

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:

**«Веб-додаток для персоналізованого підбору книг з
використанням рекомендаційних алгоритмів»**

Виконала студентка групи ІІЗм-24-1
ХАРЧЕНКО Юлія Валеріївна

Керівник роботи:
ВАКАЛЮК Тетяна Анатоліївна

Рецензент:
САВІЦЬКИЙ Роман Святославович

Житомир – 2025

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
«18» вересня 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **ХАРЧЕНКО Юлія Валеріївна**

Керівник роботи: **ВАКАЛЮК Тетяна Анатоліївна**

Тема роботи: **«Веб-додаток для персоналізованого підбору книг з використанням рекомендаційних алгоритмів»,**

затверджена Наказом закладу вищої освіти від **«18» вересня 2025 р. № 491/с**

Вихідні дані для роботи: **веб-додатки та системи персоналізованого підбору книг; рекомендаційні алгоритми та їх застосування для персоналізації пропозицій книг у веб-додатках.**

Консультанти з магістерської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Вакалюк Т.А.	22.09.2025р.	22.09.2025р.
2	Вакалюк Т.А.	22.09.2025р.	22.09.2025р.
3	Вакалюк Т.А.	22.09.2025р.	22.09.2025р.

РЕФЕРАТ

Кваліфікаційна робота освітнього ступеня «магістр» присвячена процесу персоналізованого підбору книг з використанням рекомендаційних алгоритмів. Пояснювальна записка складається з вступу, трьох розділів та списку використаної літератури. Пояснювальна записка до кваліфікаційної магістерської роботи містить 74 сторінок, 29 ілюстрацій, 11 таблиць, 5 додатків та 36 посилань на зовнішні джерела.

Метою роботи є створення веб-додатка для персоналізованого підбору книг та розробка зручного графічного інтерфейсу для користувачів. У роботі описано процес розробки програми, яка аналізує інтереси користувачів та пропонує книги на основі рекомендаційних алгоритмів. Досліджено способи побудови та впровадження алгоритмів рекомендацій, проведено тестування ефективності підбору літератури.

КЛЮЧОВІ СЛОВА: РЕКОМЕНДАЦІЙНИЙ АЛГОРИТМ, ПЕРСОНАЛІЗАЦІЯ, ВЕБ-ДОДАТОК, ПІДБІР КНИГ, ГРАФІЧНИЙ ІНТЕРФЕЙС.

ABSTRACT

The master's qualification work is dedicated to the process of personalized book recommendations using recommendation algorithms. The explanatory note consists of an introduction, three chapters, and a list of references. The explanatory note of the master's qualification work contains 74 pages, 29 illustrations, 11 table, 5 appendices and 36 references.

The aim of the work is to develop a web application for personalized book recommendations and to design a user-friendly graphical interface. The work describes the development process of a program that analyzes user preferences and suggests books based on recommendation algorithms. Methods for building and implementing recommendation algorithms are studied and the effectiveness of the book recommendation system is tested.

KEYWORDS: RECOMMENDATION ALGORITHM,
PERSONALIZATION, WEB APPLICATION, BOOK RECOMMENDATION,
GRAPHICAL USER INTERFACE.

ЗМІСТ

ВСТУП.....	6
РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО РЕКОМЕНДАЦІЙ	
КНИГ	9
1.1. Постановка завдання	9
1.2. Огляд наявних аналогів	10
1.3. Вибір програмних засобів для реалізації	15
1.4. Огляд методів, моделей та алгоритмів	18
Висновки до розділу 1	26
РОЗДІЛ 2. РОЗРОБКА ТА ПРОЕКТУВАННЯ МОДЕЛІ	
РЕКОМЕНДАЦІЙ	27
2.1. Підготовка та аналіз даних	27
2.2. Розробка моделі рекомендацій	31
2.3. Проектування веб-додатку	38
Висновки до розділу 2	42
РОЗДІЛ 3. ОПИС ПРОГРАМНОГО ПРОДУКТУ	43
3.1. Графічний інтерфейс	43
3.2. Відображення результатів	48
Висновки до розділу 3	58
ВИСНОВКИ	59
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	60
ДОДАТКИ	65

ВСТУП

Сучасні веб-платформи містять велику кількість інформації, через це користувачам стає дедалі складніше самотійно знаходити потрібний контент. Це особливо помітно у сфері книжкових сервісів, де кількість творів швидко зростає, а пошук цікавої книги без допоміжних інструментів займає багато часу. Тому питання побудови зручних механізмів навігації, персонального добору книг та організації свого читання є актуальним як для користувачів, так і для розробників веб-додатків.

Рекомендаційні системи активно застосовуються у більшості сучасних онлайн-сервісів. У наукових дослідженнях описуються різні підходи від простих методів на основі жанрів і рейтингів до складних моделей. Однак навіть порівняно прості рекомендаційні алгоритми здатні суттєво покращити взаємодію користувача з веб-платформою, якщо вони правильно інтегровані в архітектуру системи та працюють стабільно. Для практичних веб-сервісів особливо важливою є можливість виділення рекомендаційного модуля як окремої частини, яка не впливає на роботу всієї системи у випадку її тимчасового відключення. Це дозволяє підтримувати роботу сервісу навіть за умов обмежених ресурсів.

Практична цінність подібних систем полягає в тому, що вони допомагають користувачеві швидше знаходити книги, які можуть його зацікавити, зберігати особисті списки для читання та організовувати взаємодію з контентом у зручній формі. Для розробників це дає можливість створювати більш сучасні, привабливі та функціональні веб-додатки.

У цій роботі розглядається створення веб-додатку з рекомендаційною системою, що працює на основі даних про книги, жанри, оцінки та взаємодії користувачів. Система використовує прості загальнодоступні методи: добір рекомендацій за жанрами, врахування оцінок і визначення схожості між книгами. Окрему увагу приділено розробці архітектури, де рекомендаційний модуль існує як відокремлений компонент. Це дозволяє зберегти

працездатність головного сайту навіть тоді, коли рекомендаційна частина тимчасово недоступна.

Метою роботи є розробка веб-додатку з персональними рекомендаціями книг на основі простих алгоритмів та побудова архітектури, у якій рекомендаційний модуль може працювати незалежно від основної системи.

Для досягнення поставленої мети необхідно виконати такі **завдання**:

- 1) Проаналізувати існуючі підходи до побудови веб-платформ з рекомендаційною функціональністю.
- 2) Спроекувати структуру веб-додатку та його окремих модулів.
- 3) Розробити базу даних для зберігання інформації про книги, користувачів, списки та оцінки.
- 4) Створити рекомендаційний модуль, який формуватиме добірки книг за жанрами, рейтингами та схожістю.
- 5) Реалізувати взаємодію між основним застосунком і рекомендаційним модулем.

Об'єктом дослідження є процес формування рекомендацій у веб-додатках.

Предметом дослідження є методи побудови рекомендацій на основі жанрів, оцінок і схожості книг та їх інтеграція у модульну архітектуру веб-додатку.

Методи дослідження. У роботі використовуються методи системного та порівняльного аналізу при виборі структури застосунку, методи моделювання даних для побудови схеми бази даних, підходи веб-розробки (React, Node.js), прості алгоритмічні методи для формування рекомендацій (фільтрація за жанрами, рейтингові списки, обчислення схожості між книгами), елементи модульного підходу до побудови архітектури.

Наукова новизна роботи полягає у використанні розділеної архітектури, у якій рекомендаційний модуль може працювати незалежно від основної частини веб-додатку, у реалізації комбінованого підходу до

рекомендацій книг на основі жанрів, оцінок та схожості, що дозволяє підвищити релевантність результатів та у забезпеченні можливості роботи системи навіть за вимкненого рекомендаційного модуля, що підвищує надійність і стійкість всього застосунку.

Практичне значення одержаних результатів полягає у створенні веб-додатку, який дозволяє користувачам знаходити книги, що відповідають їхнім вподобанням, зберігати власні списки та оцінювати твори та отримувати персоналізовані рекомендації без складних обчислювальних моделей. Розробникам дозволяє застосовувати модульну архітектуру, де рекомендаційний блок легко підтримувати, оновлювати та розширювати.

Апробація. За результатами дослідження опубліковано тези у матеріалах конференцій [1, 2].

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ ПІДХОДІВ ДО РЕКОМЕНДАЦІЙ КНИГ

1.1. Постановка завдання

Сучасні веб-додатки, які розраховані для підбору книг, мають мати високу точність рекомендацій, швидкість обробки запитів і зручний інтерфейс для користувачів. Основна проблема полягає в розробці алгоритму, який може працювати ефективно навіть при невеликих обсягах даних та враховувати зміни в вподобаннях читачів.

Завдання, які необхідно вирішити для створення ефективної системи рекомендацій:

1) Визначення проблеми:

- визначити вимоги до системи рекомендацій,
- врахувати як можна працювати з малими наборами даних,
- визначити, які методи рекомендацій варто використовувати.

2) Створення моделі:

- представити вподобання користувача у числовому вигляді,
- проаналізувати поведінку читача за історичними даними,
- використати методи статистики та машинного навчання.

3) Реалізація алгоритмів:

- вибір і впровадження підходу до фільтрації,
- застосування алгоритмів машинного навчання для підвищення точності прогнозування.

4) Інтеграція у систему:

- визначити архітектуру,
- створити зручний інтерфейс.

5) Аналіз ефективності:

- перевірити точність рекомендацій,
- оптимізувати параметри на основі тестування.

Таким чином, для успішної реалізації рекомендаційної системи необхідно дослідити існуючі методи, обрати оптимальні підходи та

впровадити алгоритми, які забезпечать точність, швидкодію та адаптивність системи до змін у вподобаннях читачів.

1.2. Огляд наявних аналогів

На ринку існує багато різних сервісів, що надають персоналізовані книжкові рекомендації. Для аналізу обрала три популярні - це Goodreads, LibraryThing та BookBub. Розглянемо кожен з них детальніше.

Goodreads - одна з найбільш популярних платформ для зберігання та обговорення книжок. Платформа дозволяє користувачам оцінювати книги, залишати рецензії та отримувати персоналізовані рекомендації на основі історії прочитаного та улюблених жанрів.

Цей веб-ресурс використовує фільтрацію як контентну, так і колаборативну. Контентна фільтрація ґрунтується на аналізі характеристик самих книжок. Алгоритм аналізує жанри, тематику, автора, ключові слова, анотації та інші параметри книг, які вже прочитали та оцінили. На основі цих даних система підбирає нові книги, які мають схожі риси. Колаборативна фільтрація використовує дані про вподобання інших користувачів, які мають схожий смак із вами. Алгоритм аналізує, які книги сподобалися людям, що мають схожі оцінки на ті ж книги, що й ви, і пропонує вам ті твори, які вони також високо оцінили.

Поєднання цих двох методів дозволяє цьому сайту рекомендувати книги не тільки за об'єктивними характеристиками, а й на основі спільних уподобань користувачів, що робить рекомендації більш персоналізованими та точними. Зовнішній вигляд сайту на рис 1.1.

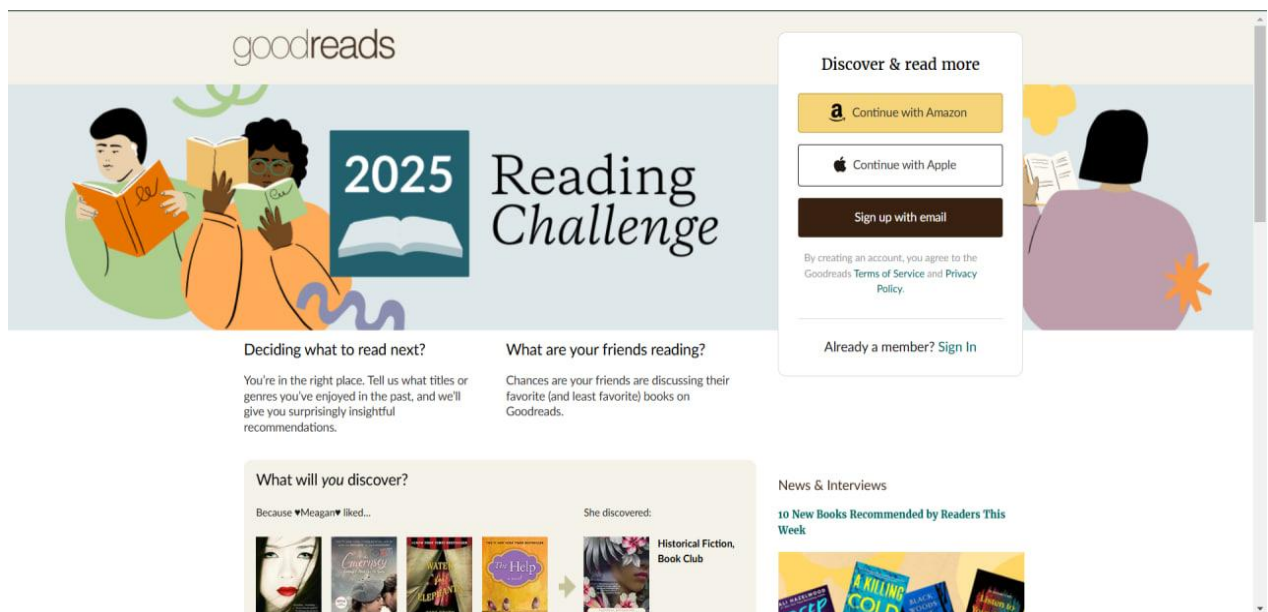


Рис. 1.1. - Вигляд сайту Goodreads

До переваг цього сервісу можна віднести перш за все велику базу користувачів і книг, завдяки якій він здатен формувати досить надійні рекомендації. Активна спільнота читачів теж відіграє свою роль, бо люди активно обговорюють прочитане та діляться рецензіями. Крім того, сервіс дозволяє створювати власні списки книжок і має інтеграцію з Amazon, тому за потреби книги можна легко придбати.

Однак є й недоліки. Інтерфейс виглядає застарілим і не надто зручним у використанні. Рекомендаційний алгоритм часто віддає перевагу більш популярним творам, через що унікальні літературні вподобання користувача враховуються не надто глибоко. Також сервіс не надто добре підходить для пошуку маловідомих або нішевих книг.

LibraryThing - платформа, орієнтована на книголюбів, які хочуть якимось систематизувати свою бібліотеку та отримувати різні рекомендації. Вона використовує співпадіння у книжкових колекціях інших користувачів для надання персоналізованих пропозицій.

Сайт застосовує метод порівняння книжкових колекцій користувачів для формування рекомендацій. Тобто платформа аналізує, які книги є у вашій бібліотеці, і шукає інших користувачів із подібними колекціями. На основі цього аналізу система пропонує книги, які є у схожих користувачів, але яких

ще немає у вас. Детальніше, додаються книги у свій каталог, а система фіксує їхні назви, авторів, жанри та інші характеристики. Алгоритм LibraryThing аналізує інші бібліотеки та шукає тих, у кого є значна кількість спільних книжок із вашими. Якщо багато людей, у яких є подібні книги та також мають у своїх бібліотеках певні інші книги, яких у вас ще немає, система пропонує вам їх як потенційно цікаві.

Цей метод нагадує колаборативну фільтрацію, але більшою мірою орієнтований на цілісні бібліотеки, а не лише на оцінки книг. Це дозволяє формувати рекомендації на основі того, які книги реально зберігають і читають користувачі, а не просто оцінюють. На рис 1.2. можна детально побачити як він виглядає.

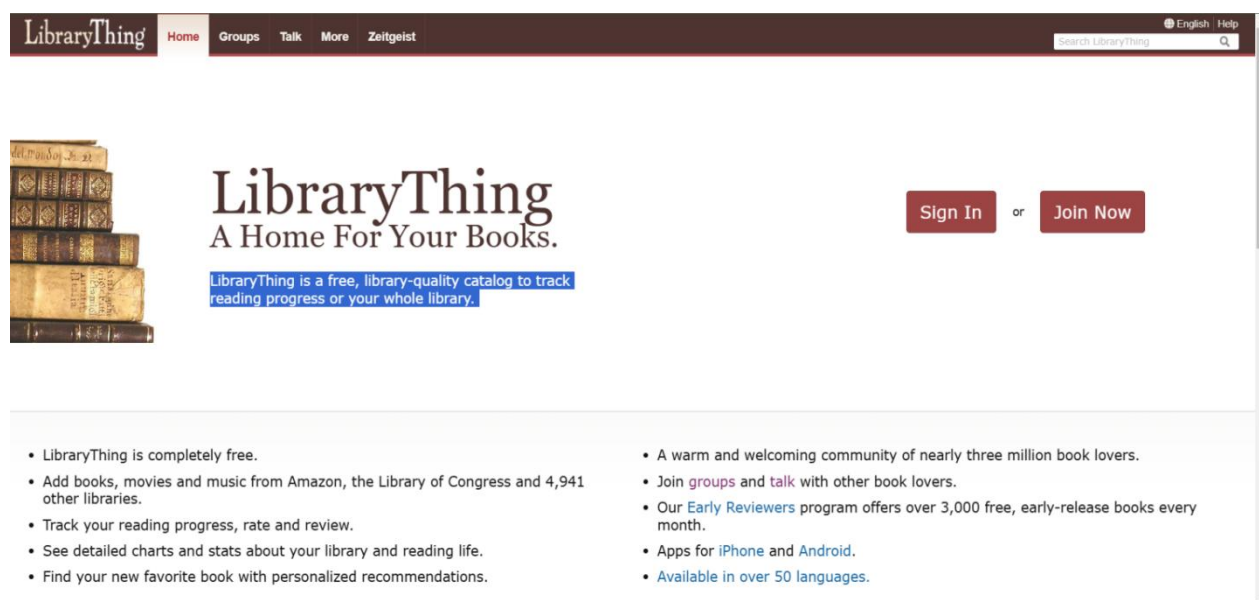


Рис. 1.2. - Вигляд сайту LibraryThing

Сервіс добре підходить для роботи з рідкісними чи нішевими книгами, а також підтримує детальні метадані, завдяки чому можна дуже точно описувати кожне видання. Він не має нав'язливої реклами чи комерційного ухилу й орієнтований переважно на людей, які колекціонують рідкісні або спеціалізовані книжки.

Однак тут також є певні мінуси. Багато даних доводиться вводити вручну, а спільнота тут значно менш активна, ніж на Goodreads. Дизайн і загальний функціонал виглядають застарілими, тож новачкам інтерфейс

може здатися заплутаним. Крім того, у безкоштовній версії діють обмеження на кількість книг, які можна додати.

BookBub - сервіс, що спеціалізується на персоналізованих книжкових пропозиціях і знижках. Він вивчає читацькі вподобання та підбирає акційні книги, які можуть зацікавити користувача.

Аналіз вподобань відбувається шляхом збору інформації про улюблені жанри, авторів і попередні покупки, після чого система знаходить найкращі пропозиції від видавців і авторів. Тобто при реєстрації на сайті користувач обирає улюблені жанри. Він також може слідувати за улюбленими авторами, щоб отримувати сповіщення про їхні нові книги чи знижки на їхні твори. Платформа враховує книги, якими цікавився користувач, які він додавав у список бажаних або купував через сервіс. Сайт може аналізувати загальні тенденції серед користувачів із подібними смаками. Таким чином, користувач отримує добірку книг за зниженими цінами, що відповідають його вподобанням.

Так сервіс не просто показує знижені ціни, а підбирає саме ті акції, які найбільше відповідають вашим літературним інтересам. На рис 1.3. можна побачити який вигляд має сайт.

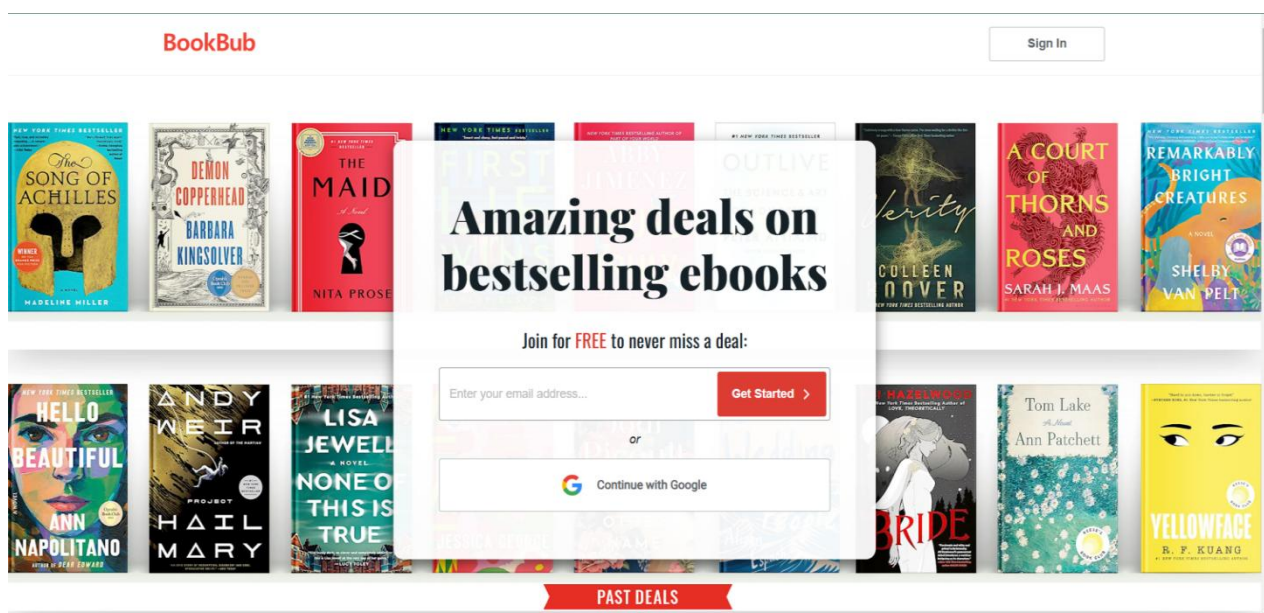


Рис. 1.3. - Вигляд сайту BookBub

Перевагою цього сервісу є те, що він добре інтегрується з електронними магазинами, тому особливо підходить тим, хто любить читати в цифровому форматі або шукає бюджетні варіанти книг. Тут можна стежити за улюбленими авторами й отримувати сповіщення про їхні нові роботи. Також щодня можна переглядати добірки зі знижками чи навіть безкоштовними пропозиціями.

У нього є свої недоліки. Персоналізовані рекомендації не завжди точні, а сам сервіс орієнтований переважно на англomовні видання, через що вибір для інших мов помітно обмежений. Основна увага зосереджена на знижках, а не на якості рекомендацій. Також тут немає можливості додавати фізичні книги або вести власний каталог.

Проведений аналіз аналогів показує, що сучасні сервіси мають певні обмеження та недоліки: відсутність ефективних рішень для роботи з малими вибірками даних, орієнтація на популярні книги, що може не відповідати персональним уподобанням користувача та недостатній аналіз поведінкових факторів користувачів та змін їхніх вподобань з часом.

Зважаючи на це, важливо розробити систему, яка б не тільки враховувала історичні дані користувача, а й адаптувалася до його змінних вподобань та контексту читання.

Щоб порівняти всі аналоги, було розроблено таблицю (табл. 1.1).

Таблиця 1.1

Порівняння аналогів

Назва	Goodreads	LibraryThing	BookBub
Метод рекомендацій	Контентна та колаборативна фільтрація	Ручне введення оцінок, власні списки	Аналіз читацьких звичок, комбінований підхід
Персоналізація	Часто пропонує популярні книги	Базується на власних оцінках	Враховує читацькі вподобання
Соціальна взаємодія	Є групи, обговорення, відгуки	Більше для особистого використання	Мінімальна, фокус на рекомендаціях

Продовження таблиці 1.1

1	2	3	4
Зручність використання	Інтуїтивний інтерфейс з рекламою	Менш автоматизований	Простий
Доступ до книг	Тільки оцінка та обговорення, купівля через партнерів	Додавання книг вручну, каталогізація	Пропонує книги зі знижками
База даних	Велика, охоплює популярні та нішеві книги	Менша, акцент на особистій бібліотеці	Орієнтація на популярні книги
Орієнтація	Обговорення, відкриття нових книг	Каталогізація та керування особистою бібліотекою	Книги зі знижками та персональні рекомендації

1.3. Вибір програмних засобів для реалізації

Щоб розробити добре працюючий веб-сервіс, треба обрати стек технологій, який забезпечить ефективність, масштабованість та зручність у розробці й підтримці проєкту. Веб-додаток можна умовно поділити на серверну та клієнтську частини, а також на засоби для реалізації алгоритмів та бази даних.

Для реалізації саме серверної частини буде використано Node.js. Це сучасне середовище виконання JavaScript, яке дозволяє запускати код поза браузером, а саме на сервері.

Node.js працює за принципом використання подій. Це дозволяє ефективно обробляти велику кількість одночасних запитів від користувачів. Це середовище виконання працює завдяки рушію V8, що дозволяє виконувати код швидко, що робить його чудовим вибором для продуктивних веб-додатків [3]. Node.js відомий своїм менеджером пакетів npm, завдяки якому розробники мають доступ до тисячі різних готових рішень. Він також підходить для мікросервісної архітектури, таким чином можна легко

розширювати проєкт. У інтернеті є велика кількість прикладів кодів, різна документація, тому це дозволить спростити розробку.

Таким чином, я обрала Node.js, бо він є асинхронним, швидкодіючим, має великий набір пакетів, маштабований та є доволі популярним серед спільноти.

Для зберігання даних буде використовуватись MongoDB. Це нереляційна база даних, яка зберігає дані у форматі JSON-подібних документів. Вона має певні відмінності з реляційними базами даних. Для прикладу розглянемо різницю між MongoDB та MySQL (табл. 1.2).

Таблиця 1.2

Різниця між нереляційною та реляційною базами[4]

Функція	MongoDB	MySQL
Структура	Документ (JSON)	Таблиці та рядки
Гнучкість схеми	Динамічна	Фіксована
Запити	JSON-запити	SQL-запити
Зв'язки між даними	Вбудовані документи	Складні зв'язки
Масштабування	Горизонтальне	Вертикальне

MongoDB підтримує JSON-подібний формат, що зручно для зберігання динамічних даних, тобто дані зберігаються у вигляді колекцій документів. Ця база даних не вимагає фіксованої структури, що дозволяє легко змінювати формат збережених записів. Також, вона підходить для великих навантажень і горизонтального масштабування. Має високу швидкість обробки запитів. Таке швидке читання та запис даних відбувається завдяки індексації та кешуванню [5].

Важливою перевагою є те, що у Node.js є офіційна бібліотека Mongoose для роботи з цією базою даних. Ця бібліотека дозволяє визначати схеми, валідувати дані та виконувати запити у зручній формі [6].

Тому MongoDB буду використовувати через свою гнучкість, маштабованість, більш високу продуктивність та сумісність із Node.js.

Для реалізації інтерфейсу веб-додатку, тобто клієнтської частини, буде використано React.js. Це JavaScript-бібліотека для створення інтерфейсів користувача. Вона використовує компонентний підхід для побудови динамічних веб-інтерфейсів.

Кожен елемент інтерфейсу є окремим незалежним компонентом, що сприяє повторному використанню коду. Завдяки Virtual DOM, бібліотека швидко оновлює лише необхідні частини сторінки. React.js дозволяє керувати динамічними змінами в інтерфейсі [7]. Також тут можна оптимізувати завантаження сторінки. Так само як в Node.js, тут є багато ресурсів і готових бібліотек, які спрощують розробку.

Отже, React.js має високу продуктивність, компонентний підхід, популярність та велику спільноту користувачів.

Для реалізації рекомендаційних алгоритмів у веб-додатку буде використано інструменти Python - pandas, NumPy, scikit-learn та SciPy.

Pandas забезпечує зручні структури даних та засоби аналізу табличної інформації, що дозволяє виконувати попереднє опрацювання, фільтрацію, очищення та трансформацію даних користувачів і книг [8].

NumPy використовується для високоефективних числових обчислень і роботи з багатовимірними масивами, що є основою для виконання математичних операцій у рекомендаційних алгоритмах [9].

SciPy доповнює можливості NumPy, надаючи оптимізовані математичні функції та інструменти для обчислення подібності, статистичного аналізу та лінійної алгебри [10].

Ключовою бібліотекою для реалізації алгоритмів рекомендацій є scikit-learn, яка містить широкий набір методів машинного навчання: обчислення косинусної подібності, кластеризацію, побудову моделей для прогнозування та алгоритмів контентної й колаборативної фільтрації. Вона також підтримує зручні інструменти для векторизації текстів, такі як TfidfVectorizer, які широко використовуються для побудови рекомендацій на основі контенту [11].

Отже, вибір зазначених технологій дозволяє створити ефективний, продуктивний і масштабований веб-додаток. Node.js забезпечує швидкість і масштабованість серверної частини, React.js - динамічний і зручний інтерфейс, а MongoDB - гнучке зберігання даних. Бібліотеки pandas, NumPy, SciPy та scikit-learn забезпечують ефективну обробку даних і реалізацію рекомендаційного алгоритму у Python-середовищі.

1.4. Огляд методів, моделей та алгоритмів

Багато рекомендаційних систем використовують різні підходи для створення персоналізованих рекомендацій для користувачів. Основними з цих підходів можна виділити фільтрацію на основі контенту, колаборативну фільтрацію, гібридний підхід. Також в таких системах часто використовують застосування машинного навчання.

Content-Based Filtering

Цей метод рекомендацій використовує аналіз характеристик контенту книг, таких як жанр, автор, ключові слова, описи та інше. Основна ідея полягає у тому, щоб порекомендувати користувачу ті книги, які мають подібні характеристики до тих, що він уже читав або оцінив.

Ця система працює наступним чином: вона будує профіль користувача на основі його попередніх вподобань. Якщо говорити про книги, то кожна з них описується через певний набір атрибутів, які система порівнює з вподобаними користувачем творами, щоб знайти найбільш відповідні рекомендації [12].

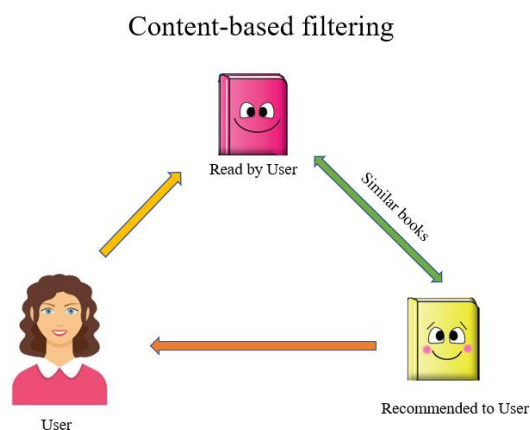


Рис. 1.4. - Схема Content-Based Filtering

На цій схемі (рис. 1.4.) детально показано як працює цей метод. Є користувач, він прочитав певну книгу, наприклад книгу автора-фантаста. Далі, система шукає схожі книжки, наприклад книжки цього ж автора або цього ж жанру - фантастики, і рекомендує їх цьому користувачеві.

В цілому, алгоритм роботи цієї системи доволі простий. Спочатку, відбувається побудова профілю користувача, тобто аналізується історія користувача, які книги він читав, які оцінки ставив, і обов'язково виділяються найпопулярніші атрибути, зазвичай це можуть бути жанри, автори, певні ключові слова.

Далі, робиться векторизація, а саме кожна книга представляється у вигляді вектору певних характеристик. Наприклад, книга може бути описана як: [наукова фантастика, пригоди, космос, 4.5].

Потім обчислюється схожість. Використовується метрика косинусної схожості або інші подібні метрики для порівняння векторів книг. Чим ближчий вектор книги до вектора профілю користувача, тим вища ймовірність, що книга буде рекомендована.

Наведу декілька конкретних прикладів роботи цього методу:

1) Користувач полюбляє читати книги в жанрі романтичне фентезі. Він прочитав такі книги як “Жорстокий принц” Голлі Блек та “Двір шипів та троянд” Сара Дж. Маас, таким чином йому порекомендує прочитати “Четверте крило” Ребекки Яррос, тому що ці книги написані в одному жанрі.

2) Користувач любить читати Стівена Кінга. Він встиг прочитати “Воно” та “Мізері”, рекомендація йому буде - “Сяйво”, “Керрі” чи інші роботи цього ж автора.

3) Користувач прочитав книги “1984” Джорджа Орвелла та “Каллокаїн” Карін Бос. Ключові слова цих творів - антиутопія та тоталітаризм. Ці слова підходять роману “Який чудесний світ новий!” Олдоса Гакслі, тому в рекомендованих також можна буде побачити цю книгу.

Переваги такого методу заключаються в тому, що рекомендації базуються на конкретних вподобаннях користувача, легко пояснити

користувачеві, чому система рекомендує ту чи іншу книгу, система не залежить від популярності книги серед інших користувачів.

Недоліками є те, що рекомендуються лише схожі книги, що обмежує можливість відкривати нові жанри, новим користувачам важко отримати релевантні рекомендації через відсутність історії, система може "заиклюватись" на певних жанрах або авторах.

Підсумовуючи переваги та недоліка вийшла наступна таблиця (табл. 1.3):

Таблиця 1.3

Переваги та недоліки Content-Based Filtering

Переваги	Недоліки
Персоналізація	Обмеженість
Прозорість	Проблема старту новачкам
Актуальність	Відсутність різноманітності

Collaborative Filtering

Колаборативна фільтрація є одним із найпопулярніших методів рекомендацій, ґрунтується на аналізі вподобань інших користувачів. Основна ідея полягає в тому, що якщо двоє людей у минулому мали схожі оцінки для певних книг, то найімовірніше їхні майбутні вподобання також будуть подібними [13].

На відміну від контентної фільтрації (Content-Based Filtering), колаборативна фільтрація не потребує аналізу вмісту книг, бо вона спирається виключно на поведінкові дані користувачів.

COLLABORATIVE FILTERING

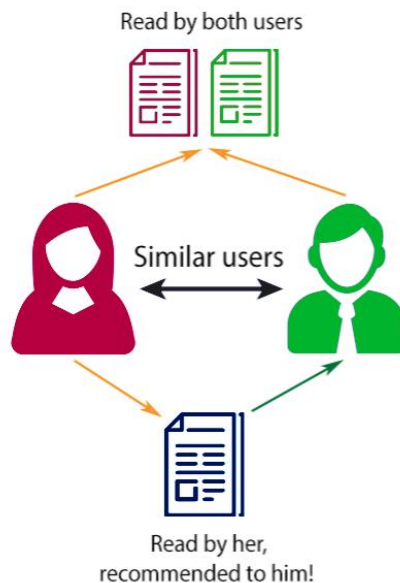


Рис. 1.5. - Схема Collaborative Filtering

Схема (рис. 1.5.) пояснюється таким чином: Є два користувачі, у них прочитаний перелік книг дуже подібний. Далі, є твір, який користувач А (зліва) прочитав, а користувач Б (справа) - цього не зробив. Наприклад, цією книгою буде Стівен Кінг “Кладовище домашніх тварин”. Тому, на основі того, що користувачі мають подібні смаки, цей твір буде порекомендовано користувачеві Б.

Колаборативна фільтрація ділиться на такі типи як: фільтрація, заснована на користувачах та фільтрація, заснована на схожості книг.

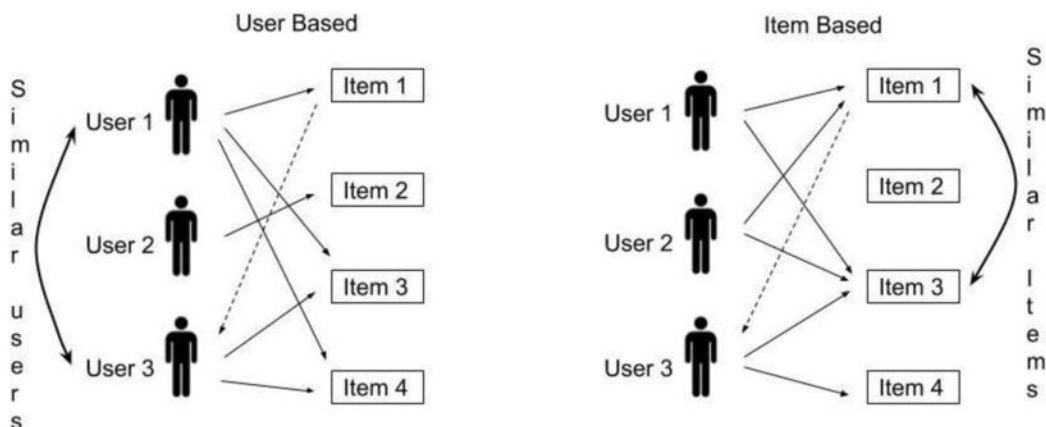


Рис. 1.6. - Схема User-based та Item-based Collaborative Filtering

На схемі (рис. 1.6.) видно, що основні відмінності цих двох типів заключаються в тому, що User-based орієнтується конкретно на схожих користувачів (User 1, 2, 3), а Item-based - на схожих книгах (Item 1, 2, 3, 4).

1. User-based Collaborative Filtering

Цей підхід знаходить користувачів зі схожими вподобаннями та використовує їхні оцінки для рекомендацій [14].

Алгоритм роботи наступний: кожен користувач оцінює за певною шкалою певну кількість книг. Таким чином, ми отримуємо матрицю рейтингів, де рядки - це користувачі, а стовпці - це книги. Далі, використовуються метрики, такі як косинусна подібність або коефіцієнт кореляції Пірсона. Виходить, що чим вищий коефіцієнт схожості між двома користувачами, тим більш імовірно, що у них спільні вподобання. Урешті-решт, книги, які високо оцінили подібні користувачі, але ще не оцінені цим користувачем, будуть відрекомендовані.

Тут можна навести приклад такий: Нехай буде користувач А та Б. Користувач А прочитав і високо оцінив “Дюну”, “1984”, “451 градусів за Фаренгейтом”. Користувач Б прочитав “Дюну” і “1984”, але ще не читав “451 градусів за Фаренгейтом”. Оскільки вподобання цих користувачів збігаються, система рекомендує “451 градусів за Фаренгейтом” користувачу Б.

Перевагами цього типу є звичайно те, що він враховує інтереси реальних користувачів, що дає більш персоналізовані рекомендації та є легким у реалізації на невеликих наборах даних.

Недоліки: якщо є велика кількість користувачів, порівняння між ними може бути обчислювально складним та нові користувачі не мають історії оцінок, через що їм складно давати рекомендації.

2. Item-based Collaborative Filtering

Замість пошуку схожих користувачів цей метод знаходить схожі книги на основі того, як їх оцінювали користувачі [15].

Алгоритм такий: спочатку, так само як у методі User-based Collaborative Filtering, відбувається формування матриці рейтингів. Після цього кроку,

використовуються метрики, такі як косинусна подібність або коефіцієнт кореляції. Наприклад, якщо багато користувачів оцінили книги А і Б схожим чином, вони вважаються подібними. У кінці, якщо користувач прочитав і високо оцінив книгу А, йому рекомендується книга Б, яка має високий рівень схожості з книгою А.

Щодо контретного прикладу, то: багато користувачів, які оцінили “Гаррі Поттера”, також високо оцінили “Персі Джексона”. Таким чином, користувачеві, який любить “Гаррі Поттера”, можна порекомендувати “Персі Джексона”.

Переваги тут наступні: уподобання користувачів можуть змінюватися, але схожість між книгами більш стабільна та не потрібно аналізувати кожного користувача окремо.

Недоліки звичайно це те, що якщо книгу ще ніхто не оцінив, її неможливо порівняти з іншими та якщо користувач хоче спробувати щось нове, система може рекомендувати занадто схожі книги.

Якщо порівнювати ці два методи за певними параметрами, то виходить наступне (табл. 1.4):

Таблиця 1.4

Порівняння методів Collaborative Filtering [16]

Характеристики	User-based	Item-based
Основний принцип	Знаходить схожих користувачів	Знаходить схожі книги
Врахування нових трендів	Враховує	Менше реагує на нові вподобання користувачів
Стійкість до змін	Менш стабільний (користувачі змінюють уподобання)	Стабільніший (книги залишаються схожими)
Проблема масштабованості	Є, чим більше користувачів, тим важче аналізувати	Менше впливає на продуктивність
Проблема старту новачкам	Важко рекомендувати новим користувачам	Важко рекомендувати нові книги

Hybrid Recommender System

Раніше було з'ясовано, що минулі два підходи мають свої певні недоліки:

1) Фільтрація на основі контенту може рекомендувати лише схожі книги, але може бути обмежена в різноманітності рекомендацій.

2) Колаборативна фільтрація, у свою чергу, може не працювати на нових або рідкісних книгах.

Тому часто, замість тих двох, використовують гібридний підхід. Цей метод поєднує у собі Content-Based (CBF) та Collaborative Filtering (CF) для покращення точності рекомендацій. Комбінація обох підходів дозволяє усунути слабкі сторони кожного з них. Використовуючи гібридний підхід, система може враховувати як властивості книг, так і вподобання користувачів, що підвищує якість рекомендацій [17].

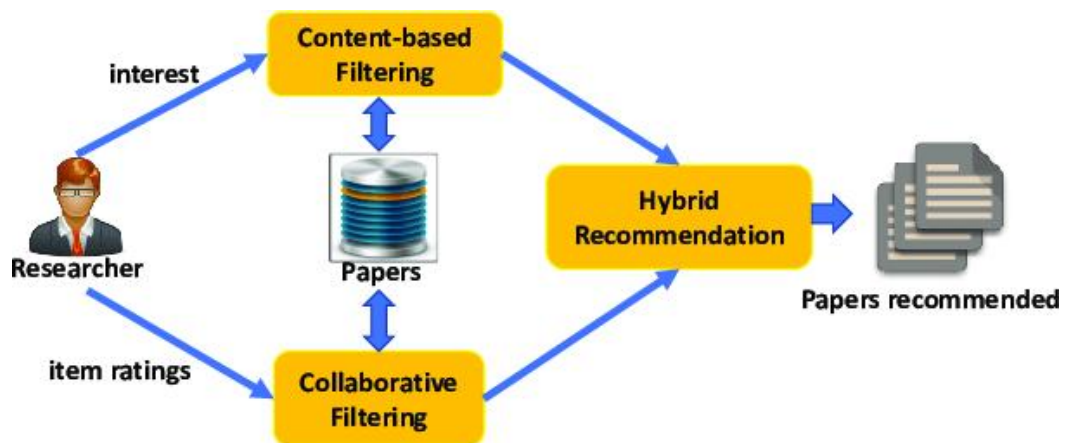


Рис. 1.7. - Схема Hybrid Recommender System

Розбираючи цю схему (рис.1.7.), можна детальніше побачити, як вона працює. Для зручності, щоб це відповідало темі мого проекту, під Researcher (дослідником) я буду мати на увазі користувача, а під Papers (паперами) - книги. Отже, є користувач у нього є певні інтереси: жанри, автори та інше, вони використовуються в фільтрації на основі контенту. Також, він часто ставить рейтинги книгам, це застосовується в колаборативній фільтрації. Далі, це все об'єднується в один гібридний підхід, який усуває проблеми, які виникали раніше. Таким чином, ми отримуємо певні рекомендовані книги.

Більш конкретний приклад цього підходу такий: користувач читає книги жанру “детектив”. Контентна фільтрація визначає його вподобання та пропонує нові книги цього жанру. Колаборативна фільтрація знаходить користувачів зі схожими смаками та рекомендує книги, які вони високо оцінили. Система комбінує ці два підходи та показує користувачеві персоналізовані рекомендації.

Взагалі, гібридні рекомендаційні системи можуть комбінувати CBF та CF різними способами.

У **Weighted Hybrid** обидва методи працюють одночасно, а їхні результати поєднуються з певними вагами [18]. Наприклад, якщо система бере 60% інформації з контентного аналізу і 40% - з колаборативної фільтрації, то, знаючи, що користувач часто читає фентезі, а люди з подібними вподобаннями читають “Гаррі Поттера”, вона може запропонувати “Персі Джексона”.

У **Cascading Hybrid** спочатку застосовується один метод, а потім інший. Зазвичай система спершу відбирає книги за контентною фільтрацією, а вже з цього звуженого списку обирає ті, що найпопулярніші серед схожих користувачів [19].

У **Feature Augmentation** один метод підсилює інший. Спершу контентна фільтрація формує детальний профіль користувача за жанрами, авторами та іншими параметрами, а потім цей профіль використовується в колаборативній фільтрації, щоб точніше підібрати подібних користувачів [20].

У **Switching Hybrid** система сама вибирає, який метод використати. Якщо в користувача вже накопичено багато оцінок, активується колаборативна фільтрація, а якщо він новий і даних мало, то використовується контентний підхід [21].

У **Model-Based Hybrid** обидва методи інтегруються в єдину модель, наприклад у нейронну мережу. У такому випадку глибокі моделі можуть

одночасно аналізувати характеристики книг і взаємодії користувачів, забезпечуючи точніші рекомендації [22].

Підсумовуючи все раніше перелічене, тепер ми можемо порівняти переваги та недоліки трьох основних підходів (табл. 1.5).

Таблиця 1.5

Порівняння переваг та недоліків розглянутих підходів

Підходи	Переваги	Недоліки
Content-Based Filtering	Добре працює для нових користувачів, дає пояснювані рекомендації	Вузьке коло рекомендацій, не враховує популярність
Collaborative Filtering	Враховує вподобання інших користувачів, відкриває новий контент	Проблема старту новачків, потребує великої кількості даних
Hybrid Recommender System	Поєднує переваги обох методів, забезпечує різноманітні рекомендації	Високі обчислювальні витрати, складність реалізації

Таким чином, аналіз трьох підходів показує, що гібридна система є найбільш кращим рішенням для створення точних і унікальних рекомендацій.

Висновки до розділу 1

У першому розділі розглянуто основні завдання системи рекомендацій книг та проаналізовано три популярні веб-сервіси. Дослідження показало, що сучасні рекомендаційні системи використовують контентну та колаборативну фільтрацію, але кожен із розглянутих сервісів має свої обмеження, наприклад орієнтацію на популярні книги або складність інтерфейсу.

Отримані результати підтверджують необхідність створення власної системи рекомендацій, яка забезпечуватиме точні, персоналізовані та адаптивні рекомендації для користувачів.

РОЗДІЛ 2. РОЗРОБКА ТА ПРОЕКТУВАННЯ МОДЕЛІ РЕКОМЕНДАЦІЙ

2.1. Підготовка та аналіз даних

У межах даного дослідження використовуються дані про книги та оцінки користувачів. Було вирішено створити власну базу даних. Основна причина цього полягає в тому, що не існувало відкритого набору даних, який би повністю відповідав потребам дослідження. Тому що необхідна була інформація про книги українською мовою, їхні жанри, авторів та оцінки користувачів. Створення власної бази дозволило контролювати якість і повноту даних, а також забезпечити можливість розширення інформації про книги і рейтинги у разі потреби.

Для збереження даних була обрана MongoDB, яка дозволяє гнучко працювати зі структурованими та напівструктурованими даними. Дані зберігаються в базі даних у двох таблицях: books та ratings.

У таблиці books зберігаються назва книг, автор, жанри, додатковий опис і інші поля (табл 2.1).

Таблиця 2.1

Основні поля таблиці books

Назва поля	Тип даних	Опис
_id	str	Внутрішній ідентифікатор книги у форматі MongoDB ObjectID
book_id	int	Внутрішній ідентифікатор книги
title	str	Назва книги
authors	str	Автори книги
genres	array	Масив жанрів
year	int	Рік першого видання книги
description	str	Короткий текстовий опис
image_url	str	Посилання на обкладинку книги

Приклад документа в таблиці books

```
{  
  "book_id": 1,  
  "title": "Кобзар",  
}
```

```

"authors": "Т. Шевченко",
"genres": [
  "Поезія"
],
"year": 1840,
"description": "Збірка поезій Т. Шевченка",
"image_url": "https://folio.com.ua/i/books/3d58b2b19fb8552b23ecfa85c6ababaa"
}

```

Це приклад того, як зберігається інформація про книгу в MongoDB. Кожен документ містить основні поля, які використовуються у системі рекомендацій.

У таблиці ratings містяться дані про оцінку, до якої книги і хто поставив її (табл 2.2).

Таблиця 2.2

Основні поля таблиці ratings

Назва поля	Тип даних	Опис
_id	str	Внутрішній ідентифікатор книги у форматі MongoDB ObjectID
user_id	int	Внутрішній ідентифікатор користувача
book_id	int	Внутрішній ідентифікатор книги
rating	int	Оцінки книг від 1 до 10
timestamp	datetime	Дата та час, коли була зроблена оцінка

Приклад документа в колекції ratings

```

{
  "_id": "51a05138-861d-4151-adf3-9cf2df245544",
  "user_id": 12,
  "book_id": 2,
  "rating": 8,
  "timestamp": "2024-10-28T14:46:00Z"
}

```

Кожен документ містить інформацію про користувача (user_id), книгу (book_id), значення оцінки (rating) та час, коли оцінка була залишена (timestamp).

На рис 2.1. представлена схема даних бази, що включає колекції books та ratings із зазначенням основних полів та типів даних. Детальнішу ER-діаграму з усіма таблицями бази та їхніми зв'язками наведено в підрозділі 2.3 (рис. 2.4).

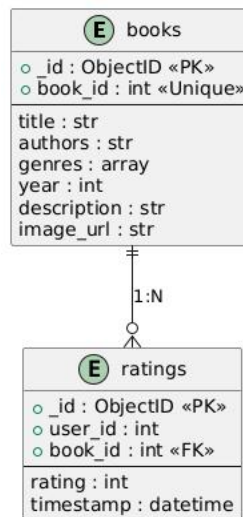


Рис. 2.1. - Схема даних бази книг та оцінок користувачів

Між таблицями `books` та `ratings` встановлено зв'язок один до багатьох (1:N). Тому що кожна книга може мати багато оцінок у таблиці `ratings`, тоді як кожна оцінка належить лише одній книзі. Така структура дозволяє аналізувати популярність книг і формувати персоналізовані рекомендації для користувачів.

Створена база даних має можливість розширюватись у майбутньому. Можна додавати нові книги та користувачів, включати додаткові дані, наприклад, мову книги, видавництво, серію чи інші характеристики, що дозволить підвищувати точність рекомендацій та масштабувати систему відповідно до нових потреб.

Під час побудови системи було виявлено, що алгоритм колаборативної фільтрації працює не зовсім коректно через проблему холодного старту. Це коли обсяг реальних даних про оцінки є недостатнім для формування точних рекомендацій [23]. Щоб вирішити цю проблему та щоб алгоритм добре працював було реалізовано функцію генерації синтетичних користувачів (`add_synthetic_ratings`). Це такі користувачі, записи яких є штучно згенерованими. Вони створюються для того, щоб збільшити кількість оцінок. Переваги такого підходу включають зменшення проблеми холодного старту, стабільніші рекомендації та можливість тестування алгоритмів на достатньо великій кількості даних. Недоліки полягають у тому, що синтетичні дані не є

реальними і можуть трохи спотворювати статистику популярності книг, тому їх використання вимагає обережності при інтерпретації результатів [24].

```
def add_synthetic_ratings(ratings_df, books_df, num_users=30):
    import random
    synthetic = []
    book_ids = books_df['book_id'].tolist()
    for uid in range(1000, 1000 + num_users):
        for bid in book_ids:
            if random.random() < 0.5: # 50% шанс, що користувач оцінить
книгу
                rating = random.randint(6, 10) # оцінки 6-10
                synthetic.append({"user_id": uid, "book_id": bid, "rating":
rating})
    if synthetic:
        ratings_df = pd.concat([ratings_df, pd.DataFrame(synthetic)],
ignore_index=True)
    return ratings_df
```

У цій функції створюються ід користувачів в певному діапазоні. Для кожного такого користувача випадковим чином обираються книги, які він оцінить. Ймовірність для кожної книги становить 50%. Далі випадково ставиться рейтинг від 6 до 10, щоб він виглядав більш реалістичнішим. Ці створені оцінки додаються разом з оцінками реальних користувачів до таблиці рейтингів.

Перед тим як реалізовувати самі алгоритми треба опрацювати дані, щоб вони не містили пропусків, непотрібних даних та іншого.

Для початку треба заповнити всі пропуски, які можуть бути. Для жанрів, якщо десь є NaN, то заповнюємо порожнім рядком. У базі даних всі жанри зберігаються у вигляді списку, а для побудови TF-IDF використовуються рядки, тому список перетворюємо на них:

```
books_df['genres_str'] = books_df.get('genres', '').fillna('').apply(
    lambda x: ' '.join(x) if isinstance(x, list) else str(x))
```

Для матриці user-item всі пропуски у рейтингах замінюємо на нулі:

```
user_item = ratings_df.pivot(index='user_id', columns='book_id',
values='rating').fillna(0)
```

Обов'язково ще треба виконати препроцесинг тексту. Це попередня обробка текстових даних, щоб підготувати їх для аналізу та побудови моделі [25].

Виконаємо перетворення тексту у нижній регістр. Наприклад, замість “Лісова пісня” буде “лісова пісня”. Також видаляємо всі знаки пунктуації, наприклад крапка, кома і інші.

```
def preprocess(text):  
    text = str(text).lower()  
    text = re.sub(r'^\w\s]', '', text)  
    return text
```

Також в назвах книг є стоп слова, такі як “він” чи “по” та інші, які не мають сенсового навантаження. Вони будуть ігноруватись при побудові TF-IDF. Оскільки в базі даних всі назви книг зберігаються українською мовою, то створюється список з українських стоп слів. Такий список дозволяє зосередитися на словах, що несуть інформацію про жанр, автора чи сюжет. Таким чином, це підвищує точність побудови профілів книг та користувачів.

```
uk_stopwords = [  
    "і", "в", "не", "на", "що", "як", "з", "до", "по", "за", "у", "та", "для",  
    "від", "це", "а", "але", "ви", "вона", "вони", "ми", "його", "її", "ім"  
]
```

Перед побудовою TF-IDF векторів, всі важливі дані - назва, автори та жанри об'єднуємо в один рядок. Вони всі проходять через функцію preprocess.

```
books_df['content'] = books_df.get('authors', '').apply(preprocess) + ' ' + \  
    books_df.get('title', '').apply(preprocess) + ' ' + \  
    books_df['genres_str'].apply(preprocess)
```

2.2. Розробка моделі рекомендацій

У роботі для надання рекомендацій використовується гібридний метод, який поєднує в собі контентно-орієнтований та колаборативний підходи. Такий підхід дозволяє максимально компенсувати недоліки кожного з алгоритмів окремо. Content-Based Filtering ефективний для нових книг, тоді як Collaborative Filtering добре працює для користувачів з історією оцінок. Гібридна модель дозволяє отримати більш точні та персоналізовані рекомендації, поєднуючи сильні сторони обох підходів.

Content-Based Filtering формує свої рекомендації на основі характеристик книги, а саме авторів, назви та жанрів. Цей метод добре працює для нових книг, які ще не мають рейтингів, і дозволяє будувати

рекомендації на основі змісту книги та ключових ознак жанру чи стилю автора.

Спочатку, створюється TF-IDF матриця. Ми представляємо всі текстові дані у вигляді чисел, тобто все з тексту перетворюємо на числа (вектори). Кожен вектор показує, наскільки важливі окремі слова для конкретної книги. Це дозволяє визначати значущість термінів і виділяти ті, які найкраще описують жанр або тему. Перед побудовою матриці всі українські стоп-слова видаляємо, щоб уникнути часто вживаних, але смислово неважливих слів. У матриці TF-IDF важливі слова матимуть більший ваговий коефіцієнт, а неважливі - менший [26].

Для визначення схожості між профілем користувача та описами книг використовується косинусна міра. Її формула має вигляд:

$$\text{cosine}(A, B) = \frac{\sum_{i=1}^n A_i B_i}{\sqrt{\sum_{i=1}^n A_i^2 \times \sum_{i=1}^n B_i^2}}$$

де:

- A - це вектор книги
- B - це вектор користувача
- n - це кількість ознак (TF-IDF слів)

Косинусна міра чудово працює з розрідженими векторами, якими завжди є TF-IDF. Вона не залежить від абсолютної довжини тексту і не важливо книга на 10 чи на 300 слів. Ця міра одразу показує не наскільки тексти великі, а саме наскільки вони схожі.

Тому косинусна подібність це такий стандарт у системах контентної фільтрації.

Щоб продемонструвати логіку моделі, можна навести наступну структурну схему роботи:

1. Preprocessing даних - очищення тексту, нормалізація, видалення українських стоп-слів (всі назви та описи в БД зберігаються українською).

2. TF-IDF векторизація описів книг.

3. Формування профілю користувача через усереднення TF-IDF векторів книг, які він оцінив вище певного порогу.

4. Обчислення косинусної схожості між профілем користувача і всіма доступними книгами.

5. Сортуюння й формування списку рекомендацій.

Якщо оцінювати складність методу Content-Based, то TF-IDF побудова має:

$$O(n \cdot k)$$

Де

- n - кількість книг
- k - кількість унікальних слів після всієї очистки

Обчислення косинусної схожості без оптимізацій має:

$$O(n^2)$$

Але у реальних системах зазвичай використовують оптимізацію для розріджених даних, що значно зменшує час виконання.

Отже, створюємо TF-IDF матрицю наступним чином, використовуючи стоп слова.

```
tfidf = TfidfVectorizer(stop_words=uk_stopwords)
tfidf_matrix = tfidf.fit_transform(books_df['content'])
```

Щоб легко було знаходити до кожної книги свій вектор, створюється словник, де зберігаємо індекс книги і її вектор.

```
bookid_to_idx = pd.Series(books_df.index.values,
index=books_df['book_id']).to_dict()
```

Коли є готова матриця, то можна будувати рекомендацію. Створюємо функцію, яка рахує наскільки кожна книга схожа на улюблені книги користувача. Для початку перевіряємо, якщо користувач не читав ніяких книг, то у нього нема смаку, тому повертаємо порожній список, бо неможливо визначити його смакові вподобання.

```
if not user_books:
    return {}
```

Далі створюємо порожній вектор користувача, щоб зберігати туди потім які слова, авторів, жанри він любить. Щоб отримати який у нього смак,

усереднюємо вектори всіх прочитаних книг. А саме беремо TF-IDF вектор кожної прочитаної книги, додаємо до вектора користувача і усереднюємо ($\text{user_vec} \neq \text{count}$).

```
user_vec = np.zeros(tfidf_matrix.shape[1])
count = 0
for book_id in user_books.keys():
    idx = bookid_to_idx.get(book_id)
    if idx is not None:
        user_vec += tfidf_matrix[idx].toarray().ravel()
        count += 1
if count > 0:
    user_vec /= count
```

Далі порівнюємо книги з профілем користувача, щоб знайти, які книги найбільш схожі з тими, які він любить. Якщо 1, то це дуже схоже, а якщо 0, то зовсім не схоже. Ці дані ми зберігаємо в словник з ідентифікатором книги. Обов'язково видаляємо книги, які користувач вже читав.

```
scores = cosine_similarity([user_vec], tfidf_matrix).ravel()
scores_map = {books_df.iloc[i]['book_id']: float(scores[i]) for i in
range(len(books_df))}
for b in user_books.keys():
    scores_map.pop(b, None)
```

У кінці треба нормалізувати, щоб найкраща книга мала значення 1, а всі інші були в діапазоні від 0 до 1. Це дозволяє легко порівнювати різні книги між собою. На цьому рекомендація готова.

```
max_score = max(scores_map.values(), default=1)
if max_score > 0:
    scores_map = {k: v / max_score for k, v in scores_map.items()}
```

Повний алгоритм знаходиться в додатку А.

Item-Based Collaborative Filtering створює рекомендації інакшим чином, а саме на основі поведінки інших користувачів. Тут буде використовуватись Item-Based підхід, а не User-Based. Це все пов'язано з тим, що на відміну від User-Based підходу, тут аналізується схожість між книгами, а не між самими користувачами. Це дозволяє визначати книги, які користувачам із подібними вподобаннями подобаються найбільше, навіть якщо новий користувач не має великої історії оцінок. Item-Based підхід особливо ефективний у системах з великою кількістю користувачів і книг, оскільки схожість між книгами змінюється повільніше, ніж схожість між користувачами.

Було обрано саме Item-Based Collaborative Filtering, а не User-Based. Тому що item-item стабільніший. Книги змінюються рідше, ніж користувачі. Матриця “книга-книга” менш динамічна, краще кешується. Для нових користувачів item-item працює краще завдяки тому, що рекомендації будуються через вже існуючі зв’язки між книгами. А user-user вимагає дуже багато даних і є чутливим до “шумних” користувачів. Тому item-item підхід дає стабільніші та зрозуміліші зв’язки між об’єктами.

Для початку будемо також матрицю схожості між книгами, але вже через рейтинги інших користувачів [27]. Тому першим кроком є підготовка рейтингових даних. Якщо рейтингів немає ніяких, то повертаємо пусте, оскільки рекомендації неможливо сформулювати без даних.

```
if ratings_df.empty:  
    return pd.DataFrame()
```

Далі групуємо за індифікатором користувача та книги. Якщо користувач поставив декілька оцінок одній книзі, то беремо середнє значення. Це дозволяє уникнути спотворення матриці через дублікати оцінок.

```
ratings_df = ratings_df.groupby(['user_id', 'book_id'],  
                                as_index=False)['rating'].mean()
```

Після цього формуємо User-Item матрицю, де рядки - це користувачі, колонки - книги, а значення - рейтинг. Якщо книгу користувач не оцінював, то значення буде 0. Для економії пам’яті переводимо цю матрицю в розрідженну, збільшуючи нулі.

```
user_item = ratings_df.pivot(index='user_id', columns='book_id',  
                              values='rating').fillna(0)  
sparse = csr_matrix(user_item.values)
```

Як і в контентно-орієнтованому підході, обчислюємо схожість книг, але за поведінкою користувачів. Для цього використовується косинусна схожість, яка показує наскільки схожі книги за оцінками користувачів. Результат зберігається у DataFrame, де рядки та колонки відповідають book_id.

```
item_sim = cosine_similarity(sparse.T)  
item_similarity_df = pd.DataFrame(item_sim, index=user_item.columns,  
                                  columns=user_item.columns)
```

Формуємо рекомендацію. Створюємо словник для сумування всіх оцінок. Для кожної оціненої книги береться рядок її схожості. Для кожної іншої книги рахується вагова сума (схожість * оцінка), тобто якщо книга дуже схожа і користувач високо її оцінив, то інша книга отримує великий бал.

```
scores = defaultdict(float)
for book_id, rating in user_books.items():
    if book_id not in item_similarity_df.index:
        continue
    sims = item_similarity_df.loc[book_id]
    for bid, sim in sims.items():
        if bid not in user_books:
            scores[bid] += sim * rating
```

У кінці, так само як у минулому підході, видаляємо книги, які користувач вже оцінював, щоб уникнути повторних рекомендацій. Нормалізуємо все до 0 та 1, що дозволяє порівнювати їх між різними книгами. Повний алгоритм знаходиться в додатку Б.

Однак в системі ці алгоритми не окремо використовуються один від одного, а вони поєднуються в гібридному підході. Тому що кожен із методів має свої недоліки і їхнє комбінування дозволяє компенсувати слабкі сторони один одного. У таблиці 2.3 наведено основні проблеми, властиві кожному типу алгоритму.

Таблиця 2.3

Недоліки рекомендаційних алгоритмів [28]

Проблема	Content-Based	Item-Based Collaborative
Холодний старт користувача	не працює	працює
Холодний старт книги	працює	не працює
Сильна залежність від тексту	є	не залежить від тексту
Потреба в багатьох оцінках	не потребує	потребує

Як видно з таблиці, кожен алгоритм неповний. Наприклад, Content-Based добре працює з новими книгами, але не здатний давати рекомендації новим користувачам, що не прочитали жодної книги. У свою чергу, Item-Based Collaborative чудово працює з новими користувачами, але зовсім не може рекомендувати книги, які не мають історії оцінок.

Використання гібридного алгоритму компенсує ці недоліки, бо поєднує в собі сильні сторони обох методів. Якщо коротко описувати, то алгоритм бере два набори оцінок, які ми отримали при формуванні результатів у минулих методах і об'єднуємо їх в єдиний. Однак розберемо більш детально.

Для початку знаходяться всі книги, які мають хоч якісь оцінки хоча б в одному з методів. Це необхідно, тому що книга може бути або в Content-Based, або в Item-Based Collaborative, або в обох, але для рекомендацій нам треба всі. У результаті утворюється множина всіх унікальних книжкових id.

```
all_books = set(cb_scores.keys()).union(cf_scores.keys())
```

Далі для кожної книги обчислюється гібридна оцінка. Її формула наступна:

$$\text{hybrid} = \alpha * \text{cb_scores} + (1 - \alpha) * \text{cf_scores}$$

де:

- α (або alpha) - це коефіцієнт ваги Content-Based
- `cb_scores` - це оцінка книги Content-Based
- $(1 - \alpha)$ - це вага Item-Based Collaborative
- `cf_scores` - це оцінка Item-Based Collaborative

Наприклад, alpha рівне 0.5, то ці два алгоритми важать однаково. Якщо alpha рівне 0.7, то Content-Based важить більше. Якщо alpha рівне 0.2, то Item-Based Collaborative - більше. За замовчуванням в алгоритмі використовується 0.5, тобто 50 на 50.

Додатково в коді враховується, що якщо книги немає ні в одному зі словників. У такому разі для неї автоматично використовується значення 0, щоб уникнути помилок.

```
hybrid_scores = {b: alpha * cb_scores.get(b, 0) + (1-alpha) * cf_scores.get(b, 0) for b in all_books}
```

Тобто система використовує Weighted Hybrid (зважений гібрид). Він є одним з найпоширеніших типів гібридних рекомендаційних систем. У цьому алгоритмі Content-Based та Item-Based формують свій список оцінок, а кінцевий результат - це вже їхня зважена сума. Це дозволяє компенсувати

недоліки кожного методу, керувати впливом алгоритмів через параметр α та гнучко налаштовувати модель під різні сценарії.

Коли всі гібридні оцінки обчислені, залишається відібрати найкращі рекомендації. Для цього список рекомендацій сортується за балами у спадаючому порядку, після беремо тільки перші книги, за замовчуванням 10, , але це число може бути змінене залежно від потреб.

```
top_books = sorted(hybrid_scores.items(), key=lambda x: x[1],  
reverse=True)[:top_n]
```

У результаті маємо найкращі рекомендації, сформовані на основі поєднання контекстної подібності книг та колективних вподобань інших читачів.

Таким чином, гібридний метод забезпечує більшу стабільність і універсальність рекомендацій у порівнянні з окремим використанням кожного з алгоритмів.

Повний алгоритм знаходиться в додатку В.

2.3. Проектування веб-додатку

1) Архітектура системи та взаємодія модулів

Додаток складається з трьох окремих, але тісно пов'язаних між собою компонентів, а саме клієнтської, серверної частин та рекомендаційної системи. Кожен модуль виконує власну функцію, однак вони не є самостійними сервісами, бо усі частини утворюють єдину логічну систему.

Frontend (клієнт) побудований на основі React та реалізований у вигляді Single Page Application. Цей підхід дозволяє завантажувати сторінку лише один раз, а подальше оновлення інтерфейсу відбувається динамічно через взаємодію з сервером без повного перезавантаження вікна браузера [29].

На клієнті реалізований функціонал візуалізації книжкового каталогу, профілю користувача та рекомендацій. Також, тут є багато API-запитів, які можна відправляти на backend, наприклад реєстрація, пошук, оцінювання книги і так далі. Реалізована обробка відповідей серверної частини та

відображення результатів користувачу. На frontend немає взаємодії з базою даних, бо всі операції здійснюються через http API.

Backend (сервер) - це центральна частина системи. Його задачами є приймати запити з клієнту та формувати відповіді, здійснювати CRUD-операції з базою даних MongoDB, виконувати авторизацію користувачів, звертатися до рекомендаційної системи при необхідності сформувати персоналізовані рекомендації.

Сервер виступає посередником між базою даних і рекомендатором, тому рекомендаційна система не має прямого доступу до таблиць MongoDB.

Рекомендаційна система - це окремий модуль, який відповідає за обчислення рекомендацій. Тут відбувається обробка отриманих від серверу даних, а саме книг та рейтингів. Також, тут відбувається робота з синтетичними користувачами. Реалізовано всі рекомендаційні алгоритми - Content-Based filtering, Item-Based Collaborative Filtering та Hybrid. Їхні результати повертаються на сервер у форматі JSON. Ця система не читає дані з бази напряду, щоб уникнути дублювання логіки та забезпечити цілісність системи.

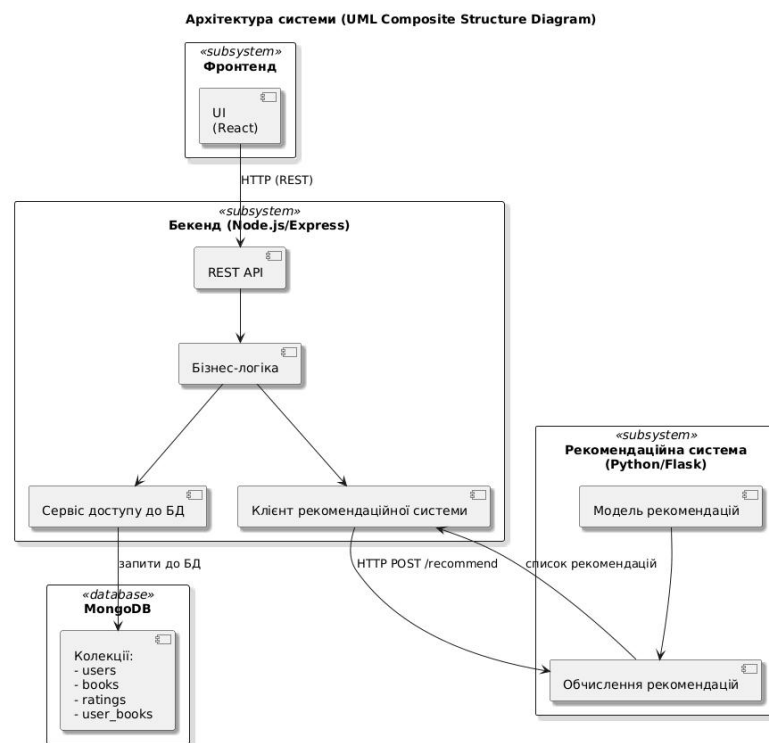


Рис. 2.2. - Діаграма компонентів

Діаграма рис. 2.2. відображає основні підсистеми програми та їх взаємодію [30]. Фронтенд надсилає HTTP-запити до бекенду. Бекенд обробляє запити, взаємодіє з MongoDB та передає дані до рекомендаційної системи. Рекомендаційна система обчислює персоналізовані рекомендації та повертає їх на сервер. MongoDB зберігає дані користувачів, книги, рейтинги та персональні списки.

2) Діаграма потоку даних

Діаграма потоку даних показує як інформація рухається між користувачем, фронтендом, бекендом, базою даних та рекомендаційною системою. Вони демонструють, які дані передаються, в якому порядку відбувається обробка запитів та формування персоналізованих рекомендацій [31].

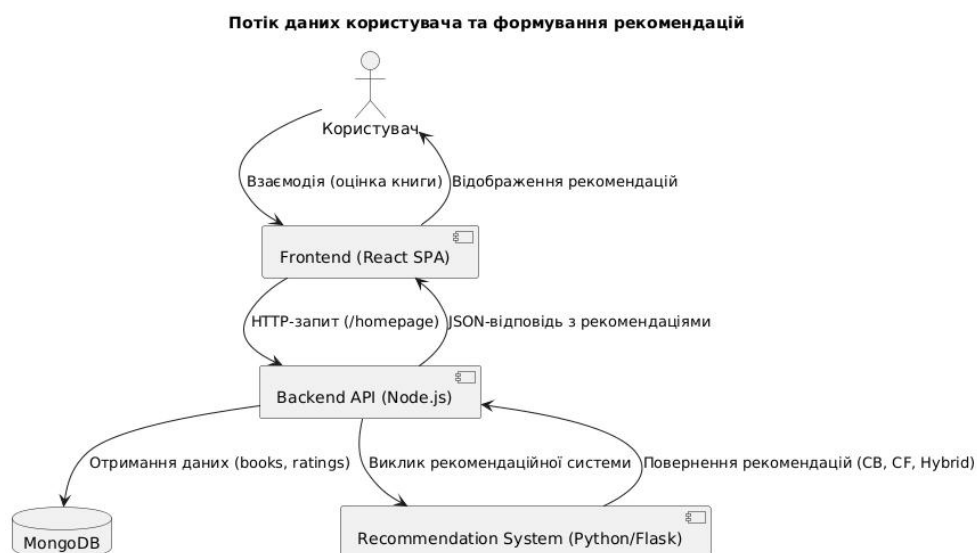


Рис. 2.3. - Діаграма потоку даних

На рис. 2.3. все працює наступним чином. Користувач взаємодіє з фронтендом, а фронтенд надсилає HTTP-запити до бекенду. Бекенд отримує дані з MongoDB і викликає рекомендаційну систему при необхідності. Ця система обчислює рекомендації та повертає результат. Бекенд відправляє ці рекомендації фронтенду, а він відображає результати користувачу.

3) Структура бази даних

ER-діаграма відображає структуру бази даних MongoDB і основні колекції, що використовуються в системі [32].

ER-діаграма бази даних

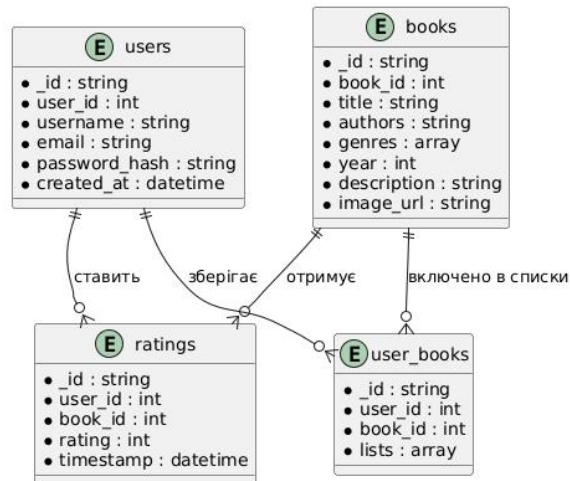


Рис. 2.4. - ER-діаграма бази даних

На діаграмі рис. 2.4. відображено чотири таблиці бази:

- users - зберігає інформацію про користувачів, його логін, електронну пошту, хеш пароля та дату створення облікового запису.
- books - містить дані про книги: назву, авторів, жанри, рік видання, опис та посилання на зображення.
- ratings - зберігає оцінки користувачів книгам, має зв'язок 1:N з користувачами і книгами.
- user_books - персональні списки користувачів, у яких зберігаються обрані книги.

4) Архітектурний стиль

Система реалізована за принципом модульного моноліту. Це такий стиль, коли система є єдиним логічним цілим, але складається з чітко виділених модулів, які виконують окремі функції [33].

Ця система не мікросервісна, бо хоча фронтенд, бекенд та рекомендаційна система працюють на різних портах, але вони не мають власних баз даних, не працюють незалежно, не масштабуються окремо, жорстко пов'язані через API-передачу даних. Рекомендаційний модуль є обчислювальним, а не самостійним сервісом у архітектурному сенсі [34].

Це також не класичний моноліт, бо компоненти фізично розділені. Вони запускаються окремо та взаємодіють виключно через HTTP. Тому

система є модульним монолітом, бо логічно модульна, але технічно є цілісною [35].

Модульний моноліт спрощує підтримку системи, оскільки всі компоненти знаходяться в межах однієї логічної системи і не потребують складної координації між незалежними сервісами. Взаємодія між модулями є прозорою, бо модулі обмінюються даними чітко визначеними API-запитами, що робить систему більш передбачуваною та легкою для налагодження. Важливою перевагою є можливість майбутнього масштабування. Рекомендаційну систему можна буде виділити в окремий мікросервіс без значного переписування логіки решти системи.

Однак він має свої обмеження. Масштабувати систему окремими модулями складно, бо всі компоненти працюють як єдине ціле. Бекенд залишається центральним елементом, тому через це при великому навантаженні можуть виникати затримки. Зі збільшенням числа користувачів та обсягу даних зростає складність підтримки. Можливо доведеться оптимізувати код та ресурси для стабільної роботи [36].

Висновки до розділу 2

У другому розділі було проведено підготовку та аналіз даних, а саме заповнення пропусків, препроцесинг тексту, обробку жанрів та використання синтетичних користувачів для покращення роботи алгоритмів при проблемі холодного старту. Було розроблено гібридну модель рекомендацій, яка поєднує Content-Based та Item-Based Collaborative Filtering, що дозволяє компенсувати недоліки кожного підходу та підвищити точність рекомендацій. Спроековано веб-додаток у вигляді модульного моноліту, де фронтенд, бекенд та модуль рекомендацій взаємодіють через чітко визначені API, що забезпечує цілісність системи, прозору взаємодію модулів та можливість подальшого масштабування. Структура бази даних та архітектура системи дозволяють ефективно обробляти дані, формувати персоналізовані рекомендації та підтримувати подальший розвиток системи.

РОЗДІЛ 3. ОПИС ПРОГРАМНОГО ПРОДУКТУ

3.1. Графічний інтерфейс

Графічний інтерфейс реалізовано на основі фреймворку React, що дозволяє створювати динамічні елементи інтерфейсу та швидко оновлювати контент на сторінці без її перезавантаження.

Інтерфейс складається з кількох основних сторінок, кожна з яких відповідає окремому функціональному блоку системи:

1) Реєстрація / Авторизація

На рис. 3.1. розташована форма для входу або реєстрації. Вона містить поля для введення логіна та пароля, а також кнопку для надсилання даних на сервер. У випадку, якщо користувач не має облікового запису, він може перейти до форми реєстрації, де створюється новий профіль. Сторінка реалізована таким чином, щоб мінімізувати кількість дій необхідних для входу та забезпечити швидкий доступ до персоналізованих рекомендацій. Усі введені дані проходять базову валідацію, що підвищує безпеку та зменшує кількість можливих помилок.

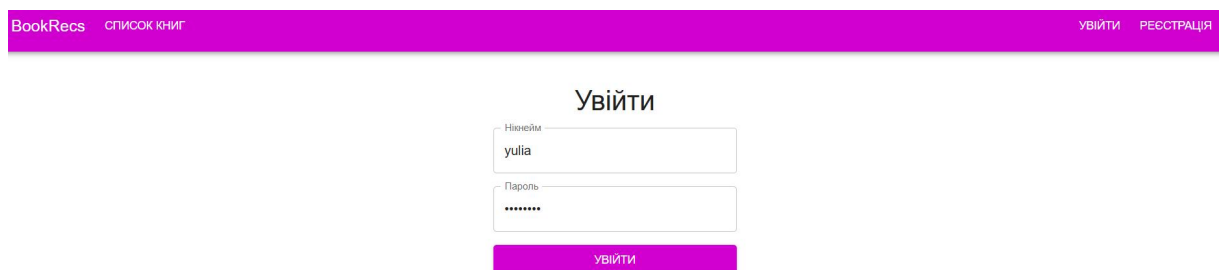


Рис. 3.1. - Сторінка Увійти

2) Головна сторінка

Вона відображає списки книг, містить базову навігацію та може показувати популярні або основні добірки незалежно від рекомендаційної системи.

Якщо відвідувач не авторизований, система показує підбірку з 12 популярних книг, сформовану на основі загального рейтингу бібліотеки (рис. 3.2.). Це дозволяє навіть новим користувачам отримати початковий огляд контенту.

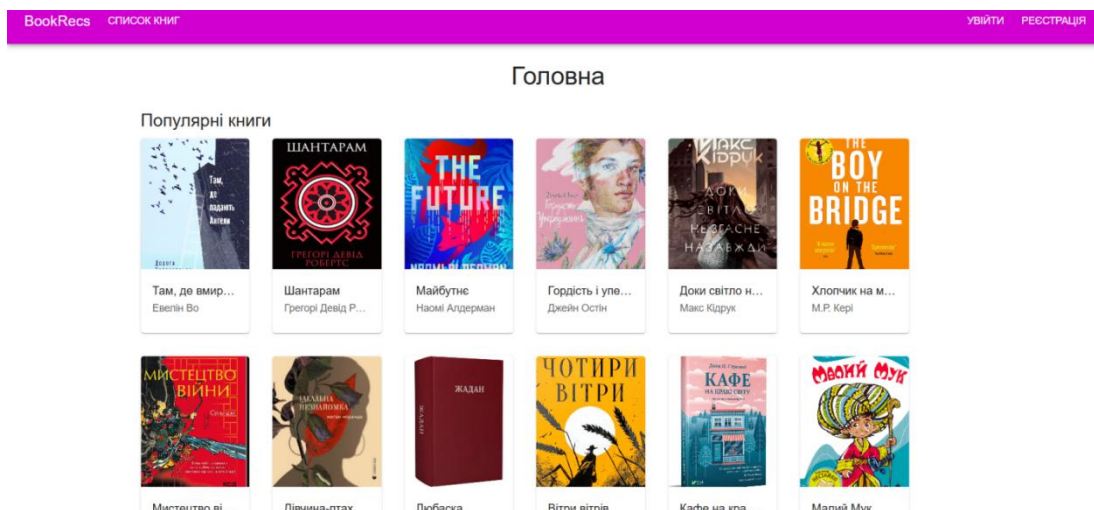


Рис. 3.2. - Головна сторінка для не авторизованих

Після авторизації зміст головної сторінки змінюється: замість універсальних добірок з'являються персональні рекомендації, побудовані алгоритмом на основі оцінок і вподобань конкретного користувача (рис. 3.3.). Таким чином, головна сторінка адаптується до кожного користувача, забезпечуючи індивідуальний досвід.

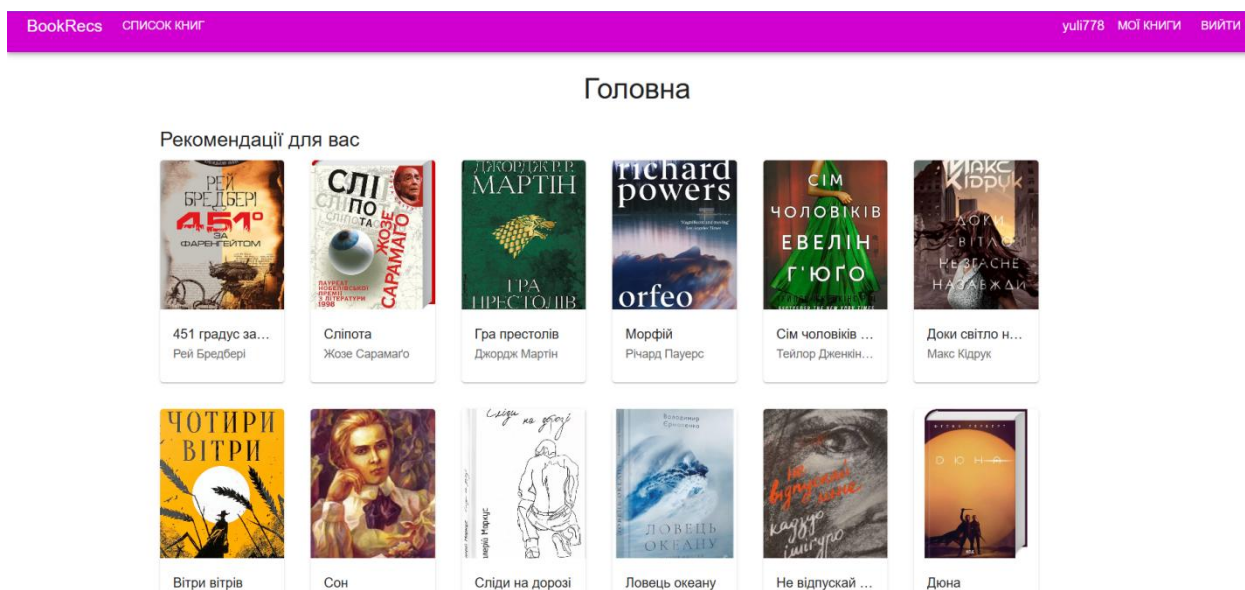


Рис. 3.3. - Головна сторінка для зареєстрованих

3) Усі книги та пошук

Сторінка Усі книги (рис. 3.4.) містить повний набір книг, що зберігаються у базі даних. Вона підтримує пагінацію, що дозволяє зручно переглядати велику кількість записів без перевантаження інтерфейсу. Кожна

книга відображається у вигляді картки з обкладинкою, назвою та автором, що забезпечує швидке візуальне сприйняття.

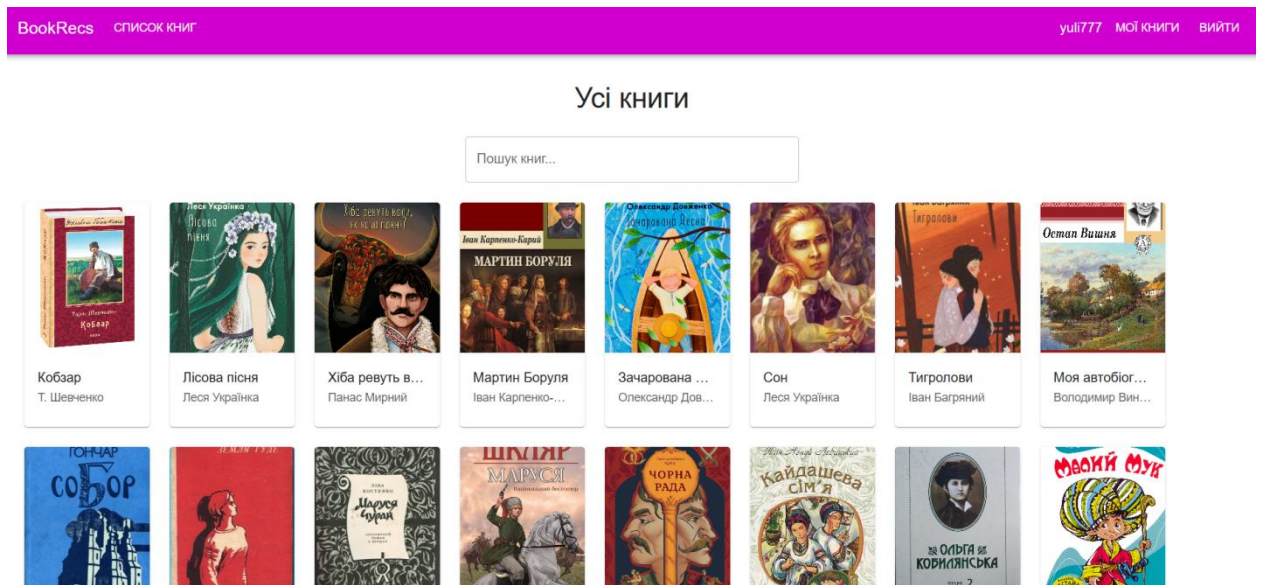


Рис. 3.4. - Сторінка Усі книги

Також сторінка містить інтерактивний пошук за назвою книги. Результати оновлюються одразу після введення тексту, що створює ефект швидкого пошуку та значно полегшує навігацію серед великої кількості елементів (рис. 3.5). Це особливо зручно для користувачів, які вже мають конкретний запит.

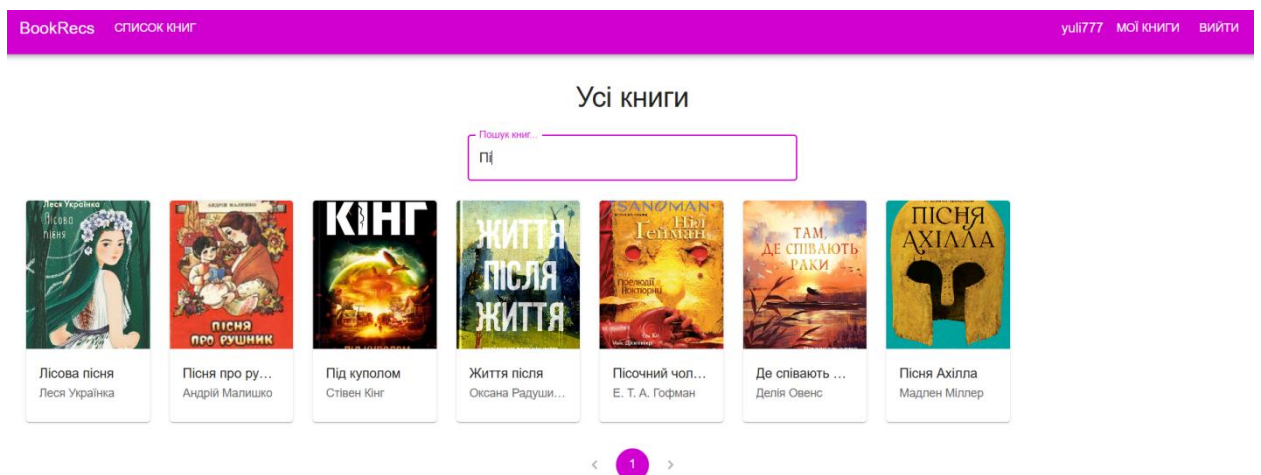


Рис. 3.5. - Результати пошуку

4) Сторінка книги

На сторінці окремої книги наведено детальну інформацію про неї, а саме назву, список авторів, жанри, рік видання, опис та обкладинку. Також тут відображається середній рейтинг книги серед усіх користувачів.

Якщо користувач не авторизований, він може лише переглядати інформацію, але йому недоступні можливості взаємодії, такі як виставлення оцінки або додавання книги до власних списків (рис. 3.6.).

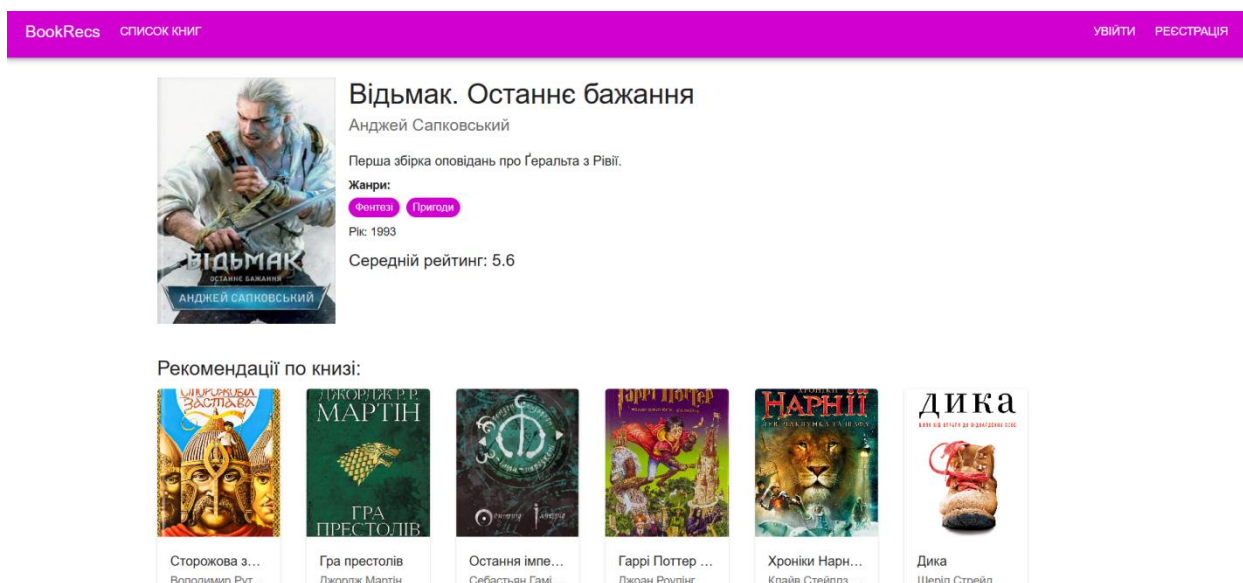


Рис. 3.6. - Сторінка книги для не зареєстрованого

У авторизованого користувача відображається рейтинг, який цей користувач ставив. Якщо користувач додавав книгу в свої списки, то будуть відображатись назви цих списків. Також цю книгу він може знову додати в новий або інші його списки (рис. 3.7.). Такий підхід створює персоналізовану взаємодію та розширює можливості користувача формувати власну колекцію.

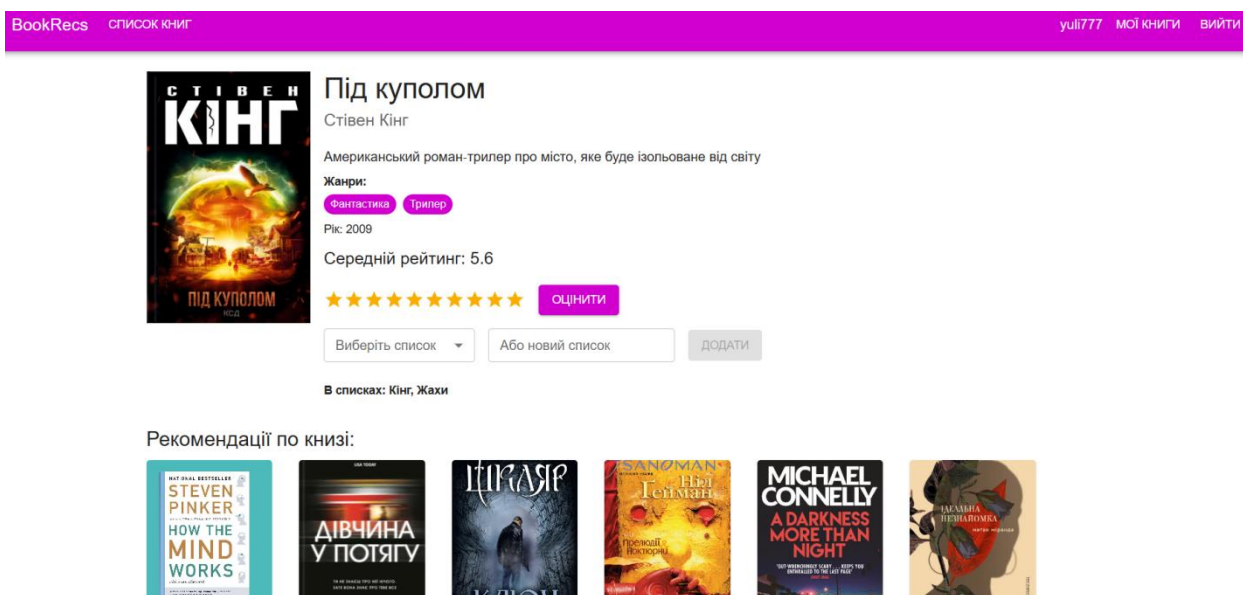


Рис. 3.7. - Сторінка книги для авторизованого

5) Мої книги

На сторінці Моїх книг можна переглянути свій список оцінених книг та власні створені списки. Вони розташовані на окремих двох вкладках.

У розділі оцінених книг (рис. 3.8.) користувач може побачити всі книжки, яким він залишив рейтинг. Відображаються обкладинка, назва та виставлена ним оцінка. Пагінація дозволяє легко переглядати великі добірки.

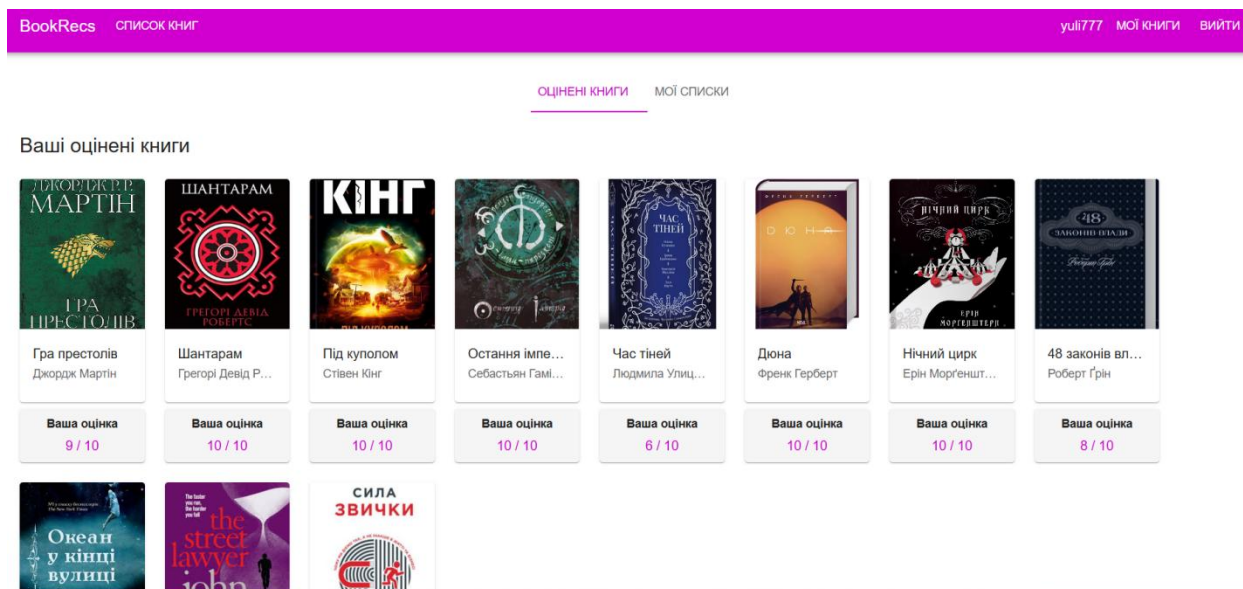


Рис. 3.8. - Сторінка Оцінених книг

У вкладці Мої списки (рис. 3.9.) користувач може переглядати власні тематичні добірки, наприклад “Кінг”, або “Жахи”, або “укр”. Тут представлені назва списку, кількість книг у ньому та кнопка для видалення списку. Це дозволяє організовувати власну колекцію у зручному форматі.

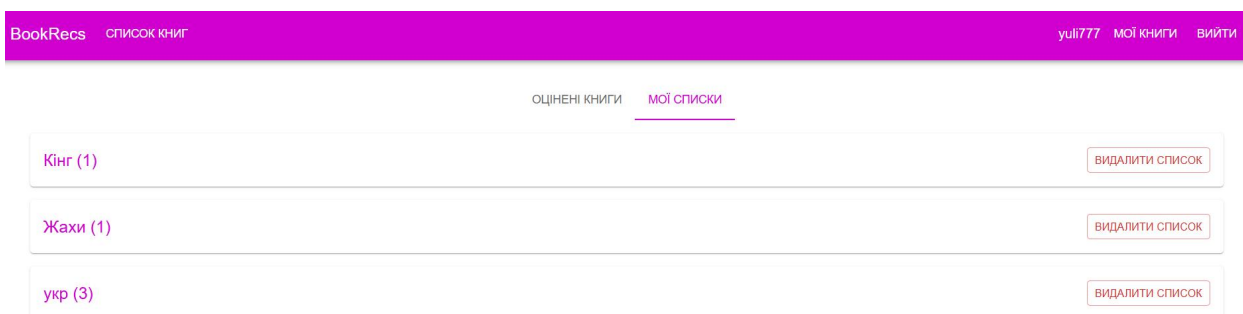


Рис. 3.9. - Сторінка Мої списки

6) Сторінка списку

На сторінці конкретного списку (рис. 3.10) відображається його назва, перелік книг, що до нього належать, а також кнопки керування. Користувач може видаляти книги зі списку або додавати нові. Така сторінка фактично є

інструментом персональної організації бібліотеки, де користувач структурує свої інтереси тематично або за власними критеріями.

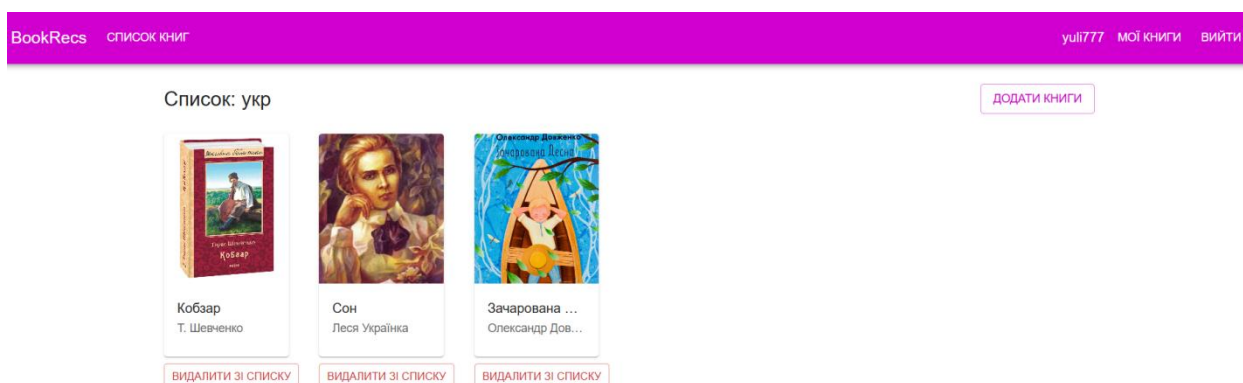


Рис. 3.10. - Сторінка списку

3.2. Відображення результатів

1. Рекомендація для користувача

Щоб коректно проаналізувати виведені рекомендації, передусім необхідно розглянути профіль користувача за списком оцінених ним книг, оскільки саме вони формують жанрову основу та поведінкові сигнали, на яких базується робота моделі.

Всього у користувача, якого ми аналізуємо, оцінено 11 книг (рис. 3.11.).

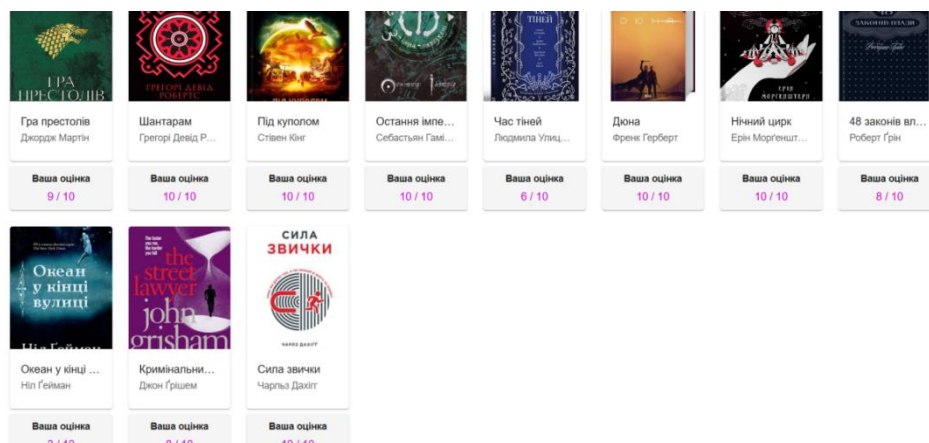


Рис. 3.11. - Список всіх оцінених книг

Таблиця 3.1

Основні улюблені жанри (мають 9-10)

Книга	Жанри	Оцінка
Гра престолів	фентезі, пригоди	9
Шантарам	роман, пригоди	10

Продовження таблиці 3.1

1	2	3
Під куполом	фантастика, трилер	10
Остання імперія	фентезі, пригоди	10
Дюна	наукова фантастика	10
Нічний цирк	фентезі, роман, містика	10
Сила звички	нон-фікшн, психологія	10

По кількості високо оцінених книг можна сказати, що фентезі та пригоди - це жанри, які найбільше подобаються користувачу. Є містика або магічний реалізм, тобто він любить твори подібні на Геймана. Користувач цінить жорсткі трилери та фантастику (Кінг) та наукову фантастику (Дюна), бо по 10 у рейтингу. Також тут є оцінка 10 на нон-фікшн про саморозвиток, тому це має свою вагу.

Непопулярні жанри вийшли історичні драми, бо книга “Час тіней” отримала лише 6, та містична психологічна проза “Океан у кінці дороги” - 3. Важливо це врахувати, бо система справедливо не рекомендує подібні твори.

Головна

Рекомендації для вас

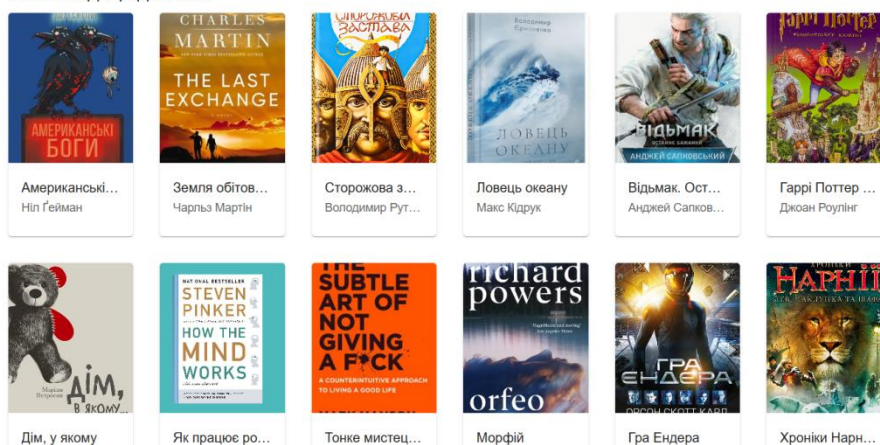


Рис. 3.12. - Список рекомендацій

Отже, користувач отримав наступні рекомендації (рис. 3.12.): “Американські боги” (Ніл Гейман), “Земля обітована”, “Сторожова застава”, “Ловець океану”, “Відьмак. Останнє бажання”, “Гаррі Поттер”, “Дім, у

якому”, “Як працює розум”, “Тонке мистецтво пофігізму”, “Морфій”, “Гра Ендера” та “Хроніки Нарнії”.

Додатково я виводжу логи (Додаток Г), щоб зрозуміти як модель визначила рекомендації для кожної книги. Вони приблизно мають ось такий вигляд (фрагмент):

Book 48 - Content-Based:

[(95, np.float64(0.38640776003242594)), (27, np.float64(0.08559178617529177)), (37, np.float64(0.08145305401422058))]

Book 48 - Collaborative:

[(38, 0.6273885280522581), (99, 0.6197506830096349), (83, 0.5610494542072252)]

У виведених логах видно, що у всіх книгах у топах з’являються: “Гра престолів” (id - 27), “Остання імперія” (37), “48 законів влади” (83), “Нічний цирк” (82), “Час тіней” (38), бо багато людей читають драми або сучасну прозу разом з фентезі, “Кримінальний роман” (99), бо трилери часто читають разом із Кінгом, та “Сила звички” (111).

Отже, а тепер проаналізуємо топ рекомендацій. “Американські боги” - це найсильніша рекомендація та абсолютно правильна. Тому що у користувача високі оцінки на містику, фентезі, а саме темне, які ідеальні під Геймана. У логах Content-Based та Item-Based Collaborative дають оцінку 0,94. Якщо аналізувати виведені логи, то виходить наступне. У Content-Based виводить, що книга 48:

- схожа на 95 (“Океан у кінці дороги - той самий автор”) - 0,386,
- схожа на 27 (“Гра престолів”) - 0,085,
- схожа на 37 (“Остання імперія”) - 0,081.

У Collaborative схожа на книги 38, 99, 83, вони мають середній вплив.

Через оцінку 10 для “Нічного цирку”, система бачить патерн “магічний реалізм, темне фентезі, Гейман-подібний тон”. Тому і вийшла оцінка 0,94.

Далі пропонують фентезі - “Відьмак”, “Хроніки Нарнії”, “Гаррі Поттер”, “Дім, у якому” та “Сторожова застава”. Це все Content-Based логіка,

бо вона узгоджена з оцінками на “Останню імперію”, “Гру престолів” та “Нічний цирк”.

У рекомендації є українська та світова пригодницька література, а саме “Земля обітована” та “Ловець океану”. Система видала ці книжки в першу десятку через спільні жанрові теги з “Шантарам” і “Сторожовою заставою”.

З наукової фантастики - “Гра Ендера” та “Морфій”. Це все через високу оцінку “Дюни”. По додатково виведеним логам видно, що book 71 (це Дюна) тягне sci-fi твори ближче.

“Сила звички” сильно вплинула на рекомендації, тому тут також є нон-фікшн, а саме “Як працює розум” та “Тонке мистецтво пофігізму”. Дивлячись по логам, то нон-фікшн порекомендувало, бо книга 116 (“Як працює розум”) має Content-Based з

- 111 (“Сила звички”) - 0,1966,
- 35 (“Під куполом”) - 0,1550,
- 83 (“48 законів влади”) - 0,0791.

Item-Based Collaborative зв’язок також сильний, бо

- 37 (“Остання імперія”) - 0,6412,
- 38 (“Час тіней”) - 0,5745,
- 35 (“Під куполом”) - 0.5560.

Тобто він бачить групи користувачів, які читають і нон-фікшн, і фентезі разом, що є типовою поведінкою на книжкових сервісах.

Отже, система виявила, що користувач - фанат фентезі, любить містичні та темні сюжети, йому подобаються епічні та масштабні світи, він іноді читає нон-фікшн із саморозвитку та не любить занадто ліричну та психологічну прозу. І рекомендації виводять таке саме. Content-Based дає комбінацію фентезі та sci-fi, Collaborative додає нон-фікшн та популярні книги у подібних читачів, а гібрид утримує баланс.

2. Рекомендація для користувача при холодному старті

Користувач аналізується той самий, оцінені книги - також ті самі. Його рекомендації на рис. 3.13.

Головна

Рекомендації для вас

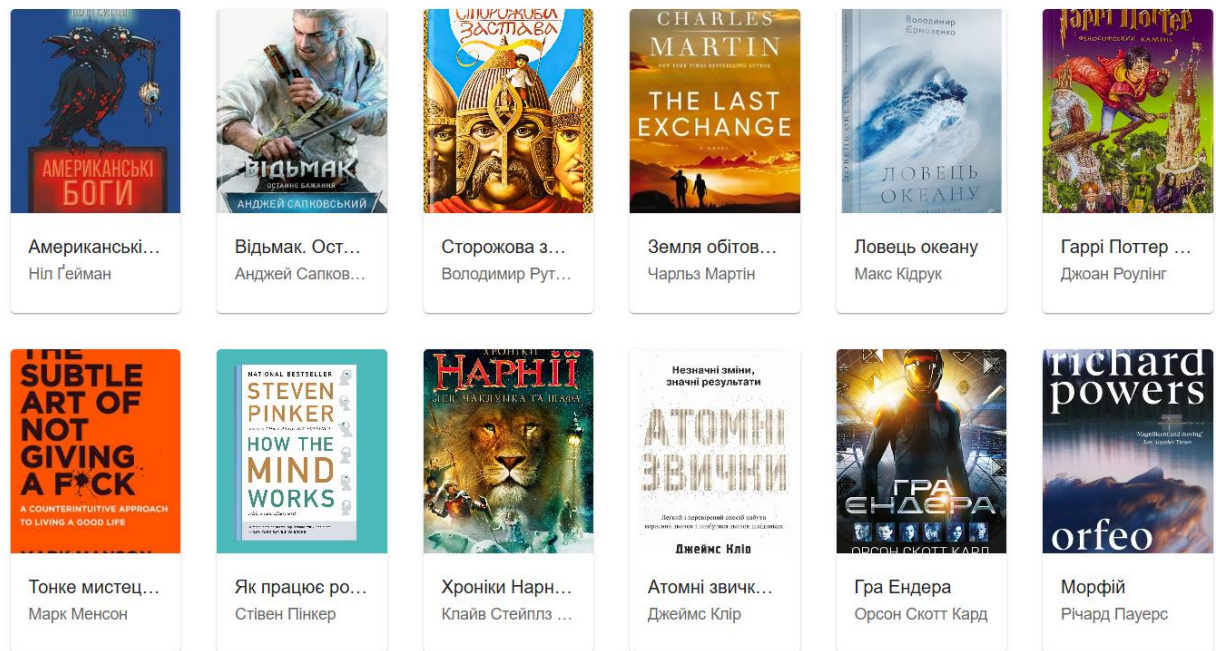


Рис. 3.13. - Список рекомендацій при холодному старті

У цьому прикладі сайт працює в умовах холодного старту, коли для деяких книг або навіть користувача загалом ще немає достатньо даних для Collaborative Filtering. Це добре видно у логах (Додаток Д), бо для всіх рекомендованих книг Collaborative повертає порожній список.

Book 48 - Collaborative: []

У рекомендаціях важать тільки жанри, ключові слова та опис книг. Кожна рекомендація будується через порівняння книги з тими, що користувач уже оцінив. За логами видно, що практично в кожному Content-Based топі присутні ті ж самі книги. Це означає, що саме вони визначають жанровий профіль користувача і формують схожість. Книга потрапляє в рекомендації, якщо вона близька до фентезі (27, 37), пригоди (28), sci-fi (71), нон-фікшн (111). Саме тому вийшов такий топ (рис. 3.13.).

Не дивлячись на холодний старт рекомендації виходять все одно правильні. Немає жодної рекомендації з жанрів, які користувач не любить.

У холодному старті іноді з'являються книги, які не потрапили у звичайний режим через відсутність даних Collaborative Filtering.

Книга “Атомні звички. Як набути добрих звичок і позбутися поганих” з’являється у холодному старті через Content-Based алгоритм. Тому що він бачить зв’язок із оціненими користувачем книгами у жанрах нон-фікшн та саморозвиток (“Сила звички”). У звичайному режимі ця книга не потрапляє до топу, бо Collaborative Filtering не підказує її (синтетичні користувачі або спільні оцінки не дали сигналу).

“Дім, у якому” у жанрах фентезі, драма потрапляє у звичайний режим завдяки Collaborative Filtering. Користувачі, що читають фентезі та пригоди, часто оцінювали цю книгу. У холодному старті Content-Based не дав їй високої ваги, бо жанрова схожість з оціненими книгами була слабшою.

3. Рекомендація для іншого користувача

Переглянемо список оцінених книг для цього користувача (рис. 3.14.). Всього оцінок - 5. Це дозволяє скласти базові уявлення про його смакові уподобання.

[ОЦІНЕНІ КНИГИ](#) [МОЇ СПИСКИ](#)

Ваші оцінені книги

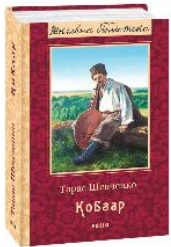
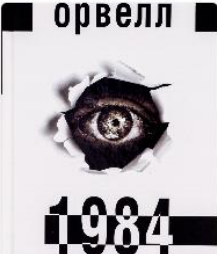
				
Кобзар Т. Шевченко	1984 Джордж Орвелл	Там, де вмир... Евелін Во	Майбутнє Наомі Алдерман	П'ять мов лю... Гері Чепмен
Ваша оцінка 9 / 10	Ваша оцінка 9 / 10	Ваша оцінка 7 / 10	Ваша оцінка 9 / 10	Ваша оцінка 8 / 10

Рис. 3.14. - Список оцінених книг

Список оцінених книг

Книга	Жанри	Оцінка
Кобзар	поезія	9
1984	роман, антиутопія	9
Там, де вмирають птахи	роман, сатира	7
Майбутнє	наукова фантастика, антиутопія	9
П'ять мов любові	психологія, стосунки	8

Користувач віддає перевагу класичній літературі та антиутопіям (оцінки 9 на “1984” та “Майбутнє”). Також, високі оцінки має поезія та психологія (“Кобзар” - 9, “П'ять мов любові” - 8). Вони теж формують значну частину смаку. Менш улюблені жанри - це трагікомічні романи (7 на “Там, де вмирають птахи”). Профіль цього користувача показує інтерес до класики, антиутопій, поезії та літератури про стосунки і психологію.

Його рекомендації вийшли наступні (рис. 3.15.):

Головна

Рекомендації для вас

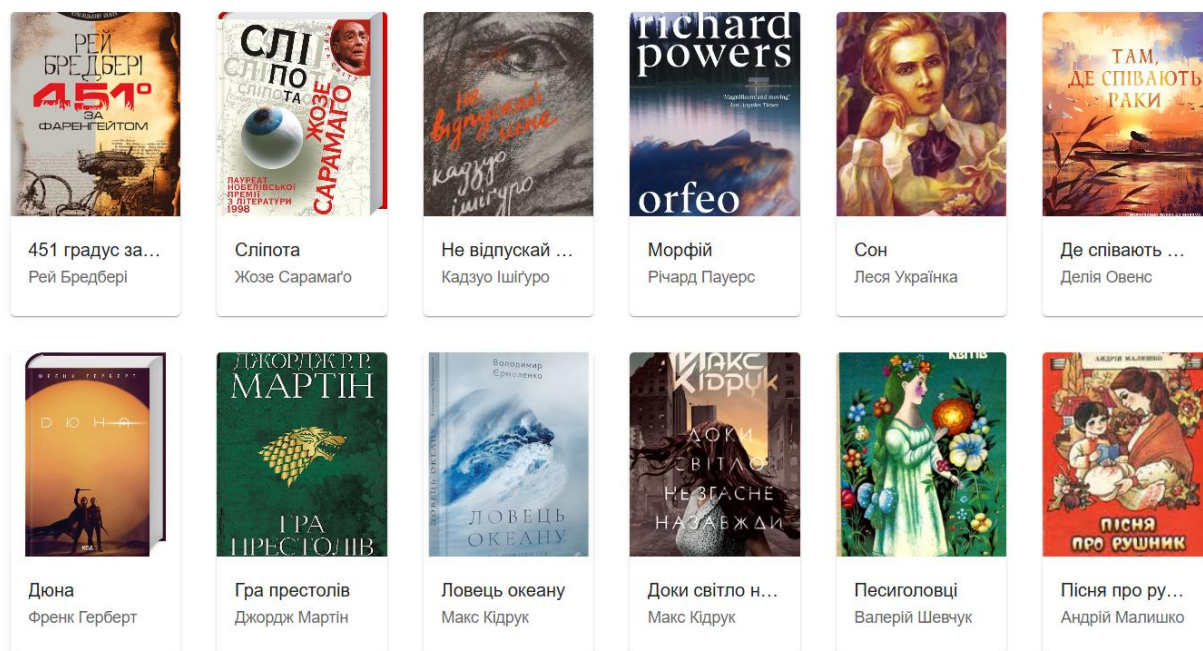


Рис. 3.15. - Список рекомендованих книг

Система коректно враховала жанрові переваги, бо через антиутопії (“1984”, “Майбутнє”) у рекомендаціях вивело “451 градус за Фаренгейтом”, “Сліпота”, “Не відпускай мене”. Поезія та класика формують рекомендації на українські твори: “Кобзар”, “Сон”, “Пісня про рушник”.

Отже, Content-Based з’єднав книги по жанровій та тематичній схожості, а Collaborative підтвердив популярні серед інших користувачів вибори. Наукова фантастика (“Дюна”) та пригодницькі мотиви (“Ловець океану”, “Доки світло не згасне назавжди”) потрапили завдяки змішаному впливу оцінених книг і поведінки інших користувачів.

4. Рекомендація до книги

У цьому пункті буде аналізуватись книга “Мандрівний замок Хаула” (рис. 3.16.). Це дозволяє оцінити рекомендації, побудовані не для користувача, а для конкретного твору.

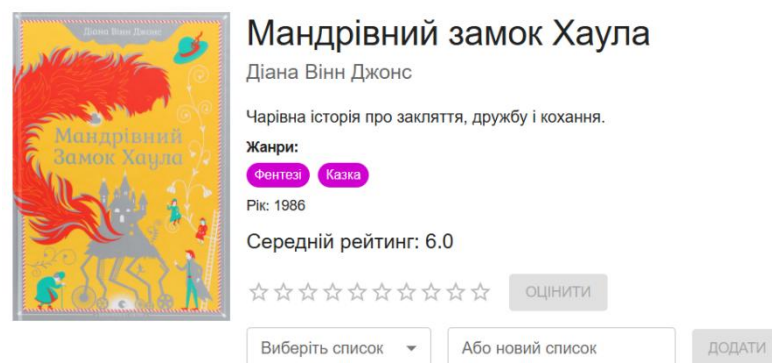


Рис. 3.16. - Назва та дані про книгу

Рекомендації до неї вийшли наступні (рис. 3.17.):

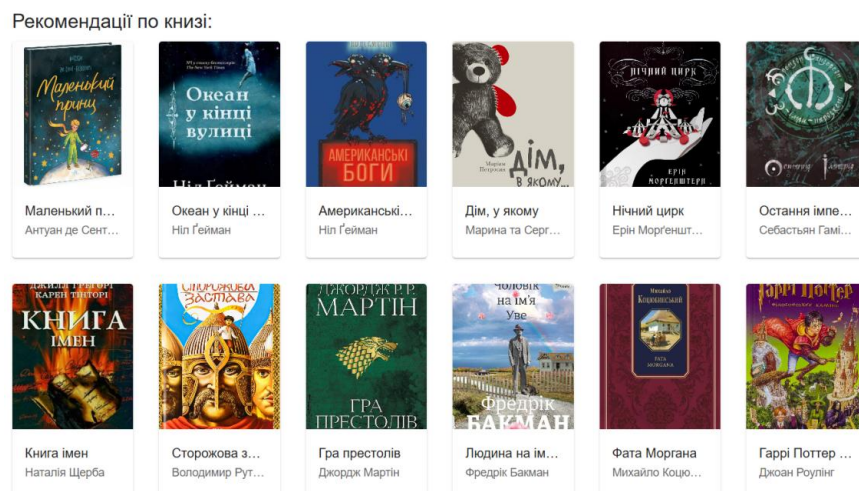


Рис. 3.17. - Рекомендації до книги “Мандрівний замок Хаула”

Для наглядності в таблиці 3.3 приведу додаткову інформацію, щоб краще розуміти чому такі рекомендації. Це дає змогу простежити взаємозв'язок між коефіцієнтами релевантності та жанровими характеристиками.

Таблиця 3.3

Список книг в рекомендації

Книга	Жанри	Оцінка релевантності книги
Маленький принц	казка, філософія	0,838
Океан у кінці дороги	фентезі, містика	0,676
Американські боги	фентезі, міфологія	0,615
Дім, у якому	фентезі, драма	0,607
Нічний цирк	фентезі, роман, містика	0,592
Остання імперія	фентезі, пригоди	0,565
Книга імен	фентезі, підліткова література	0,563
Сторожова застава	пригоди, дитяча література, фентезі	0,543
Гра престолів	фентезі, пригоди	0,514
Людина на ім'я Уве	комедійний роман, драма	0,5
Фата Моргана	роман	0,494
Гаррі Поттер і філософський камінь	фентезі, пригоди	0,474

Content-Based алгоритм бачить схожість книги “Мандрівний замок Хаула” з іншими фентезі та казковими творами, хоча коефіцієнти схожості у логах невеликі (0,056-0,071).

Collaborative Filtering додає книги, які користувачі, що оцінювали схожі твори, також читали. З таких книг, найвищий вплив мають “Людина на ім'я Уве” (0,765), “Фата Моргана” (0,757) та “Океан у кінці дороги” (0,667). Це означає, що популярність серед читачів значно підвищує рейтинг рекомендацій.

Отже, система комбінує жанрову схожість та поведінку інших користувачів. Content-Based забезпечує зв'язок із фентезі та казками, магічними або чарівними сюжетами. А Collaborative підкреслює популярні серед читачів книги, що допомагає видавати більш релевантні та цікаві рекомендації.

Таким чином, рекомендації для “Мандрівного замку Хаула” включають як класичні казкові та фентезійні твори (“Маленький принц”, “Нічний цирк”), так і популярні сучасні книги, які можуть зацікавити читача завдяки поведінці інших користувачів.

5. Рекомендація до книги при холодному старті

Так само як і у випадку з рекомендаціями для користувачів при холодному старті, для книг список Collaborative теж є порожнім, тобто рекомендації ґрунтуються виключно на Content-Based алгоритмі (рис. 3.18.).

Рекомендації по книзі:

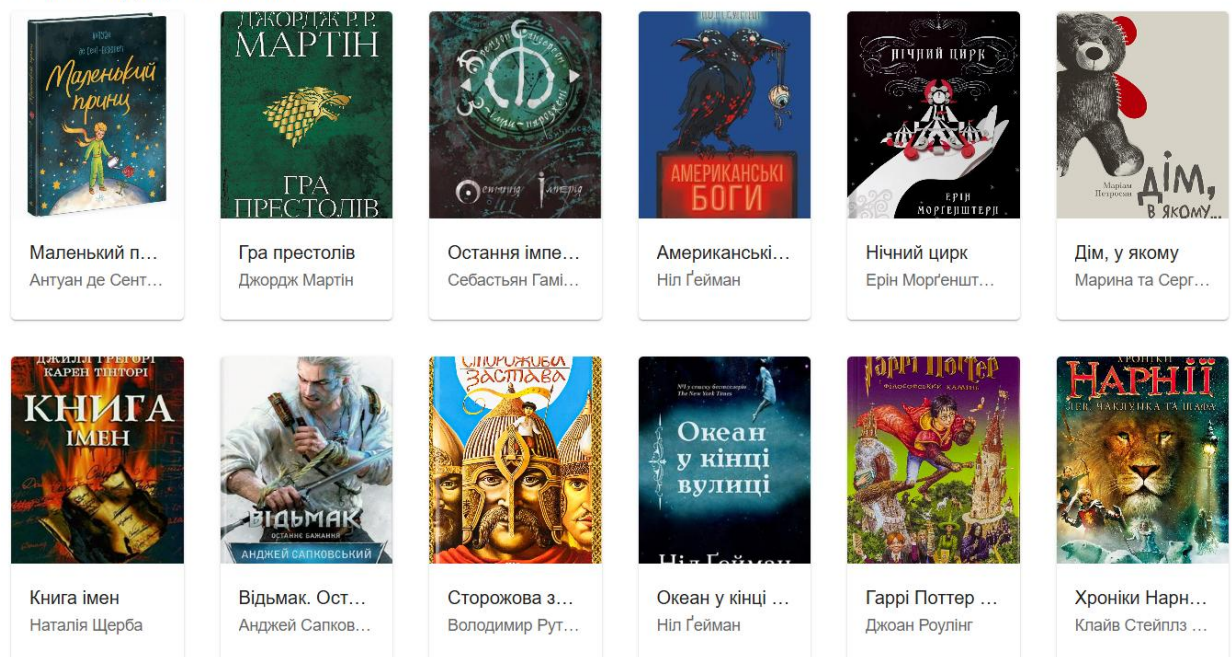


Рис. 3.18. - Рекомендації до книги “Мандрівний замок Хаула” при холодному старті

Так як вони будуються на контентному алгоритмі, в рекомендаціях ми отримали, що “Маленький принц” має 0,5, бо він має спільні риси з казковими та фентезійними елементами, що близькі до “Мандрівного замку Хаула”. А в інших випадках як “Гра престолів” або “Американські боги”, цей

бал нижчий, бо жанрова та тематична схожість з “Мандрівним замком Хаула” явно менша, але все одно достатня для включення до топ-рекомендацій.

У порівнянні зі звичайним режимом, де рекомендації для “Мандрівного замку Хаула” формуються з урахуванням двох підходів, топ у холодному старті трохи відрізняється. Деякі популярні серед користувачів книги отримали менший рейтинг, а частина рекомендацій з’являється лише завдяки жанровій та тематичній схожості. У той же час ключові фентезійні та казкові твори залишаються у топі, що демонструє, що Content-Based алгоритм здатний підтримати логічну послідовність рекомендацій навіть без даних від інших користувачів.

Висновки до розділу 3

У третьому розділі було описано вигляд програмного продукту та механізми формування рекомендацій. Графічний інтерфейс реалізовано на базі React, що забезпечує динамічну навігацію між різними сторінками.

Було проаналізовано рекомендації для користувачів та окремих книг. Гібридна модель Content-Based та Collaborative Filtering дозволяє формувати персоналізовані рекомендації з урахуванням жанрових уподобань та поведінки інших читачів. При холодному старті система працює на основі Content-Based алгоритму, що забезпечує логічну послідовність рекомендацій навіть без даних Collaborative Filtering.

ВИСНОВКИ

У результаті проведеного дослідження створено веб-додаток для персоналізованих рекомендацій книг із використанням гібридного підходу, що поєднує Content-Based та Item-Based Collaborative Filtering. Розроблена система дозволяє формувати точні та релевантні рекомендації, враховуючи жанрові уподобання користувачів та поведінку інших читачів. Також вона ефективно працює у режимі холодного старту.

Проаналізовано сучасні алгоритми рекомендацій та реалізовано препроцесинг даних, обробку жанрів і використання синтетичних користувачів, що підвищило точність прогнозування. Тестування рекомендацій для конкретних користувачів і книг підтвердило ефективність гібридного підходу. Тому що Content-Based забезпечує жанрову та тематичну релевантність, а Collaborative Filtering підсилює рекомендації популярними книгами.

Архітектура веб-додатку на базі React з модульною структурою забезпечує динамічний інтерфейс, незалежність рекомендаційного модуля та можливість масштабування. Результати дослідження демонструють практичну цінність розробленої системи для бібліотечних сервісів, онлайн-магазинів та навчальних платформ. Отримані рішення створюють основу для подальшого вдосконалення системи, інтеграції нових джерел даних та підвищення персоналізації рекомендацій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Харченко Ю.В., Вакалюк Т.А. Проблема холодного старту в рекомендаційних алгоритмах для книжкових веб-додатків. VIII Всеукраїнська науково-технічна конференція «Комп'ютерні технології: інновації, проблеми, рішення» (02-03 грудня 2025 р., м. Житомир) (подано до друку).

2. Харченко Ю.В., Вакалюк Т.А. Гібридні рекомендаційні алгоритми для персоналізованого підбору книжок. VIII Всеукраїнська науково-технічна конференція «Комп'ютерні технології: інновації, проблеми, рішення» (02-03 грудня 2025 р., м. Житомир) (подано до друку).

3. Shah H., Soomro T. R. Node.js challenges in implementation. Global Journal of Computer Science and Technology. Vol. 17, No. 2. 2017. pp. 73–83. https://www.researchgate.net/profile/Hezbollah-Shah/publication/318310544_Nodejs_Challenges_in_Implementation/links/59634e42458515a3575451a6/Nodejs-Challenges-in-Implementation.pdf

4. Györödi C., Györödi R., Pecherle G., Olah A. A comparative study: MongoDB vs. MySQL. 13th International Conference on Engineering of Modern Electric Systems (EMES). 2015. pp. 1–6. <https://doi.org/10.1109/EMES.2015.7158433>

5. Chauhan A. A review on various aspects of MongoDB databases. International Journal of Engineering Research & Technology (IJERT). Vol. 8, No. 5. 2019. pp. 90–92. <https://www.ijert.org/research/a-review-on-various-aspects-of-mongodb-databases-IJERTV8IS050031.pdf>

6. Cayres L. U., de Lima B. S., Garcia R. E., Correia R. C. M. Analysis of Node.js application performance using MongoDB drivers. Information Technology and Systems. ICITS 2020. Advances in Intelligent Systems and Computing. Vol. 1137. 2020. pp. 213–222. https://doi.org/10.1007/978-3-030-40690-5_21

7. Lazuardy M. F. S., Anggraini D. Modern front-end web architectures with React.js and Next.js. Research Journal of Advanced Engineering and Science. Vol.

7, No. 1. 2022. pp. 132–141. <https://irjaes.com/wp-content/uploads/2022/02/IRJAES-V7N1P162Y22.pdf>

8. McKinney W. pandas: a foundational Python library for data analysis and statistics. Python for High Performance and Scientific Computing. Vol. 14, No. 9. 2011. pp. 1–9. <https://docslib.org/doc/4231522/a-foundational-python-library-for-data-analysis-and-statistics>

9. Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Oliphant, T. E. Array programming with NumPy. Nature. Vol. 585. 2020. pp. 357–362. <https://doi.org/10.1038/s41586-020-2649-2>

10. Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., van Mulbregt, P. SciPy 1.0: Fundamental algorithms for scientific computing in Python. Nature Methods. Vol. 17. 2020. pp. 261–272. <https://doi.org/10.1038/s41592-019-0686-2>

11. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Duchesnay, É. Scikit-learn: Machine learning in Python. Journal of Machine Learning Research. Vol. 12. 2011. pp. 2825–2830. https://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf?source=post_page

12. Phalle Ms, Bhushan Prof. Content Based Filtering And Collaborative Filtering: A Comparative Study. Journal of Advanced Zoology. Vol. 45. 2024. pp. 96–100. <https://doi.org/10.53555/jaz.v45iS4.4158>

13. Su X., Khoshgoftaar T. M. A survey of collaborative filtering techniques. Mathematical Problems in Engineering. 2009. <https://doi.org/10.1155/2009/421425>

14. Wang H., Shen Z., Jiang S., Sun G., Zhang R.-J. User-based collaborative filtering algorithm design and implementation. Journal of Physics: Conference Series. Vol. 1757. 2021. 012168. <https://doi.org/10.1088/1742-6596/1757/1/012168>

15. Sarwar B., Karypis G., Konstan J., Riedl J. Item-based Collaborative Filtering Recommendation Algorithms. ACM WWW Conference Proceedings. 2001. <https://doi.org/10.1145/371920.372071>

16. Lee J., Sun M., Lebanon G. A comparative study of collaborative filtering algorithms. arXiv preprint. 2012. <https://doi.org/10.48550/arXiv.1205.3193>
17. Çano E. Hybrid Recommender Systems: A Systematic Literature Review. *Intelligent Data Analysis*. Vol. 21. 2017. pp. 1487–1524. <http://dx.doi.org/10.3233/IDA-163209>
18. Do H.-Q., Le T.-H., Yoon B. Dynamic Weighted Hybrid Recommender Systems. 22nd International Conference on Advanced Communication Technology (ICACT). 2020. pp. 644–650. <https://doi.org/10.23919/ICACT48636.2020.9061465>
19. Burke R. Hybrid Web Recommender Systems. *Lecture Notes in Computer Science*. Vol. 4321. 2007. https://doi.org/10.1007/978-3-540-72079-9_12
20. Liu, W., Du, Z., Zhao, H., Zhang, W., Zhao, X., Wang, G., ... & Xu, J. Inference computation scaling for feature augmentation in recommendation systems. arXiv preprint. 2025. <https://doi.org/10.48550/arXiv.2502.16040>
21. Ghazanfar M., Prugel-Bennett A. An improved switching hybrid recommender system using Naive Bayes classifier and collaborative filtering. The 2010 IAENG International Conference on Data Mining and Applications, Hong Kong. 2010. pp. 17–19. <http://eprints.soton.ac.uk/id/eprint/268483>
22. Hrnjica B., Music D., Softic S. Model-based recommender systems. *Trends in Cloud-based IoT*. 2020. https://doi.org/10.1007/978-3-030-40037-8_8
23. Gope J., Jain S. K. A survey on solving cold start problem in recommender systems. 2017 International Conference on Computing, Communication and Automation (ICCCA). 2017. pp. 133–138. <https://doi.org/10.1109/CCAA.2017.8229786>
24. Monti D., Rizzo G., Morisio M. All you need is ratings: A clustering approach to synthetic rating datasets generation. arXiv preprint. 2019. <https://doi.org/10.48550/arXiv.1909.00687>
25. Kadhim A. I. An evaluation of preprocessing techniques for text classification. *International Journal of Computer Science and Information Security*. Vol. 16, No. 6. 2018. pp. 22–32. <https://www.researchgate.net/profile/Ammar->

Kadhim-

4/publication/329339664_An_Evaluation_of_Preprocessing_Techniques_for_Text_Classification/links/5c1b6aa6a6fdccfc705ae648/An-Evaluation-of-Preprocessing-Techniques-for-Text-Classification.pdf

26. Permana A. H. J. P. J., Wibowo A. T. Movie recommendation system based on synopsis using content-based filtering with TF-IDF and cosine similarity. International Journal on Information and Communication Technology (IJoICT). Vol. 9, No. 2. 2023. pp. 1–14. <https://doi.org/10.21108/ijoict.v9i2.747>

27. Deshpande M., Karypis G. Item-based top-N recommendation algorithms. ACM Transactions on Information Systems. Vol. 22, No. 1. 2004. pp. 143–177. <https://doi.org/10.1145/963770.963776>

28. Widayanti, R., Chakim, M. H. R., Lukita, C., Rahardja, U., & Lutfiani, N. Improving recommender systems using hybrid techniques of collaborative filtering and content-based filtering. Journal of Applied Data Sciences. Vol. 4, No. 3. 2023. pp. 289–302. <https://doi.org/10.47738/jads.v4i3.115>

29. Fink G., Flatow I. Introducing Single Page Applications. Pro Single Page Application Development. 2014. https://doi.org/10.1007/978-1-4302-6674-7_1

30. Bell D. UML basics: The component diagram. IBM Global Services. 2004. pp. 1–10. https://softwareresearch.net/fileadmin/user_upload/Teaching/WS13/SE_UML_basics-The_component_diagram.pdf

31. Whitten J. L., Bentley L. D. Systems analysis and design methods. McGraw-Hill. 2005. <https://dl.acm.org/doi/abs/10.5555/1200604>

32. Chen P. P. S. The entity-relationship model—toward a unified view of data. ACM Transactions on Database Systems. Vol. 1, No. 1. 1976. pp. 9–36. <https://doi.org/10.1145/320434.320440>

33. Su R., Li X. Modular Monolith: Is this the trend in software architecture? 1st International Workshop on New Trends in Software Architecture. 2024. pp. 10–13. <https://doi.org/10.1145/3643657.3643911>

34. Kamisetty A. Microservices vs. Monoliths: Comparative Analysis for Scalable Software Architecture Design. Engineering International. Vol. 11, No. 2. 2023. pp. 99–112.
https://www.researchgate.net/publication/387645461_Microservices_vs_Monoliths_Comparative_Analysis_for_Scalable_Software_Architecture_Design
35. Al-Qora'n L. F., Al-Said Ahmad A. Modular Monolith Architecture in Cloud Environments: A Systematic Literature Review. Future Internet. Vol. 17, No. 11. 2025. 496. <https://doi.org/10.3390/fi17110496>
36. Faustino D., Gonçalves N., Portela M., Silva A. R. Stepwise migration of a monolith to a microservice architecture: Performance and migration effort evaluation. Performance Evaluation. Vol. 164. 2024. 102411.
<https://doi.org/10.1016/j.peva.2024.102411>

ДОДАТКИ

Рекомендаційний алгоритм Content-Based Filtering

```

def build_cb_matrix(books_df):
    books_df = books_df.copy()
    books_df['genres_str'] = books_df.get('genres', '').fillna('').apply(
        lambda x: ' '.join(x) if isinstance(x, list) else str(x)
    )
    books_df['content'] = books_df.get('authors', '').apply(preprocess) + ' ' + \
        books_df.get('title', '').apply(preprocess) + ' ' + \
        books_df['genres_str'].apply(preprocess)
    tfidf = TfidfVectorizer(stop_words=uk_stopwords)
    tfidf_matrix = tfidf.fit_transform(books_df['content'])
    bookid_to_idx = pd.Series(books_df.index.values,
index=books_df['book_id']).to_dict()
    return tfidf_matrix, bookid_to_idx

def content_based_scores_for_user(user_books, books_df, tfidf_matrix,
bookid_to_idx):
    if not user_books:
        return {}
    user_vec = np.zeros(tfidf_matrix.shape[1])

    count = 0
    for book_id in user_books.keys():
        idx = bookid_to_idx.get(book_id)
        if idx is not None:
            user_vec += tfidf_matrix[idx].toarray().ravel()
            count += 1
    if count > 0:
        user_vec /= count

    scores = cosine_similarity([user_vec], tfidf_matrix).ravel()
    scores_map = {books_df.iloc[i]['book_id']: float(scores[i]) for i in
range(len(books_df))}

    for b in user_books.keys():
        scores_map.pop(b, None)

    max_score = max(scores_map.values(), default=1)
    if max_score > 0:
        scores_map = {k: v / max_score for k, v in scores_map.items()}

    return scores_map

```

Рекомендаційний алгоритм Item-Based Collaborative Filtering

```

def build_ibcf_matrix(ratings_df):
    if ratings_df.empty:
        return pd.DataFrame()
    ratings_df = ratings_df.groupby(['user_id', 'book_id'],
as_index=False)['rating'].mean()

    user_item = ratings_df.pivot(index='user_id', columns='book_id',
values='rating').fillna(0)
    sparse = csr_matrix(user_item.values)
    item_sim = cosine_similarity(sparse.T)
    item_similarity_df = pd.DataFrame(item_sim, index=user_item.columns,
columns=user_item.columns)
    return item_similarity_df

def cf_scores_for_user(user_books, item_similarity_df):
    scores = defaultdict(float)
    for book_id, rating in user_books.items():
        if book_id not in item_similarity_df.index:
            continue
        sims = item_similarity_df.loc[book_id]
        for bid, sim in sims.items():
            if bid not in user_books:
                scores[bid] += sim * rating
    if scores:
        max_score = max(scores.values())
        if max_score > 0:
            scores = {k: v / max_score for k, v in scores.items()}
    return dict(scores)

```

Гібридний алгоритм

```
def combine_scores(cb_scores, cf_scores, alpha=0.5, top_n=10):  
    all_books = set(cb_scores.keys()).union(cf_scores.keys())  
    hybrid_scores = {b: alpha * cb_scores.get(b, 0) + (1-alpha) * cf_scores.get(b,  
0) for b in all_books}  
    top_books = sorted(hybrid_scores.items(), key=lambda x: x[1],  
reverse=True)[:top_n]  
    return top_books
```

Лог рекомендації користувачу

Book 48 - Content-Based: [(95, np.float64(0.38640776003242594)), (27, np.float64(0.08559178617529177)), (37, np.float64(0.08145305401422058))]

Book 48 - Collaborative: [(38, 0.6273885280522581), (99, 0.6197506830096349), (83, 0.5610494542072252)]

Content-Based: Book 95 -> 0.3864 Book 27 -> 0.0856 Book 37 -> 0.0815

Collaborative: Book 38 -> 0.6274 Book 99 -> 0.6198 Book 83 -> 0.5610

Book 31 - Content-Based: [(27, np.float64(0.20150051426038443)), (111, np.float64(0.17982180699452915)), (38, np.float64(0.07742222270024189))]

Book 31 - Collaborative: [(83, 0.606687814028925), (82, 0.605712860211563), (27, 0.6016742263719357)]

Content-Based: Book 27 -> 0.2015 Book 111 -> 0.1798 Book 38 -> 0.0774

Collaborative: Book 83 -> 0.6067 Book 82 -> 0.6057 Book 27 -> 0.6017

Book 67 - Content-Based: [(27, np.float64(0.16958889661721754)), (37, np.float64(0.16138854174726777)), (28, np.float64(0.0910353668631334))]

Book 67 - Collaborative: [(27, 0.6629855491870713), (38, 0.5901483219111896), (99, 0.5817525118896392)]

Content-Based: Book 27 -> 0.1696 Book 37 -> 0.1614 Book 28 -> 0.0910

Collaborative: Book 27 -> 0.6630 Book 38 -> 0.5901 Book 99 -> 0.5818

Book 47 - Content-Based: [(71, np.float64(0.16945534562844616)), (27, np.float64(0.10251250065966884)), (28, np.float64(0.10206516348928177))]

Book 47 - Collaborative: [(83, 0.5811223694796518), (82, 0.5585670762846733), (38, 0.4772944933844444)]

Content-Based: Book 71 -> 0.1695 Book 27 -> 0.1025 Book 28 -> 0.1021

Collaborative: Book 83 -> 0.5811 Book 82 -> 0.5586 Book 38 -> 0.4773

Book 42 - Content-Based: [(27, np.float64(0.17043140758645897)), (37, np.float64(0.16219031367599762)), (28, np.float64(0.09148762698570632))]

Book 42 - Collaborative: [(28, 0.4957929915602791), (82, 0.4902221696037451), (27, 0.4863128016370674)]

Content-Based: Book 27 -> 0.1704 Book 37 -> 0.1622 Book 28 -> 0.0915

Collaborative: Book 28 -> 0.4958 Book 82 -> 0.4902 Book 27 -> 0.4863

Book 25 - Content-Based: [(27, np.float64(0.15753322109947454)), (37, np.float64(0.14991581015695415)), (28, np.float64(0.08456387689278586))]

Book 25 - Collaborative: [(28, 0.5712314001550421), (83, 0.5608684664074725), (27, 0.5287609116276896)]

Content-Based: Book 27 -> 0.1575 Book 37 -> 0.1499 Book 28 -> 0.0846

Collaborative: Book 28 -> 0.5712 Book 83 -> 0.5609 Book 27 -> 0.5288

Book 41 - Content-Based: [(27, np.float64(0.08042143365567189)), (37, np.float64(0.07653271034723994)), (82, np.float64(0.07280177660728619))]

Book 41 - Collaborative: [(111, 0.6284184680732404), (82, 0.6272205067690074), (37, 0.6234082721961863)]

Content-Based: Book 27 -> 0.0804 Book 37 -> 0.0765 Book 82 -> 0.0728

Collaborative: Book 111 -> 0.6284 Book 82 -> 0.6272 Book 37 -> 0.6234

Book 116 - Content-Based: [(111, np.float64(0.19659407167050136)), (35, np.float64(0.1550148903321502)), (83, np.float64(0.0790683103412205))]

Book 116 - Collaborative: [(37, 0.6412215036744151), (38, 0.5745352870509867), (35, 0.5559640793554348)]

Content-Based: Book 111 -> 0.1966 Book 35 -> 0.1550 Book 83 -> 0.0791

Collaborative: Book 37 -> 0.6412 Book 38 -> 0.5745 Book 35 -> 0.5560

Book 117 - Content-Based: [(111, np.float64(0.30603122160943635)), (83, np.float64(0.18400210152380272)), (35, np.float64(0.0))]

Book 117 - Collaborative: [(27, 0.5889176331787022), (38, 0.5299508431678379), (37, 0.519318409491814)]

Content-Based: Book 111 -> 0.3060 Book 83 -> 0.1840 Book 35 -> 0.0000

Collaborative: Book 27 -> 0.5889 Book 38 -> 0.5300 Book 37 -> 0.5193 -----

Book 66 - Content-Based: [(71, np.float64(0.18909369444714166)), (35, np.float64(0.08663801330479981)), (99, np.float64(0.03215726480280596))]

Book 66 - Collaborative: [(38, 0.6789070687357304), (27, 0.6223509862493581), (83, 0.5760401037229983)]

Content-Based: Book 71 -> 0.1891 Book 35 -> 0.0866 Book 99 -> 0.0322

Collaborative: Book 38 -> 0.6789 Book 27 -> 0.6224 Book 83 -> 0.5760

Book 77 - Content-Based: [(27, np.float64(0.17200887424131456)), (71, np.float64(0.15686887518537845)), (35, np.float64(0.07187340506067963))]

Book 77 - Collaborative: [(99, 0.5980602405218663), (82, 0.5587354091691329), (38, 0.5425035691141689)]

Content-Based: Book 27 -> 0.1720 Book 71 -> 0.1569 Book 35 -> 0.0719

Collaborative: Book 99 -> 0.5981 Book 82 -> 0.5587 Book 38 -> 0.5425

Book 63 - Content-Based: [(27, np.float64(0.13003296837467293)), (37, np.float64(0.12374531330564989)), (28, np.float64(0.06980173358288574))]

Book 63 - Collaborative: [(83, 0.5918351469143739), (27, 0.5672217090124616), (99, 0.5395467758743487)]

Content-Based: Book 27 -> 0.1300 Book 37 -> 0.1237 Book 28 -> 0.0698

Collaborative: Book 83 -> 0.5918 Book 27 -> 0.5672 Book 99 -> 0.5395

Лог рекомендації користувачу при холодному старті

Book 48 - Content-Based: [(95, np.float64(0.38640776003242594)), (27, np.float64(0.08559178617529177)), (37, np.float64(0.08145305401422058))]

Book 48 - Collaborative: []

Content-Based: Book 95 -> 0.3864 Book 27 -> 0.0856 Book 37 -> 0.0815

Collaborative:

Book 42 - Content-Based: [(27, np.float64(0.17043140758645897)), (37, np.float64(0.16219031367599762)), (28, np.float64(0.09148762698570632))]

Book 42 - Collaborative: []

Content-Based: Book 27 -> 0.1704 Book 37 -> 0.1622 Book 28 -> 0.0915

Collaborative:

Book 67 - Content-Based: [(27, np.float64(0.16958889661721754)), (37, np.float64(0.16138854174726777)), (28, np.float64(0.0910353668631334))]

Book 67 - Collaborative: []

Content-Based: Book 27 -> 0.1696 Book 37 -> 0.1614 Book 28 -> 0.0910

Collaborative:

Book 31 - Content-Based: [(27, np.float64(0.20150051426038443)), (111, np.float64(0.17982180699452915)), (38, np.float64(0.07742222270024189))]

Book 31 - Collaborative: []

Content-Based: Book 27 -> 0.2015 Book 111 -> 0.1798 Book 38 -> 0.0774

Collaborative:

Book 47 - Content-Based: [(71, np.float64(0.16945534562844616)), (27, np.float64(0.10251250065966884)), (28, np.float64(0.10206516348928177))]

Book 47 - Collaborative: []

Content-Based: Book 71 -> 0.1695 Book 27 -> 0.1025 Book 28 -> 0.1021

Collaborative:

Book 25 - Content-Based: [(27, np.float64(0.15753322109947454)), (37, np.float64(0.14991581015695415)), (28, np.float64(0.08456387689278586))]

Book 25 - Collaborative: []

Content-Based: Book 27 -> 0.1575 Book 37 -> 0.1499 Book 28 -> 0.0846

Collaborative:

Book 117 - Content-Based: [(111, np.float64(0.30603122160943635)), (83, np.float64(0.18400210152380272)), (35, np.float64(0.0))]

Book 117 - Collaborative: []

Content-Based: Book 111 -> 0.3060 Book 83 -> 0.1840 Book 35 -> 0.0000

Collaborative:

Book 116 - Content-Based: [(111, np.float64(0.19659407167050136)), (35, np.float64(0.1550148903321502)), (83, np.float64(0.0790683103412205))]

Book 116 - Collaborative: []

Content-Based: Book 111 -> 0.1966 Book 35 -> 0.1550 Book 83 -> 0.0791

Collaborative:

Book 63 - Content-Based: [(27, np.float64(0.13003296837467293)), (37, np.float64(0.12374531330564989)), (28, np.float64(0.06980173358288574))]

Book 63 - Collaborative: []

Content-Based: Book 27 -> 0.1300 Book 37 -> 0.1237 Book 28 -> 0.0698

Collaborative:

Book 94 - Content-Based: [(111, np.float64(0.2706667693889496)), (83, np.float64(0.14862401679370082)), (35, np.float64(0.0))]

Book 94 - Collaborative: []

Content-Based: Book 111 -> 0.2707 Book 83 -> 0.1486 Book 35 -> 0.0000

Collaborative:

Book 77 - Content-Based: [(27, np.float64(0.17200887424131456)), (71, np.float64(0.15686887518537845)), (35, np.float64(0.07187340506067963))]

Book 77 - Collaborative: []

Content-Based: Book 27 -> 0.1720 Book 71 -> 0.1569 Book 35 -> 0.0719

Collaborative:

Book 66 - Content-Based: [(71, np.float64(0.18909369444714166)), (35, np.float64(0.08663801330479981)), (99, np.float64(0.03215726480280596))]

Book 66 - Collaborative: []

Content-Based: Book 71 -> 0.1891 Book 35 -> 0.0866 Book 99 -> 0.0322

Collaborative:
