Міністерство освіти і науки України

Національний університет «Києво-Могилянська академія»

Факультет інформатики

Кафедра інформатики

Кваліфікаційна робота

освітній ступінь – бакалавр

на тему: «**РОЗРОБКА WEB-ОРІЄНТОВАНОЇ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ**»

Виконав: студент 4-го року навчання,

Освітньої програми «Комп'ютерні науки», 122

Ракітенко Дмитрій Андрійович

Керівник Горборуков В.В.,

кандидат технічних наук, доцент

Рецензент

_____

(прізвище та ініціали)

Кваліфікаційна робота захищена з оцінкою

_____

Секретар ЕК

_____

«____» _____

20____ р.

Київ – 2024

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ
Зав.кафедри
інформатики, доцент, к. ф.-м. н.
С. С. Гороховський

_____
(підпис)

“____”_____2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

студента Ракітенка Дмитрія Андрійовича факультету інформатики 4-го курсу

ТЕМА Розробка web-орієнтованої рекомендаційної системи

Зміст ТЧ до кваліфікаційної роботи:

Індивідуальне завдання
Календарний план
Зміст
Вступ
Розділ 1: Теоретичні засади та практичне використання рекомендаційних систем
Розділ 2: Розробка веб-сайту “ANI-WORLD”
Розділ 3: Розробка рекомендаційної системи для “ANI-WORLD”
Висновки
Список використаних джерел

Дата видачі “___” _____ 2024 р. Керівник _____
(підпис)

Завдання отримав _____
(підпис)

Тема: Розробка web-орієнтованої рекомендаційної системи

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	листопад	
2.	Огляд літератури за темою роботи	листопад - грудень	
3.	Створення практичної частини роботи	січень - березень	
4.	Написання текстової частини до курсової роботи	березень - квітень	
5.	Надання роботи керівнику	кінець квітня	
6.	Коригування роботи	кінець квітня – початок травня	
7.	Подання роботи на кафедру для перевірки на плагіат	тиждень до захисту	
8.	Захист кваліфікаційної роботи	кінець травня	

Студент Ракітенко Д. А.

Керівник Горборуков В. В.

ЗМІСТ

Перелік прийнятих скорочень

API – Application Programming Interface, прикладний програмний інтерфейс;

SPA – Single Page Application, застосунок на основі єдиної сторінки;

URL – Uniform Resource Locator, інтернет-адреса;

JSON – JavaScript Object Notation, текстовий формат обміну даними;

JWT – JSON Web Token, метод шифрування даних;

SVD – Singular Value Decomposition, сингулярний розклад матриці;

LDA – Latent Dirichlet Allocation, латентне розміщення Діріхле;

MCDA - Multiple-criteria decision analysis, теорія прийняття рішень на основі кількох конфліктуючих критеріїв;

АНОТАЦІЯ

Робота присвячена розробці веб-сайту з рекомендаційною системою. Реалізована система вирішує проблему “холодного старту” та недоліки методів колаборативної фільтрації. Для реалізації системи використано та вдосконалено LDA, k-NN та TF-IDF методи, що застосовуються для генерації рекомендацій, розроблено гібридний метод рекомендацій.

Ключові слова: веб-розробка, react.js, node.js, mongoDb, flask, рекомендаційна система, проблема “холодного старту”, LDA, SVD, TF-IDF, k-NN, колаборативна фільтрація, фільтрація за змістом

ВСТУП

Перші досягнення в галузі сучасних рекомендаційних систем, створення системи колаборативної фільтрації “Tapestry”, датуються 1992-м роком, а найперший приклад рекомендаційної системи “Grundy” було створено Елейн Річ у 1979-му році. З того часу було здійснено низку досліджень у цій галузі, а потреба у системах, що здатні генерувати якісні персоналізовані рекомендації, продовжує зростати. Наразі рекомендації пропонує більшість веб-платформ: онлайн-магазини, сервіси перегляду відео, читання книжок тощо. Низка популярних сервісів, таких як “Netflix”, “Imdb”, “Ebay”, “Youtube” тощо особливу увагу приділяють вдосконаленню власних рекомендаційних систем для залучення нових користувачів та утримування існуючих, слідкують за останніми дослідженнями в цій галузі.

У рекомендаційних системах сьогодення використовується багато методів, що стали можливими завдяки здобуткам останніх років в галузях машинного навчання та штучного інтелекту, проте найскладнішою для розробників лишається задача коректного використання алгоритмів, оцінки їх ефективності та коректного підбору даних для їх роботи, тож питання розробки рекомендаційних систем, зокрема гібридного типу, є надзвичайно актуальним. Дана робота намагається вирішити проблему “холодного старту” та пропонує гібридний метод для нівелювання недоліків колаборативної фільтрації.

Метою даної роботи є реалізація веб-сайту для перегляду даних про анімаційні серіали та наданням їм оцінок і гібридної персоналізованої рекомендаційної системи для нього з використанням алгоритмів машинного навчання. Для досягнення даної мети необхідно виконати наступні завдання:

1) Здійснити аналіз поняття рекомендаційної системи, вдалих прикладів вже розроблених систем в популярних сервісах, дослідити особливості реалізації сучасної рекомендаційної системи та проблеми, що виникають під час неї.

2) Здійснити аналіз популярних типів рекомендаційних систем, широко використовуваних ефективних алгоритмів генерації рекомендацій різних типів, способів оцінки ефективності роботи рекомендаційної системи.

3) Розробити веб-сайт для перегляду даних анімаційних серіалів та надання їм оцінок з використанням сучасних технологій веб-розробки та зручним інтерфейсом користувача.

4) Розробити для створеного веб-сайту гібридну рекомендаційну систему з використанням фільтрації за вмістом та колаборативної фільтрації цікавих для поточного користувача анімаційних серіалів.

5) У створеній системі подолати проблему холодного старту та запропонувати рішення проблем колаборативної фільтрації.

Об’єктом дослідження є методи сучасних веб-орієнтованих рекомендаційних систем, зокрема для генерації рекомендацій для веб-сайтів перегляду фільмів та читання книжок.

Методами дослідження є вивчення та аналіз наявних здобутків в галузі рекомендаційних систем на основі джерел, розробка моделі власного методу рекомендацій, порівняння методів рекомендаційних систем.

Робота складається з трьох розділів.

Перший розділ присвячено аналізу наукових здобутків у галузі рекомендаційних систем, розгляду історичного підґрунтя даної сфери та прикладів вдалих реалізацій рекомендаційних систем в комерційних продуктах. Також проведено дослідження основних проблем розробки рекомендаційної системи, аналіз типів систем та популярних алгоритмів генерації рекомендацій та оцінки ефективності роботи системи.

Другий розділ присвячено розробці веб-сайту для перегляду анімаційних серіалів, надання оцінок серіалам, обліку користувачів з можливістю створювати персоналізований профіль користувача.

Третій розділ присвячено розробці гібридної рекомендаційної системи для веб-сайту з використанням методів колаборативної фільтрації та фільтрації за змістом.

Реалізовано веб-сайт та гібридну рекомендаційну систему для нього. В системі розроблено гібридний метод рекомендацій на основі колаборативної фільтрації та фільтрації за змістом і вирішено проблему “холодного старту”.

ОСНОВНА ЧАСТИНА

РОЗДІЛ 1: Теоретичні засади та практичне використання рекомендаційних систем

1.1 Місце рекомендаційних систем в житті сучасного користувача

Продавці з давніх-давен намагалися рекламувати свій товар, аби максимально підвищити прибуток, проте саме розвиток сучасних технологій зробив можливим застосування і персоналізованої реклами, що базується не лише на уявленнях автора рекламного оголошення про смаки потенційного покупця, а і на тих діях, які користувач сам здійснює під час використання певної платформи. Найкраще персоналізована реклама себе зарекомендувала саме в контексті веб-орієнтованих продуктів: онлайн магазинів, веб-сайтів для перегляду відео, читання текстового матеріалу тощо, бо збір метаданих та смаків користувачів онлайн платформи є набагато легшим, ніж, наприклад, збір відгуків на певний товар, придбаний фізичною особою безпосередньо в магазині. Аби підібрати для потенційного клієнта влучні персональні рекомендації, зацікавленим у їх ефективності компаніям доводиться докладати багато зусиль, щоб розробити таку рекомендаційну систему для своїх продуктів, що збільшить їхній прибуток. Через це багатошарові гібридні рекомендаційні системи присутні у більшості популярних застосунків та платформ, зокрема веб-орієнтованих, та базуються на низці підходів з галузі інформатики, в тому числі і нетривіальних алгоритмах машинного навчання, що існують завдяки здобуткам науковців-математиків.

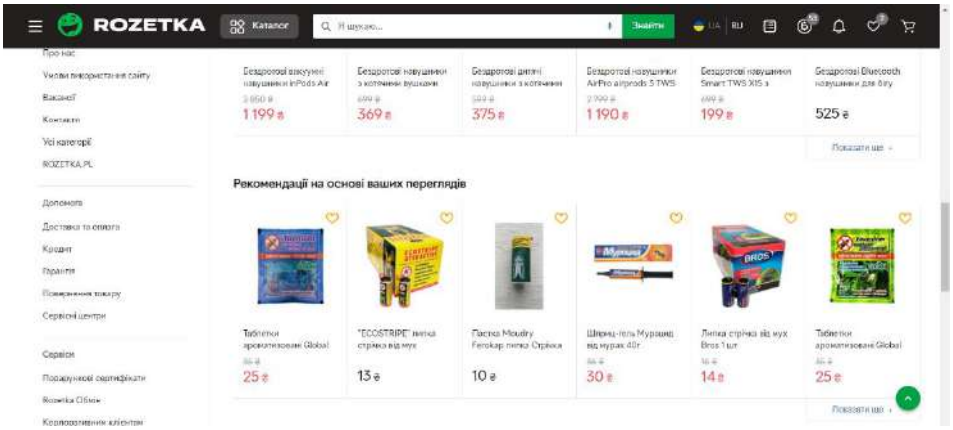
1.2 Приклади рекомендаційних систем в сучасних продуктах

Розглянемо деякі продукти, що підтвердили ефективність генерованих рекомендацій своєю популярністю та світовим визнанням. “Netflix”, відома платформа для перегляду розважальних відеоматеріалів, вважається передовим продуктом у галузі рекомендацій. За офіційними джерелами компанії, для ефективної роботи системи про користувача збираються наступні дані: взаємодія з сервісом, оцінки переглянутих матеріалів, історія переглядів тощо. Проводиться порівняння з іншими користувачами, що мають схожі смаки,

аналізуються метадані про наявні на платформі розважальні відео, такі як жанр, актори, рік створення тощо. Також платформа збирає дані про час та тривалість перегляду [1]. Усі ці дані в подальшому використовуються для роботи алгоритмів самої системи, що після їх оброблення надає персоналізовані рекомендації.

Мал. 1.1 Основна сторінка платформи Netflix [2]

Відображаються рекомендації у вигляді персоналізованих списків, де різні алгоритми відповідають за генерацію відповідної вибірки відео на основі зібраних даних. Присутні як і рекомендації, що однакові для всіх користувачів, такі як “Trending Now” (популярно зараз) та “New Releases” (нещодавно викладені відео), так і особиста вибірка, що генерується системою окремо для кожного глядача в залежності від вище описаної інформації про нього. Іншим вдалим прикладом є система популярного українського інтернет магазину Rozetka, що на основі попередньо придбаних, переглянутих товарів та поставлених їм оцінок створює персоналізовані списки товарів.



Мал. 1.2 Рекомендації Rozetka на основі переглядів

Іншими прикладами популярних платформ з просунутими рекомендаційними системами є Amazon та AliExpress, що також збирають низку даних для генерації спільних та персоналізованих списків рекомендацій для користувачів.

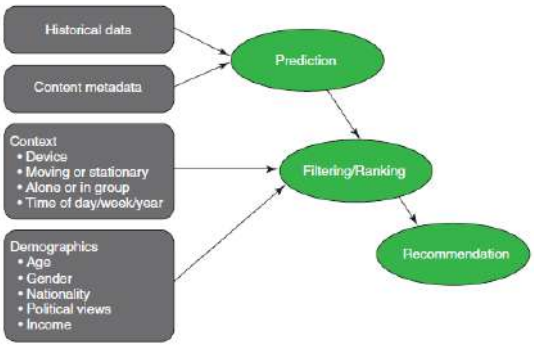
1.3 Поняття рекомендаційної системи

Хоча раніше розглянуті приклади й демонструють можливості сучасних платформ, вони не розкривають тих програмних та архітектурних рішень, що забезпечують ефективність роботи рекомендаційних систем. Фахівець в галузі, Кім Фальк, дає наступне визначення: рекомендаційна система вираховує та надає користувачеві влучний продукт відповідно до знань про нього, продукт та його взаємодію з продуктом [3]. Доцільно виділити такі основні аспекти побудови рекомендаційної системи: предметна область, мета, контекст, рівень персоналізації, інтерфейс користувача та рекомендаційні алгоритми. Предметна область вимагає визначення типу платформи, для якої треба генерувати рекомендації. Очевидно, що цей процес може відрізнятися, наприклад, для онлайн-магазину та порталу перегляду кінофільмів. Важливо визначити мету, допомогти досягти якої мають рекомендації, для користувача та самої платформи, наприклад, для такого сервісу як “Netflix” основною метою є зменшення часу, який користувач буде витрачати на пошук відео, що зацікавить його, а відповідно і максимальне звуження списку відео, які йому треба буде переглянути, та їх вибір з великого каталогу платформи [3]. Контекст позначає влучність рекомендацій відповідно до різних додаткових факторів: часу, пори року, країни

проживання користувача тощо. Рівень персоналізації позначає, наскільки рекомендації підібрані спільно для всіх користувачів, чи лише для однієї конкретної особи, відповідно можна зустріти деякою мірою персоналізовані або взагалі не персоналізовані рекомендації, проте важливо врахувати, що обидва типи рекомендацій будуються на основі даних про користувачів, продукти, та їх взаємодію. Зручний інтерфейс рекомендацій важливий, щоб максимально донести результати роботи рекомендаційної системи до кінцевого клієнта, а алгоритми рекомендацій відповідають за вирахування списку тих продуктів, що можуть зацікавити користувача, на основі зібраних даних. Саме з цим аспектом створення рекомендаційної системи доводиться стикатися розробникам програмного забезпечення.

1.4 Збір даних для рекомендаційної системи

Дані, що рекомендаційна система отримує для обробки грають велику роль у її кінцевій ефективності та впливають на оцінку її роботи. Алгоритми потребують таких даних, які зазвичай важко отримати в умовах реального світу та великого навантаження на платформи через наявність такої кількості користувачів, що перевищує каталог пропонованих продуктів. Зазвичай збирають дані про користувачів та їх взаємодію з продуктом, наприклад, для сервісу перегляду кінофільмів доцільно збирати інформацію про те, які фільми переглянув глядач, залишені ним рейтинги, з якими елементами інтерфейсу він взаємодіяв на сторінці якогось фільму. Реалізація збору даних на програмному рівні потребує додання відповідного функціоналу до застосунку та збереження усіх статистичних даних в його базі даних для подальшого передання до рекомендаційної системи.



Мал. 1.3 Дані, що зазвичай збираються для рекомендаційних систем [3]

1.5 Проблема “холодного старту”

Проблема “холодного старту” зумовлена тим, що ті дані, без яких рекомендаційна система не може працювати ефективно, недоступні для нових користувачів, що перший раз заходять на нову платформу, де про них ще не зібрано жодної інформації, яка допомогла би згенерувати рекомендації. Таким чином, одним з завдань, що поставлені перед розробниками, є отримання хоч якихось уявлень про смаки нових користувачів. Існує кілька підходів досягнення цієї мети. Очевидно, що одним з них є використання неперсоналізованих рекомендацій для таких користувачів, поки про них не буде зібрано достатню кількість інформації, наприклад, користувачем буде обрано рейтинги для певної кількості продуктів на платформі. Іншим способом є пропозиція користувачу обрати рейтинг для кількох популярних продуктів або вказати якусь додаткову інформацію про себе: інтереси, вік, стать, улюблені жанри (у випадку платформ перегляду кінофільмів, читання книжок) тощо. Наприклад, “Netflix” впровадила можливість “швидкого старту” рекомендаційної системи шляхом опитування нових користувачів про їх улюблені кінофільми та серіали, проте такий збір інформації не варто нав’язувати користувачеві, тож він не має бути обов’язковим, бо може відбити бажання деяких клієнтів користуватися платформою через те, що вона хоче знати про них більше, ніж потрібно. Рішенням даної проблеми може бути використання алгоритмів рекомендаційної системи, що розраховані саме на нових користувачів та працюють радше з одразу доступними метаданими, ніж з інформацією, що збирається в реальному часі. Такий підхід передбачає

можливість переходу до використання крім метаданих і накопиченої інформації та рейтингів, якщо користувач почне активно користуватися платформою.

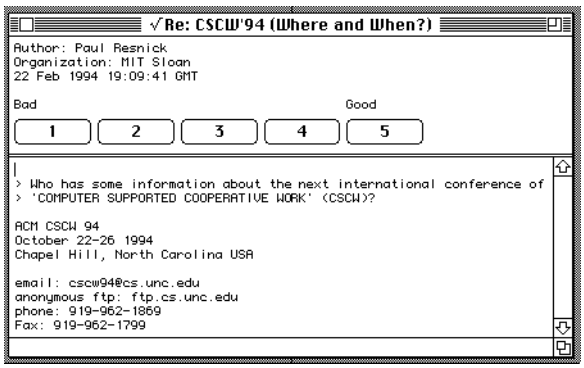
1.6 Класифікація рекомендаційних систем

Як і дані, що рекомендаційна система використовує для обчислень, її ефективність також визначають і обрані методи генерації рекомендацій. Підхід має обиратися відповідно до предметної області, масштабності платформи, порівнянню кількості активних користувачів з кількістю доступних продуктів (зазвичай у веб-орієнтованих застосунках користувачі переважають). За обраним типом фільтрації (мається на увазі фільтрації продуктів, що зацікавлять користувача) алгоритми рекомендаційних систем класифікують наступним чином: фільтрація за змістом та колаборативна (спільна) фільтрація. Колаборативна фільтрація покладається на аналіз дій користувача, його взаємодії з продуктами, відповідно для таких методів проводиться збір даних в реальному часі, та на їх ефективність негативно впливає вище описана проблема “холодного старту”. Також спільна фільтрація часто включає порівняння смаків користувача з вподобаннями подібних до нього (схожість також визначається за допомогою окремих алгоритмів). Фільтрація за змістом покладається на пошук продуктів, схожих до тих, що подобалися користувачу, або відповідають його вподобанням (опису профілю, метаданим про нього) [4]. Також можна здійснювати пошук продуктів, подібних до того, на сторінку з яким перейшов користувач, та відображати їх в якості рекомендацій на ній. Для цього алгоритмічно вираховується схожість продуктів за допомогою метаданих про них.

1.7 Історична довідка

У 1992 році науковці Белкін та Крофт проаналізували та порівняли фільтрацію інформації та видобуток інформації і визначили, що видобуток інформації є фундаментальною функцією пошукового двигуна, а рекомендаційна система переважно базується на фільтрації інформації. Tapestry, представлена того самого року, вважається однією з найперших рекомендаційних систем з використанням колаборативної фільтрації. Система покладалася на погляди

людей в маленькій групі, проте вона не могла обслуговати велику кількість користувачів, бо передбачала, що вони попарно знають одне одного [5]. Після цього було розроблено сервіс рекомендації новин “GroupLens”, що вже використовував технології поділу користувачів на групи (за допомогою алгоритмів знаходження сусідів). Система надавала користувачам можливість залишати рейтинги до статей новин.



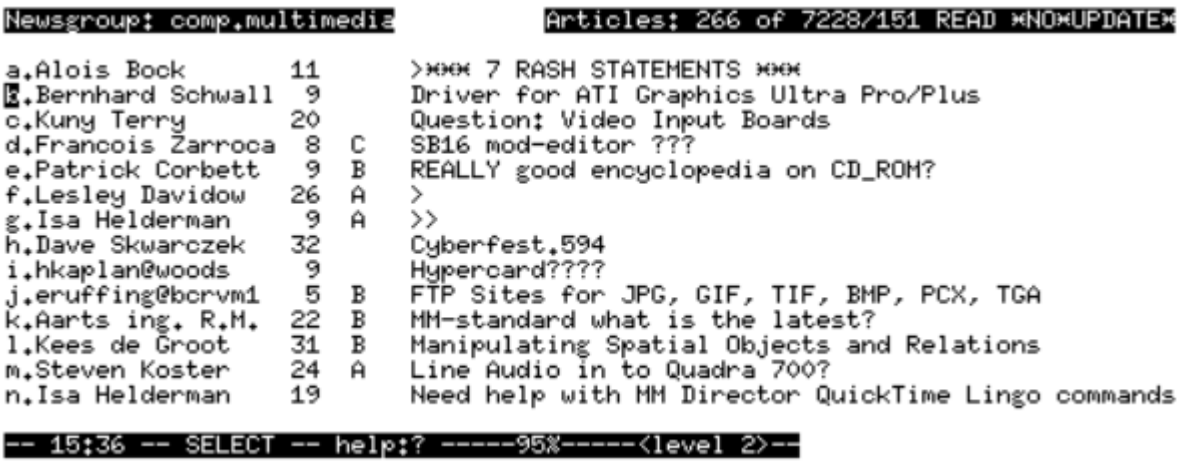
Мал. 1.4 GroupLens. Інтерфейс з наданням рейтингів [6]

Після створення рейтингів будувалася матриця рейтингів, а на її основі намагалися алгоритмічно передбачити потенційні значення неоцінених статей, проте очевидно, що такий підхід сильно ускладнював роботу алгоритму для великої кількості користувачів [6].

message #	Ken	Lee	Meg	Nan
1	1	4	2	2
2	5	2	4	4
3			3	
4	2	5		5
5	4	1		1
6	7	2	5	7

Мал. 1.5. Grouplens. Матриця рейтингів [6]

Система після вирахування надавала передбачені рейтинги різних статей для своїх користувачів [6]. Таким чином, “GroupLens” є першою платформою з використанням системи повноцінної колаборативної фільтрації та поділом на групи користувачів.



Мал. 1.6 Grouplens. Передбачені рейтинги (A, B, C, D, E) статей [6]

“Net Perceptions”, заснована в 1996 році, стала першою компанією, що пропонувала комерційний рекомендаційний двигун. Її клієнтами стали популярні сервіси, в тому числі Amazon та BestBuy. У 1997 році дослідницька лабораторія “GroupLens” створила датасет фільмів “MovieLens”, який продовжували оновлювати до 2019 року. Цей датасет став одним з найпопулярніших для досліджень у галузі рекомендаційних систем [5]. Вдосконалення рекомендаційних систем продовжується і сьогодні через їх велику потребу для різноманітних цілей та предметних областей, проте початок цим дослідженням поклали саме такі проєкти, як “Tapestry” та “Grouplens”. Сучасні рекомендаційні системи складні та гібридні, а різноманіття використовуваних у них алгоритмів надає розробникам можливість періодично оцінювати та покращувати їхні рекомендаційні системи.

1.8 Базові алгоритми рекомендаційних систем

1.8.1 Оцінка подібності

Однією з головних задач, що постають перед рекомендаційною системою, є розрахунок оцінок схожості користувачів чи продуктів з використанням даних про них. Загально проблему подібності можна визначити так: маємо два об’єкти i_1 та i_2, тоді схожість між цими об'єктами задається функцією $sim(i_1, i_2)$. Очікується, що значення функції буде зростати в залежності від того, наскільки об’єкти схожі та змінюватись від 0 до 1. Доцільно порівнювати знаходження схожості зі

знаходженням відстані між двома об’єктами. Поширеними функціями подібності є косинус подібності, коефіцієнт кореляції Пірсона та коефіцієнт Жакара [3, с. 153].

Коефіцієнт Жакара доцільно застосовувати для знаходження схожості між двома об’єктами на основі бінарних даних (0 або 1). В теорії множин коефіцієнт визначають формулою:

$$J(A,B) = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

де A та B – дві множини. Тобто, на прикладі покупки товарів покупцями, аби знайти схожість двох товарів a та b за коефіцієнтом Жакара, слід визначити, скільки користувачів купили обидва товари та поділити результат на кількість користувачів, що купили лише товар a чи лише товар b [3, с. 156]. Таким чином, буде отримано оцінку подібності двох товарів від 0 до 1, проте коефіцієнт Жакара у галузі рекомендаційних систем є сенс застосовувати лише у випадку бінарних даних: факту покупки товару, вподобання статті тощо. У випадку ширшої розмірності даних використовується косинус подібності та коефіцієнт кореляції Пірсона. Наприклад, доцільно застосовувати такі алгоритми у випадку знаходження схожості між двома кінофільмами на основі їх рейтингів.

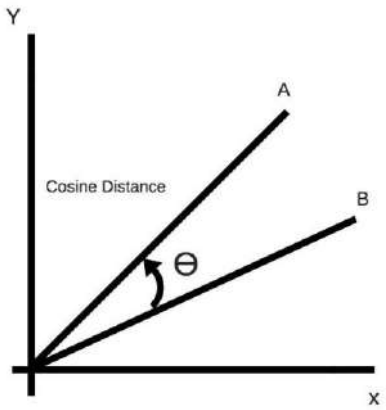
Data type	Data example	Similarity
Unary data: This can be data containing only likes or only transactions of items bought.	User 1 likes movie 2 User 2 likes movie 2 User 3 likes movie 1	Jaccard similarity
Binary data: Data where there are two possible values. Like/dislike for example.	User 1 dislikes movie 1 User 1 likes movie 2 User 2 likes movie 2 User 3 likes movie 1	Jaccard similarity
Quantitative Data	User 1 gave 4/10 stars to movie 2 User 2 gave 10/10 stars to movie 2 User 3 gave 1/10 stars to movie 1	Pearson or cosine similarity

Мал. 1.7 Приклади даних для алгоритмів подібності [3]

Косинус подібності використовується для знаходження подібності між двома векторами. Ігнорується різниця у довжині векторів, а подібність розраховується саме за напрямком векторів. Косинус подібності визначається як скалярний добуток векторів, поділений на добуток їхніх довжин. Можна визначити схожість за такою формулою:

$$sim(A, B) = \cos\theta = \frac{A * B}{\|A\| * \|B\|}$$

де A, B – вектори, A * B – скалярний добуток між ними, $\|A\|$ – довжина вектора A, а θ – кут між векторами A та B. Цей метод гарно зарекомендував себе у розробці рекомендаційних систем, бо може працювати з розрідженими даними, що часто трапляється у випадку з рекомендаціями, бо очевидно, що кожен користувач в реальному часі буде взаємодіяти лише з малою частиною усіх доступних продуктів, а косинус подібності в матрицях з осями користувач – продукт буде ігнорувати порожні значення. Найчастіше цей підхід застосовується для вимірювання схожості документів в обробці природньої мови, книжок, фільмів і відео в рекомендаційних системах та зображень в задачах комп'ютерного зору [7].



Мал. 1.8 Косинус подібності між векторами A та B [9]

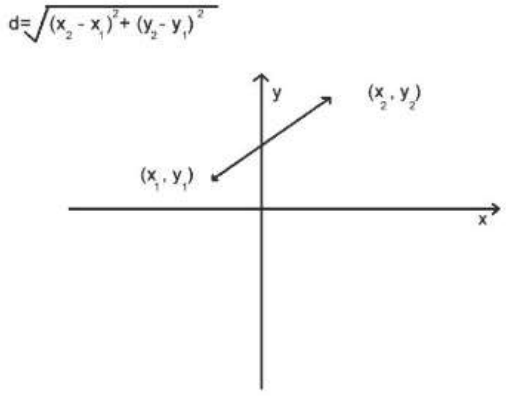
Коефіцієнт кореляції Пірсона дозволяє визначити схожість між двома змінними. Він може часто давати результати, що не сильно відрізняються від розрахунку за допомогою косинусу подібності. Кореляція змінюється від -1 до 1, де обидва граничні значення вважаються показником ідентичності двох

порівнюваних змінних [8]. Значення 1 вказує на позитивну кореляцію, тобто випадку повної подібності елементів з даних, а -1 вказує на протилежну закономірність: негативну кореляцію, тобто показує, що кожен елемент з даних протилежний відповідному у іншого користувача чи продукту [9, с. 111]. Це може бути корисно, коли система рекомендує користувачеві такий продукт, який не подобається тим користувачам, з якими в нього протилежні смаки, тобто коефіцієнт кореляції Пірсона наближений до -1. Визначити коефіцієнт Пірсона для вибірки даних (що якраз застосовується для визначення схожості в теорії рекомендаційних систем) можна за формулою:

$$r = \frac{[n(\sum xy) - \sum x \sum y]}{\sqrt{[n(\sum x^2) - (\sum x)^2][n(\sum y^2) - (\sum y)^2]}}$$

де x та y – незалежні змінні, а n – розмір вибірки [8].

Для оцінки ефективності рекомендаційних систем також застосовується Евклідова відстань. ЇЇ можна визначити як довжину відрізку, що поєднує дві точки у n – вимірному просторі.



Мал. 1.9 Евклідова відстань між двома точками у 2 – вимірному просторі [9]

Загальна формула Евклідової відстані задається як:

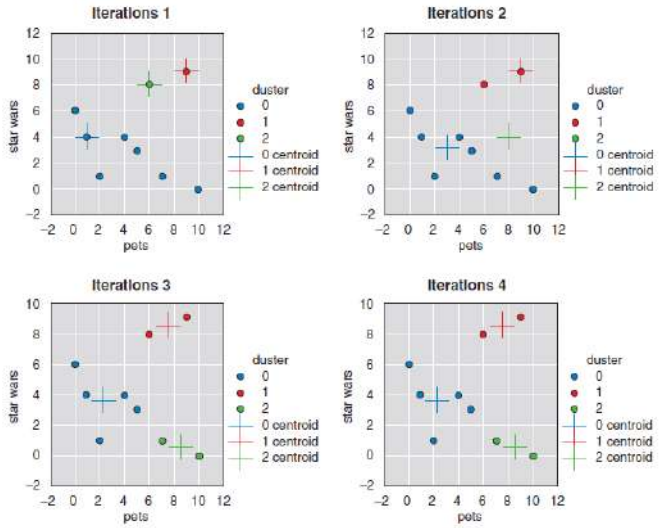
$$d(v_1, v_2) = \sqrt{\sum_{i=1}^{n} (q_i - r_i)^2}$$

Де $v1(q1, q2, \dots, qn), v2(r1, r2, \dots, rn)$ – вектори. Значення може змінюватися до нескінченності, при цьому чим воно нижче, тим більш подібними вважаються вектори [9, с. 109].

1.8.2 Кластеризація

Розподіл продуктів та користувачів на групи грає важливу роль у генерації рекомендацій, бо час виконання алгоритмів зростає при збільшенні об'єму даних. Відповідно спочатку варто звузити групу користувачів чи продуктів, які будуть використовуватись алгоритмом для передбачення потенційних рекомендацій. Кластеризація полягає в тому, аби розподілити множину довільних об'єктів на групи подібних. Цей підхід застосовується при вибірці схожих користувачів, обробці зображень, групуванні документів. Кластеризація вважається алгоритмом машинного навчання без вчителя [10].

Кластеризація методом к-середніх доволі часто застосовується при розробці рекомендаційних систем. Алгоритм намагається розподілити множину даних на k попередньо визначених груп, що не мають спільних елементів. Метод ставить на меті максимально уподібнити елементи всередині кластеру (групи), при цьому роблячи кластери різними. Алгоритм розподіляє дані між кластерами так, що досягається мінімальна сума відстаней від елементів до центроїд кластерів. Центроїдом при цьому вважається середнє арифметичне усіх значень елементів кластеру [10]. Зазвичай алгоритм реалізується наступним чином: обирається к центроїд довільним чином та циклічно виконується спроба досягнути мінімальної суми відстаней від елементів до центроїд. Для цього під час кроку циклу для кожного елементу знаходиться центроїд, до якого він має мінімальну відстань, а після визначення кластерів (співставлення центроїд та елементів) вираховується сума відстаней між центроїдами та елементами з їхніх кластерів. Цикл продовжується, поки результат попередньої його ітерації не буде менше поточної, тобто до досягнення мінімальної суми [3, с. 168].



Мал. 1.10 Приклад ітерацій алоритму к середніх (групи користувачів за рейтингами фільмів “Star Wars” та “Pets”) [3]

Варто зазначити, що алгоритм к-середніх не є ідеальним та не завжди повертає очікувані результати, наприклад, він має негативний результат для розділення точок на графіку, що зображають два півмісяці. Для такої задачі доцільніше скористатися алгоритмом спектральної кластеризації [9, с. 122]. Цей приклад демонструє те, що кластеризація може негативно вплинути на ефективність роботи рекомендаційної системи, тому, на відміну від знаходження подібності, що є невід’ємною частиною будь-якої рекомендаційної системи, кластеризація застосовується не завжди.

1.8.3 Сингулярний розклад матриці

Сингулярний розклад матриці дозволяє представити певну матрицю великої розмірності у вигляді структури з меншою розмірністю. Це досягається завдяки тому, що алгоритм визначає та ігнорує менш важливу частину матриці та генерує з неї структуру бажаної розмірності. Саме у колаборативній фільтрації цей підхід вперше застосував Саймон Фанк, що допомогло йому перемогти у змаганні “Netflix prize” [9, с. 130]. Ідея алгоритму полягає в такій факторизації матриці:

$$A = U\Sigma V^T$$

де U та V – ортогональні матриці, а Σ – діагональна матриця [11, с. 1].

1.8.4 TF-IDF

Алгоритм TF-IDF визначає наскільки певне слово важливе в документі відносно колекції документів. За допомогою текстової векторизації слова в документі перетворюються у значення їх важливості. Алгоритм можна визначити наступним чином:

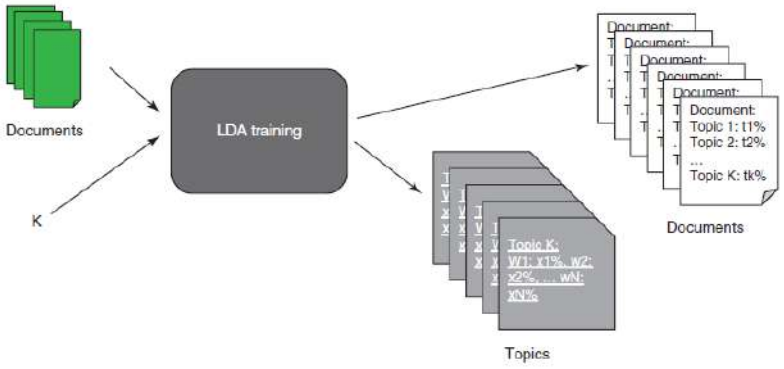
$$TF = \frac{\text{кількість входжень слова в документ}}{\text{кількість слів в документі}}$$

$$IDF = \log(\frac{\text{загальна кількість документів}}{\text{кількість документів, що містять слово}})$$

TF-IDF визначається як добуток TF та IDF [12]. У фільтрації за змістом цей алгоритм вважається класичним для визначення подібності текстових даних, проте останнім часом частіше застосовуються більш складні алгоритми, такі як латентне розміщення Діріхле. Їх ефективність вища та генеровані рекомендації відповідно краще.

1.8.5 Латентне розміщення Діріхле

Латентне розміщення Діріхле – генеративна модель, що на основі певної колекції текстових документів генерує їхні теми та тематичний розподіл. На вхід алгоритму подається набір документів та кількість тем, яку треба згенерувати, а в якості результату алгоритм обраховує набір тем та список векторів, що демонструє, як часто кожна тема зустрічається в документі [3, с. 264].



Мал. 1.11 Схема роботи алгоритму латентного розміщення Діріхле [3]

1.8.6 Оцінка ефективності роботи рекомендаційної системи

Для оцінки ефективності моделей машинного навчання можна застосовувати низку метрик.

Однією з найпопулярніших метрик є точність, що визначається як результат від ділення правильних передбачень на усі передбачення, зроблені моделлю.

RMSE (root mean square error) – інша популярна метрика в машинному навчанні, що математично задається як:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$

де y_i – справжні значення, $\hat{y}_i$ – відповідні передбачені моделлю значення, а n – розмір вибірки.

Бінарна метрика класифікації також часто застосовуються в рекомендаційних системах. Для її опису пояснимо наступні терміни: TP (true positive) – обидва справжній та передбачений класи позитивні, TN (true negative) – обидва справжній та передбачений класи негативні, FP (false positive) – справжній клас негативний, а передбачуваний – позитивний, FN (false negative) - протилежно до FP [9, с. 149]. Відповідно до цих термінів можна визначити наступні співвідношення:

$$\text{влучність} = \frac{TP}{TP + FP}$$

$$\text{повнота} = \frac{TP}{TP + FN}$$

F1 – міра є середнім гармонійним між влучністю та повнотою:

$$F_1 = 2 * \frac{\text{влучність} * \text{повнота}}{\text{влучність} + \text{повнота}}$$

1.9 Колаборативна фільтрація

Колаборативна фільтрація покладається на дані, зібрані в реальному часі під час користування певною платформою, а не на метадані: дії користувачів, рейтинги, покупки тощо. Правильно розроблена система колаборативної фільтрації з великою вірогідністю здатна згенерувати таку рекомендацію, що зацікавить користувача та при цьому виявиться неочікуваною для нього. Колаборативна фільтрація може здійснюватися на основі користувачів або на основі продуктів. Можливе знаходження користувачів зі схожими смаками та обрання в якості рекомендацій поточного користувача продуктів, які подобаються групі з подібними смаками, до якої він належить, проте які він ще не дивився чи не купував в залежності від предметної області платформи. Другий підхід передбачає знаходження для поточного користувача таких продуктів, що подібні до того, що йому вже подобалося раніше [3, с. 184]. Методи реалізації конкретної рекомендаційної системи з колаборативною фільтрацією сильно залежать від предметної області та особливих вимог до системи, проте зазвичай фільтрація обирається на основі кількості користувачів чи продуктів: якщо користувачів більше, то доцільніше обрати саме фільтрацію на основі продуктів, бо це зменшить навантаження під час обчислень рекомендацій, складність яких зростає зі збільшенням кількості користувачів. Для колаборативної фільтрації між об’єктами (користувачами чи предметами) слід визначати подібність, для чого можна використовувати вище розглянуті методи: коефіцієнт Жакара, косинус подібності, коефіцієнт кореляції Пірсона тощо. Ці методи є найпопулярнішими для знаходження схожості. Також слід виконувати поділ на групи, для чого можна використати відповідні алгоритми кластеризації, такі як алгоритм к-середніх. Кластеризація не є універсальним виходом, тому застосовуються і інші методи. Одним з найлегших методів є визначення числа N як кількості сусідів, що мають входити в групу. Такий підхід гарантує, що кожна група буде мати достатню кількість елементів для обробки, проте ці елементи можуть виявитися не подібними між собою. З іншого боку, можна розподіляти елементи на групи відповідно до пов’язаних з подібністю умов, наприклад,

вимагати, щоб схожість усіх елементів певної групи перевищувала певне значення. Такий підхід потенційно надасть більш гарні рекомендації, проте може призвести до того, що деякі продукти в групи взагалі не потраплять, бо нема достатньо схожих на них продуктів [3, с. 194]. Після знаходження схожості та поділу на групи система має передбачити потенційний рейтинг продуктів для користувача та відповідно запропонувати йому продукти з найвищим для нього рейтингом. Одним з найпопулярніших способів передбачення оцінок є використання вище описаного алгоритму сингулярного розкладу матриці, а саме – його модифікація для роботи з розрідженими матрицями, запропонована Саймоном Фанком.

1.10 Фільтрація за змістом

Для фільтрації за змістом використовуються метадані про продукти та користувачів. Зазвичай більше уваги надається саме метаданим про продукт. Наприклад, у випадку фільму дуже корисними є його опис, перелік ключових слів, жанрів, рік створення, режисер тощо. Ці дані можуть використовуватися, аби згенерувати для користувача рекомендації швидше, ніж він зробить таку кількість дій на платформі, що необхідна для роботи алгоритмів колаборативної фільтрації та побудови матриці користувач-продукт, тобто доцільно використовувати обидва типи фільтрації в одній системі, щоб забезпечити високу якість рекомендацій за допомогою колаборативної фільтрації та частково подолати проблему “холодного старту” за допомогою фільтрації за змістом. Цей тип фільтрації часто будується на алгоритмах природньої мови, зокрема на вище описаних методах: TF-IDF застосовується для векторизації текстових даних (описів, метаданих) з подальшим визначенням схожості продуктів за отриманими векторами (застосовуються алгоритми подібності, такі як косинус подібності тощо) та латентне розміщення Діріхле, що також може застосовуватися для виявлення належності текстових документів до певних визначених тематик. У сучасних рекомендаційних системах цьому методу надається перевага, бо використання TF-IDF може давати сухі результати через простоту самого алгоритму. Як і у випадку з колаборативними фільтрами, вибір методів проводиться в залежності

від предметної області, потреб до системи тощо, а описано лише найпопулярніші та найдоступніші підходи реалізації генерації рекомендацій.

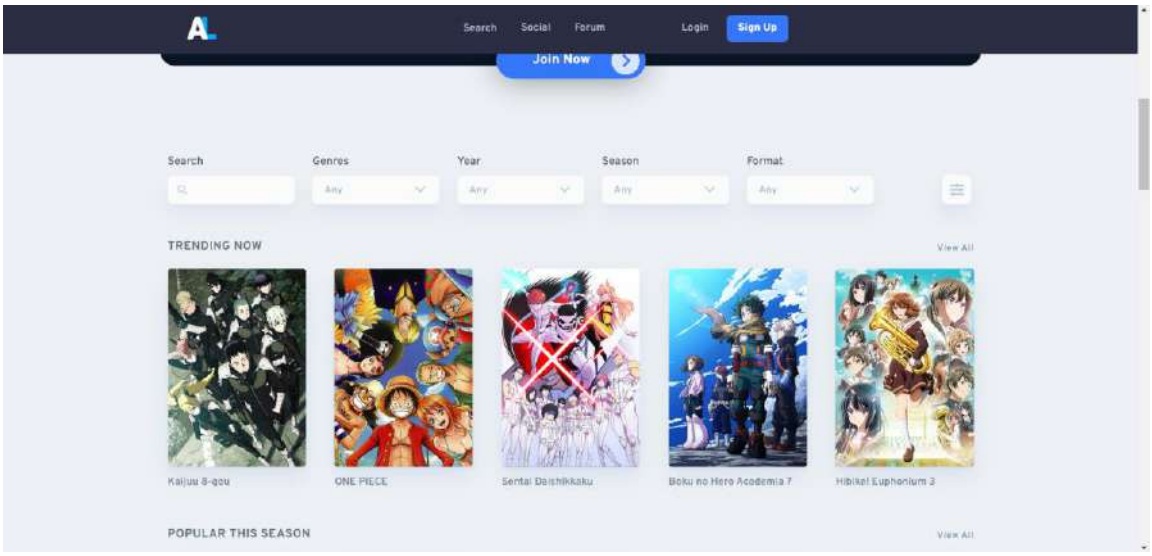
Висновки до розділу 1

Даний розділ містить загальне знайомство з поняттям рекомендаційної системи, приклади вдалих реалізацій таких систем, зокрема ознайомлення з системою платформи “Netflix”, та деякі історичні відомості про перші рекомендаційні системи, такі як “Tapestry” та “GroupLens”. Окрім цього, в цьому розділі було проаналізовано основні задачі та проблеми, що постають при розробці рекомендаційної системи: збір даних та проблему “холодного старту”. Було пояснено різницю між типами рекомендаційних систем (системами колаборативної фільтрації та фільтрації за змістом) та описано основні алгоритми, що застосовуються при розробці рекомендаційної системи, а саме: алгоритми для оцінки подібності, кластеризації (зокрема методом к-середніх), сингулярний розклад матриці, TF-IDF, латентне розміщення Діріхле. Також розглянуто основні метрики для оцінки ефективності роботи алгоритмів машинного навчання, що застосовуються в рекомендаційних системах: точність, RMSE та бінарну метрику.

РОЗДІЛ 2: Розробка веб-сайту “ANI-WORLD”

2.1 Загальний опис поставленої задачі

Для демонстрації ілюстративного прикладу практичного використання рекомендаційних систем було обрано розробити веб-сайт з даними про різноманітні анімаційні серіали. Візуальне оформлення та вміст веб-сайту надихався популярними рішеннями з аналогічної предметної області. Веб-сайт не надає можливості переглядати наявні анімаційні серіали, бо вони захищені відповідними ліцензіями, проте містить інформацію про них та дозволяє користувачеві веб-сайту залишати оцінки до серіалів, що симулює взаємодію користувача з аналогічними реальними платформами. Вдалими прикладами таких веб-сайтів є anilist.co та myanimelist.net.



Мал. 2.1 Головна сторінка anilist.co

Також подібний функціонал мають платформи з даними про фільми, такі як imdb.com.

2.2 План розробки веб-сайту

Розробку даного веб-застосунку було умовно поділено на наступні кроки:

1) Проектування та розробка клієнтської частини застосунку.
2) Використання взаємодії з арі для отримання даних про анімаційні серіали.
3) Проектування та створення бази даних користувачів.

4) Створення серверної частини веб-сайту.
5) Поєднання клієнтської та серверної частини та підготовка до імплементації рекомендаційної системи.

Розробку рекомендаційної системи буде розглянуто окремо.

2.3 Функціонал веб-сайту

Розроблений вебсайт має наступний функціонал:

1) Перегляд даних про анімаційні серіали, отримані з api (доступні такі дані, як назва, зображення, рік створення, кількість епізодів, студія-творець, джерело сюжету, короткий опис, персонажі з їхніми зображеннями, посилання на відео трейлера)
2) Пошук потрібного анімаційного серіалу.
3) Можливість залишати оцінки анімаційним серіалам.
4) Профіль користувача з можливістю додати особистий опис, улюблені жанри та переглянути список оцінених серіалів.

В подальшому для веб-сайту буде розроблено рекомендаційну систему.

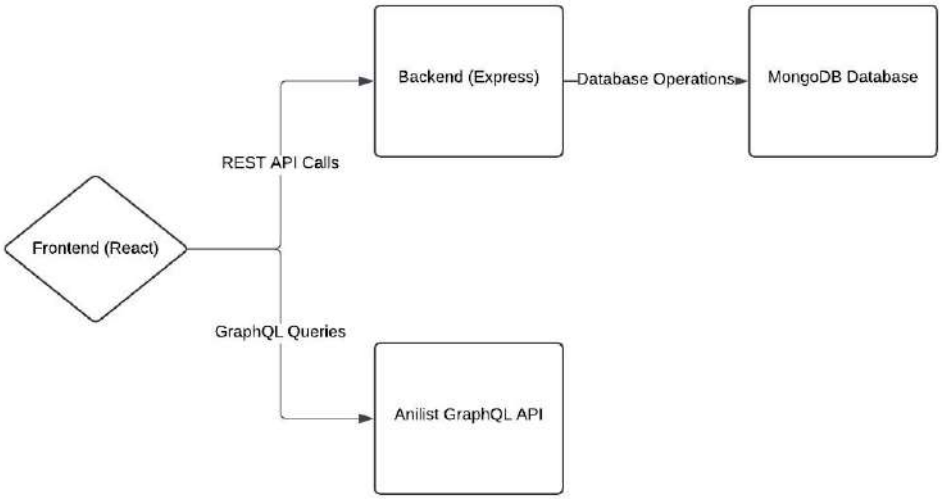
2.4 Використані інструменти

Для створення вище описаної веб-платформи було використано наступні технології: React.js (react.dev) та Material Ui (mui.com) для розробки швидкої та інтуїтивної клієнтської частини, node.js (nodejs.org) та express (expressjs.com) для розробки серверної частини застосунку, нереляційну базу даних MongoDb (mongodb.com) і api Anilist v2 (anilist.gitbook.io/anilist-apiv2-docs) для отримання даних про анімаційні серіали за допомогою GraphQl запитів (graphql.org)

2.5 Архітектура веб-застосунку

Веб-застосунок має зрозумілу архітектуру. Клієнтська частина за допомогою api викликів взаємодіє з серверною частиною застосунку, що у свою чергу взаємодіє з базою даних. Також клієнтська частина взаємодіє з api anilist за допомогою api викликів мови запитів GraphQl для отримання даних про

анімаційні серіали. У подальшій розробці буде додано також окремий сервер рекомендаційної системи.

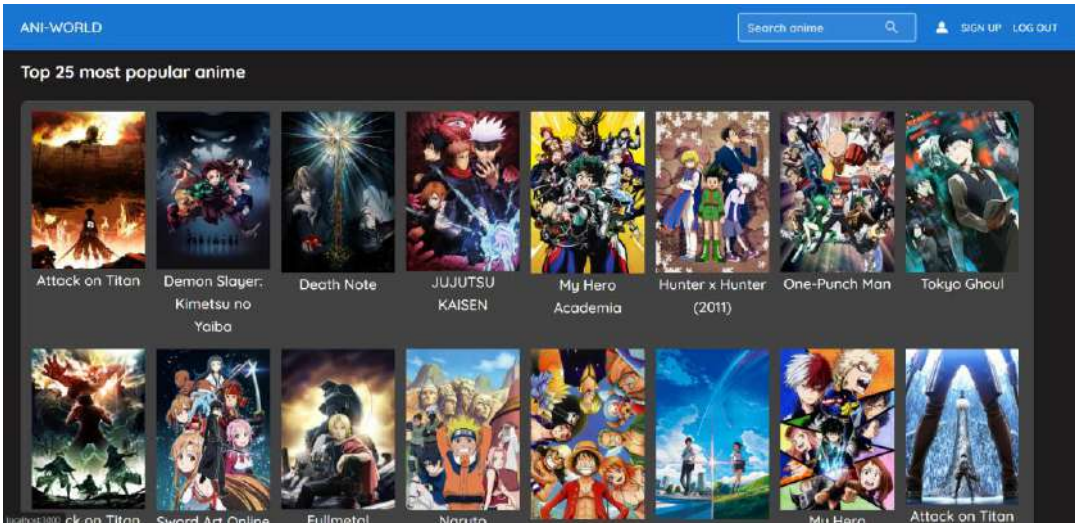


Мал. 2.2 Архітектура веб-сайту

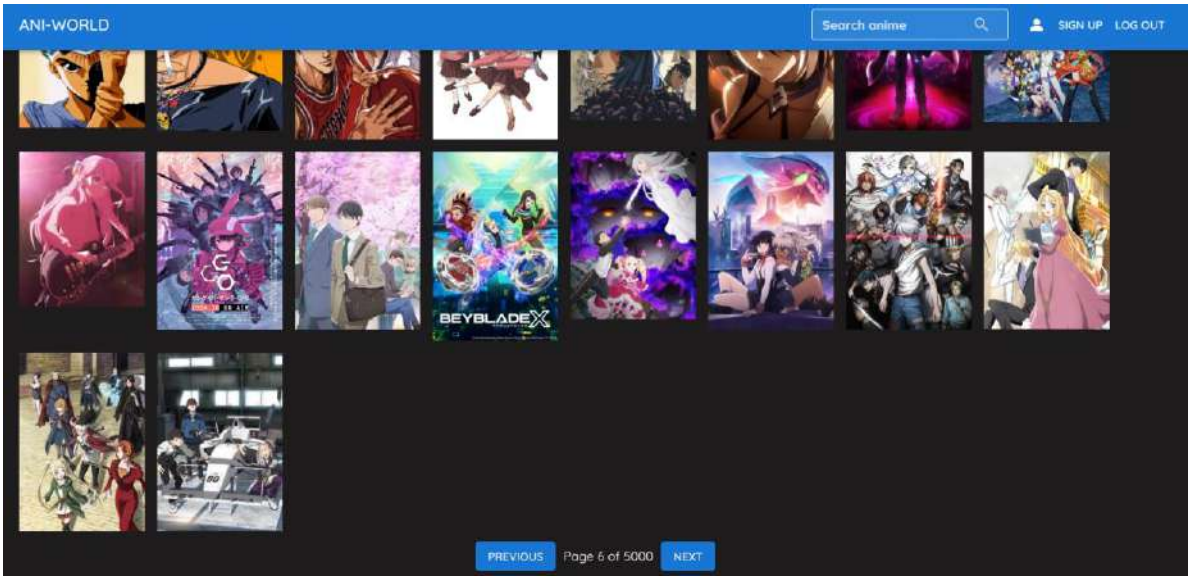
2.6 Структура клієнтської частини веб-сайту

Для веб-сайту створено роутинг по сторінках та розроблено наступні сторінки:

1) Домашня сторінка з анімаційними серіалами та можливістю переходу по сторінках, доступні усі анімаційні серіали з api anilist. На цій сторінці також відображається 25 найпопулярніших серіалів (з anilist). Такий список на справжній платформі може використовуватися для того, аби подолати проблему “холодного старту”, бо він надає неперсоналізовані рекомендації користувачам, для яких ще не зібрано достатньо даних для використання алгоритмів колаборативної фільтрації.



Мал. 2.3 Домашня сторінка. 25 найкращих серіалів

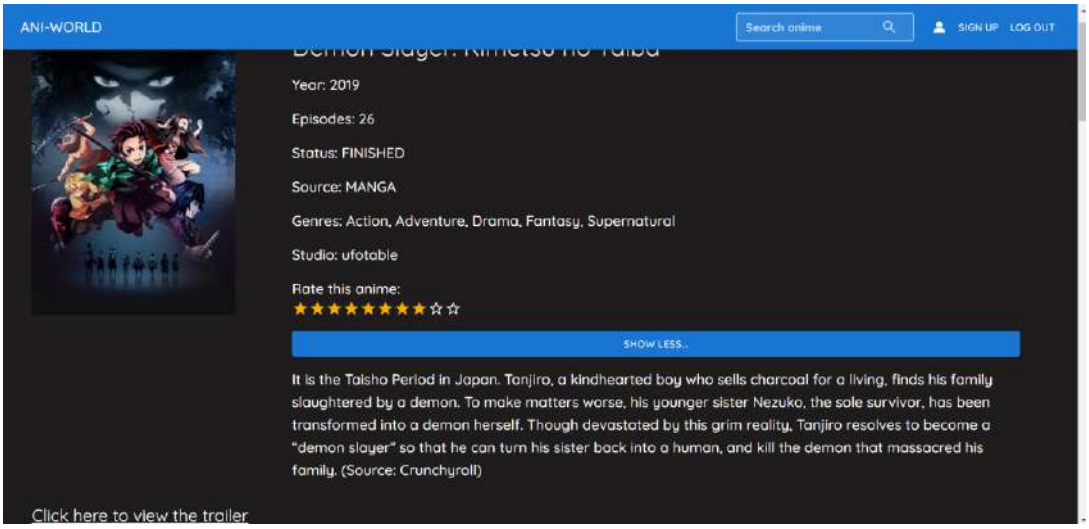


Мал. 2.4 Домашня сторінка. Перемикання сторінок

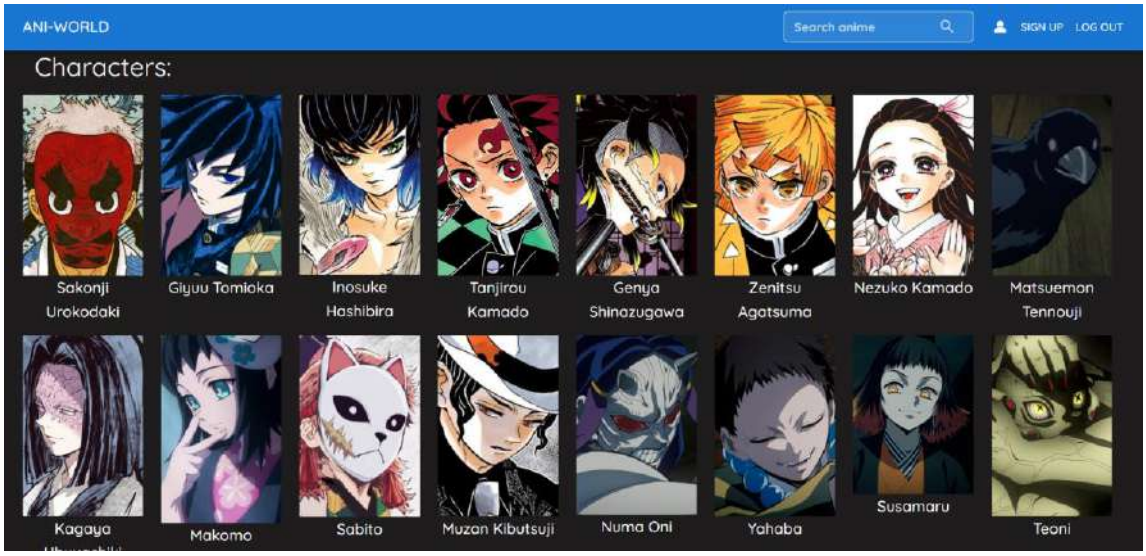
На сторінках присутні зображення та назви серіалів. Для зручного повторного використання таких списків створено окрему компоненту react для відображення переліків анімаційних серіалів.

2) Індивідуальна сторінка певного анімаційного серіалу. Ця сторінка містить відомості про обраний серіал, його опис, посилання на трейлер, список персонажів. Для списку персонажів створено окрему компоненту, як і у випадку з анімаційними серіалами. Сторінка також містить і поле, де користувач може залишити рейтинг для якогось серіалу. Рейтинги та взаємодія

користувача зі сторінками зберігаються в базі даних користувачів, що розміщена у хмарі MongoDb.

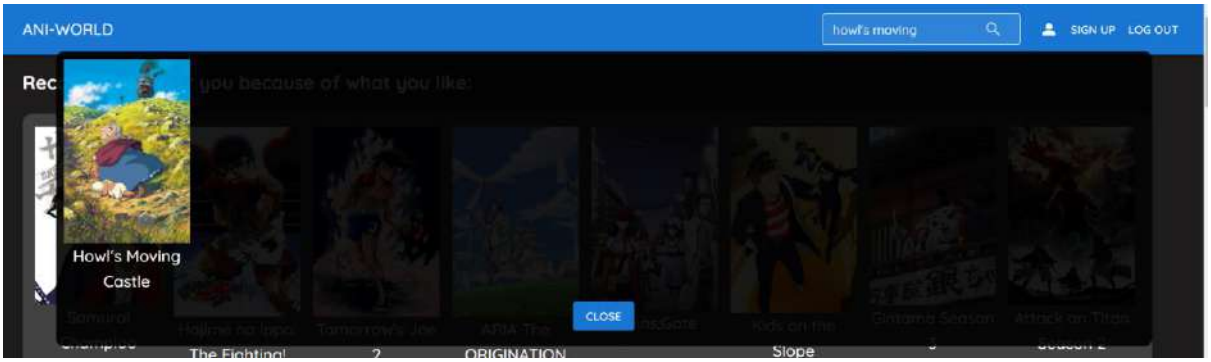


Мал. 2.5 Індивідуальна сторінка. Дані та опис, рейтинг



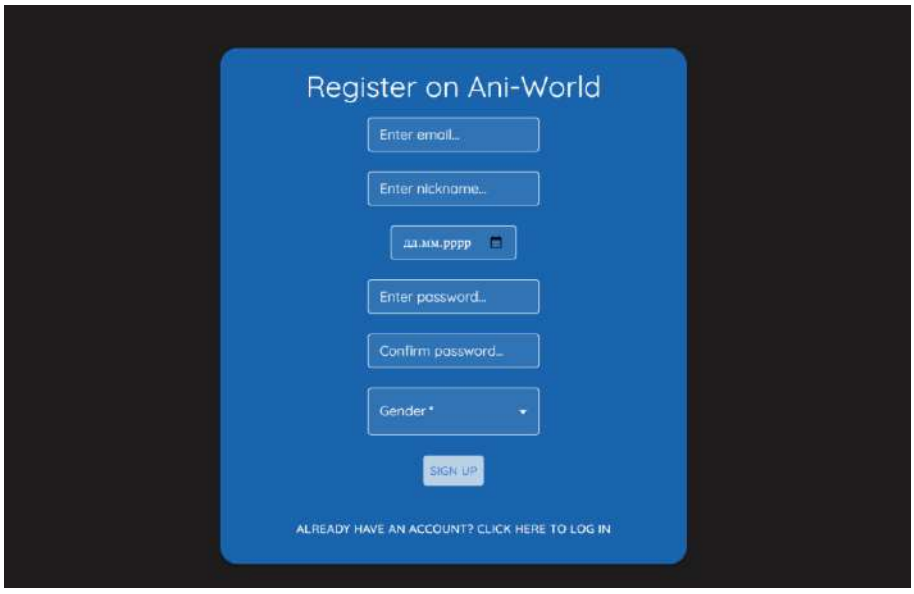
Мал. 2.6 Індивідуальна сторінка. Список персонажів

3) Навігаційна панель. Ця панель у верхній частині екрану дає можливість перейти на домашню сторінку, перейти на сторінку реєстрації, сторінку авторизації та виконати пошук потрібного анімаційного серіалу. Для пошуку по каталогу доступних анімаційних серіалів використовується запит до api anilist, та результати відображаються у списку-компоненті.



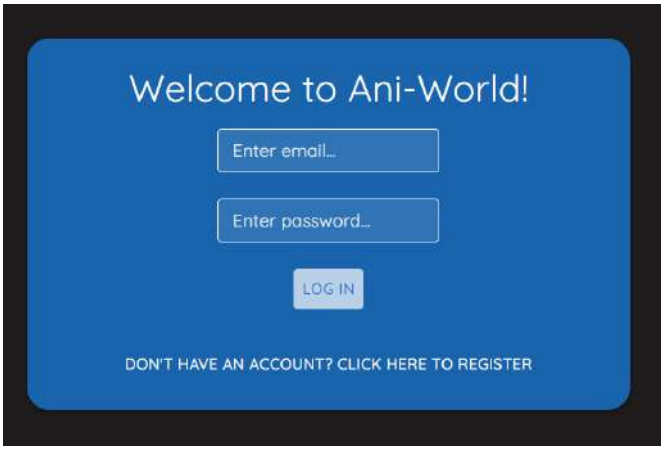
Мал. 2.7 Навігаційна панель. Пошуковий запит

4) Сторінка реєстрації. Ця сторінка призначена для реєстрації нового користувача. Вона містить поля для заповнення ім’я, паролю, електронної адреси, статі та дати народження користувача. Після успішного заповнення полів виконується запит до бази даних на додання нового користувача. Також наявна можливість швидкого переходу на сторінку авторизації. Присутня валідація полів, вони обов’язкові для заповнення користувачем.



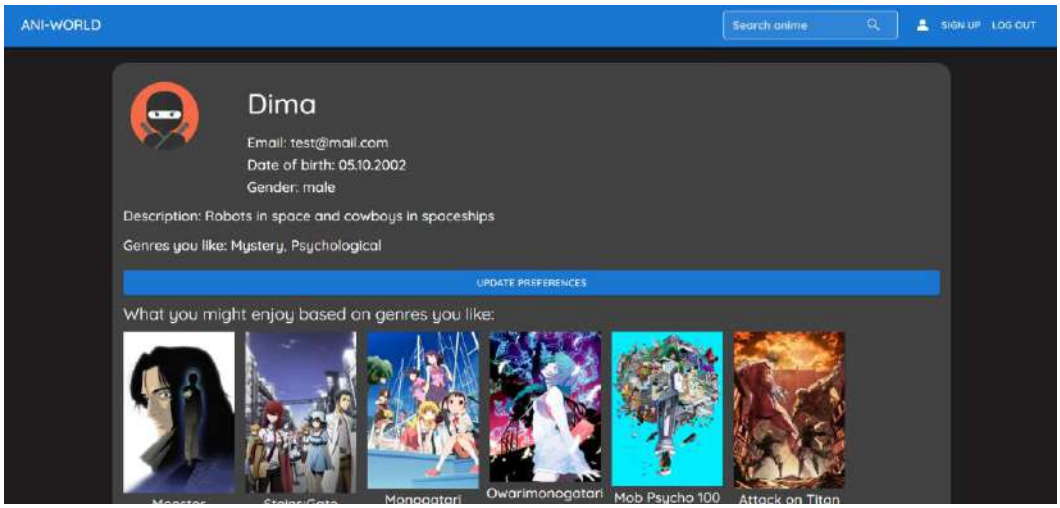
Мал. 2.8 Сторінка реєстрації

5) Сторінка авторизації. Ця сторінка надає можливість зареєстрованим користувачам авторизуватись на веб-сайті за електронною адресою та паролем.



Мал. 2.9 Сторінка авторизації користувача

6) Сторінка профілю. Ця сторінка відображає дані авторизованого користувача (ім'я, дату народження, електронну адресу та стать), дає можливість додати особистий опис та улюблені жанри, що будуть використовуватися для генерації персоналізованих рекомендацій. Також на цій сторінці відображаються оцінені анімаційні серіали.



Мал. 2.10 Сторінка профілю користувача

2.7 Опис особливостей розробки клієнтської частини застосунку

Для розробки інтерфейсу користувача використовуються елементи бібліотеки Material Ui. Створено кілька компонентів для повторного використання, наприклад, поля на сторінках реєстрації та авторизації, списки персонажів та анімаційних серіалів. Візуальний стиль елементів відредаговано відповідно до дизайну веб-сайту, який надихався оформленням вже доступних популярних

рішень. Перехід між сторінками зі збереженням SPA концепції react реалізовано за допомогою бібліотеки react-router-dom. Доступні шляхи /signup, /profile, /login та /item для зручності заміни та розширення винесено в змінні. Для збереження змінних використовується useState, а для виконання запитів, в тому числі на api anilist, використовується fetch функція з javascript та useEffect. Деякі запити для зручності та повторного використання винесено в окремий файл, фукнції з якого використовуються у декількох компонентах.

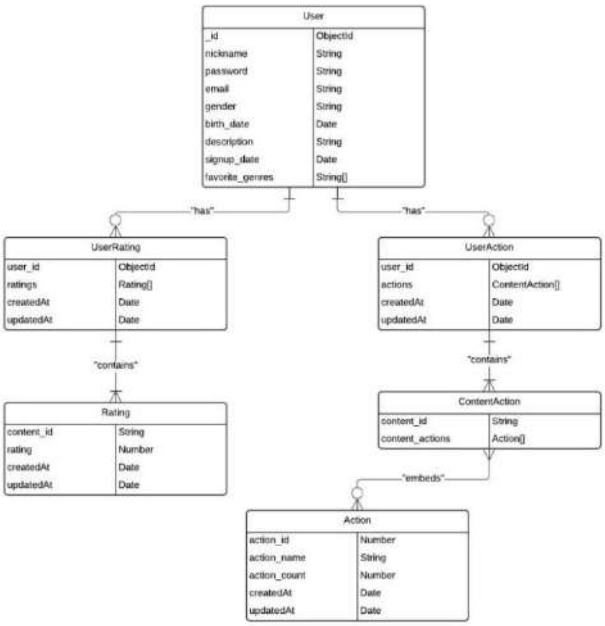
2.8 Особливості взаємодії з anilist api

Anilist api дозволяє отримувати дані про анімаційні серіали посторінково. Доступні такі дані як жанри, опис і теги, що корисні при побудові рекомендаційної системи, зокрема для фільтрації за вмістом. Запити до api потребують використання GraphQl. На веб-сайті дане api використано для отримання даних про анімаційні серіали на індивідуальних сторінках та домашній сторінці, а також на сторінці профілю. Для зручності url даного api та усі використовувані запити винесено в окремий файл. Anilist надає можливість сортування за різними критеріями, фільтрації і пошуку, полегшує наповнення веб-сайту даними в некомерційному середовищі, надає метадані про тисячі анімаційних серіалів. Реалізовано запити посторінкового отримання серіалів, отримання найпопулярніших серіалів, отримання серіалу або групи серіалів за ідентифікаційними кодами.

Мал. 2.11 GraphQl api запит за ідентифікаційними кодами

2.9 База даних користувачів

Для збереження даних користувачів, а саме їхньої реєстраційної інформації, опису, вподобань, рейтингів та дій використано нереляційну базу даних MongoDb.



Мал. 2.12 Схема бази даних користувачів

Взаємодію серверної частини та бази даних реалізовано за допомогою бібліотеки mongoose, що полегшує читання та запис в базу даних. Ця бібліотека надає можливість створювати окремі моделі для різних сутностей в базі даних за певними схемами, що дозволяє вказати імена та типи даних для документів в колекціях MongoDb. Це дозволяє зробити процес взаємодії з базою даних зручним та інтуїтивним, бо усі документи в колекціях мають однакову структуру. Валідація структури виконується на рівні mongoose, що слугує інтерфейсом між серверною частиною та базою даних. Створено окремий javascript клас-сервіс, що надає інтерфейс для підключення до бази даних та зручні функції додавання, видалення та редагування колекцій за mongoose моделями. Дані рейтингів користувачів сформовані таким чином, що їх доречно використати для колаборативної фільтрації. Для фільтрації за змістом зручно використати опис та улюблені жанри користувача. Про користувача зберігається наступна інформація

про його дії: відкрити сторінку з анімаційним серіалом, переглянути опис, перейти за посиланням на трейлер, залишити рейтинг. Збір такої інформації може бути корисним при реалізації колаборативної фільтрації та формуванню статистики по користувачах. Пароль користувача в базі даних зашифровано за допомогою bcrypt. За рахунок використання api для наповнення веб-сайту збереження анімаційних серіалів в базі даних не виконується, а рейтинг та дії прив'язані до ідентифікаційного коду серіалу, що використовується в самому api.

2.10 Серверна частина веб-сайту

Серверна частина веб-сайту взаємодіє з клієнтською, проте не взаємодіє з anilist api, бо в цьому нема потреби. Сервер розроблено за допомогою node.js та express, наявні роутер та контролер для користувача та для вмісту (анімаційних серіалів), що надають кінцеві точки api для використання у клієнтській частині застосунку. Окремі функції серверу будуть розглядатися у розділі, присвяченому розробці рекомендаційної системи. До серверу можна виконувати POST та GET запити за кінцевими точками, а передача та повернення інформації відбувається за допомогою зручного для використання у javascript формату json. Роутер та контролер користувачів виконують наступні функції: реєстрація, авторизація користувача, отримання та оновлення даних користувача.

```
userRouter.post('/login', userController.login);
userRouter.post('/sign_up', userController.signUp);
userRouter.get('/info', userController.getInfo);
userRouter.post('/update', userController.updateUser);
userRouter.get('/preferences', userController.getPrefs);
userRouter.get('/personal_recs', userController.getPersonalRecs);
```

Мал. 2.13 Роутер користувачів

Роутер та контролер змісту виконують наступні функції: додання та оновлення рейтингу анімаційного серіалу, отримання усіх рейтингів користувача, отримання рейтингу користувача за ідентифікаційним кодом окремого анімаційного серіалу, отримання усіх дій користувача, додання та оновлення дій за ідентифікаційним кодом певної дії та серіалу.

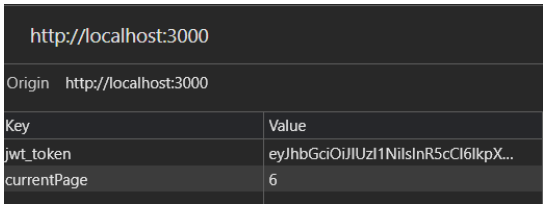
```
contentRouter.post('/rate', contentController.setRating);
contentRouter.get('/ratings', contentController.getRatings);
contentRouter.get('/rating/:content_id', contentController.getRatingByContent);
contentRouter.post('/act', contentController.updateActions);
contentRouter.get('/actions', contentController.getActionsByUser);
contentRouter.post('/similar',contentController.getSimilar);
```

Мал. 2.14 Роутер змісту

Для зручності використання у клієнтській частині застосунку посилання на кінцеві точки api серверу винесено в окремий javascript файл.

2.11 Реєстрація та авторизація користувачів

Визначення користувача у поточній сесії виконується за допомогою jwt токена. При авторизації користувача до jwt токена додається ім'я, електронна адреса та ідентифікаційний код користувача, який буде використовуватись для його визначення в поточній сесії під час запиту до api серверу з переданням токену в заголовках. Токен зберігається в локальному сховищі при авторизації. При виході користувача з платформи за допомогою кнопки "log out" значення токена звідти видаляється.



Мал. 2.13 Збережені в локальному сховищі токен користувача та поточна сторінка домашнього екрану

2.12 Потенціал для додання рекомендаційної системи

Розроблений веб-сайт надає користувачеві можливість переглядати інформацію про анімаційні серіали, проте не може порекомендувати йому серіали на основі до його попередніх дій, рейтингів, подібності серіалів тощо. Для даного веб-сайту доречна розробка рекомендаційної системи, що допоможе користувачеві знайти більше цікавих для себе анімаційних серіалів. Серед функціоналу веб-сайту наявна система рейтингів, можливість додати особисті вподобання, такі як жанри і опис, а також метадані про анімаційні серіали. Ці

дані та функції можна використати для реалізації відповідних алгоритмів колаборативної фільтрації та фільтрації за змістом.

Висновки до розділу 2

Даний розділ присвячений розробці веб-сайту “ANI-WORLD”, що дозволяє переглядати різноманітну інформацію про анімаційні серіали. Описано особливості поставленої задачі та представлено приклади схожих веб-застосунків, детально розписано план розробки веб-сайту. Згадано усі інструменти, що використовувалися при розробці. Розглянуто реалізований функціонал та особливості архітектури, надано її ілюстративну схему. Окрему увагу приділено структурі клієнтської частини веб-сайту та його інтерфейсу, показано візуальне оформлення доступних сторінок. Крім цього, проаналізовано особливості взаємодії з anilist api. Звернено увагу на структуру та процес реалізації бази даних користувачів. Також зображено її модель. Описано реалізацію серверної частини веб-сайту, її структури, наявні контролери та роутери, окремо розглянуто реалізацію реєстрації та авторизації користувачів на веб-сайті за допомогою jwt токенів. Обґрунтовано доцільність розробки рекомендаційної системи для створеного веб-сайту та доведено корисність особливостей його розробки для алгоритмів самої системи.

РОЗДІЛ 3: Розробка рекомендаційної системи для “ANI-WORLD”

3.1 Загальний опис поставленої задачі

Для створеного веб-сайту “ANI-WORLD” доречно розробити рекомендаційну систему для користувачів, що буде надавати персоналізовані та неперсоналізовані рекомендації. Слід поєднати методи колаборативної фільтрації та фільтрації за вмістом для генерації рекомендацій, що надають подібні до смаків користувача серіали та при цьому не рекомендують надто непопулярних та низько оцінених серіалів. При розробці мають бути використані дані про рейтинги користувачів, їхні описи та улюблені жанри, а також метадані про анімаційні серіали. Рекомендаційна система має бути реалізована за допомогою окремого веб-серверу та має бути настільки незалежною від основного застосунку, наскільки це можливо.

3.2 План розробки рекомендаційної системи

Процес розробки було умовно поділено на наступні етапи:

1) Проектування системи та ознайомлення з використовуваними технологіями.
2) Завантаження та підготовка датасету anilist, його попередня обробка та фільтрація.
3) Розробка функцій для застосування алгоритмів колаборативної фільтрації та фільтрації за змістом.
4) Інтеграція розроблених функцій у веб-сервер і створення відповідних кінцевих точок арі рекомендаційної системи.
5) Налагодження взаємодії веб-сервера рекомендацій з основним сервером anilist арі, розширення контролерів та роутерів користувача та змісту функціоналом генерації рекомендацій.
6) Розширення клієнтської частини, додання арі запитів, налаштування відображення результатів в інтерфейсі користувача.
7) Подальше дослідження та оцінка ефективності роботи системи та покращення результатів роботи.

3.3 Використані технології

Для розробки системи було обрано мову python та веб-сервер Flask. Також для реалізації колаборативної фільтрації та фільтрації за вмістом використані python бібліотеки scikit-learn, ast, pandas, suprise та numpy. Застосовується датасет з даними про анімаційні серіали з вебсайту anilist.co (kaggle.com/datasets/barrettotte/anilistanimedata). Обрання мови python зумовлено її зручністю та доцільністю для вирішення задач машинного навчання, обробки природньої мови та фільтрації датасетів. Хоча ця мова не вирізняється швидкодією, наявні в ній інструменти дозволяють швидко візуалізувати та оцінювати отримані результати роботи рекомендаційної системи, що полегшує процес її покращення та оптимізації, правильної підготовки даних до обробки.

3.4 Функціонал рекомендаційної системи

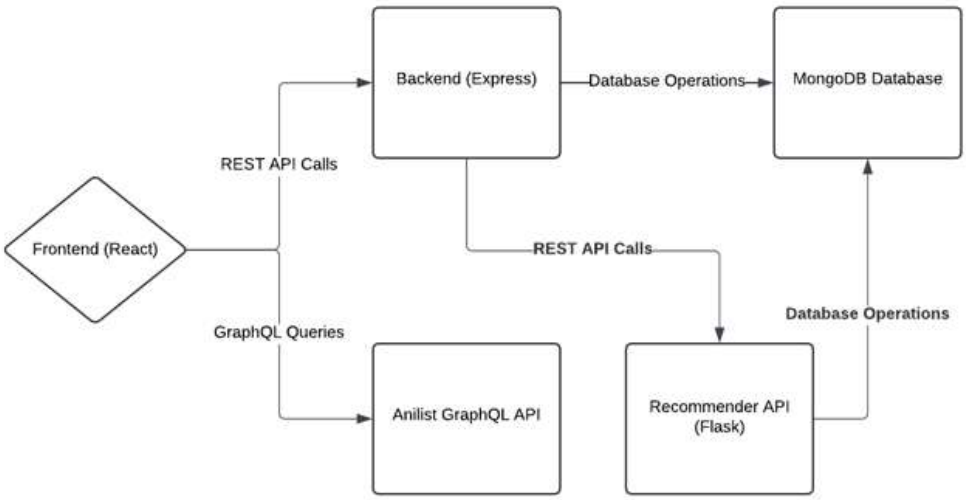
Рекомендаційна система має поєднувати в собі техніки фільтрації за змістом та колаборативної фільтрації для того, щоб:

1) На індивідуальних сторінках серіалів показувати схожі за описом, метаданими та тематикою серіали.
2) На сторінці профілю користувача показувати персональні рекомендації відповідно до його опису профілю та улюблених жанрів.
3) На домашній сторінці користувача показувати персоналізовані рекомендації на основі високо оцінених анімаційних серіалів користувачем та порівнянні з оцінками інших користувачів.

3.5 Оновлена архітектура веб-застосунку

Після інтеграції арі-серверу рекомендаційної системи архітектура веб-застосунку незначно ускладнюється. Нова її компонента, Flask сервер рекомендаційної системи, надає арі для генерації рекомендацій, що використовується основним Express сервером для отримання рекомендацій. Обидва сервери використовують спільну базу даних користувачів в MongoDb, проте сервер рекомендацій не змінює її, а лише отримує з неї інформацію про

користувача. Дані, що треба передавати між серверами для їх роботи зведено до мінімуму, що робить їх більш незалежними та зручними для використання.



Мал. 3.1 Оновлена архітектура веб-сайту

3.6 Структура серверу рекомендацій

Сервер рекомендацій має три кінцеві точки, що забезпечують отримання подібних серіалів, рекомендацій на основі смаків користувача та рекомендацій головної сторінки.

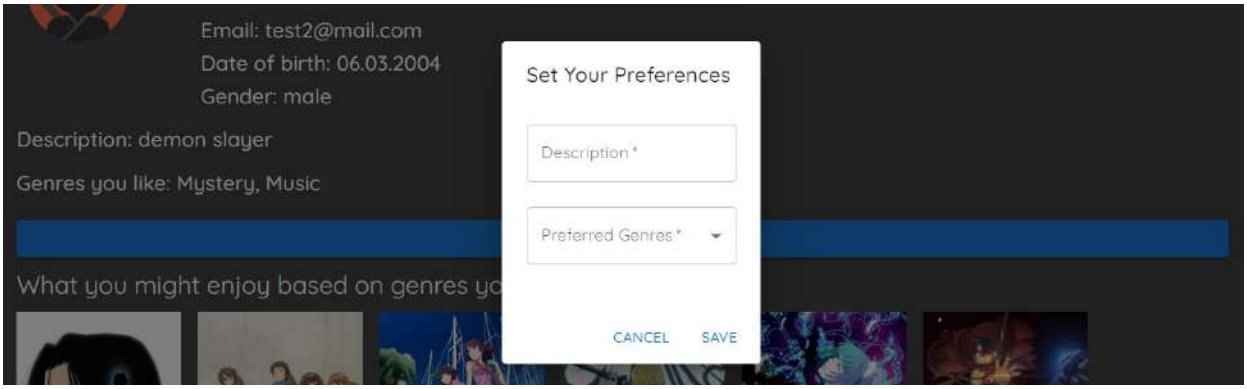
```python
@app.route("/recommend/similar-items", methods=["POST"])
def recommend_similar_items():
    item_data = request.get_json()
    anime_set = prepare_anime_set()
    topic_recs = recommend_similar_content_lda(anime_set, item_data)
    term_recs = recommend_similar_content_tfidf(anime_set, item_data)
    return {"topic_recs": topic_recs[:10], "term_recs": term_recs[:10]}, 200
```

Мал. 3.2 Кінцева точка отримання подібних серіалів

Функції, що відповідають за генерацію самих рекомендацій, винесено в окремий файл, а безпосередньо на кінцевих точках виконується лише мінімальна обробка запитів. Три кінцеві точки використовуються серверною частиною веб-сайту для отримання рекомендацій та відображення результату в інтерфейсі користувача.

3.7 Вирішення проблеми “холодного старту”

Перед реалізацією інших методів рекомендацій слід врахувати проблему холодного старту, тобто надати користувачеві можливість отримувати рекомендації навіть у тому випадку, якщо він ще не був достатньо активним на веб-сайті, щоб отримувати персоналізовані рекомендації на домашній сторінці. Для цього користувачу надається можливість вказати свої улюблені жанри та надати короткий опис своїх смаків.



Мал. 3.3 Обрання смаків та опису

На основі наданих даних користувач отримує персоналізовані рекомендації, для яких ніяка інша інформація не використовується, що вирішує проблему “холодного старту”.

Для отримання рекомендацій за жанрами користувача розроблено наступний алгоритм:

1) Знаходяться усі анімаційні серіали, що мають спільні жанри з тими, що вказані користувачем.
2) Отриманий результат сортується за середніми рейтингами серіалів, що отримуються з датасету.
3) Обирається 10 серіалів з найвищим рейтингом з тих, що входять в перетин жанрів.

```python
# knowledge-based recommendation

def recommend_by_genres(anime_data, pref_genres):
    anime_data["genres"] = anime_data["genres"].apply(lambda genre_lst: ast.literal_eval(genre_lst))
    filtered_data = anime_data[anime_data["genres"].apply(lambda genre_lst: set(genre_lst).intersection(pref_genres)).astype(bool)]
    filtered_data_sorted = filtered_data.sort_values(by="mean_score", ascending=False)
    recommended_ids = filtered_data_sorted["id"].tolist()
    return recommended_ids
```

Мал 3.4 Функція рекомендацій за жанрами

Отримання рекомендацій за описом є більш складною задачею, бо воно потребує обробки опису та його порівняння з описами самих анімаційних серіалів. У системі даний метод реалізовано наступним чином:

1) Датасет серіалів фільтрується так, щоб в ньому не було серіалів з середнім рейтингом нижче ніж 75 зі 100, бо погані серіали, навіть якщо вони подібні до смаків користувача, система рекомендувати не має.
2) Виконується векторизація описів, включаючи опис користувача, за допомогою TF-IDF методу.
3) За допомогою косинусу подібності визначається подібність серіалів, та вони сортуються за спаданням подібності.
4) Обирається 10 найподібніших за описом серіалів.

```python
# content-based recommendation (TF - IDF)

def recommend_by_description(anime_data, description):
    anime_data = anime_data[anime_data["mean_score"] > 75]
    anime_data["cleaned_description"] = anime_data["description"].apply(lambda html_desc: strip_html(html_desc.strip()))
    descriptions = anime_data["cleaned_description"].tolist()
    descriptions.append(description)
    ids = anime_data["id"].tolist()
    vectorizer = TfidfVectorizer(stop_words="english", lowercase=True, max_df=0.90, min_df=2)
    matrix = vectorizer.fit_transform(descriptions)
    similarity = linear_kernel(matrix[-1], matrix[:-1])
    id_sim_dict = dict(zip(ids, similarity.flatten()))
    sorted_dict = dict(sorted(id_sim_dict.items(), key=lambda item: item[1], reverse=True))
    return list(sorted_dict.keys())
```

Мал 3.5 Функція рекомендацій за описом

Після обрання смаків для користувача одразу доступні рекомендації, з яких він може почати своє ознайомлення з веб-сайтом.

Мал. 3.6 Рекомендації за жанрами “Кохання”, “Спорт”

Мал. 3.7 Рекомендації за описом “I enjoy space battles and huge spaceships!”

3.8 Подібні серіали

3.8.1 Подібність серіалів за метаданими

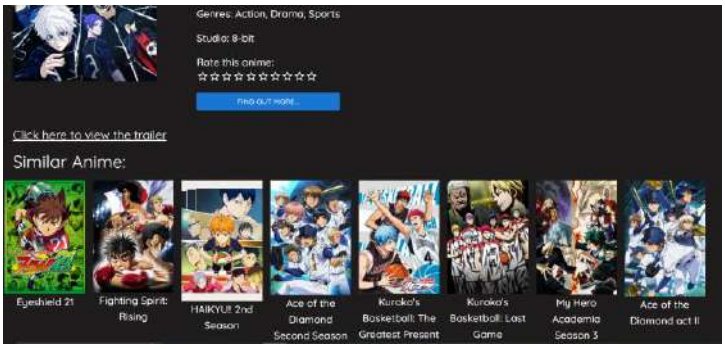
Коли користувач переходить на індивідуальну сторінку якогось серіалу, він має можливість також переглянути і подібні серіали. Доцільно скористатися доступними метаданими, такими як: опис, жанри, теги, назви студій-творців серіалів. Для цього в системі розроблено наступний метод фільтрації за змістом:

1) Датасет серіалів фільтрується так, щоб в ньому не було серіалів з середнім рейтингом нижче ніж 75 зі 100.
2) Виконується підготовка даних до векторизації та видалення зайвої інформації.
3) Створюються TF-IDF вектори для підготовлених даних.
4) Знаходиться подібність серіалів за допомогою косинусу подібності.
5) Обирається 10 найподібніших до даного за метаданими серіалів.

```python
def recommend_similar_content_tfidf(anime_data, target_data):
    anime_data = anime_data[anime_data["mean_score"] > 75]
    anime_texts = prepare_anime_soup(anime_data)
    target_combined_text = prepare_target_soup(target_data)
    corpus = anime_texts + [target_combined_text]
    ids = anime_data["id"].tolist()
    vectorizer = TfidfVectorizer(stop_words='english', lowercase=True, max_df=0.90, min_df=2)
    matrix = vectorizer.fit_transform(corpus)
    similarity = linear_kernel(matrix[-1], matrix[:-1])
    id_sim_dict = dict(zip(ids, similarity.flatten()))
    sorted_dict = dict(sorted(id_sim_dict.items(), key=lambda item: item[1], reverse=True))
    return list(sorted_dict.keys())
```

Мал. 3.8 Функція подібності серіалів за метаданими

Даний метод вже є повноцінною рекомендаційною системою фільтрації за змістом, що надає подібні серіали, дозволяючи користувачеві одразу знайти схожі серіали на ті, що йому сподобались.



Мал. 3.9 Подібні серіали до серіалу “Bluelock” (спортивна тематика)

3.8.2 Подібність серіалів за тематикою

Хоча подібність за метаданими надає влучні рекомендації в більшості випадків, цей метод не завжди здатний вловити тематичні особливості анімаційних серіалів, а також може надокучити користувачеві очевидними рекомендаціями. Для того, аби подолати цю проблему, в системі розроблено метод визначення подібності серіалів за спільною тематикою. Цей метод складніше попереднього та потребує більшої підготовки та певних обчислень, що не проводяться в реальному часі.

Підготовча частина складається з наступних кроків:

1) З датасету отримати теги серіалів та провести їхню обробку: токенізацію та лематизацію (приведення різних граматичних форм слова до однієї форми).
2) Зберегти створений список тегів для кожного серіалу в окремий файл для подальшого перевикористання.

3) Створити вектори з частотою входження слів для кожного набору тегів, щоб в подальшому використати їх для моделі латентного розміщення Діріхле.
4) Визначити коректні параметри LDA моделі.
5) Натренувати LDA модель з використанням отриманих векторів.
6) Зберегти отримані вектори та модель для подальшого використання.
7) На основі отриманого розподілу тем визначити косинус подібності анімаційних серіалів.
8) Знайти серіали з найкращим значенням подібності за допомогою сортування.

Після виконання такої підготовки метод швидко працює в реальному часі, виконуються наступні кроки:

1) Датасет серіалів фільтрується так, щоб в ньому не було серіалів з середнім рейтингом нижче ніж 75 зі 100.
2) Збережений раніше список тегів отримується та поєднується з тегами поточного серіалу, для якого визначається подібність.
3) Завантажуються попередньо збережені вектори та натренована LDA модель.
4) Визначається розподіл тем та їхня подібність на основі розподілу.
5) Серіали з найкращим значенням подібності знаходяться за допомогою сортування.

Даний метод інколи надає менш очевидні рекомендації, ніж подібність за метаданими, проте може показати і гірший результат, бо не завжди вдається визначити влучну тематику.

```
def recommend_similar_content_lda(anime_data, target_data, model_path='anime_lda.pkl', vectorizer_path='anime_count.pkl'):
    anime_data = anime_data[anime_data["mean_score"] > 75]
    target_tag = list(map(lambda tag: tag["name"], target_data["tags"]))
    corpus = []
    with open('preprocessed_corpus.txt', 'r', encoding='utf-8') as f:
        corpus = f.read().split('\n')
    corpus += [preprocess_corpus(' '.join(target_tag))]
    ids = anime_data["id"].tolist()
    vectorizer = joblib.load(vectorizer_path)
    lda = joblib.load(model_path)
    matrix = vectorizer.transform(corpus)
    lda_result = lda.transform(matrix)
    similarity_scores = cosine_similarity(lda_result[-1].reshape(1, -1), lda_result[:-1])
    id_sim_dict = dict(zip(ids, similarity_scores.flatten()))
    sorted_dict = dict(sorted(id_sim_dict.items(), key=lambda item: item[1], reverse=True))
    return list(sorted_dict.keys())
```

Мал. 3.10 Функція рекомендацій за тематикою

Даний метод ефективно доповнює спосіб знаходження подібних серіалів за метаданими, бо вони перекривають деякі недоліки одне одного: очевидність першого методу перекривається непередбачуваністю латентного розміщення Діріхле, а неточність другого методу перекривається передбачуваністю першого методу, тож об’єднана система фільтрації за змістом забезпечує більш якісні рекомендації. Наприклад, отримані рекомендації до футуристичного серіалу “Neon Genesis Evangelion” включають як і тематично подібні серіали, так і серіали та фільми з того самого циклу “Evangelion”, так і неочікуваний серіал “Hellsing Ultimate”, що містить подібні військові теми, філософські теми тощо.

Мал. 3.11 Рекомендації за тематикою до серіалу “Neon Genesis Evangelion”

3.9 Унікальний гібридний метод рекомендацій

3.9.1 Проблема колаборативних методів

Колаборативні методи рекомендацій містять кілька проблем, які розроблена рекомендаційна система намагається вирішити:

1) Такі методи вимагають присутності активної користувацької бази, що залишає оцінки серіалам.
2) Такі методи не завжди враховують актуальність рейтингів, а відповідно користувач отримує серед рекомендацій те, що його вже не цікавить.

3.9.2 Вирішення проблеми відсутності користувацької бази

У випадку відсутності достатньої кількості рейтингів можлива генерація бази користувачів вручну. Не варто генерувати таку базу випадковим чином, бо отримані рейтинги будуть використовуватися для передбачення рейтингів поточного користувача. Доцільно скористатися деякими статистичними даними

про наявні оцінки, наприклад, середнім рейтингом кожного серіалу. На основі середніх оцінок можливо згенерувати набір користувачів з випадковими оцінками серіалів, що мають певне відхилення від середньої оцінки серіалу, проте дозволяють відобразити смаки людей стосовно певних серіалів. Після генерації доцільно зберегти отримані рейтинги для повторного використання.

```python
def generate_synthetic_ratings(average_scores, rating_scale=100, deviation_factor=12, max_rating_amount=50):
    all_ratings = []
    for anime_id, avg_score in average_scores.items():
        deviations = np.random.normal(loc=0, scale=deviation_factor, size=np.random.randint(2, (max_rating_amount + 1)))
        ratings = avg_score + deviations
        ratings = np.clip(ratings, 0, rating_scale)
        ratings_prepared = list(map (lambda rating: int(rating / 10), ratings))
        i = 1
        for rating in ratings_prepared:
            entry = {}
            entry["user_id"] = f'synthethic{i}'
            entry["anime_id"] = str(anime_id)
            entry["rating"] = rating
            all_ratings.append(entry)
            i += 1
    return all_ratings
```

Мал. 3.12 Функція генерації рейтингів

Отримані рейтинги відображають загальні тенденції оцінок та не містять неочікуваних для системи користувачів, тож їхнє використання гарантує високу якість рекомендацій поточного користувача.

```
commender > ≡ synthethic_ratings.txt
1   "synthethic1", "anime_id": "6843", "rating": 2}, {"user_id": "synthethic2", "anime_id": "6843", "rating": 4}, {"user_id": "synthethic3
```

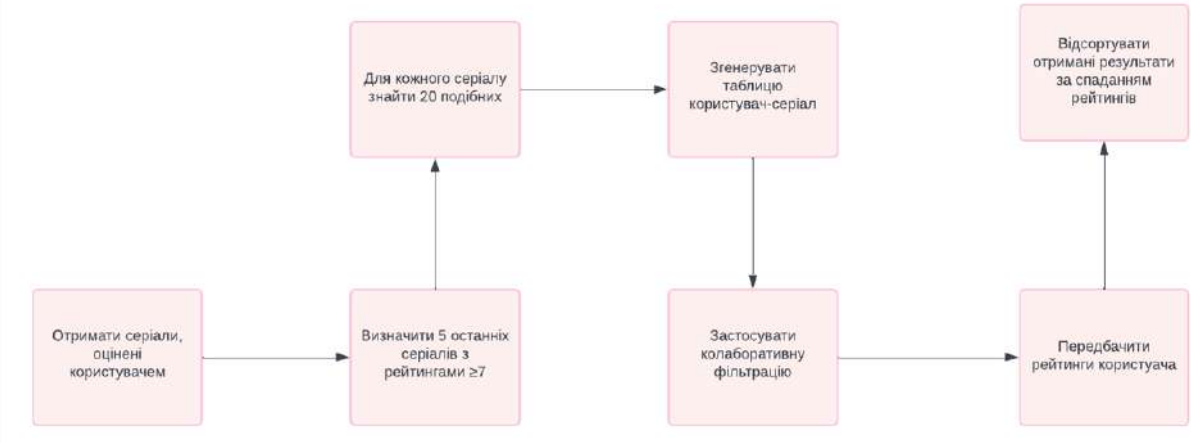
Мал. 3.13 Збережені рейтинги

3.9.3 Реалізація гібридного алгоритму основної сторінки

Згенеровані рейтинги користувачів дозволяють скористатися методами колаборативної фільтрації, проте такий метод буде недостатньо персоналізованим та актуальним для поточного користувача. Аби зробити рекомендації більш персоналізованими та актуальними, розроблено гібридний метод рекомендацій на основі рейтингів користувачів. Цей метод показує кращі рекомендації, ніж методи колаборативної фільтрації в чистому вигляді.

Ідея методу полягає в тому, щоб застосовувати колаборативну фільтрацію до попередньо обраної за допомогою фільтрації за змістом множини серіалів, подібних до тих, що подобалися користувачеві раніше, враховуючи актуальність рейтингів. Алгоритм можна описати так:

1) Отримати серіали, оцінені користувачем.
2) Для 5 серіалів, рейтинг для яких залишали останнім, та наданий рейтинг не нижче 7, знайти по 20 подібних серіалів. Для вирахування подібності використовується функція подібності за метаданими серіалів.
3) На основі згенерованих рейтингів та рейтингів поточного користувача згенерувати таблицю користувач-рейтинг.
4) До отриманої таблиці застосувати колаборативну фільтрацію.
5) Передбачити рейтинги користувача для серіалів та відсортувати результати за спаданням рейтингу.



Мал. 3.14 Схема гібридного методу

В результаті використання цього методу отримаємо рекомендації, що залишаються персоналізованими та актуальними навіть для свіжих користувачів. Користувачеві достатньо лишити 5 рейтингів, щоб отримувати рекомендації на їхній основні. За рахунок використання алгоритму, користувач в якості рекомендацій отримає як і продовження переглянутих ним серіалів, подібні до них серіали, так і неочікувані для нього серіали з високим рейтингом. Отримані рекомендації будуть оновлюватися в реальному часі залежно від останніх рейтингів користувача, тож вони завжди залишаються актуальними.

```python
def find_similar_anime(user_ratings, rated_anime, anime_data):
    high_rated = list(filter(lambda entry: entry["rating"] > 6, user_ratings["ratings"]))
    high_rated = sorted(high_rated, key=lambda item: item["updatedAt"], reverse=True)
    if not high_rated:
        return None
    else:
        if len(high_rated) > 5:
            high_rated = high_rated[:5]
    ids = list(map(lambda entry: entry["content_id"], high_rated))
    interesting_anime = list(filter(lambda entry: str(entry["id"] )in ids, rated_anime))
    similar_ids = []
    for anime in interesting_anime:
       similar_ids += recommend_similar_content_tfidf(anime_data, anime)[:20]
    return list(map(lambda id: str(id), set(similar_ids)))
```

Мал. 3.15 Функція знаходження множини подібних серіалів до оцінених

```python
def recommend_collab(user_ratings, rated_anime, anime_data):
    similar_anime_ids = find_similar_anime(user_ratings, rated_anime, anime_data)
    ratings = []
    with open('synthethic_ratings.txt', 'r', encoding='utf-8') as f:
        ratings = json.load(f)
    actual_ratings = []
    for rating in user_ratings["ratings"]:
        entry = {}
        entry["user_id"] = str(user_ratings["user_id"])
        entry["anime_id"] = str(rating["content_id"])
        entry["rating"] = rating["rating"]
        actual_ratings.append(entry)
    all_ratings = ratings + actual_ratings
    ratings_df = pd.DataFrame(data=all_ratings, columns=["user_id", "anime_id", "rating"])
    user_anime_table = pd.pivot_table(data=ratings_df, values="rating", columns="anime_id", index="user_id")
    reader = Reader(rating_scale=(0, 100))
    dataset = Dataset.load_from_df(ratings_df[['user_id', 'anime_id', 'rating']], reader)
    algorithm = KNNBasic(sim_options={'user_based': True})
    training_set = dataset.build_full_trainset()
    algorithm.fit(training_set)
    anime_to_predict = user_anime_table.columns[~user_anime_table.loc[str(user_ratings["user_id"])].notna()]
    if similar_anime_ids:
        anime_to_predict = list(filter(lambda anime_id: anime_id in similar_anime_ids, anime_to_predict))
    predictions = []
    for anime_id in anime_to_predict:
        predicted_rating = algorithm.predict(str(user_ratings["user_id"]), anime_id).est
        predictions.append((anime_id, predicted_rating))
    sorted_preds = sorted(predictions, key=lambda item: item[1], reverse=True)
    return list(map(lambda pred: pred[0], sorted_preds))
```

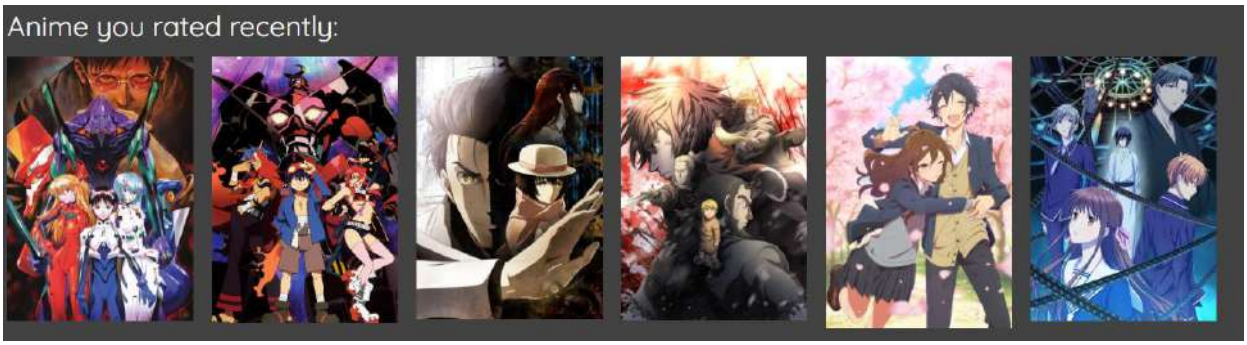
Мал. 3.16 Функція знаходження рекомендацій гібридним методом

Такий метод надає цікаві користувачеві рекомендації, застосовуючи алгоритми як колаборативної, так і фільтрації за змістом. Після того, як користувач залишить перші 5 рейтингів, цей метод буде надавати для нього рекомендації на домашній сторінці. Поєднання обох методів гарантує появу серед рекомендацій анімаційних серіалів, що найбільше подобаються користувачам веб-сайту серед тих, які подібні до тих, що високо оцінені поточним користувачем.

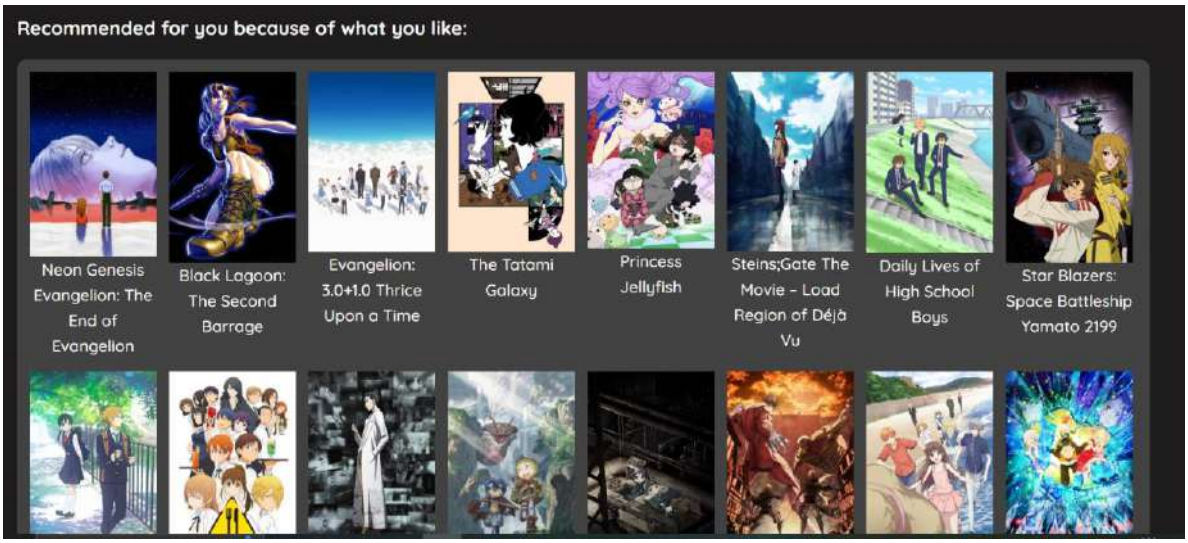
Порівняння SVD та методу k-найближчих сусідів за допомогою RMSE показало, що другий метод дає кращі результати, тому він використовується у фінальній версії алгоритму.

```
SVD RMSE: 1.3269800722559018
k-NN RMSE: 1.2489596160470882
```

Мал. 3.17 Порівняння SVD та k-NN



Мал. 3.18 Залишені оцінки



Мал. 3.19 Рекомендації на основі оцінок

Висновки до розділу 3

У даному розділі було описано розробку рекомендаційної системи для веб-сайту, розробленого у попередньому розділі. Було надано план розробки, згадано використані технології та зображено оновлену архітектуру веб-сайту. Було описано структуру Flask серверу рекомендацій та функціонал системи. Окрему увагу було приділено розробленим методам рекомендацій та вирішеним проблемам, зокрема проблемі “холодного старту” та недоліках колаборативної фільтрації. Детально описано реалізовані алгоритми для знаходження рекомендацій на основі смаків користувача, знаходження подібних серіалів та гібридного методу рекомендацій, що поєднує в собі генерацію рейтингів користувачів, фільтрацію за змістом та колаборативну фільтрацію.

ВИСНОВКИ

В процесі виконання даної роботи було успішно виконано поставлену мету та проаналізовано теоретичні засади рекомендаційних систем.

В першому розділі роботи було проаналізовано та детально описано особливості реалізації рекомендаційних систем, їхню історію, класифікацію, проблеми, що часто виникають при їхній розробці та основні методи генерації рекомендацій.

Другий розділ присвячений розробці веб-сайту. Було вичерпно описано процес розробки, зображено архітектуру, представлено використані інструменти та методи, а також кінцевий результат – розроблений веб-сайт.

Третій розділ роботи присвячений розробці рекомендаційної системи для веб-сайту. В ньому детально описано розроблену систему, представлено оновлену архітектуру веб-сайту та надано пояснення створених методів рекомендацій. Розроблені методи вирішують актуальні проблеми рекомендаційних систем, з якими стикаються розробники.

Отже, отримано наступні результати:

1) Розроблено веб-сайт для перегляду інформації про анімаційні серіали з можливістю залишати їм оцінки.
2) Для веб-сайту розроблено рекомендаційну систему, що вирішує актуальні проблеми рекомендаційних систем.
3) Вирішено проблему “холодного старту” за допомогою методу рекомендацій на основі опису та улюблених жанрів (системи переваг), що надається користувачем.
4) Вирішено недоліки колаборативного методу шляхом реалізації гібридного методу, що поєднує в собі та вдосконалює основи колаборативної фільтрації та фільтрації за вмістом.
5) Натреновано та оптимізовано модель латентного розміщення Діріхле для знаходження подібних серіалів за їх тегами.

Розроблений гібрідний метод, натренована модель латентного розміщення Діріхле та розроблений підхід до вирішення проблеми “холодного старту” можуть бути рекомендовані до використання у подібних рекомендаційних системах. Доведено їхню ефективність на прикладах генерованих системою рекомендацій. Можливе розширення даної рекомендаційної системи за допомогою інших методів, наприклад, доцільна імплементація в систему нейронної мережі з глибинним навчанням для вищої якості рекомендацій, також можливе застосування теорії MCDA систем для покращення якості рекомендацій, тож розроблені методи можна легко вдосконалити, що доводить їхню гнучкість в умовах сучасного світу.

Список використаних джерел

1. How Netflix’s Recommendations System Works [Електронний ресурс] – Режим доступу до ресурсу: https://help.netflix.com/en/node/100639.
2. Netflix Algorithm: Everything You Need to Know About the Recommendation System of the Most Popular Streaming Portal [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://recostream.com/blog/recommendation-system-netflix.
3. Falk K. Practical Recommender Systems / Kim Falk. – Shelter Island: Manning, 2019. – 432 с. – (ISBN 9781617292705).
4. Shriver D. Assessing the Quality and Stability of Recommender Systems / Shriver David, 2018. – 88 с.
5. Dong Z. A Brief History of Recommender Systems [Електронний ресурс] / Z. Dong, Z. Wang, J. Xu. – 2022. – Режим доступу до ресурсу: https://arxiv.org/pdf/2209.01860.
6. Resnik P. GroupLens: An Open Architecture for Collaborative Filtering of Netnews [Електронний ресурс] / P. Resnik, N. Iacovou, M. Suchak – Режим доступу до ресурсу: http://ccs.mit.edu/papers/CCSWP165.html.
7. Karabiber F. Cosine Similarity [Електронний ресурс] / Fatih Karabiber – Режим доступу до ресурсу: https://www.learndatasci.com/glossary/cosine-similarity/.
8. Stewart K. Pearson’s correlation coefficient [Електронний ресурс] / Ken Stewart. – 2024. – Режим доступу до ресурсу: https://www.britannica.com/topic/Pearsons-correlation-coefficient.
9. Banik R. Recommendation Systems with Python / Rounak Banik., 2018. – (ISBN 9781788993753).
10. Dabbura I. K-means Clustering: Algorithm, Applications, Evaluation Methods, and Drawbacks [Електронний ресурс] / Imad Dabbura. – 2018. – Режим доступу до ресурсу: https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a.

11. Singular value decomposition [Електронний ресурс] – Режим доступу до ресурсу: https://ocw.mit.edu/courses/18-06sc-linear-algebra-fall-2011/d273f75ee2552a5c3c35ccab37e5edce_MIT18_06SCF11_Ses3.5sum.pdf.
12. Karabiber F. TF-IDF — Term Frequency-Inverse Document Frequency [Електронний ресурс] / Fatih Karabiber – Режим доступу до ресурсу: https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/.