie Brown
6433          S.W.A.T.
10794    Freedomland
12938    Lakeview Terrace
27869    The Hateful Eight
Name: title, dtype: object
```

Рисунок. 2.14 - Рекомендаційні фільми для «Pulp Fiction»

Далі спрогнозуємо, наскільки фільм може сподобатися користувачеві, поставивши йому рейтинг. Функція використовує поєднання різних методів: спільне фільтрування, фільтрування на основі вмісту та зважене оцінювання, щоб зробити цей прогноз якомога точним. Він бере ідентифікатор користувача та ідентифікатор фільму, а потім повертає прогнозований рейтинг, який показує, наскільки цей конкретний фільм сподобається конкретному користувачеві.

На рис. 2.15 зображено прогнозований гібридний рейтинг для `userId=1` та `movieId=5`, а лістинг коду наведено в додатку Д.

```
Прогнозований гібридний рейтинг для userId=1 та movieId=5 становить: 1.9644784657605807
```

Рисунок. 2.15 - Прогнозований гібридний рейтинг

У цьому розділі генеруватимемо рекомендації щодо фільмів для користувачів за допомогою Hybrid Movie Recommender System. Для цього потрібен ідентифікатор користувача. Функція спочатку використовує спільну фільтрацію для прогнозування рейтингів фільмів на основі минулої поведінки користувача. Потім він використовує фільтрацію на основі вмісту, щоб знайти фільми, схожі на останній фільм, який користувач переглянув. Ця функція поєднує ці два набори рекомендацій, надаючи їм однакову вагу, щоб створити остаточний список із 10 найпопулярніших рекомендованих фільмів. Цей гібридний підхід спрямований на надання більш точних і персоналізованих пропозицій щодо фільмів.

На рис. 2.16 зображено рекомендацію фільмів для старих користувачів (userId=10), а лістинг коду наведено в додатку Є.

```
Title: The Million Dollar Hotel (2000), Genres: drama, thriller, Director: wimwenders
Title: Point Break (1991), Genres: action, thriller, crime, Director: kathrynbigelow
Title: A Brief History of Time (1991), Genres: documentary, Director: erroldmorris
Title: Dr. Jekyll and Mr. Hyde (1941), Genres: drama, horror, sciencefiction, Director: victorflaming
Title: The Soft Skin (1964), Genres: drama, romance, Director: françoistruffaut
Title: Dr. Jekyll and Mr. Hyde (1920), Genres: drama, horror, sciencefiction, Director: johns.robertson
Title: Abbott and Costello Meet Dr. Jekyll and Mr. Hyde (1953), Genres: comedy, horror, sciencefiction, Director: charleslamont
Title: Children (2006), Genres: horror, sciencefiction, foreign, Director: ragnarbragason
Title: Drive-In Horrorshow (2009), Genres: horror, Director:
Title: The Two Faces of Dr. Jekyll (1960), Genres: horror, sciencefiction, romance, Director: terencefisher
Title: The Two Faces of Dr. Jekyll (1960), Genres: horror, sciencefiction, romance, Director: terencefisher
```

Рисунок. 2.16 - Рекомендація фільмів для старих користувачів (userId=10)

Рекомендація найпопулярніші фільми (1 підхід):

Спочатку спробуємо вирішити проблему «холодного запуску», шляхом рекомендації новим користувачам останніх фільмів з найвищим рейтингом. Використаємо DataFrame ідентифікаторів фільмів і їх зважених балів для створення списку з n найкращих фільмів за останні min_year років. Рекомендації сортуються за балом, який поєднує як популярність, так і оцінки користувачів. Функція розроблена як гнучка, що дозволяє вказати кількість рекомендацій і часові рамки для останніх фільмів.

На рис 2.17 зображено рекомендацію фільмів для нових користувачів (1 підхід), а лістинг коду наведено в додатку Ж.

```

Рекомендовані фільми:
Title: Deadpool (2016), Genres: action, adventure, comedy, Director: timmiller
Title: Guardians of the Galaxy Vol. 2 (2017), Genres: action, adventure, comedy, Director: jamesgunn
Title: Captain America: Civil War (2016), Genres: adventure, action, sciencefiction, Director: anthonyrusso
Title: Wonder Woman (2017), Genres: action, adventure, fantasy, Director: pattyjenkins
Title: Zootopia (2016), Genres: animation, adventure, family, Director: byronhoward
Title: Zootopia (2016), Genres: animation, adventure, family, Director: byronhoward
Title: Your Name. (2016), Genres: romance, animation, drama, Director: makotoshinkai
Title: Beauty and the Beast (2017), Genres: family, fantasy, romance, Director: billcondon
Title: Baby Driver (2017), Genres: action, crime, Director: edgarwright

```

Рисунок. 2.17 - Рекомендація фільмів для старих користувачів (1 підхід)

Рекомендація найпопулярніші фільми (2 підхід):

Інший підхід полягає в рекомендації фільмів з найвищим рейтингом у різних жанрах. Він виходить за рамки універсального методу та надає жанрові списки найкращих n фільмів за останні `min_year` років. Таким чином, нові користувачі можуть із самого початку вибрати фільми відповідно до своїх жанрових уподобань. Функція повертає словник, у якому кожен жанр зіставляється з DataFrame найкращих рекомендацій, що робить її універсальним інструментом для персоналізованих пропозицій.

На рис. 2.18 зображено рекомендацію фільмів для нових користувачів (1 підхід), а лістинг коду наведено в додатку 3.

```

Title: American Sniper (2014), Genres: war, action, Director: clinteastwood
Title: Fury (2014), Genres: war, drama, action, Director: davidayer
Title: Unbroken (2014), Genres: drama, war, Director: angelinajolie
Title: Testament of Youth (2014), Genres: history, drama, war, Director: jameskent
Title: Beasts of No Nation (2015), Genres: drama, war, Director: caryfukunaga
Title: Eye in the Sky (2015), Genres: drama, thriller, war, Director: gavinhood
Title: Land of Mine (2015), Genres: drama, history, war, Director: martinzandvliet
Title: Hacksaw Ridge (2016), Genres: drama, history, war, Director: melgibson
Title: Allied (2016), Genres: action, drama, war, Director: robertzemeckis
Title: War for the Planet of the Apes (2017), Genres: drama, sciencefiction, war, Director: mattr Reeves
thriller
Рекомендовані фільми:
Title: Mission: Impossible - Rogue Nation (2015), Genres: action, adventure, thriller, Director: christophermcquarrie
Title: Gone Girl (2014), Genres: mystery, thriller, drama, Director: davidfincher
Title: John Wick (2014), Genres: action, thriller, Director: chadstahelski
Title: Nightcrawler (2014), Genres: crime, drama, thriller, Director: dangilroy
Title: The Imitation Game (2014), Genres: history, drama, thriller, Director: mortentyldum
Title: The Hunger Games: Mockingjay - Part 1 (2014), Genres: sciencefiction, adventure, thriller, Director: francislawrence
Title: Room (2015), Genres: drama, thriller, Director: lennyabrahamson
Title: Spotlight (2015), Genres: drama, thriller, history, Director: tommccarthy
Title: Get Out (2017), Genres: mystery, thriller, horror, Director: jordanpeele
Title: The Fate of the Furious (2017), Genres: action, crime, thriller, Director: f.garygray

```

Рисунок. 2.18 - Рекомендація фільмів для старих користувачів (2 підхід)

2.6 Аналіз результатів

У ході тестування було застосовано кілька моделей, зокрема SVD, KNNWithZScore та SVDpp. Згідно з отриманими результатами, модель SVD показала найкращу продуктивність за показником кореня середньоквадратичної похибки (RMSE). Після оптимізації гіперпараметрів SVD значно перевершила інші моделі, забезпечуючи більш точні прогнози рейтингів і була обрана для створення API для реалізації рекомендаційної системи.

Гібридний підхід до прогнозування рейтингу фільмів поєднує в собі колаборативну фільтрацію, контент-орієнтовану фільтрацію та вагове оцінювання. Такий підхід забезпечує більш точні та персоналізовані рекомендації. Наприклад, прогнозований рейтинг для користувача з ідентифікатором 1 і фільму з ідентифікатором 5 складав 1.89, що є результатом зваженого середнього значення трьох підходів.

Для користувачів, які мають історичні дані, система використовує комбінацію колаборативної фільтрації та контент-орієнтованої фільтрації. Такий підхід забезпечує більш точні рекомендації, враховуючи як минулі рейтинги користувачів, так і схожість фільмів з останніми переглянутими. Наприклад, для користувача з ідентифікатором 10 були рекомендовані фільми, які відповідають його попереднім вподобанням і мають високі рейтинги.

Для вирішення проблеми «холодного старту» було розроблено два підходи: рекомендація популярних фільмів та рекомендація фільмів за жанрами. Перший підхід пропонує новим користувачам топ-рейтингові та останні популярні фільми, що забезпечує загальну популярність і високу оцінку. Другий підхід дозволяє новим користувачам отримати рекомендації відповідно до їх жанрових уподобань, що робить рекомендації більш персоналізованими з самого початку.

Загалом, рекомендаційна система для фільмів продемонструвала високу ефективність і здатність до адаптації. Використання гібридного підходу дозволило поєднати переваги кількох методів і забезпечити більш точні

прогнози. Система успішно справляється як із старими, так і з новими користувачами, надаючи їм персоналізовані та релевантні рекомендації.

Висновки до другого розділу

Було розглянуто основні підходи до рекомендаційних систем, зокрема колаборативну фільтрацію, контент-орієнтовані підходи та гібридні моделі. Колаборативна фільтрація використовує інформацію про схожість між користувачами або елементами для рекомендацій, контент-орієнтовані підходи використовують характеристики елементів для рекомендацій, а гібридні моделі комбінують обидва підходи.

Було описано методи та алгоритми інтелектуального аналізу даних, які можуть бути використані для рішення задачі рекомендаційних систем для фільмів. Ці методи включають базові моделі, такі як глобальне середнє значення, середнє значення користувача та середнє значення елемента, а також складніші методи, такі як колаборативна фільтрація та контентна фільтрація. Гібридні підходи комбінують ці методи для отримання більш точних та персоналізованих рекомендацій.

Застосування цих методів та алгоритмів інтелектуального аналізу даних дозволить розробити ефективну рекомендаційну систему для фільмів, яка забезпечить високу точність рекомендацій та покращить досвід користувачів у виборі фільмів. Гібридний підхід, який об'єднує переваги різних методів, може виявитися особливо ефективним для досягнення цієї мети.

В даному розділі було досліджено джерела даних для проведення аналізу. Було розглянуто ключові аспекти цього набору даних у табличному поданні. Було виконано очищення та коригування набору даних.

Реалізовано базові моделі, включаючи середній рейтинг та зважений середній рейтинг. RMSE склав 0.92 Surprise. Обчислено зважений рейтинг для покращення рекомендацій. Розроблено функцію для прогнозування рейтингів фільмів для користувачів. Згенеровано рекомендації для користувачів за допомогою гібридної системи рекомендацій.

Було проведено аналіз результатів та тепер можемо перейти до безпосереднього створення застосунку.

РОЗДІЛ 3 СТВОРЕННЯ ВЕБЗАСТОСУНКУ ТА ІНТЕГРАЦІЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ

3.1 Проектування бази даних та обробка даних

Для початку спроектуємо на базі наявних даних базу даних. Виконаємо це за допомогою системи керування базами даних (СКБД) MySQL, оскільки вона забезпечує широку підтримку розробників і користувачів, демонструє високу продуктивність при роботі з великими обсягами даних, легко інтегрується з іншими технологіями та мовами програмування, а також має розвинуті механізми безпеки.

На рис. 3.1 зображено ER-діаграму бази даних, спроектованої системи.

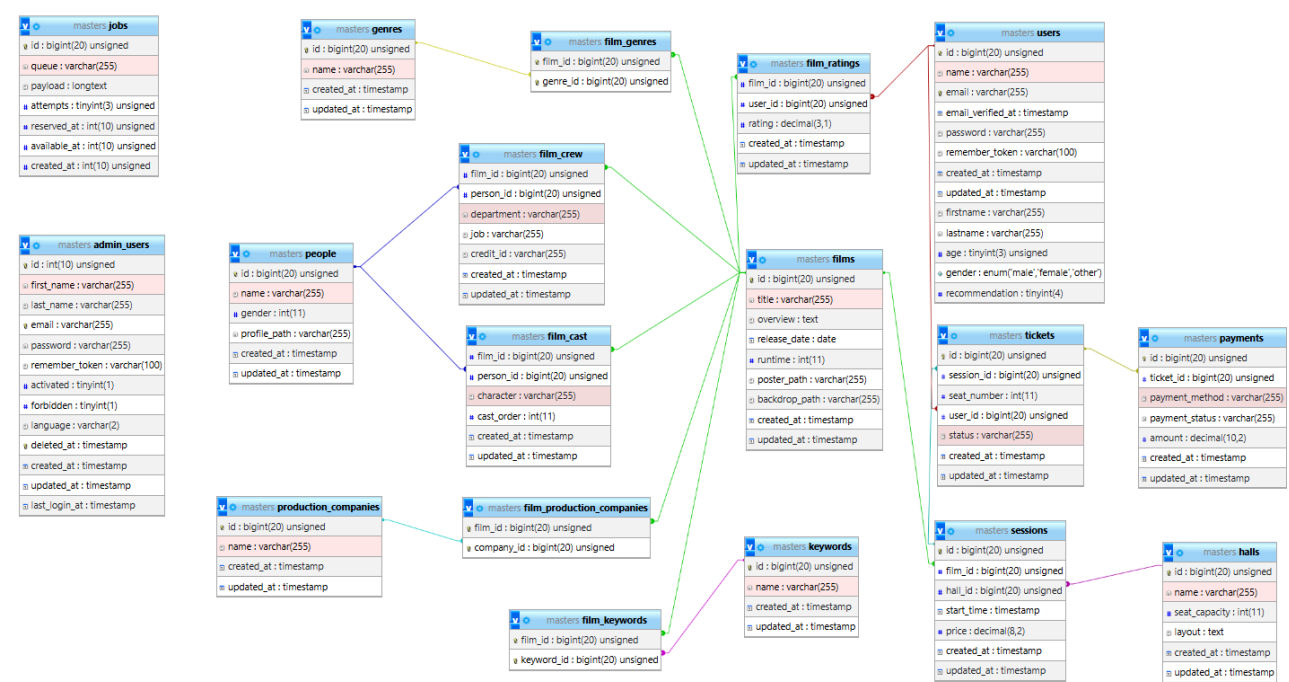


Рисунок. 3.1 - ER-діаграма бази даних, спроектованої системи

Опис головних таблиць, полів та їх призначення наведені в табл. 3.1-3.10.

Табл. 3.1 відповідає за збереження загальної інформацію про фільм, включаючи назву, короткий опис, дату виходу, тривалість та постер.

Таблиця 3.1

Структура таблиці films

Поле	Ключ	Тип	Призначення
id	PK	bigint(20)	Унікальний ідентифікатор фільму.
title	-	varchar(255)	Назва фільму.
overview	-	text	Короткий опис фільму.
release_date	-	date	Дата виходу фільму.
runtime	-	int(11)	Тривалість фільму (у хвилинах).
poster_path	-	varchar(255)	Шлях до постера фільму.
backdrop_path	-	varchar(255)	Шлях до бекграунду фільму.
created_at	-	timestamp	Дата та час створення запису.
updated_at	-	timestamp	Дата та час останнього оновлення запису.

Табл. 3.2 відповідає за збереження оцінок користувача, які він виставляє після перегляду.

Таблиця 3.2

Структура таблиці film_ratings

Поле	Ключ	Тип	Призначення
id	PK	bigint(20)	Унікальний ідентифікатор запису.
film_id	FK	bigint(20)	Ідентифікатор фільму, до якого належить рейтинг.
user_id	FK	bigint(20)	Ідентифікатор користувача, який залишив рейтинг.
rating	-	decimal(3,1)	Оцінка, надана користувачем фільму (наприклад, від 0.0 до 10.0).
created_at	-	timestamp	Дата та час створення запису.
updated_at	-	timestamp	Дата та час останнього оновлення запису.

Табл. 3.3 відповідає за збереження користувачів у системі.

Таблиця 3.3

Структура таблиці users

Поле	Ключ	Тип	Призначення
1	2	3	4
id	PK	bigint(20)	Унікальний ідентифікатор користувача.

Продовження табл. 3.3

1	2	3	4
name	-	varchar(255)	Ім'я користувача.
email	-	varchar(255)	Email користувача.
password	-	varchar(255)	Хешований пароль.
firstname	-	varchar(255)	Ім'я (перше) користувача.
age	-	tinyint(3)	Вік користувача.
gender	-	enum('male', 'female', 'other')	Стать користувача.
recommendation	-	tinyint(4)	Джерело надсилання рекомендацій
created_at	-	timestamp	Дата та час створення запису.
updated_at	-	timestamp	Дата та час останнього оновлення запису.

Табл. 3.4 відповідає за збереження жанрів у системі.

Таблиця 3.4

Структура таблиці genres

Поле	Ключ	Тип	Призначення
id	PK	bigint(20)	Унікальний ідентифікатор жанру.
name	-	varchar(255)	Назва жанру
created_at	-	timestamp	Дата та час створення запису.
updated_at	-	timestamp	Дата та час останнього оновлення запису.

Табл. 3.5 відповідає за збереження зв'язків між фільмами та жанрами у системі.

Таблиця 3.5

Структура таблиці film_genres

Поле	Ключ	Тип	Призначення
id	PK	bigint(20)	Унікальний ідентифікатор запису.
film_id	FK	bigint(20)	Ідентифікатор фільму.
genre_id	FK	bigint(20)	Ідентифікатор жанру.
created_at	-	timestamp	Дата та час створення запису.
updated_at	-	timestamp	Дата та час останнього оновлення запису.

Табл. 3.6 відповідає за ключові слова до фільмів у системі.

Таблиця 3.6

Структура таблиці keywords

Поле	Ключ	Тип	Призначення
id	РК	bigint(20)	Унікальний ідентифікатор ключового слова.
name	-	varchar(255)	Ключове слово
created_at	-	timestamp	Дата та час створення запису.
updated_at	-	timestamp	Дата та час останнього оновлення запису.

Табл. 3.7 відповідає за зв'язок між ключовими словами і фільмами у системі.

Таблиця 3.7

Структура таблиці film_keywords

Поле	Ключ	Тип	Призначення
id	РК	bigint(20)	Унікальний ідентифікатор запису.
film_id	FK	bigint(20)	Ідентифікатор фільму.
keyword_id	FK	bigint(20)	Ідентифікатор ключового слова.
created_at	-	timestamp	Дата та час створення запису.
updated_at	-	timestamp	Дата та час останнього оновлення запису.

Табл. 3.8 відповідає за збереження адміністраторів у системі.

Таблиця 3.8

Структура таблиці admin_users

Поле	Ключ	Тип	Призначення
1	2	3	4
first_name	-	varchar(255)	Ім'я адміністратора.
last_name	-	varchar(255)	Прізвище адміністратора.
email	-	varchar(255)	Email адміністратора.
password	-	varchar(255)	Хешований пароль.
remember_token	-	varchar(100)	Токен для функції запам'ятати мене.
activated	-	tinyint(1)	Чи активований акаунт (1 - так, 0 - ні).
language	-	varchar(2)	Обрана мова інтерфейсу.

Продовження табл. 3.8

1	2	3	4
created_at	-	timestamp	Дата та час створення запису.
updated_at	-	timestamp	Дата та час останнього оновлення запису.
last_login_at	-	timestamp	Дата та час останнього входу адміністратора.

Табл. 3.9 відповідає за збереження запланованих дій у системі.

Таблиця 3.9

Структура таблиці jobs

Поле	Ключ	Тип	Призначення
id	РК	bigint(20)	Унікальний ідентифікатор завдання.
queue	-	varchar(255)	Назва черги, до якої належить завдання.
payload	-	longtext	Дані завдання в серіалізованій формі.
attempts	-	tinyint(3)	Кількість спроб виконання завдання.
reserved_at	-	int(10)	Час резервування завдання.
available_at	-	int(10)	Час, коли завдання стане доступним для виконання.
created_at	-	int(10)	Час створення завдання.

Після того, як було побудовано базу даних за допомогою міграцій, необхідно імпортувати дані у створену таблиці. Для цього скористаємося сідерами, що надаються фреймворком Laravel. Це спеціальні класи, що призначені для заповнення бази даних тестовими значеннями.

Аналогічним чином було заповнено всі інші таблиці, дані яких наявні у csv-файлах. Таблиці, які ще не мали даних в файлах, були заповнені з використанням бібліотеки Faker [38], що надає можливість отримання багатьох характеристик випадковим чином.

Аналогічно було заповнено інші таблиці.

На рис. 3.2 зображено фрагмент класу FilmSeeder для заповнення таблиці films.

На рис. 3.3 зображено заповнення таблиці users за допомогою бібліотеки Faker.

```

class FilmSeeder extends Seeder
{
    function isValidDate($date, $format = 'Y-m-d')
    {
        $dateTime = DateTime::createFromFormat($format, $date);
        return $dateTime && $dateTime->format($format) === $date;
    }

    public function run()
    {
        try {
            $filePath = storage_path('app/csv/movies.csv');
            $rows = array_map('str_getcsv', file($filePath));
            array_shift($rows); // Skip header row

            foreach ($rows as $row) {
                foreach ($row as $key => $value) {
                    if ($value === '[]' || $value === '' || $value === '0') {
                        $row[$key] = null;
                    }
                }
                if (!isset($row[8]) || !$row[8] || !is_numeric($row[5])) {
                    continue;
                }

                if (!Film::find((int)$row[5])) {
                    Film::create(
                        [
                            'id' => $row[5],
                            'title' => $row[8],
                            'overview' => $row[9] ?? null,
                            'release_date' => isset($row[14]) && $this->isValidDate($row[14]) ? $row[14] : null,
                            'runtime' => isset($row[16]) && is_numeric($row[16])
                                ? $row[16] : null,
                            'poster_path' => $row[11] ?? null,
                            'created_at' => now(),
                            'updated_at' => now(),
                        ]
                    );
                }
            }
        }
    }
}

```

Рисунок. 3.2 - Фрагмент класу FilmSeeder для заповнення таблиці films

```

class UsersTableSeeder extends Seeder
{
    /**
     * Run the database seeds.
     */
    public function run(): void
    {
        $faker = \Faker\Factory::create();

        for ($i = 0; $i < 1000; $i++) {
            User::create([
                'firstname' => $faker->firstName,
                'lastname' => $faker->lastName,
                'name' => $faker->name,
                'age' => $faker->numberBetween(int1: 14, int2: 65),
                'gender' => $faker->randomElement(['male', 'female', 'other']),
                'email' => $faker->unique()->safeEmail,
                'password' => Hash::make(value: 'password'),
            ]);
        }
    }
}

```

Рисунок. 3.3 - Фрагмент класу UserTableSeeder для заповнення таблиці users

3.2 Розробка адмін-панелі та API вебзастосунку

Тепер створимо адмін-панель, щоб адміністратори могли редагувати вміст вебзастосунку. Скористаємося бібліотекою Laravel Craftable [39] для її створення. Після встановлення необхідних розширень за допомогою інструмента керування залежностями Composer [40]. Отримуємо згенерований застосунок з ролями та авторизацією.

На рис. 3.4 зображено головну сторінку адмін-панелі.

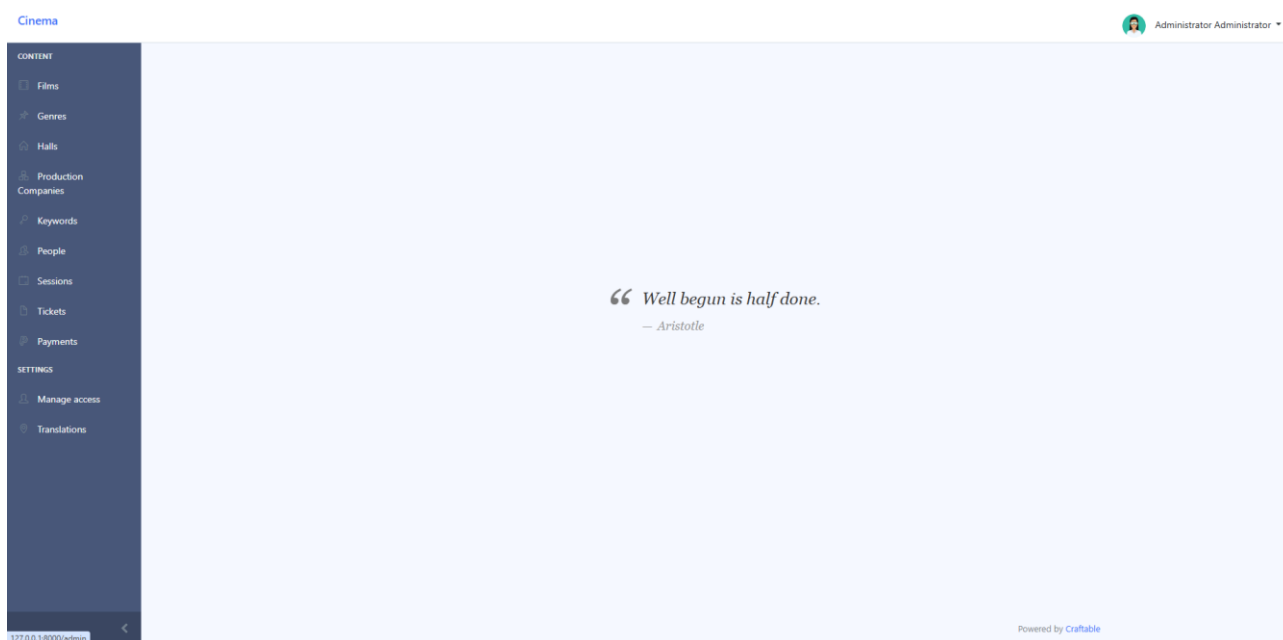


Рисунок. 3.4 - Головна сторінка адмін-панелі

Далі згенеруємо всі необхідні CRUD, що адміністратор міг повністю керувати вмістом. Для прикладу розглянемо клас FilmController.

Для цього виконаємо наступну команду, що відповідає за створення CRUD для певної сутності: `php artisan admin:generate films`. Після того, як команда виконалася отримуємо готовий контролер, вигляд, форму та лістинг.

Далі редагуємо їх, щоб вказані файли відповідали логіці розроблюваного вебзастосунку.

На рис. 3.5-3.6 зображено методи класу FilmsController, що відповідають за перегляд всіх фільмів та редагування певного фільму.

```

public function index(IndexFilm $request)
{
    // create and AdminListing instance for a specific model and
    $data = AdminListing::create(Film::class)->processRequestAndGet(
        // pass the request with params
        $request,

        ['id', 'title', 'release_date', 'runtime', 'poster_path', 'backdrop_path'],

        ['id', 'title', 'overview', 'poster_path', 'backdrop_path']
    );
    $data->load('genres');
    $data->load('productionCompanies');
    $data->transform(function ($film) {
        $film->companies = $film->productionCompanies->take(3)->pluck('name')->implode(', ');
        $film->genre = $film->genres->take(3)->pluck('name')->implode(', ');
        return $film;
    });
    if ($request->ajax()) {
        if ($request->has('key: bulk')) {
            return [
                'bulkItems' => $data->pluck('id')
            ];
        }
        return ['data' => $data];
    }

    return view('admin.film.index', ['data' => $data]);
}

```

Рисунок. 3.5 - Метод index класу FilmsController

```

public function update(UpdateFilm $request, Film $film)
{
    // Sanitize input
    $sanitized = $request->getSanitized();
    $genres = $sanitized['genres'] ?? [];
    $cast = $sanitized['cast'] ?? [];
    $crew = $sanitized['crew'] ?? [];
    $keywords = $sanitized['keywords'] ?? [];
    $productionCompanies = $sanitized['production_companies'] ?? [];
    unset($sanitized['genres'], $sanitized['cast'], $sanitized['crew'],
    $sanitized['keywords'], $sanitized['production_companies']);

    $film->update($sanitized);

    if (!empty($genres)) {
        $genreIds = array_column($genres, 'id');
        $film->genres()->sync($genreIds);
    }
    if (!empty($cast)) {
        $castIds = array_column($cast, 'id');
        $film->cast()->sync($castIds);
    }
    if (!empty($crew)) {
        $crewIds = array_column($crew, 'id');
        $film->crew()->sync($crewIds);
    }
    if (!empty($keywords)) {
        $keywordIds = array_column($keywords, 'id');
        $film->keywords()->sync($keywordIds);
    }
    if (!empty($productionCompanies)) {
        $productionCompanyIds = array_column($productionCompanies, 'id');
        $film->productionCompanies()->sync($productionCompanyIds);
    }

    if ($request->ajax()) {
        return [
            'redirect' => url('admin/films'),
            'message' => trans('brackets/admin-ui::admin.operation.succeeded'),
        ];
    }

    return redirect('admin/films');
}

```

Рисунок. 3.6 - Метод update класу FilmsController

Аналогічним чином генеруємо та адаптуємо всі необхідні сутності системи.

На рис. 3.7-3.8 зображено вигляди сутності Film для перегляду всіх фільмів та оновлення певного фільму.

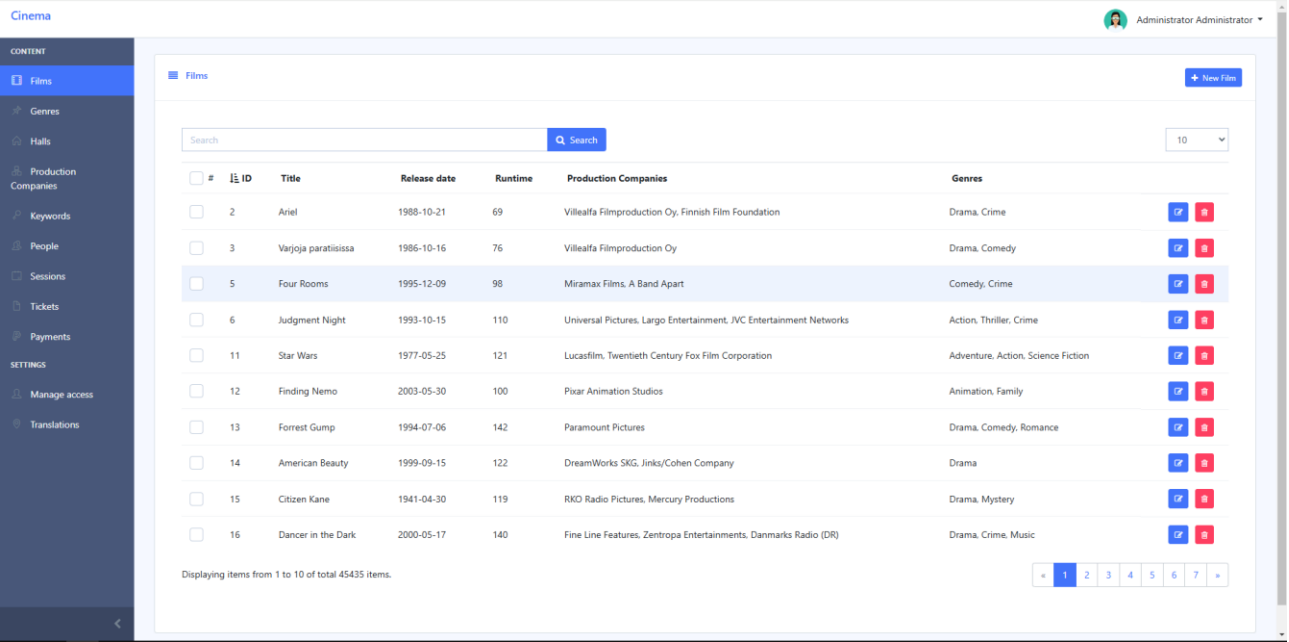


Рисунок. 3.7 - Вигляд сторінки перегляду всіх фільмів

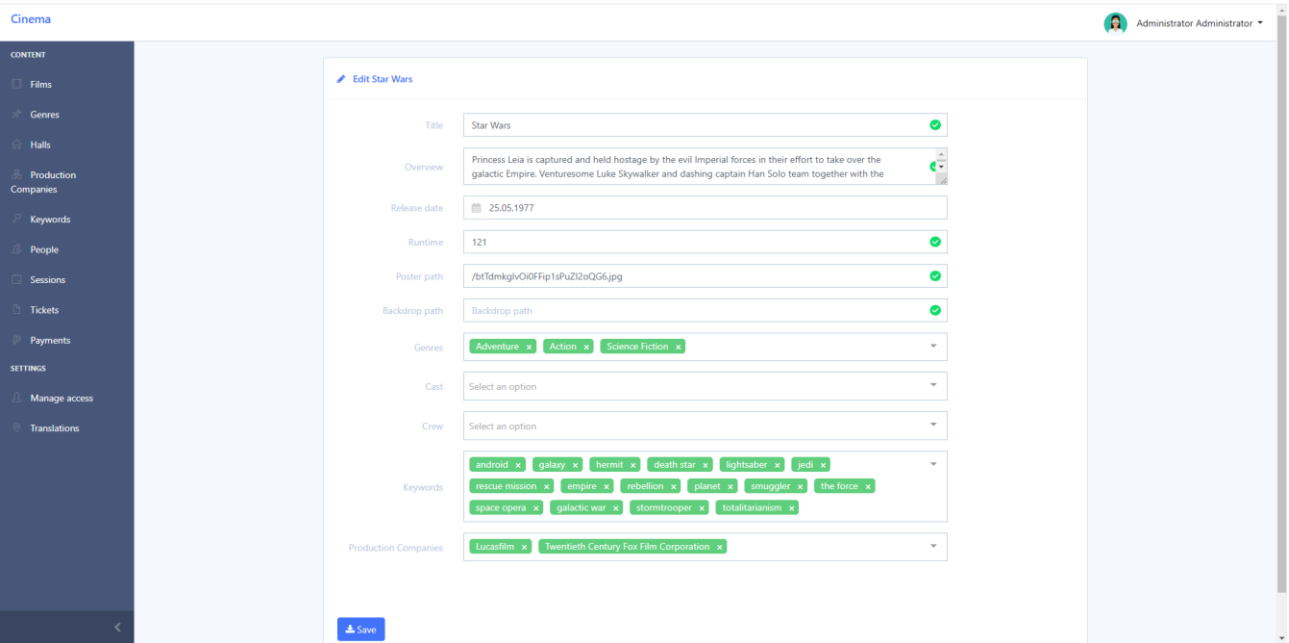


Рисунок. 3.8 - Вигляд сторінки перегляду оновлення певного фільму

Наступним кроком було створено API для повернення даних клієнту для подальшого його відображення.

Було розроблено 4 контролери, які відповідають за отримання фільмів, сеансів, квитків та оплат, а також за авторизацію користувачів. Для прикладу розглянемо отримання всіх фільмів та отримання сеансів певного фільму.

На рис. 3.9 зображено реалізацію метода index контролера FilmController.

```
public function index(Request $request)
{
    try {
        $query = Film::query()->with(['genres', 'productionCompanies', 'cast',
        'crew']);

        if ($request->has('title')) {
            $query->where('title', 'like', '%' . $request->title . '%');
        }
        if ($request->has('genre')) {
            $query->whereHas('genres', function ($q) use ($request) {
                $q->where('name', $request->genre);
            });
        }
        if ($request->has('production_company')) {
            $query->whereHas('productionCompanies', function ($q) use ($request) {
                $q->where('name', $request->production_company);
            });
        }
        if ($request->has('cast')) {
            $query->whereHas('cast', function ($q) use ($request) {
                $q->where('name', $request->cast);
            });
        }
        if ($request->has('crew')) {
            $query->whereHas('crew', function ($q) use ($request) {
                $q->where('name', $request->crew);
            });
        }
        if ($request->has('rating')) {
            $query->where('rating', $request->rating);
        }
        if ($request->has('release_date')) {
            $query->where('release_date', $request->release_date);
        }

        return response()->json(['films' => $query->paginate(10)->items()]);
    } catch (\Exception $e) {
        return response()->json(['message' => 'An error occurred'], 500);
    }
}
```

Рисунок. 3.9 - Реалізація метода index контролера FilmController

На рис. 3.10 зображено результат роботи метода index контролера FilmController, отримані за допомогою Postman.

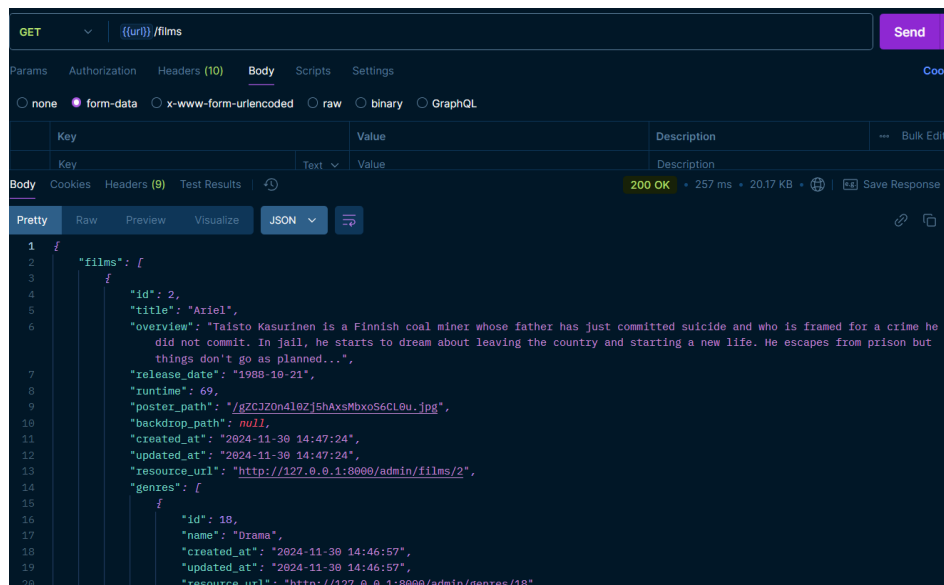


Рисунок. 3.10 - Результат роботи метода index контролера FilmController

На рис. 3.11 зображено реалізацію метода index контролера SessionController.

```
public function index($id)
{
    try {
        $film = Film::findOrFail($id);
        $sessions = $film->sessions()
            ->where('start_time', '>', Carbon::now())
            ->with(['hall'])
            ->get();

        return response()->json($sessions);
    } catch (\Exception $e) {
        return response()->json(['message' => 'Film not found'], status: 404);
    }
}
```

Рисунок. 3.11 - Реалізація метода index контролера SessionController

На рис. 3.12 зображено результат роботи метода index контролера SessionController.

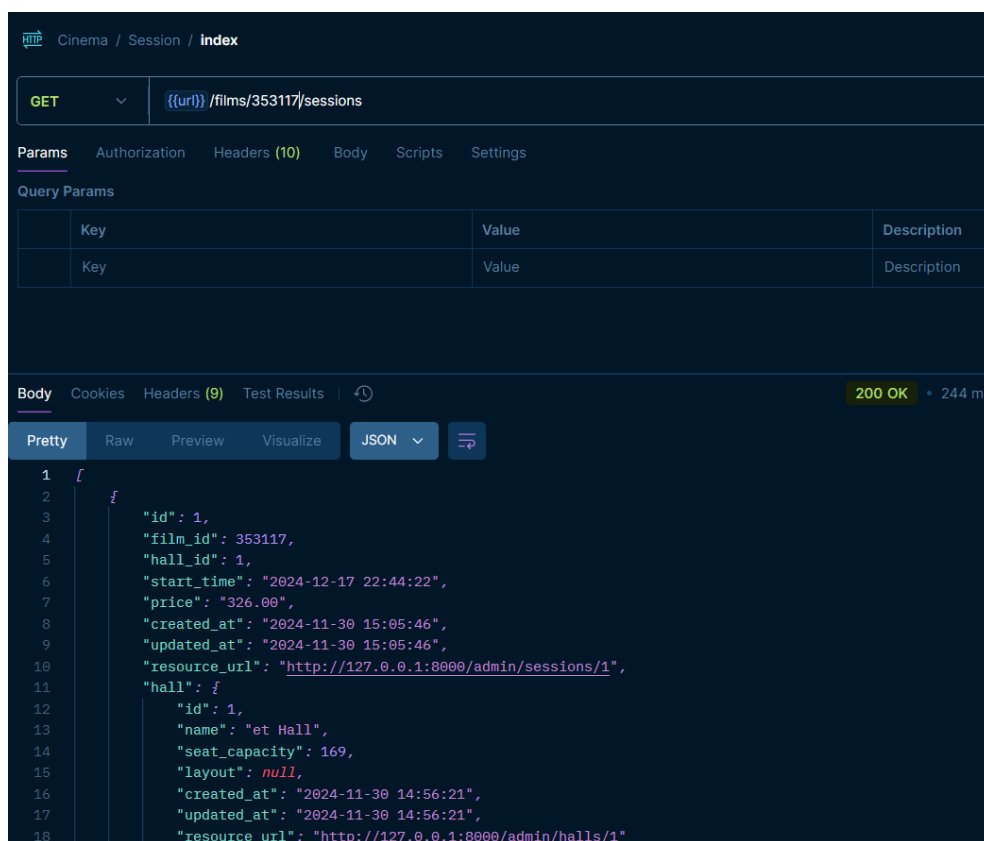


Рисунок. 3.12 - Результат роботи метода index контролера SessionController

3.3 Інтеграція рекомендаційної системи у вебзастосунок

Тепер розглянемо інтеграцію розробленої рекомендаційної системи зі вебзастосунком. Розпочнемо з створення API для розробленої рекомендаційної

системи. Для цього збережемо отриману модель та попередньо обчислених даних за допомогою бібліотеки `joblib` у відповідні файли.

На рис. 3.13 зображено навчання та збереження моделі у файл.

```
reader = Reader(rating_scale=(1, 5))
train_data = Dataset.load_from_df(ratings_df[['userId', 'movieId', 'rating']], reader)
param_grid = {
    'n_factors': [125, 150, 175],
    'n_epochs': [25, 30, 35],
    'lr_all': [0.008, 0.01, 0.012],
    'reg_all': [0.15, 0.2, 0.25]
}
grid_search = GridSearchCV(SVD, param_grid, measures=['rmse'], cv=3, refit=True)

grid_search.fit(train_data)
joblib.dump(grid_search, MODEL_PATH)
```

Рисунок. 3.13 - Навчання та збереження моделі у файл

Після збереження моделі можемо перейти до безпосереднього створення API. Для його розробки скористаємося фреймворком `FastAPI`, що надає можливість швидкого створення API. При запуску сервера необхідно записати в пам'ять збережену модель.

На рис. 3.14 зображено ініціалізація застосунку `FastAPI` та отримання збереженої моделі при запуску сервера.

```
def load_resources():
    global svd_model, cosine_sim_matrix, weighted_df, movies_df, ratings_df, movies_df_clean
    svd_model = joblib.load(MODEL_PATH)
    cosine_sim_matrix = joblib.load(SIM_MATRIX_PATH)
    weighted_df = joblib.load(WEIGHTED_SCORES_PATH)
    movies_df = pd.read_csv(MOVIES_PATH, low_memory=False)
    movies_df_clean = joblib.load(MOVIES_CLEAN_PATH)
    ratings_df = pd.read_csv(RATINGS_PATH)
    print("Models and datasets loaded successfully.")

@app.on_event("startup")
def startup_event():
    load_resources()
```

Рисунок. 3.14 - Ініціалізація застосунку `FastAPI` та отримання збереженої моделі при запуску сервера

Тепер створимо відповідні ендпоїнти для отримання рекомендацій для старих та нових користувачів.

На рис. 3.15 зображені ендпоїнти для отримання рекомендацій для старих та нових користувачів.

```
@app.get("/recommendations/{user_id}")
def get_recommendations(user_id: int, n: int = 10):
    try:
        recommended_movies = hybrid_recommendation(user_id, n)
        return {"user_id": user_id, "recommended_movies": recommended_movies}
    except Exception as e:
        return {"error": str(e)}
```

```
@app.get("/recommendations")
def get_recommendations_for_new_user(n: int = 10, min_year: int = 10):
    try:
        recommendations = recommend_for_new_user_top_rating_movies(
            weighted_df, movies_df_clean, n=n, min_year=min_year
        )
        recommendations_list = recommendations['id'].tolist()
        return {"recommended_movies": recommendations_list}
    except Exception as e:
        return {"error": str(e)}
```

Рисунок. 3.15 - Ендпоїнти для отримання рекомендацій для старих та нових користувачів

Перевіримо роботу створених ендпоїнтів за допомогою Postman.

На рис. 3.16 зображено отримані фільми для користувача з id=5.

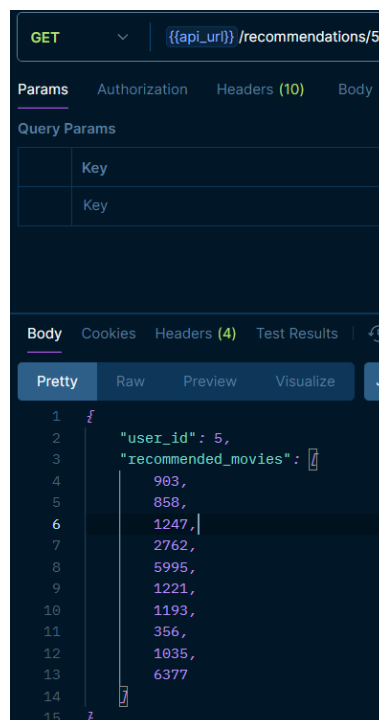


Рисунок. 3.16 - Отримані фільми для користувача з id=5

На рис. 3.17 зображено отримані фільми для нового користувача.

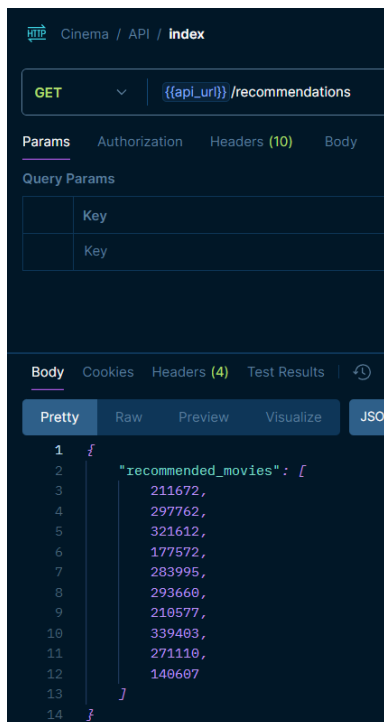


Рисунок. 3.17 - Отримані фільми для нового користувача

Також передбачимо оновлення даних для моделі для цього створимо ендпоінт, який буде отримувати актуальні csv-файли з вебзастосунку і використовувати їх для періодичного перенавчання моделі.

На рис. 3.18 зображено ендпоінт для оновлення csv-файлів.

```

entities = {
    "movies": "movies.csv",
    "keywords": "keywords.csv",
    "credits": "credits.csv",
    "ratings": "ratings_small.csv"
}

@app.post("/upload/{entity}")
async def upload_file(entity: str, file: UploadFile = File(...)):
    file_path = entities.get(entity)
    if os.path.exists(file_path):
        os.remove(file_path)

    content = await file.read()
    with open(f"./{file_path}", "wb") as f:
        f.write(content)
    return {"filename": file_path, "status": "uploaded"}

```

Рисунок. 3.18 - Ендпоінт для оновлення csv-файлів

Тепер перейдемо до інтеграції створеного API у вебзастосунок. Для реалізації скористаємося підходом запланованих команд та черг. Реалізацію цих

підходів фреймворк Laravel, їх необхідно лише налаштувати. Після змін в конфігураційних файлах приступимо до створення необхідного функціоналу. Для цього створимо job, певна дія, посилення на яку зберігається в певній таблиці і виконується, коли до неї доходить черга. Реалізуємо звернення до рекомендаційної системи для певного користувача і відправку рекомендацій йому на пошту.

На рис. 3.19 зображено фрагмент створеного job для обробки рекомендацій певного користувача.

```
private function sendMail(): void
{
    $recommendation = $this->getRecommendation();
    if (empty($recommendation)) {
        $recommendation = $this->getRecommendation(retry: true);
    }
    $films = [];
    foreach ($recommendation as $id) {
        $film = Film::find($id);

        if ($film) {
            $film->load('genres', 'productionCompanies', 'cast', 'crew');
            $sessions = $film->sessions->whereBetween('start_time', [now(), now()->addMonth()]);

            if ($sessions->isEmpty())
                $films[] = [
                    'id' => $film->id,
                    'title' => $film->title,
                    'overview' => $film->overview,
                    'release_date' => $film->release_date,
                    'runtime' => $film->runtime,
                    'genres' => $film->genres->pluck('name')->toArray(),
                    'companies' => $film->productionCompanies->pluck('name')->toArray(),
                    'cast' => $film->cast->pluck('name')->toArray(),
                    'crew' => $film->crew->pluck('name')->toArray(),
                ];
        }
    }

    if (count($films) === 0) {
        Log::info('message: 'No recommendation for user ' . $this->user->id);
        return;
    }

    Mail::to($this->user->email)->send(new RecommendationMail($this->user, $films));
}
```

Рисунок. 3.19 - Фрагмент створеного job для обробки рекомендацій певного користувача

Тепер створимо команду, що буде виконувати створені вище jobs щотижня.

створеного вище, щоб зберегти файл на стороні API для подальших донавань рекомендаційної системи.

На рис. 3.22 зображено отриманий csv-файл на стороні API.

ue	runtime	spoken_languages	status	tagline	title	video
1	69	<null>	<null>	<null>	Ariel	<null>
2	76	<null>	<null>	<null>	Varjoja paratiisissa	<null>
3	98	<null>	<null>	<null>	Four Rooms	<null>
4	110	<null>	<null>	<null>	Judgment Night	<null>
5	121	<null>	<null>	<null>	Star Wars	<null>
6	100	<null>	<null>	<null>	Finding Nemo	<null>
7	142	<null>	<null>	<null>	Forrest Gump	<null>
8	122	<null>	<null>	<null>	American Beauty	<null>
9	119	<null>	<null>	<null>	Citizen Kane	<null>
10	140	<null>	<null>	<null>	Dancer in the Dark	<null>
11	87	<null>	<null>	<null>	The Dark	<null>
12	126	<null>	<null>	<null>	The Fifth Element	<null>
13	153	<null>	<null>	<null>	Metropolis	<null>

Рисунок. 3.22 - Отриманий csv-файл на стороні API

Висновки до третього розділу

У цьому розділі було спроектовано та побудовано базу даних з наявних даних у csv-файлах та згенерованих випадковим чином за допомогою розширення Faker. Докладно описано головні таблиці побудованої бази даних.

Також, було створено адмін-панель за допомогою розширення Laravel Craftable та адаптовано під потреби системи. Реалізовано сутності для зручного керування вмістом адміністраторами.

Реалізовано API вебзастосунку для реалізації клієнту кінотеатру. Були створені ендпоїнти для отримання інформації про один фільм, про всі фільми, про сеанс та певний сеанс, а також реалізацію оплати, оцінювання фільмів та авторизації.

Було створено API для рекомендаційної системи та реалізовано процес оновлення інформації. Було створено функціонал у вебзастосунку для розсилки рекомендацій фільмів на основі отриманих даних з API рекомендаційної системи щотижня. Для цього було використано вбудований у фреймворк Laravel функціонал, такий як: черги та заплановані команди. Було протестовано коректність роботи розсилки листів та оновлення даних.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи магістра було проведено дослідження принципів побудови та проєктування вебзастосунку з елементами машинного навчання для кінотеатру. Виконання поставленої задачі було розділено на 3 розділи.

У розділі «Аналіз предметної області та обґрунтування актуальності теми роботи» визначена постановка завдання для кваліфікаційної роботи, включаючи формулювання функціональних вимог та визначення основних етапів виконання проєкту. Здійснено аналіз аналогів продукту, що сприяло встановленню критеріїв розробки, таких як перелік активних та майбутніх фільмів, докладний опис фільму, коректна робота сторонніх API, коректна робота сповіщень, необов'язкова авторизація для купівлі квитка та наявність персоналізованих рекомендацій. Також було проведено аналіз існуючих архітектур, в результаті чого обрано використання схеми RESTful API. Основний вибір технологічного стеку для реалізації включає в себе використання Laravel для розробки бекенду, React для фронтенду та Python для створення рекомендаційної системи. Вимоги до серверного обладнання та робочої станції користувача визначені та додатково обґрунтовані в тексті.

У розділі «Дослідження та реалізація рекомендаційної системи фільмів» було розглянуто основні підходи до рекомендаційних систем, зокрема колаборативну фільтрацію, контент-орієнтовані підходи та гібридні моделі. Колаборативна фільтрація використовує інформацію про схожість між користувачами або елементами для рекомендацій, контент-орієнтовані підходи використовують характеристики елементів для рекомендацій, а гібридні моделі комбінують обидва підходи. Було описано методи та алгоритми інтелектуального аналізу даних, які можуть бути використані для рішення задачі рекомендаційних систем для фільмів. Ці методи включають базові моделі, такі як глобальне середнє значення, середнє значення користувача та середнє значення елемента, а також складніші методи, такі як колаборативна фільтрація

та контентна фільтрація. Гібридні підходи комбінують ці методи для отримання більш точних та персоналізованих рекомендацій. Застосування цих методів та алгоритмів інтелектуального аналізу даних дозволить розробити ефективну рекомендаційну систему для фільмів, яка забезпечить високу точність рекомендацій та покращить досвід користувачів у виборі фільмів. Гібридний підхід, який об'єднує переваги різних методів, може виявитися особливо ефективним для досягнення цієї мети. В даному розділі було досліджено джерела даних для проведення аналізу. Було розглянуто ключові аспекти цього набору даних у табличному поданні. Було виконано очищення та коригування набору даних. Реалізовано базові моделі, включаючи середній рейтинг та зважений середній рейтинг. RMSE склав 0.92 Surprise. Обчислено зважений рейтинг для покращення рекомендацій. Розроблено функцію для прогнозування рейтингів фільмів для користувачів. Згенеровано рекомендації для користувачів за допомогою гібридної системи рекомендацій.

У розділі «Створення вебзастосунку та інтеграція рекомендаційної системи» У цьому розділі було спроектовано та побудовано базу даних з наявних даних у csv-файлах та згенерованих випадковим чином за допомогою розширення Faker. Докладно описано головні таблиці побудованої бази даних. Також, було створено адмін-панель за допомогою розширення Laravel Craftable та адаптовано під потреби системи. Реалізовано сутності для зручного керування вмістом адміністраторами. Реалізовано API вебзастосунку для реалізації клієнту кінотеатру. Були створені ендпоїнти для отримання інформації про один фільм, про всі фільми, про сеанс та певний сеанс, а також реалізацію оплати, оцінювання фільмів та авторизації. Було створено API для рекомендаційної системи та реалізовано процес оновлення інформації. Було створено функціонал у вебзастосунку для розсилки рекомендацій фільмів на основі отриманих даних з API рекомендаційної системи щотижня. Для цього було використано вбудований у фреймворк Laravel функціонал, такий як: черги та заплановані команди. Було протестовано коректність роботи розсилки листів та оновлення даних.

Отже, завдання на кваліфікаційну роботу виконано. Реалізований програмний продукт є готовим до використання та відповідає сучасним вимогам до застосунку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Голубовська О. А. Коронавірусна хвороба 2019 / О. А. Голубовська, А. В. Шкурба, О. В. Безродна. – Київ: Професійні видання. Україна, 2023. – 300 с.
2. Spider-Man: No Way Home (2021) - IMDb [Електронний ресурс] – Режим доступу до ресурсу: https://www.imdb.com/title/tt10872600/?ref=mv_sr_srg_0_tt_8_nm_0_in_0_q_no%2520way%2520home.
3. Avatar: The Way of Water (2022) - IMDb [Електронний ресурс] – Режим доступу до ресурсу: https://www.imdb.com/title/tt1630029/?ref=mv_sr_srg_0_tt_8_nm_0_in_0_q_a_vatar%25202.
4. Документація фреймворку Laravel [Електронний ресурс] – Режим доступу до ресурсу: <https://laravel.com/docs/9.x>.
5. FastAPI framework, high performance, easy to learn, fast to code, ready for production [Електронний ресурс] – Режим доступу до ресурсу: <https://fastapi.tiangolo.com/>.
6. Документація бібліотеки React [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.legacy.reactjs.org/>.
7. Welcome! - The Apache HTTP Server Project [Електронний ресурс] – Режим доступу до ресурсу: <https://httpd.apache.org/>.
8. Документація MySQL [Електронний ресурс] – Режим доступу до ресурсу: <https://dev.mysql.com/doc/>.
9. Biehl, Matthias. RESTful API Design / Matthias Biehl. – API-University Press, 2016. – 10 с.
10. Ricci F. Recommender Systems Handbook / F. Ricci, L. Rokach, B. Shapira. – New York: Springer, 2022. – 1060 с.
11. Що таке машинне навчання? [Електронний ресурс] – Режим доступу до ресурсу: <https://gigacloud.ua/blog/navchannja/scho-take-mashinne-navchannja-jak-pracjue-ta-de-vikoristovuetsja>.

12. Welcome to Python.org [Електронний ресурс] – Режим доступу до ресурсу: <https://www.python.org/>.
13. Зараз у кіно, Афіша – Кінотеатр Multiplex | Київ, Харків, Дніпро, Херсон і вся Україна [Електронний ресурс] – Режим доступу до ресурсу: <https://multiplex.ua/>.
14. Планета Кіно в Києві (Blockbuster) – мережа кінотеатрів [Електронний ресурс] – Режим доступу до ресурсу: <https://planetakino.ua/>.
15. What is REST? [Електронний ресурс] – Режим доступу до ресурсу: <https://restfulapi.net/>.
16. GraphQL | A query language for your API [Електронний ресурс] – Режим доступу до ресурсу: <https://graphql.org/>.
17. JetBrains: Essential tools for software developers and teams [Електронний ресурс] – Режим доступу до ресурсу: <https://www.jetbrains.com/>.
18. Посібник PHP [Електронний ресурс] – Режим доступу до ресурсу: <https://www.php.net/manual/en/index.php>.
19. HTML basics - Learn web development | MDN [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics.
20. CSS: Cascading Style Sheets | MDN [Електронний ресурс] – Режим доступу до ресурсу: <https://developer.mozilla.org/en-US/docs/Web/CSS>.
21. Сучасний підручник з JavaScript [Електронний ресурс] – Режим доступу до ресурсу: <https://uk.javascript.info/>.
22. Документація Symfony [Електронний ресурс] – Режим доступу до ресурсу: <https://symfony.com.ua/doc/current/index.html>.
23. Документація Yii2 [Електронний ресурс] – Режим доступу до ресурсу: <https://www.yiiframework.com/doc/guide/2.0/en>.
24. Mikowski M. Single Page Web Applications: JavaScript end-to-end / M. Mikowski, J. Powell. – New York: Manning, 2013. – 432 с.
25. Документація Angular [Електронний ресурс] – Режим доступу до ресурсу: <https://angular.io/docs>.

26. Документація Vue.js [Електронний ресурс] – Режим доступу до ресурсу: <https://vuejs.org/guide/introduction.html>.
27. Open Server Panel [Електронний ресурс] – Режим доступу до ресурсу: https://programy.com.ua/ua/open_server/.
28. Bringing MySQL to the web [Електронний ресурс] – Режим доступу до ресурсу: <https://www.phpmyadmin.net/>.
29. NumPy documentation – NumPy v1.26 Manual [Електронний ресурс] – Режим доступу до ресурсу: <https://numpy.org/doc/stable/index.html>.
30. pandas – Python Data Analysis Library [Електронний ресурс] – Режим доступу до ресурсу: <https://pandas.pydata.org/>.
31. scikit-learn: machine learning in Python – scikit-learn 1.5.2 documentation [Електронний ресурс] – Режим доступу до ресурсу: <https://scikit-learn.org/stable/>.
32. Surprise · A Python scikit for recommender systems. [Електронний ресурс] – Режим доступу до ресурсу: <https://surpriselib.com/>.
33. Welcome to Flask — Flask Documentation (3.1.x) [Електронний ресурс] – Режим доступу до ресурсу: <https://flask.palletsprojects.com/en/stable/>.
34. Phung D. Encyclopedia of Machine Learning and Data Science / D. Phung, G. Webb, C. Sammut. – New York: Springer, 2020. – 1975 с.
35. Golub G. Matrix Computations / G. Golub, C. Van Loan. Matrix Computations. – The Johns Hopkins University Press, 2013. – 756 с.
36. Root Mean Square Error (RMSE) [Електронний ресурс] – Режим доступу до ресурсу: <https://itwiki.dev/data-science/ml-reference/ml-glossary/root-mean-square-error>.
37. What is Mean Absolute Error? Formula & Significance [Електронний ресурс] – Режим доступу до ресурсу: <https://www.deepchecks.com/glossary/mean-absolute-error/>.
38. FakerPHP / Faker [Електронний ресурс] – Режим доступу до ресурсу: <https://fakerphp.org/>.

39. Craftable - Laravel CRUD generator [Электронный ресурс] – Режим доступа до ресурсу: <https://getcrafter.com/>.
40. A Dependency Manager for PHP [Электронный ресурс] – Режим доступа до ресурсу: <https://getcomposer.org/>

ДОДАТКИ

Базові моделі

```
# Розділення даних на навчальні та тестові набори (75% навчальних, 25% тестових)
train_df, test_df = train_test_split(ratings_df, test_size=0.25, random_state=42)

# Функція для обчислення RMSE
def calculate_rmse(y_true, y_pred):
    return sqrt(mean_squared_error(y_true, y_pred))

# Функція для обчислення MSE
def calculate_mse(y_true, y_pred):
    return mean_squared_error(y_true, y_pred)

from sklearn.model_selection import train_test_split

# Обчислення середнього глобального рейтингу на основі навчальної вибірки
global_mean_rating = train_df['rating'].mean()

# Прогнозування середнього глобального рейтингу для тестового набору
test_df['global_mean_rating'] = global_mean_rating

# Обчислення середніх оцінок користувачів та товарів на основі навчальної вибірки
user_mean_ratings = train_df.groupby('userId')['rating'].mean().reset_index()
item_mean_ratings = train_df.groupby('movieId')['rating'].mean().reset_index()

# Об'єднання середніх оцінок з тестовим набором
test_df = pd.merge(test_df, user_mean_ratings, on='userId', how='left').rename(
    columns={'rating_y': 'user_mean_rating', 'rating_x': 'rating'})
test_df = pd.merge(test_df, item_mean_ratings, on='movieId', how='left').rename(
    columns={'rating_y': 'item_mean_rating', 'rating_x': 'rating'})
```

```

# Заповнення NAs середнім глобальним рейтингом

test_df['user_mean_rating'].fillna(global_mean_rating, inplace=True)
test_df['item_mean_rating'].fillna(global_mean_rating, inplace=True)

# Обчислення середньої оцінки користувацького елементу

test_df['user_item_mean_rating'] = (test_df['user_mean_rating'] + test_df['item_mean_rating']) / 2

#####

# Обчислення середньозваженої оцінки

# Ініціалізація змінної для зберігання найкращого RMSE та відповідної ваги

best_rmse = float('inf')

best_w = 0

# Перебирання можливих ваги з кроком 0.1

for w in [i * 0.1 for i in range(11)]:

    test_df['weighted_mean_rating_tmp'] = w * test_df['user_mean_rating'] + (1 - w) *
test_df['item_mean_rating']

    rmse = calculate_rmse(test_df['rating'], test_df['weighted_mean_rating_tmp'])

    if rmse < best_rmse:

        best_rmse = rmse

        best_w = w

# Використання найкращої ваги для розрахунку остаточного середньозваженого рейтингу

test_df['weighted_mean_rating'] = best_w * test_df['user_mean_rating'] + (1 - best_w) *
test_df['item_mean_rating']

#####

# Константна модель з оцінкою 2.5 для всіх пар користувач-елемент

test_df['constant_rating'] = 2.5

# Розрахування середньої оцінки користувача та кількість оцінок на одного користувача

user_stats = train_df.groupby('userId')['rating'].agg(['mean', 'count']).reset_index()

# Коефіцієнт демпфування

damping_factor = 0.2

```

```

# Прогнозування затухаючої середньостатистичної оцінки для тестового набору

test_df = test_df.merge(user_stats, on='userId', how='left')

test_df['damped_user_mean_rating'] = ((1 - damping_factor) * test_df['mean']) + (damping_factor *
global_mean_rating)

# Розрахування RMSE для демпфованої моделі середньостатистичного користувача

rmse_damped_user_mean = calculate_rmse(test_df['rating'], test_df['damped_user_mean_rating'])

# Розрахування RMSE для моделей глобального середнього, середнього значення користувача та середнього
значення елемента

rmse_global_mean = calculate_rmse(test_df['rating'], test_df['global_mean_rating'])

rmse_user_mean = calculate_rmse(test_df['rating'], test_df['user_mean_rating'])

rmse_item_mean = calculate_rmse(test_df['rating'], test_df['item_mean_rating'])

rmse_user_item_mean = calculate_rmse(test_df['rating'], test_df['user_item_mean_rating'])

rmse_weighted_mean = calculate_rmse(test_df['rating'], test_df['weighted_mean_rating'])

rmse_constant = calculate_rmse(test_df['rating'], test_df['constant_rating'])

rmse_damped_user_mean = calculate_rmse(test_df['rating'], test_df['damped_user_mean_rating'])

# Вивід середньоквадратичне значення RMSE та середньоквадратичне значення MSE для всіх моделей

print(f"RMSE для моделі глобального середнього рейтингу: {rmse_global_mean:.2f}")

print(f"RMSE для моделі середньої оцінки користувача: {rmse_user_mean:.2f}")

print(f"RMSE для моделі середнього значення рейтингу елемента: {rmse_item_mean:.2f}")

print(f"RMSE для моделі середнього рейтингу користувач-елемент: {rmse_user_item_mean:.2f}")

print(f"RMSE для моделі середньозваженої оцінки: {best_rmse:.2f}, with setting the best weight is
{best_w:.2f}")

print(f"RMSE для моделі з константою: {rmse_constant:.2f}")

print(f"RMSE для моделі з демпфуванням середньої оцінки користувача: {rmse_damped_user_mean:.2f}")

```

Колаборативна фільтрація

```
# Створення об'єкту Reader із зазначенням шкали оцінювання
reader = Reader(rating_scale=(1, 5))

# Завантаження даних до Surprise Dataset
train_data = Dataset.load_from_df(train_df[['userId', 'movieId', 'rating']], reader)
test_data = Dataset.load_from_df(test_df[['userId', 'movieId', 'rating']], reader)

# Збирання trainset та testset
trainset = train_data.build_full_trainset()
testset = test_data.build_full_trainset().build_testset()

# Список алгоритмів
algorithms = [SVD, KNNWithZScore, SVDpp]

# Параметри для пошуку по сітці
param_grid = {
    SVD: {
        'n_factors': [125, 150, 175],
        'n_epochs': [25, 30, 35],
        'lr_all': [0.008, 0.01, 0.012],
        'reg_all': [0.15, 0.2, 0.25]
    },
    KNNWithZScore: {
        'k': [35, 40, 45],
        'min_k': [1, 3, 5, 7],
        'sim_options': {
            'name': ['msd', 'cosine', 'pearson'],
            'user_based': [True, False]
        }
    }
},
```

```

SVDpp: {
    'n_factors': [20, 50, 100, 140, 150, 160],
    'n_epochs': [20, 25],
    'lr_all': [0.001, 0.002, 0.005, 0.009, 0.01, 0.011],
    'reg_all': [0.18, 0.2, 0.22, 0.4, 0.6, 0.8]
}
}

best_models = {}
best_params = {}
best_rmse_scores = {}

# Виконання GridSearchCV для кожного алгоритму
for algo in algorithms:

    gs = GridSearchCV(algo, param_grid[algo], measures=['rmse'], cv=3)
    gs.fit(train_data)

    # Отримання найкращої моделі, найкращих параметрів та найкращої оцінки RMSE
    best_models[algo.__name__] = gs.best_estimator_['rmse']
    best_params[algo.__name__] = gs.best_params_['rmse']
    best_rmse_scores[algo.__name__] = gs.best_score_['rmse']

# Візуалізація результатів RMSE
plt.figure(figsize=(10, 6))
plt.bar(best_rmse_scores.keys(), best_rmse_scores.values(), color='blue')
plt.xlabel('Алгоритми')
plt.ylabel('Найкращий показник RMSE')
plt.title('Найкращі показники RMSE за алгоритмами')
plt.show()

# Виведення найкращої моделі, найкращих параметрів та найкращої оцінки RMSE
for algo_name, model in best_models.items():
    print(f"Найкраща модель для {algo_name}:\n", model)
    print(f"Найкращі параметри для {algo_name}:\n", best_params[algo_name])
    print(f"Найкращий показник RMSE для {algo_name}:\n", best_rmse_scores[algo_name])

```

```
# Визначення найкращого алгоритму на основі найнижчого показника RMSE

best_algo = min(best_rmse_scores, key=best_rmse_scores.get)

# Виведення найкращого алгоритму, найкращих параметрів та найкращої оцінки RMSE

print(f"Найкращий алгоритм: {best_algo}")

print(f"Найкращі параметри: {best_params[best_algo]}")

print(f"Найкращий показник RMSE: {best_rmse_scores[best_algo]}")
```

Зважені рейтинги

```
# Фільтрація фільми з відсутніми показниками vote_average або vote_count
movies_df = movies_df[(movies_df['vote_average'].notnull()) | (movies_df['vote_count'].notnull())]

# Обчислення R як середнє значення голосів для кожного фільму
R = movies_df['vote_average']

# Обчислення v як кількість голосів для кожного фільму
v = movies_df['vote_count']

# Обчислення m як 90-й перцентиль підрахунку голосів для всіх фільмів
m = movies_df['vote_count'].quantile(0.9)

# Обчислення C як середнє значення голосів для всіх фільмів
C = movies_df['vote_average'].mean()

# Розрахунок середньозваженої оцінки для кожного фільму за формулою IMDB
# Ця формула допомагає збалансувати середній рейтинг (R) із загальним середнім (C)
# Вона також враховує кількість голосів (v) та мінімальний поріг голосування (m)
movies_df['weighted_average'] = (R * v + C * m) / (v + m)

# Визначення коефіцієнт затухання часу
current_year = 2020
movies_df['time_decay_factor'] = 1 / (current_year - movies_df['year'] + 1)

# Нормалізація доходу від 0 до 1
max_revenue = movies_df['revenue'].max()
min_revenue = movies_df['revenue'].min()
movies_df['normalized_revenue'] = (movies_df['revenue'] - min_revenue) / (max_revenue - min_revenue)

# Ініціалізація MinMaxScaler
scaler = MinMaxScaler()
```

```

# Підбирання та перетворення стовпців "popularity", "weighted_average", "time_decayfactor_factor" та
"normalized_revenue"

scaled = scaler.fit_transform(

    movies_df[['popularity', 'weighted_average', 'time_decay_factor', 'normalized_revenue']]

)

# Створення DataFrame з масштабованих даних

weighted_df = pd.DataFrame(

    scaled,

    columns=['popularity', 'weighted_average', 'time_decay_factor', 'normalized_revenue']

)

# Встановити індекс нового DataFrame'а на відповідність "original_title" фільмів

weighted_df.index = movies_df['id']

# Розрахуйте «оцінку» на основі зваженої комбінації факторів

# Налаштуйте ваги за потреби (наприклад, 40% середньозважене_середнє, 40% популярність, 5%
фактор_розпаду_часу, 15% нормалізований_дохід)

weighted_df['score'] = (

    weighted_df['weighted_average'] * 0.4 +

    weighted_df['popularity'] * 0.4 +

    weighted_df['time_decay_factor'] * 0.05 +

    weighted_df['normalized_revenue'] * 0.15

)

# Відсортуйте DataFrame на основі стовпця «оцінка» за спаданням

# Це призведе до того, що фільми з найвищою змішаною оцінкою опиняться на вершині

weighted_df_sorted = weighted_df.sort_values(by='score', ascending=False)

# Показати 10 найкращих фільмів з найвищими оцінками

# Це фільми з високим рейтингом, популярні, можливо, нещодавні та з хорошими зборами

top_10_movies = weighted_df_sorted.head(10)

# Приєднання фрейма даних top_10_movies до оригінального movies_df, щоб отримати більше інформації про фільм

result_df = pd.merge(top_10_movies, movies_df[['id', 'original_title', 'year', 'revenue']], on='id',
how='left')

```

```
# Вивести результат DataFrame  
result_df[['original_title', 'year', 'revenue', 'score']]
```

Зважені рейтинги

```
def get_recommendations(title, cosine_sim):

    # Отримати індекс фільму, який відповідає назві

    idx = indices[title]

    # Отримати парні оцінки схожості всіх фільмів з цим фільмом

    sim_scores = list(enumerate(cosine_sim[idx]))

    # Відсортуйте фільми на основі оцінок схожості

    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Отримайте оцінки 10 найбільш схожих фільмів

    sim_scores = sim_scores[1:11]

    # Отримайте індекси фільмів

    movie_indices = [i[0] for i in sim_scores]

    # Повернути 10 найбільш схожих фільмів

    return movies_df['title'].iloc[movie_indices]


count = CountVectorizer(stop_words='english')

count_matrix = count.fit_transform(movies_df['bag_of_words'])

cosine_sim2 = cosine_similarity(count_matrix, count_matrix)

movies_df = movies_df.reset_index()

indices = pd.Series(movies_df.index, index=movies_df['title'])
```

Прогнозований гібридний рейтинг

```
# Створення об'єкту Reader із зазначенням шкали оцінювання
reader = Reader(rating_scale=(1, 5))

# Завантаження даних до Surprise Dataset
train_data = Dataset.load_from_df(train_df[['userId', 'movieId', 'rating']], reader)
trainset = train_data.build_full_trainset()

algo = SVD()
algo.fit(trainset)

def hybrid_predicted_rating(userId, movieId):
    # Прогнозування спільної фільтрації
    collaborative_rating = algo.predict(userId, movieId).est

    # Прогнозування фільтрації на основі вмісту
    sim_scores = list(enumerate(cosine_sim2[movieId]))
    sim_scores = sorted(cosine_sim2, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:11] # Топ-10 схожих фільмів
    movie_indices = [i[0] for i in sim_scores]

    similar_movies = movies_df.iloc[movie_indices]
    similar_movies['est'] = similar_movies['id'].apply(lambda x: algo.predict(userId, x).est)
    content_rating = similar_movies['est'].mean()

    # Отримання зваженої оцінки
    weighted_score = weighted_df.loc[movies_df.loc[movieId, 'id'], 'score']

    # Гібридне передбачення з використанням середньозваженого значення collaborative_rating, content_rating та weighted_score
    final_rating = (0.5 * collaborative_rating) + (0.2 * content_rating) + (0.3 * weighted_score)

    return final_rating
```

```
# Спрогнозування рейтингу для userId=1 та movieId=5

predicted_rating = hybrid_predicted_rating(userId=1, movieId=5)

print(f"Прогнозований гібридний рейтинг для userId=1 та movieId=5 становить: {predicted_rating}")
```

Рекомендації щодо фільмів для старих користувачів

```
# Утиліта для показу детальної інформації про рекомендовані фільми

def show_movie_details(movie_ids, movies_df):

    # Отримання інформації про фільм, таких як назва, жанр та режисер, з фрейму даних movies_df
    details_df = movies_df[movies_df['id'].isin(movie_ids)][['title', 'year', 'genres', 'director']]

    # Відображення деталей

    print("Рекомендовані фільми:")

    for index, row in details_df.iterrows():

        print(f"Title: {row['title']} ({row['year']}), Genres: {'', '.join(row['genres'])}, Director: {row['director']}")


def hybrid_recommendation(user_id, n=10):

    user_ratings = ratings_df[ratings_df['userId'] == user_id]

    predictions = []

    for index, row in user_ratings.iterrows():

        pred = algo.predict(row['userId'], row['movieId']).est

        predictions.append((row['movieId'], pred))

    top_collab_movies = [x[0] for x in sorted(predictions, key=lambda x: x[1], reverse=True)[:n]]

    last_watched_movieId = user_ratings.iloc[-1]['movieId']

    if last_watched_movieId in movies_df['id'].values:

        watched_movie_idx = movies_df[movies_df['id'] == last_watched_movieId].index[0]

        similar_movies = list(enumerate(cosine_sim2[watched_movie_idx]))

        sorted_similar_movies = sorted(similar_movies, key=lambda x: x[1], reverse=True)[1:n + 1]

        top_content_movies = [movies_df.iloc[i[0]]['id'] for i in sorted_similar_movies]

    else:

        print(f"Movie ID {last_watched_movieId} not found in movies_df.")

        top_content_movies = []

    collab_weighted_scores = fetch_weighted_scores(top_collab_movies, movies_df, weighted_df)

    content_weighted_scores = fetch_weighted_scores(top_content_movies, movies_df, weighted_df)
```

```

combined_scores = {}

for movie_id, score in collab_weighted_scores.items():
    combined_scores[movie_id] = combined_scores.get(movie_id, 0) + 0.5 * score

for movie_id, score in content_weighted_scores.items():
    combined_scores[movie_id] = combined_scores.get(movie_id, 0) + 0.5 * score

sorted_movies = sorted(combined_scores.keys(), key=lambda x: combined_scores[x], reverse=True)

return sorted_movies[:n]

# Тестування гібридної функції рекомендацій

user_id = 10

recommended_movies = hybrid_recommendation(user_id)

print(f"Рекомендовані фільми для {user_id}: {recommended_movies}")

show_movie_details(recommended_movies, movies_df)

```

Рекомендації щодо фільмів для нових користувачів (1 підхід)

```
def recommend_for_new_user_top_rating_movies(df, movies_df, n=10, min_year=5):  
  
    current_year = 2024  
  
    # Відсортування DataFrame на основі стовпця 'score' за спаданням.  
  
    sorted_df = df.copy()  
  
    sorted_df = pd.merge(sorted_df, movies_df[['id', 'year']], on='id', how='left')  
  
    sorted_df['year'] = sorted_df['year'].fillna(0).astype(int)  
  
    sorted_df = sorted_df[sorted_df['year'] >= (current_year - min_year)]  
  
    sorted_df = sorted_df.sort_values(by='score', ascending=False)  
  
    # Повернення найкращих n фільмів.  
  
    return sorted_df.head(n)  
  
  
# Отримання 10 найкращих рекомендованих фільмів для нового користувача  
  
top_movies = recommend_for_new_user_top_rating_movies(weighted_df, movies_df[['id', 'year']], n=10,  
min_year=8)  
  
show_movie_details(top_movies['id'], movies_df)
```

Рекомендації щодо фільмів для нових користувачів (2 підхід)

```
def recommend_for_new_user_by_genre(df, movies_df, genres, n=10, min_year=5):

    recommendations = {}

    movies_df = movies_df[['id', 'year', 'genres']]

    current_year = 2024

    for genre in genres:

        # Фільтрація фільмів за жанрами

        genre_movies_df = movies_df[movies_df['genres'].apply(lambda x: genre in x if x else False)]

        # Об'єднання з df DataFrame для отримання балів

        genre_df = pd.merge(df, genre_movies_df[['id', 'year']], on='id', how='left')

        # Фільтрація фільмів, випущені за останні min_років

        genre_df['year'] = genre_df['year'].fillna(0).astype(int)

        genre_df = genre_df[genre_df['year'] >= (current_year - min_year)]

        # Сортування за кількістю балів

        genre_df = genre_df.sort_values(by='score', ascending=False)

        # Отримання найкращих n фільмів для цього жанру

        recommendations[genre] = genre_df.head(n)

    return recommendations

# Перелік жанрів для розгляду

# Виділіть унікальні жанри

genres_set = set()

for genres_list in movies_df['genres']:

    genres_set.update(genres_list)
```

```
# Перетворення на список
genres_list = list(genres_set)

# Отримання 10 найкращих рекомендованих фільмів для нового користувача для кожного жанру
top_movies_by_genre = recommend_for_new_user_by_genre(weighted_df, movies_df, genres_list, n=10, min_year=10)
for genre in genres_list:
    print(genre)
    show_movie_details(top_movies_by_genre[genre]['id'], movies_df)
```