Вінницький національний технічний університет
(повне найменування вищого навчального закладу)
Факультет інтелектуальних інформаційних технологій та автоматизації
(повне найменування інституту, назва факультету (відділення))
Кафедра комп'ютерних наук
(повна назва кафедри (предметної, циклової комісії))

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія надання рекомендацій для платформи огляду книг»

Виконав: студент 2-го курсу, групи 2КН-21м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
_____ Зайчик В.О.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
_____ Арсенюк І.Р.
(прізвище та ініціали)
«15» 12 2022 р.

Опонент: к.т.н., професор каф. ПЗ
_____ Кабачій В. В.
(прізвище та ініціали)
«17» грудня 2022 р.

Допущено до захисту
Завідувач кафедри КН
_____ д.т.н., проф. Яровий А.А.
(прізвище та ініціали)
«16» грудня 2022 р.

Вінниця ВНТУ - 2022 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації Кафедра комп'ютерних наук
Рівень вищої освіти II-й (магістерський)
Галузь знань – 12 «Інформаційні технології»
Спеціальність – 122 «Комп'ютерні науки»
Освітньо-професійна програма – «Системи штучного інтелекту»

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
Д.т.н., проф. Яровий А.А.
15. 09 **2022 року**

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Зайчик Владислав Олександрович
(прізвище, ім'я, по батькові)

1. Тема роботи: «Інформаційна технологія надання рекомендацій для платформи огляду книг»
керівник роботи_ к.т.н., доцент кафедри КН Арсенюк І. Р.,
затверджені наказом вищого навчального закладу від «14» 09 2022 року №205
2. Строк подання студентом роботи 18 листопада 2022 року.
3. Вхідні дані до роботи:
вхідні дані – датасет з користувачами, книгами та рецензіями у форматі csv, набір даних для навчання, мова програмування – об'єктно-орієнтовна.
4. Зміст текстової частини:
Вступ, аналіз сучасного стану розвитку інформаційних технологій надання рекомендацій для вибору книг, розробка інформаційної технології надання рекомендацій для вибору книг, програмна реалізація інформаційної технології надання рекомендацій для платформи огляду книг, економічна частина, висновки, перелік використаних джерел, додатки.

Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень): Алгоритм загального функціонування системи, Загальна структурна схема, UML-діаграма класів, Вигляд інтерфейсу користувача Загальна схема роботи методу колаборативної фільтрації на графі, Матрична факторизація,

5. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконав прийняв
1-3	Арсенюк І. Р., к.т.н., доц. каф. КН	14.09.22р.	14.09.22р.
4	Бурєнікова Н. В., к. е. н., доц. каф. ЕПВМ	13.09.22р.	13.09.22р.

6. Дата видачі завдання 14. 09. 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примі
1	Аналіз сучасного рівня інформаційних технологій розпізнавання математичних формул. Постановка задач дослідження	14.09.2022р. - 01.10.2022р.	Розділ 1
2	Побудова моделей розпізнавання математичних формул на основі нейронної мережі та функціонування нейронної мережі	02.10.2022р - 16.10.2022р.	Розділ 2
3	Практичне застосування та оцінка ефективності розроблених моделей	17.10.2022р - 07.11.2022р.	Розділ 3
4	Підготовка економічної частини	08.11.2022р. 21.11.2022р.	Розділ 4
5	Апробація та/або впровадження результатів дослідження	23.11.2022р. 01.12.2022р.	Додаток В
6	Оформлення пояснювальної записки, графічного матеріалу та презентації	02.11.2022р. 14.12.2022р.	Пояснювальна записка

Студент _____ Зайчик В.О.
(підпис)

Керівник роботи _____ Арсенюк І. Р.
(підпис)

АНОТАЦІЯ

УДК 004.8

Зайчик В. О. Інформаційна технологія надання рекомендацій для платформи огляду книг. Магістерська кваліфікаційна робота зі спеціальності 122 – комп’ютерні науки, освітня програма - комп’ютерні науки. Вінниця: ВНТУ, 2022. 114 с.

На укр. мові. Бібліогр.: 29 назв; рис.: 12; табл. 7.

Дана магістерська кваліфікаційна робота присвячена розробці програмного забезпечення для надання рекомендацій книг. Були розглянуті та проаналізовані існуючі методи надання рекомендацій, як найбільш перспективний, було обрано нейромережевий метод. Було проаналізовано різні парадигми штучних нейронних мереж та обґрунтовано вибір для даної задачі нейромережі згорткової нейронної мережі. Архітектура обраного типу мережі налічує 336 входів, один прихований шар із 130 нейронів та вихідний шар із 15 нейронів. Було спроектовано програму надання рекомендацій щодо вибору книг, написану мовою програмування Java та Python з використанням бібліотеки TensorFlow. Доречність рекомендацій розробленої програми на 3% краще аналогу, а швидкодія прискорена на 25%.

Графічна частина складається з 7 плакатів.

У економічному розділі розраховано суму витрат на розробку та виготовлення нового технічного рішення, яка складає 751055 гривень, спрогнозовано орієнтовану величину витрат по кожній з статей витрат, розраховано чистий прибуток, термін окупності витрат для виробника 0,62 роки та економічний ефект для споживача при використанні даної розробки.

Ключові слова: рекомендація, граф, нейронна мережа, колаборативна фільтрація, матрична факторизація.

ABSTRACT

Zaichyk V. O. Information technology for providing recommendations on book review platform. Master's thesis in the specialty 122 - computer sciences, educational program - computer science. Vinnytsia: VNTU, 2022. 114 p.

In Ukrainian language. Bibliographer: 29 titles; fig .: 12; table 7.

This master's thesis is devoted to the development of software for providing book recommendations. The existing methods of providing recommendations were considered and analyzed, and the neural network method was chosen as the most promising. Different paradigms of artificial neural networks were analyzed and the choice of a convolutional neural network for this problem was justified. The architecture of the selected network type has 336 inputs, one hidden layer of 130 neurons, and an output layer of 15 neurons. A book recommendation program written in Java and Python using the TensorFlow library was designed. The appropriateness of the recommendations of the developed program is 3% better than the analogue, and the speed of action is accelerated by 25%.

The graphic part consists of 7 posters.

In the economic section, the amount of costs for the development and production of a new technical solution is calculated, which is 751,055 hryvnias, the estimated amount of costs for each of the cost items is predicted, the net profit is calculated, the payback period for the manufacturer is 0.62 years, and the economic effect for the consumer when using this developments.

Keywords: recommendation, graph, neural network, collaborative filtering, matrix factorization.

ЗМІСТ

ВСТУП

Актуальність. Інтернет і пов'язані з ним технології стрімко розвиваються з кожним днем. Інформації, що проходить через мережу, з кожною секундою стає все більше. Відстежити та обрати серед неї необхідну без спеціальних пошуково-фільтраційних засобів стало неможливо.

Інструменти для пошуку та фільтрації інформації в Інтернеті можуть бути різними: від пошукових систем, до систем рекомендацій. Ці інструменти будуть актуальні до тих пір, поки всесвітня павутина буде розвиватися, тому що простому користувачеві буде просто неможливо відфільтрувати необхідні знання без цих інструментів [1].

Рекомендаційна система відрізняється від пошукової тим, що в першому випадку користувач не може впливати на дані безпосередньо, оскільки фільтр формується не на основі запиту користувача, а на основі його дій у мережі чи системі, в якій він знаходиться. Тобто інформаційний фільтр зміниться просто тому, що користувач користується мережею.

Станом на сьогодні були розроблені різні типи систем рекомендацій: на основі перегляду текстів, порівняльних думок, оцінок користувачів, моделей купівлі, профілів користувачів тощо. Ці системи змінили спосіб роботи електронної комерції в Інтернеті та соціальних медіа: від рекомендацій друзів у Facebook до купівлі продуктів на Amazon і вибір фільмів і музики на Netflix. Система рекомендацій діє як сімейство систем фільтрації інформації, які надають рекомендації користувачам на основі їхніх вподобань.

Найбільш очевидною метою системи рекомендацій є рекомендація відповідних продуктів користувачеві. Як говорив Стів Джобс: «Часто люди не знають, чого хочуть, доки їм це не покажуть». Посилаючись на його слова, ми можемо сказати, що одна з другорядних цілей системи рекомендацій — показати користувачам продукти, які вони раніше не бачили і можуть їм сподобатися.

Правильний підбір рекомендацій може допомогти підвищити загальну задоволеність користувачів, що підвищує ймовірність того, що споживач знову скористається веб-сайтом або додатком [2].

Одним із найвідоміших користувачів і першопроходців систем рекомендацій є веб сайт - Amazon.com. Amazon використовує рекомендації для персоналізації інтернет-магазину для кожного клієнта, що приносить до 35% річного доходу Amazon. Іншим відомим прикладом системи рекомендацій є алгоритм, який використовує Netflix. Згідно з відкритими даними, 75% того, що користувачі дивляться на Netflix, надходять із рекомендацій фільмів. У статті «The Netflix Recommender System: Algorithms, Business Value, and Innovation», написаній керівниками Netflix, автори стверджують, що система рекомендацій економить компанії близько 1 мільярда доларів щороку. За даними Spotify, впровадження нового алгоритму рекомендацій допомогло збільшити кількість користувачів щомісяця з 75 мільйонів до 100 мільйонів.

Актуальність теми досліджень даної магістерської роботи полягає в тому, що обсяг інформації сьогодні швидко зростає. Це створює проблему для користувачів під час вибору продуктів, які вони дійсно хочуть купити, або послуг, на які вони дійсно хотіли б підписатися. Велике значення в цьому випадку набуває система рекомендацій, яка зможе допомогти користувачу, серед великої кількості даних, знайти бажану інформацію.

До таких систем відноситься платформа для рецензування книг. Наразі вже існують подібні платформи для оглядів, які співіснують із системами електронної комерції (Amazon, eBay). Однак суттєвим недоліком таких платформ є те, що вони в першу чергу орієнтовані на підтримку бізнесу, а не на інтереси користувачів, а також обмежені інструменти оцінки книг. Моя система пропонує вдосконалити існуючі системи рекомендацій, надавши користувачам більше інструментів і характеристик для впливу на оцінку книги, а також відокремити таку платформу від факторів, введених певними бізнес-чинниками. Однак, щоб створити таку

систему, необхідно вибрати метод, за допомогою якого створюються ці рекомендації.

Зв'язок роботи з науковими програмами, планами, темами. Магістерська кваліфікаційна робота виконана відповідно до напряму наукових досліджень кафедри комп’ютерних наук Вінницького національного технічного університету 22 К1 «Моделі, методи, технології та пристрої інтелектуальних інформаційних систем управління, економіки, навчання та комунікацій» та плану наукової та навчально-методичної роботи кафедри.

Мета і завдання дослідження. Метою дослідження магістерської кваліфікаційної роботи є підвищення точності надання рекомендацій книг та прискорення швидкодії процесу обрахунку ймовірних рекомендацій. Для досягнення поставленої мети слід розв’язати такі завдання:

- розглянути та проаналізувати існуючі програмні реалізації розв’язання задачі надання рекомендацій книг
- запропонувати математичну модель для інформаційної технології надання рекомендацій книг;
- запропонувати етапи інформаційної технології та на їх основі розробити структуру та алгоритм роботи програмного засобу;
- виконати програмну реалізацію запропонованої інформаційної технології надання рекомендацій книг;
- провести тестування програмного продукту та виконати аналіз отриманих результатів.

Об’єкт дослідження - процес надання рекомендацій щодо вибору книг.

Предметом дослідження є програмне забезпечення для вибору книг на базі платформи книжкового рецензування.

Методи дослідження – це застосовувані методи при процесі розробки програмного забезпечення, такі як теорії алгоритмів та застосування сучасних шаблонів проектування.

Практичним значенням одержаних результатів даної роботи є розроблене програмне забезпечення, що надає рекомендації для користувачів платформи огляду книг.

Для досягнення поставленої мети необхідно такі завдання:

- проаналізувати технічний рівень систем рекомендацій для платформи огляду книг;
- обґрунтувати доцільність створення програмного модуля для надання рекомендацій для платформи огляду книг;
- розробити структуру програмного модуля для надання рекомендацій для платформи рецензування книг;
- провести аналіз та обґрунтувати вибір засобів розробки;
- реалізувати програмну реалізацію програмних рекомендацій для платформи рецензування книг;
- провести тестування розробленої програми та проаналізувати отримані результати.

Наукова новизна одержаних результатів полягає в наступному: Удосконалено метод обрахунку можливих рекомендацій, що відрізняється від відомих тим, що матрицю факторизації було замінено на граф, який дозволяє знизити вимоги до потужності обчислювальних ресурсів без вагомих втрат точності надання рекомендацій, та дозволяє збільшити їх якість.

Достовірність теоретичних положень магістерської кваліфікаційної роботи підтверджується строгістю постановки задач, коректним застосуванням математичних методів під час доведення наукових положень, строгим виведенням аналітичних співвідношень, порівнянням результатів з відомими, та збіжністю результатів математичного моделювання з результатами, що отримані під час впровадження розроблених програмних засобів.

Особистий внесок здобувача. Усі результати, наведені у магістерській кваліфікаційній роботі, отримані самостійно.

Апробація результатів роботи. Результати досліджень було апробовано на всеукраїнській L науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії, я[1].

Публікації. За результатами досліджень опубліковано тезу доповіді науково-практичної конференції [1], тези доповіді міжнародної науково-практичної конференції [2].

1 АНАЛІЗ СУЧАСНОГО СТАНУ РОЗВИТКУ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ СИСТЕМ РЕКОМЕНДАЦІЙ ДЛЯ ВИБОРУ КНИГ

1.1 Аналіз існуючих методів розв’язання задачі надання рекомендецій щодо вибору книг

Рекомендаційна система – це тип машинного навчання, який використовує дані, щоб допомогти передбачити, звузити та знайти те, що люди шукають серед експоненціально зростаючої кількості варіантів. Системи рекомендацій працюють за допомогою алгоритму штучного інтелекту, що зазвичай пов’язаний із машинним навчанням, який використовує великі дані, щоб пропонувати або рекомендувати додаткові продукти та послуги споживачам. Вони можуть ґрунтуватися на різних критеріях, зокрема минулих покупках, історії пошуку, демографічній інформації та інших факторах. Системи рекомендацій дуже корисні, оскільки вони допомагають користувачам знаходити продукти та послуги, які вони могли б не знайти самостійно.

Системи рекомендацій створюються таким чином, щоб розуміти вподобання, попередні рішення та характеристики людей і продуктів, використовуючи дані, зібрані про їхню взаємодію. До них належать перегляди, кліки, оцінки "подобається" та покупки. Завдяки своїй здатності передбачати інтереси та бажання споживачів на високо персоніфікованому рівні, системи рекомендацій є улюбленими серед постачальників контенту та продуктів. Вони можуть спонукати споживачів практично до будь-якого продукту чи послуги, які їх цікавлять, від книг до відео, уроків здоров’я та одягу [3].

Рекомендації системи персоналізовані під кожного користувача на основі його відомих інтересів і кожен з них бачить різні рекомендації. Як випливає з назви, пов’язані елементи – це рекомендації, схожі на певний елемент. Для прикладу, в електронній бібліотеці Google Books користувачі, які переглядають сторінку книги

з математики, також можуть бачити панель пов’язаних книжок, наприклад математичних або наукових. Система рекомендацій допомагає користувачам знаходити переконливий вміст серед великої кількості інформації. Google Book Store пропонує мільйони книг, тоді як YouTube пропонує мільярди відео. З кожним днем додається все більше програм, відео, книг, статей і так далі. Користувачі можуть знайти новий цікавий вміст або шляхом пошуку. Однак механізм рекомендацій може показувати елементи, які користувачі, можливо, не думали шукати самостійно.

Існують різні методи для розробки ефективної системи рекомендацій, два з яких складають основу для розробки інших підходів. Це методи фільтрування за вмістом та спільного фільтрування.

Методи, що базуються на вмісті (content-based), використовують перелік функцій елемента та порівнюють його з елементами, яким раніше віддавали перевагу певні користувачі. Елементи, що відповідають схожості, рекомендуються користувачеві. Основна функція фільтрування на базі вмісту працює у два етапи. Спочатку відбувається збереження профілю користувача на основі функцій елементів, яким користувач найчастіше віддає перевагу. Ці ознаки використовуються для відображення подібності одного предмета з іншим за допомогою рівняння подібності. Після цього він порівнює характеристики кожного елемента з профілем користувача та рекомендує ті, що мають високий ступінь подібності. Для системи, що базується на вмісті, потрібно побудувати профіль елемента, який є записом основних характеристик цього елемента. Ці характеристики легко виявити. Наприклад: у книзі, запис може містити список жанрів, авторів, рік випуску та країну [4].

Content-based фільтрація є найбільш простим і природним методом, який можна застосувати як рекомендацію, адже він не вимагає від користувача зворотного зв’язку. Іноді одного вибору достатньо, щоб рекомендувати користувачеві багато елементів системи. Цей підхід також розширюється

природним шляхом для випадків, коли інформація про товари добре організована та доступна, наприклад, фільми, пісні, продукти та книги. Але в той же час це також є обмеженням фільтрування на основі вмісту, оскільки опис товару не завжди присутній і це створює труднощі у вимірюванні подібності між предметами. Такі рекомендаційні системи мають обмеження для отримання подібних результатів і є статичними протягом часу.

Колаборативне фільтрування є найпопулярнішим і найпоширенішим методом рекомендацій. Його основою є те, що користувачі, яуі мають однаковий інтерес, схильні надавати однакові переваги новим та майбутнім елементам. Цей прийом працює за двома принципами. По-перше, він служить критерієм вибору групи подібних людей, думки яких будуть накопичені як основа для рекомендації (найближчих сусідів). По-друге, він також використовує ці думки, щоб сформувати більшу групу та мати більший вплив на рекомендацію. Прийоми спільного фільтрування передбачали дуже великі масиви даних та непрямі сфери застосування, такі як фінанси, прогнозування погоди, зондування довкілля, електронна комерція тощо [5].

Методи спільного фільтрування використовують набір даних переваг/оцінок, наданих користувачами для елементів, щоб передбачити нові елементи, які можуть сподобатися активному користувачеві. Модель може бути виражена у вигляді преференцій / рейтингової матриці порядку $m \times n$, де m – кількість користувачів $(U_1, U_2, U_3, ..., U_m)$, а n – кількість елементів $(I_1, I_2, I_3, ..., I_n)$, оцінених користувачами. Значення комірки r_{ij} матриці – це рейтинг елемента j, який було надано користувачем i. Ці оцінки можуть бути неявними (наприклад, придбання товару) або явними (відгуки користувача за шкалою k). Результат спільних прийомів може бути двох типів: перший – прогнозування r_{ij}, числове значення показує перевагу користувача елемент j, а друге – список рекомендацій з N найпопулярніших елементів, які користувачеві можуть сподобатися найбільше.

Основною функцією рекомендації є передбачення корисності елемента для користувача. Система рекомендацій характеризує, як користувач *U* зацікавлений у пункті з певним ступенем уподобання або рейтингу *r(u, i)*. Кожен користувач має свій профіль користувача, що описує його смак, симпатії, антипатії або оцінку чи відгук певному предмету. Кожен елемент характеризується набором функцій, наприклад, для книги набір функцій може містити ідентифікатор книги, жанри, авторів, дату виходу, країну [6].

Фільтрування, засноване на знаннях, використовує знання або інформацію користувачів, предметів та їх взаємозв'язки. Системи, що використовують данний метод описують, як конкретний предмет відповідає вимогам користувача. Це вимагає окремих знань про користувачів та елементів, специфічних для домену (предметної області).

Гібридний метод фільтрування – це той, що поєднує переваги двох або більшої кількості методів фільтрування та долає їх обмеження. Ці методи забезпечують доволі високу ефективність і покращують результати рекомендацій. Гібридні методи можуть використати одну з таких стратегій для розробки методу гібридного фільтрування:

Використовування колаборативних та content-based фільтрів для вироблення окремих рекомендацій, а потім створення лінійної комбінації цих двох рекомендацій, для надання єдиної рекомендації.

Колаборативне фільтрування може бути використано разом з результатами фільтрування на основі вмісту для обчислення подібності між користувачами та знайденими сусідами з метою прогнозування рекомендації.

Методи, засновані на вмісті, можуть бути додані до спільних характеристик фільтрування, таких як модель прихованих факторів із підходом, заснованим на вмісті [7].

Звичайний імовірнісний метод поєднання спільної та колаборативної техніки для прогнозування рекомендацій.

Проаналізувавши усі вищенаведені методи фільтрування данних для системи рекомендацій, було вирішено зупинитися на гібридному методі, скомбінувавши метод колаборативного фільтрування та метод фільтрування, що заснований на вмісті. Такий метод дозволить виключити недоліки кожного підметоду та досягнути рекомендацій, які якнайбільше співпадатимуть з інтересами коритстувача.

Використання рекомендаційних систем повинно бути чітко обґрунтовано у кожному випадку їх використання, адже це є досить затратною технологією для реалізацією та інтеграції [7].

Завдяки постійному розвитку штучного інтелекту, рекомендації більше не націлені на широку аудиторію чи навіть на певний її сегмент. Використовуючи механізми рекомендацій на основі глибокого навчання, маркетологи сьогодні можуть націлювати споживачів за допомогою гіпер-персоналізованих рекомендацій на індивідуальному рівні на основі таких показників, як особистість, місцезнаходження, інтереси, онлайн-поведінка в реальному часі тощо. Це не тільки дозволить маркетологам заохочувати онлайн-трафік за допомогою таргетної реклами або електронного маркетингу, а й зменшить роздратування клієнтів і рівень відтоку. Це і є основною перевагою та метою використання систем надання рекомендацій.

Однак, перш ніж реалізовувати таку систему в бізнесі, той хто створює її має подолати наступні недоліки та труднощі, пов’язані з цією технологією:

1) Потрібні значні інвестиції. Механізми рекомендацій — це значні інвестиції не лише фінансово, але й з точки зору часу: потрібен тривалий час і глибокий досвід, щоб створити ефективний механізм рекомендацій власними силами. Окрім необхідних спеціалістів із обробки даних та іншого персоналу спеціалістів, потрібно буде врахувати витрати на етап відкриття та аналізу

(включно з техніко-економічним обґрунтуванням, щоб переконатися, що це правильний шлях для вашого бізнесу), етап впровадження прототипу, мінімальний життєздатний розробка продукту (MVP), а потім остаточний випуск і розгортання. Навіть коли механізм рекомендацій активний, подорож ще не закінчена: система потребуватиме постійного моніторингу та вдосконалення, щоб переконатися, що він працює якомога оптимальніше, що призводить до постійних витрат.

2) Проблема «холодного старту». Покладатися виключно на дані користувача має свої недоліки, одним із яких є проблема «холодного запуску». Це проблема, коли новий користувач входить в систему або нові товари додаються до каталогу, і тому алгоритму буде важко передбачити смак або вподобання нового користувача або рейтинг нових товарів, що призводить до менш точних рекомендацій. Однак модель глибокого навчання здатна оптимізувати взаємозв’язки між клієнтом і продуктом, аналізуючи контекст продукту та деталей користувача, як-от опис продукту, зображення та поведінку користувачів. Потім він може давати змістовні рекомендації для кожного окремого продукту чи клієнта за різними сценаріями. Результатом цього є унікальний набір рекомендацій, який враховує всі ці змінні. Оскільки ці моделі глибокого навчання не сильно покладаються на дані поведінки користувачів, вони є рішенням проблеми холодного запуску [8].

3) Проблеми конфіденційності. Чим більше алгоритм знає про клієнта, тим точнішими будуть його рекомендації. Однак багато клієнтів не наважуються передавати особисту інформацію, особливо враховуючи кілька гучних випадків витоку даних клієнтів останнім часом. Однак без цих даних клієнта система рекомендацій не може ефективно працювати. Тому побудова довіри між бізнесом і клієнтами є ключовою. Багато компаній процвітають завдяки системам рекомендацій. Незважаючи на те, що вони створюють величезні можливості, життєво важливо знати про численні виклики, властиві цій технології, щоб використовувати її якомога повніше [8].

4) Відсутність можливостей аналізу даних. Як і всі технології на основі штучного інтелекту, механізми рекомендацій покладаються на дані. Якщо у вас немає високоякісних даних або якщо ви не можете правильно їх опрацювати та проаналізувати, ви не зможете максимально використати механізм рекомендацій. Щоб забезпечити найкращу якість даних, задайте собі чотири запитання: наскільки вони нещодавні? Наскільки вони «шумні»? Наскільки вона різноманітна? Як швидко ви можете внести нові дані в модель глибокого навчання? Механізми рекомендацій на основі глибокого навчання можуть вимагати високої обчислювальної складності. Якщо дані, які передаються в модель, менш точні або цінні, результат буде менш корисним. Отже, перш ніж інвестувати в механізми рекомендацій, переконайтеся, що ваш бізнес відповідає вимогам комплексної аналітики даних [9].

1.2 Аналіз існуючих систем надання рекомендацій для вибору книг

На даний час існує багато сервісів, що містять і можливість огляду книг і систему надання рекомендацій, що базуються на цих оглядах частково, або повністю.

Google Books – це інтернет-сервіс від Google Inc., яка здійснює пошук повного тексту книг і журналів, які Google відсканувала, перетворила їх на текст за допомогою оптичного розпізнавання символів (OCR) та зберігається у його цифровій базі даних. Книги надаються видавцями та авторами через Партнерську програму Google Books або бібліотечними партнерами Google через Бібліотечний проект. Крім того, Google співпрацює з кількома видавцями журналів для оцифровки своїх архівів.

Google Books може стати найбільшою онлайн-сукупністю людських знань, та сприяння демократизації знань. Однак його також критикують за потенційні

порушення авторських прав та відсутність редагування для виправлення багатьох помилок, внесених у скановані тексти процесом OCR [10].

Станом на жовтень 2015 року кількість відсканованих назв книг перевищувала 25 мільйонів, але процес сканування сповільнився в американських академічних бібліотеках. У 2010 році Google підрахував, що у світі існує близько 130 мільйонів різних заголовків, і заявив, що має намір сканувати їх усі. Станом на жовтень 2021 року Google відсвяткував 17 років Google Books і надав кількість відсканованих книг понад 40 мільйонів назв.

Також ця платформа має систему рекомендацій для надання наступних кних. Серед плюсів цієї системи варто відзначити, що вона має найбільший архів данних, серед яких можна знайти ці рекомендації.

Мінусом є те, що самі рекомендації базуються лише на вже прочитаних раніше книгах, тобто будуть підбиратися книги від прочитаних раніше авторів, або книги схожі за жанром [11].

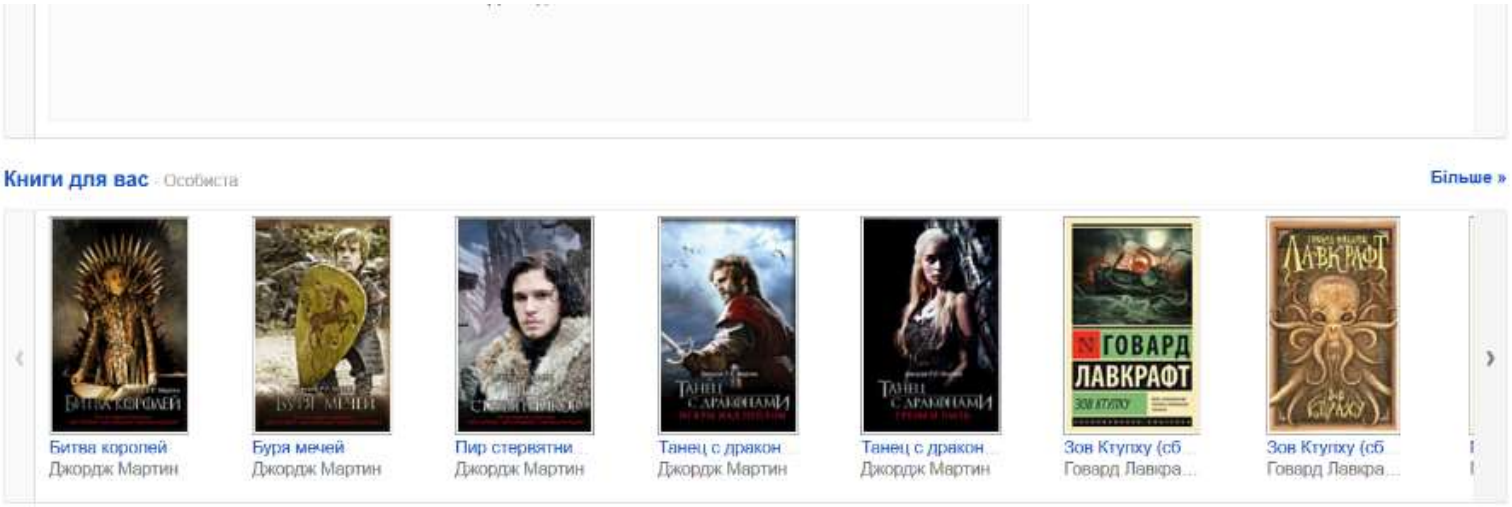


Рисунок 1.1 – Приклад роботи сервісу Google Books

Amazon Books - це мережа роздрібних книжкових магазинів, що належить інтернет-магазину Amazon. Перший магазин відкрився 2 листопада 2015 року в

Сіетлі, штат Вашингтон. Станом на 2022 рік Amazon Books має загалом 25 магазинів, які планують розширити в інші місця.

Зараз магазин працює у мережі Інтернет та має систему надання рекомендацій, що базується на оцінці та продажах.

Недоліки:

- система фільтрації базується більше на кількості продажів ніж на оцінці;
- можливість користуватися системою рекомендацій в повній мірі можна лише роблячи покупки на даному сервісі.

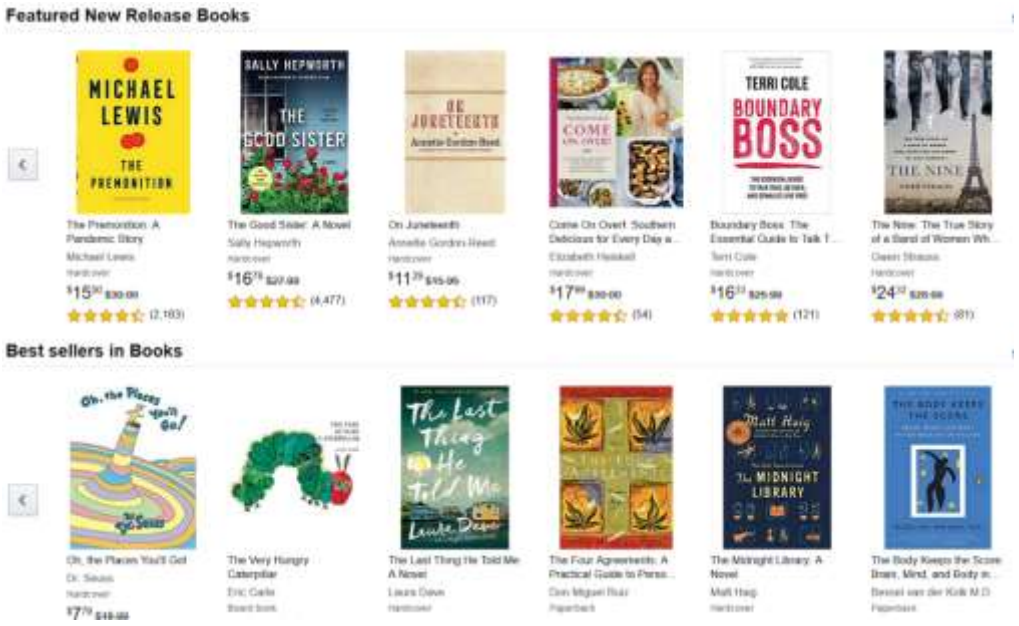


Рисунок 1.2 – Приклад роботи сервісу Google Books

Goodreads – це американський веб-сайт із соціальної каталогізації, який дозволяє людям здійснювати пошук у його базі даних книг, анотацій, цитат та відгуків. Користувачі можуть реєструватися та реєструвати книги для створення бібліотечних каталогів та списків читання. Вони також можуть створити власні групи пропозицій книг, опитувань, опитувань, блогів та дискусій. Офіси веб-сайту

розташовані в Сан-Франциско [9]. Компанія належить інтернет-магазину Amazon. До переваг програми можна віднести:

- наявність мобільної версії;
- велика база інформацій;
- зручний інтерфейс.

Недоліки:

- сервіс лише англомовний;
- сервіс надає доступ до інформації лише тим користувачам, у яких є аккаунт apple, amazon або facebook;
- відсутність або мала кількість української та іншої літератури.

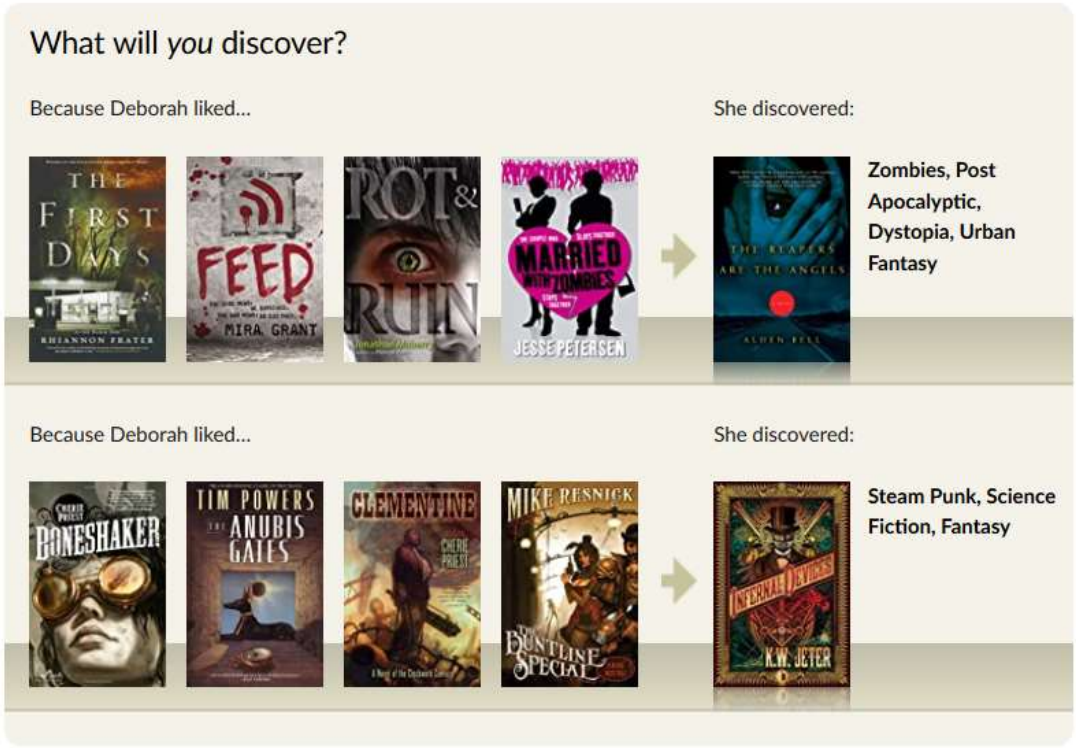


Рисунок 1.3 – Приклад роботи сервісу Goodreads

Reedsy Discovery - британська стартап-фірма, що займається онлайн-авторськими послугами, що базується в Лондоні, яка служить мостом, що об'єднує авторів та видавничих фрілансерів у видавничій галузі. Незважаючи на те, що вона

служить головним чином інформаційним центром для найвищих видавничих талантів, фірма розширилася, пропонуючи різноманітні інструменти та послуги для майбутніх авторів. Серед недоліків програми є такі:

- сервіс англомовний;
- мала кількість українських книг та авторів;
- немає можливості додавати нові книги до бази знань.

Рисунок 1.4 – Приклад роботи сервісу Reedsy Discovery

LoveReading – це британський сервіс огляду книг та рекомендацій [14]. Особливістю сервісу є те, що на рекомендації впливають не лише оцінки користувачів, але й критики та експерти, що вносить значну долю в рекомендації.

До переваг програми LoveReading віднесемо:

- Зручний інтерфейс та дизайн;
- Можливість додавання влисних книг та підбірок;
- максимальне заохочення користувача у процес огляду.

До недоліків програми LoveReading віднесемо те, що сервіс переважно

британський, тому рекомендації формуються користувачами з одного регіону.

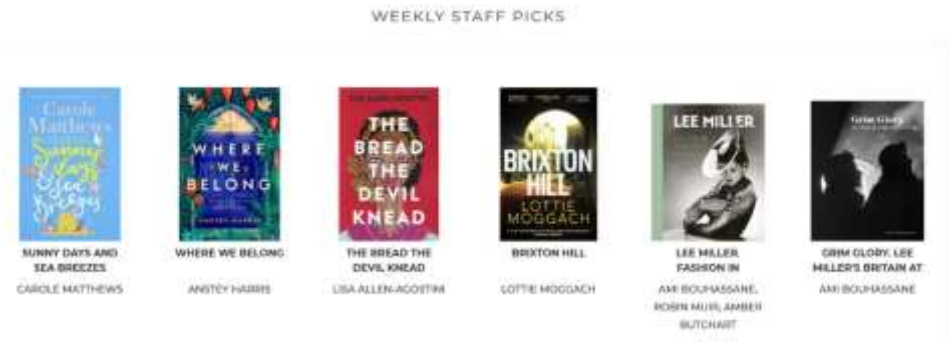


Рисунок 1.5 – Приклад роботи сервісу LoveReading

BookBub - це служба пошуку книг, яка була створена, щоб допомогти читачам знаходити нові книги та авторів [14]. Компанія пропонує безкоштовні електронні книги зі знижкою, вибрані редакційною командою, а також рекомендації щодо книг, оновлення від авторів та статті про книги. Послуга безкоштовна для читачів і включає веб-сайт та персоналізовані розсилки електронною поштою. The Guardian назвав BookBub "Основою електронних книг". BookBub має понад 15 мільйонів користувачів у США, Великобританії, Канаді та Австралії. Для видавців та авторів BookBub пропонує маркетингові інструменти, які призначені допомогти їм охопити читачів та продати більше книг. Компанія також управляє роздрібною торгівлею аудіокнигами під назвою Chirp.

Недоліки BookBub:

- доступна лише США;
- лише англомовні книги;
- відсутність кураторських оглядів.

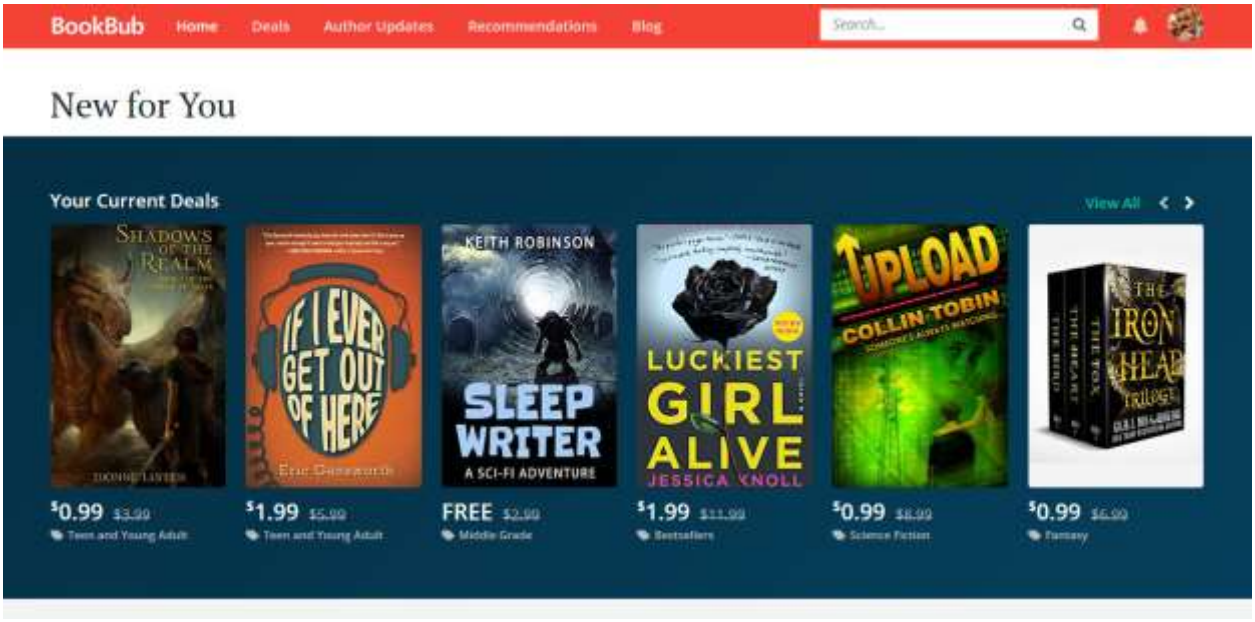


Рисунок. 1.6 – Приклад роботи сервісу BookBub

Серед проаналізованих програмних систем, LoveReading та GoodReads є максимально наближеними за своїми характеристиками до розроблюваного програмного модуля надання рекомендацій для платформи огляду книг. Саме тому оберемо їх як прототипи [10].

1.3 Постановка задачі створення інформаційної технології надання рекомендацій для платформи огляду книг

Модуль, що буде розроблений в ході дослідження, має бути представлений у вигляді веб додатку в бекенд частиною та з графічним інтерфейсом у браузері.

Так як для якісного тренування нейронної мережі повинен бути використаний великий набір даних, користувачу буде не раціонально створювати його самому. У процесі роботи програми система повинна самостійно навчатись на основі введених користувачем даних, а для тестування системи такий набір буде згенеровано автоматично.

Для доведення наукової актуальності необхідним буде модуль, який дозволить автоматизувати процес тестування роботи прогами на великому, відформатованому наборі даних.

Також необхідно розробити можливість валідації – тестування на невідформатованому наборі даних, що містить зображення різної роздільної здатності із можливими наявними вадами (шум) на них.

Отже, в кінцевому підсумку, розроблена інформаційна технологія забезпечить можливість на практиці показати результат проведеного в ході магістерської дипломної роботи дослідження.

1.4 Висновки

Проаналізовано стан сучасних можливостей надання рекомендацій, виконаний на основі багатьох літературних джерел та інформації мережі Інтернет, виявив велику кількість розроблених сучасних методів фільтрації даних. У ході аналізу було вивчено питання різних принципів розробки систем надання рекомендацій та аналіз переваг та недоліків використання машинного навчання для вирішення питань даної розробки. Виконано огляд на порівняльний аналіз існуючих технологій надання рекомендацій, для вибору книг.

Доведено доцільність розробки інформаційної системи, представленої у вигляді програмного додатку на основі гібридної математичної моделі, яка краще аналізує потреби користувача та вирішує проблему холодного старту.

Наведено і обґрунтовано вимоги до розроблюваного програмного продукту. Зроблені підсумки будуть враховані при розробці та проектуванні власної системи надання рекомендацій для платформи огляду книг.

2 РОЗРОБКА ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ СИСТЕМИ НАДАННЯ РЕКОМЕНДАЦІЙ ДЛЯ ПЛАТФОРМИ ОГЛЯДУ КНИГ

2.1 Розробка методу розв’язання задачі надання рекомендацій для вибору книг

Система рекомендацій була визначена як засіб допомоги та розширення соціального процесу використання рекомендацій інших осіб для прийняття вибору, коли у користувача системи немає достатніх особистих знань або досвіду щодо альтернатив. Системи рекомендацій вирішують проблему перевантаження інформацією, з якою зазвичай стикаються користувачі, надаючи їм персоналізований ексклюзивний контент і рекомендації щодо послуг.

Типова система рекомендацій складається з таких компонентів:

- генерація кандидатів;
- підрахунок балів;
- перерахування.

Під час першого етапу система починає з величезного набору потенційно рекомендованих кандидатів і генерує набагато меншу кількість кандидатів, завдяки фільтрам. Наприклад, генератор кандидатів в Google Book Store скорочує мільярди книг до сотень або тисяч. Модель повинна швидко оцінювати запити, враховуючи величезні розміри бібліотеки даних. Ця модель може надавати кілька генераторів кандидатів, кожен з яких висуває різні підмножини кандидатів [13].

Далі система оцінює та сортує кандидатів у рекомендації, щоб вибрати набір елементів (зазвичай близько 10) для відображення користувачеві. Якщо метод оцінює відносно невелику підмножину елементів, система може використовувати більш точну модель на основі додаткових запитів.

І, нарешті, система повинна враховувати додаткові обмеження для остаточного рейтингу. Наприклад, система видаляє елементи, які явно не сподобалися користувачеві, або підвищує рейтинг більш свіжого контенту. Пересортування також може допомогти забезпечити різноманітність, свіжість і справедливість [14].

2.2 Обґрунтування вибору порівняльного алгоритму для обраного методу фільтрації даних

Колаборативні та гібридні методи фільтрації працюють із матрицею взаємодії, яку також можна назвати матрицею рейтингу в тих рідкісних випадках, коли користувачі надають явну оцінку елементів. Завдання машинного навчання полягає в тому, щоб вивчити функцію, яка передбачає корисність предметів для кожного користувача. Матриця зазвичай величезна, дуже розріджена і більшість значень відсутні.

З іншого боку, спільна фільтрація використовує схожість між користувачами, щоб відфільтрувати елементи, які можна рекомендувати. Основна ідея полягає в тому, що користувачам зі схожою історією позитивних взаємодій також сподобаються схожі речі. Колаборативна фільтрація може бути змодельована як задача заповнення матриці, яку можна візуалізувати у вигляді таблиці. Значення в таблиці представляють рейтинги користувачів для конкретних елементів рекомендації, і очевидно, що один користувач не може мати рейтинг для всіх елементів. Задача полягає в тому, щоб заповнити пропущені через це клітинки [15].

Найпростіший алгоритм обчислює косинус або кореляційну подібність рядків (користувачів) або стовпців (елементів) і рекомендує елементи, якими користуються k — найближчі сусіди. Формулу коредяційної подібності можна подати у вигляді формули 2.1:

$$(f * g)[m,n] = \sum_{k,l} f[m-k, n-l] * g[k,l], \qquad (2.1)$$

Методи, засновані на матричній фаторизації, намагаються зменшити розмірність матриці взаємодії та апроксимувати її двома або більше малими матрицями з k – компонентами [16].

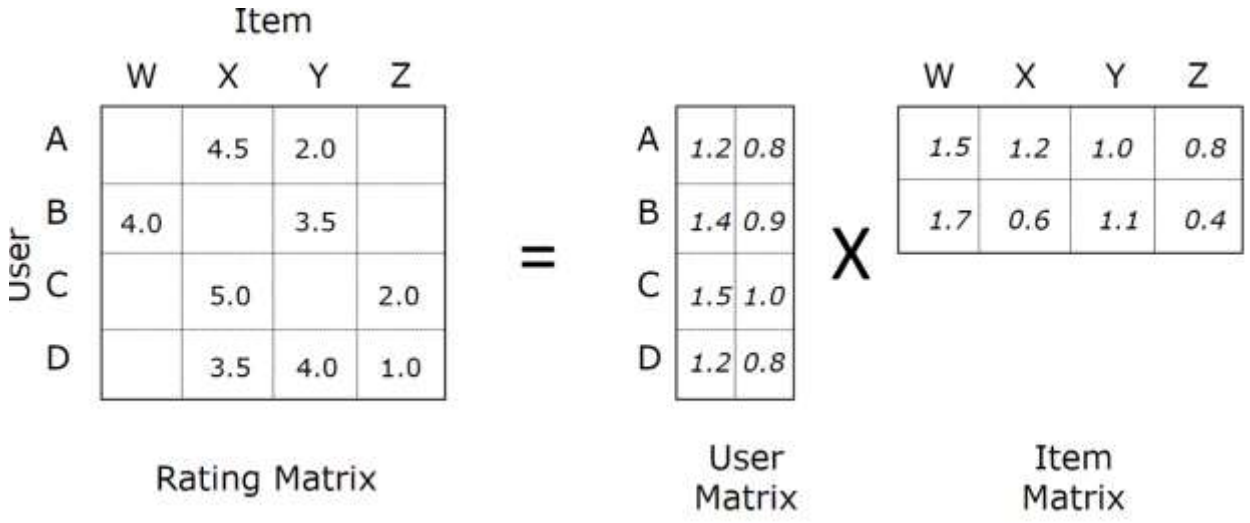


Рисунок 2.1 – Матрична факторизація

Помноживши відповідний рядок і стовпець, ви прогнозуєте рейтинг елемента за користувачем. Помилка навчання може бути отримана шляхом порівняння непорожніх оцінок із прогнозованими оцінками. Можна також упорядкувати втрату навчання, додавши термін штрафу, зберігаючи значення латентних векторів на низькому рівні.

Цю матрицю можна досить легко представити як граф, зокрема як дводольний граф. Це простий граф, який складається з двох наборів вершин. Один набір – це набір користувачів, а другий - набір елементів рекомендацій.

Однією з безпосередніх переваг графових нейронних мереж порівняно з матричною факторизацією є те, що вони здатні агрегувати зв’язки між багатьма елементами рекомендацій, тоді як матричні представлення, як правило, враховують лише прямі зв'язки. Крім того, такий граф може бути динамічним, що дозволяє рекомендаціям бути більш точними для часто змінюваного контенту[4].

Розглянемо матрицю рейтингів M розміру u × v , де u - кількість користувачів, v - кількість елементів рекомендацій. В елементах Mij цієї матриці зберігають або спостережувану оцінку користувача (користувач і оцінив елемент j) з множини дискретних можливих значень оцінки, або відсутність оцінки (зберігається зі значенням 0). Ілюстрація наведена на рисунку (рис. 1). Завдання заповнення матриці або рекомендації можна розглядати як прогнозування значення нульових записів у матриці M [18].

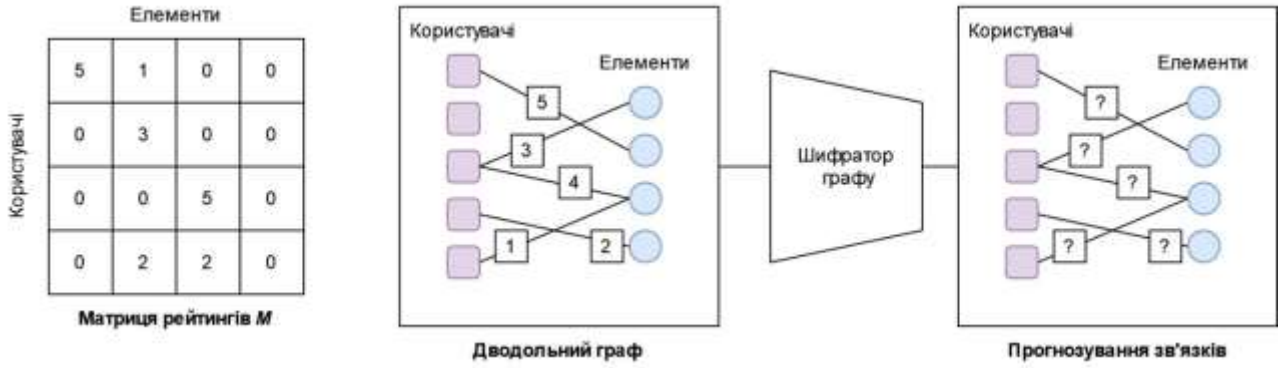


Рисунок 2.2 – Загальна схема роботи методу
колаборативної фільтрації на графі

Задачу заповнення матриці, або, інакше кажучи, надання рекомендації, можна представити як задачу прогнозування зв'язків на дводольному графі взаємодії "користувач-елемент" (рисунок 1). Дані про взаємодію можуть бути представлені неорієнтованим графом G = (W, E, R) з сутностями, що складаються з колекції вершин користувачів ui ∈ U, де i ∈ {1, ..., Nu}, та вузлів елементів vj ∈ V,

де j ∈ {1, ..., Nv }, таких, що U ∪ V = W. Ребра (ui, r, vj) ∈ E містять мітки, які представляють порядкові рівні рейтингу, тобто оцінку елементів користувачами, наприклад, r ∈ {1, ..., R} = R. Користувачі та елементи рекомендацій зображуються як вершини графа. Основною частиною моделі є шифратор, завданням якого є стиснення інформації про ребра таким чином, щоб за стисненим представленням можна було його відновити [18].

Для цього дводольний граф пропускається через декілька шарів проходження повідомлень для того, щоб вивчити представлення для кожного користувача та об'єкту. Ці шари також враховують різні типи рейтингів і застосовують різні перетворення для кожного типу. Накінець, виконується прогнозування зв'язків, що базується на кінцевому стані користувачів та елементів.

Ця схема може бути зображена у вигляді формули (2.5):

$$p\left(M_{ij} = r\right) = \frac{e^{u_i^T Q_r v_j}}{\sum_{s \in R} e^{u_i^T Q_r v_j}} \tag{2.1}$$

Де, добуток користувачів – u та елементів – v, множиться на добуток трансформації Q, що ми отримуємо з нейронної мережі. Потім, Softmax використовується для прогнозування ймовірностей для різних типів ребер [6].

Висновки. Зображено загальну схему роботи задачі заповнення матриці рейтингів в рекомендаційній системі з використанням графа. Приведено порівняння з звичайним алгоритмом матричної факторизації. Доведено доцільність використання графа для вирішення проблеми генерації оцінок елементів в рекомендаційних системах, а саме завдяки перевагам над іншими методами в точності наданих рекомендацій [17].

2.3 Обґрунтування вибору методу штучного інтелекту для інформаційної технології надання рекомендацій для платформи огляду книг

Машинне навчання — це підгалузь штучного інтелекту, яка зосереджується на створенні комп'ютерних алгоритмів, які можуть обробляти величезні набори даних, ідентифікувати повторювані шаблони та кореляції між кількома змінними та створювати математичні моделі, що їх відображають.

Системи машинного навчання, а також додатки навчання з підкріпленням, що використовують ці алгоритми, можуть фактично розширити свої можливості з часом через досвід. Чим більше даних вони обробляють, тим більше зв'язків між точками даних вони помічають і тим краще вони налаштовуватимуть свої моделі. Такі моделі, які також зазвичай використовуються для прогнозної аналітики в маркетингу, являють собою вікно в те, що думають клієнти, оскільки вони дозволяють нам сформулювати логіку конкретних моделей купівлі та пролити світло на поточні тенденції продажів або навіть передбачити розвиток у майбутньому. Наприклад, рішення на базі машинного навчання може помітити постійний зв'язок між віком клієнтів та їх перевагою щодо однієї марки над іншою.

З точки зору маркетингу, системи повинні сегментувати користувача, а саме класифікувати вас за певним архетипом клієнта або покупця відповідно до певних характеристик (схеми купівлі, інтереси, стать тощо) і націлити його на відповідну пропозицію продукту, що відображає його архетип [19].

На віртуальних ринках системи рекомендацій відіграють подібну роль, замінюючи продавців-помічників у сегментації та рекомендації продуктів клієнтам. Основна відмінність полягає в тому, що люди-продавці керуються своєю інтуїцією та досвідом, щоб дослідити невелику частку вищезгаданих змінних під час короткої розмови з покупцями. Натомість механізми рекомендацій покладаються на машинне навчання для обробки величезних наборів даних клієнтів і розглядають ширший діапазон параметрів для виконання цього процесу

класифікації та націлювання. До них належать поведінка веб-переглядача, історія покупок, використання вмісту, особиста інформація з профілів користувачів, огляди продуктів і пристрої доступу [20].

Крім того, алгоритми машинного навчання можуть враховувати величезний діапазон суто контекстних параметрів, які не пов’язані строго з клієнтами. Наприклад, із наближенням грудня механізми рекомендацій на основі машинного навчання великого веб-магазину починають рекомендувати типові різдвяні продукти. З іншого боку, потокова платформа може адаптувати свої рекомендації з дня тижня, пропонуючи сімейні фільми та документальні фільми на вихідних.

Найпопулярнішим алгоритмом навчання є стохастичний градієнтний спуск, що мінімізує втрати шляхом оновлення градієнта як стовпців, так і рядків p a q матриць.

Крім того, можна використовувати метод чергування найменших квадратів, який ітеративно оптимізує матрицю p і матрицю q за допомогою загального кроку найменших квадратів [21].

Правила асоціації також можна використовувати для рекомендації. Предмети, які часто споживаються разом, з’єднані ребром на графіку. Ви можете побачити кластери бестселерів (щільно пов’язані товари, з якими взаємодіяли майже всі) і невеликі відокремлені кластери нішевого вмісту.

Правила, видобуті з матриці взаємодії, повинні мати принаймні мінімальну підтримку та довіру. Підтримка пов’язана з частотою появи — наслідки бестселерів мають високу підтримку. Висока довіра означає, що правила не часто порушуються.

Рейтингова матриця також може бути стиснута нейронною мережею. Так званий автокодер дуже схожий на матричну факторізацію. Глибокі автокодери з кількома прихованими шарами та нелінійністю потужніші, але їх важче навчити. Нейронна мережа також може використовуватися для попередньої обробки

атрибутів елемента, щоб ми могли поєднувати підходи на основі вмісту та підходи для співпраці [22].

2.4 Обґрунтування вибору архітектури нейронної мережі для рекомендаційної системи та наведення її математичної моделі

Спільна фільтрація базується на припущенні, що схожі користувачі люблять схожі продукти. Наприклад, якщо користувачеві А подобається продукт 1, а користувачеві В схожий на користувача А, то користувачеві В, ймовірно, також сподобається продукт 1. Два користувача схожі, якщо їм подобаються однакові товари.

Наразі всі найсучасніші системи рекомендацій використовують глибоке навчання. Зокрема, Neural Collaborative Filtering (2017) поєднує в собі нелінійність нейронних мереж і матричну факторізацію. Модель створена для максимального використання простору вбудовування, використовуючи його не лише для традиційної спільної фільтрації, але й для повністю підключеної глибокої нейронної мережі. Додаткова частина має фіксувати шаблони та функції, які може пропустити матрична факторизація [23].

Переглянемо об’єкти предметної області системи надання рекомендацій:

− Цільова змінна — оцінки можуть бути явними (тобто користувач залишає відгук) або неявними (тобто передбачають позитивний відгук, якщо користувач дивиться весь фільм), у будь-якому випадку вони необхідні.

− Особливості продукту — теги та описи елементів (тобто жанрів книг), які в основному використовуються в методах на основі вмісту.

− Профіль користувача — описова інформація про користувачів може бути демографічною (тобто стать і вік) або поведінковою (тобто вподобання, середній час перебування на екрані, найчастіший час використання), здебільшого використовується для рекомендацій на основі знань.

− Контекст — додаткова інформація щодо ситуації навколо рейтингу (тобто коли, де, історія пошуку), також часто включається в рекомендації на основі знань.

Сучасні системи рекомендацій поєднують усі ці пункти, коли роблять прогноз щодо смаку користувача [24].

Щоб оцінити рейтинги, ми можемо припустити, що матриця корисності є добутком двох менших матриць. Ця ідея має сенс, якщо ми думаємо, що більшість користувачів насправді реагують на невеликий набір функцій, що відносяться до набору доступних елементів. Наприклад, користувачам може сподобатися певний жанр фільму, актор або режисер. Рекомендаційну задачу можна сформулювати так:

1. Дано матрицю корисностей M з n рядків і m стовпців (n користувачів і m елементів),
2. Знайти матрицю U з n рядків і d стовпців,
3. Знайти матрицю V з d рядків і m стовпців,

Цей процес називається UV - розкладанням. Мірою близькості UV до M зазвичай є середньоквадратична помилка, де потрібно:

1. Просумувати за всіма відомими елементами M квадрат різниці між цими елементами та відповідними елементами UV.
2. Обчислити середнє значення цих квадратів
3. Обчислити квадратний корінь із середнього значення

Щоб мінімізувати таку помилку, потрібно мінімізувати першу частину, оскільки середнє значення суми є сумою середніх, а корінь квадрата помилки є мінімальним, якщо їх аргументи рівний [25].

Щоб апроксимувати M, ми починаємо з довільних U і V, потім коригуємо кожен елемент ітераційно, щоб мінімізувати RMSE. Припустимо, що P=UV, тоді:

$$p_{ij} = \sum_{k=1}^{d} u_{ik} v_{kj}$$

(2.1)

Де pij є елементом P у рядку i та стовпці j. є елементом P у рядку i та стовпці j.

2.5 Висновки з розділу

У даному розділі було визначено та сформульовано основні етапи процесу надання рекомендацій. Після проведеного дослідження існуючиш методів визначння ймовірних оцінок елементів рекомендацій, було розроблено метод генерації рейтингів з використанням графа на основі матричної факторизації.

Для реалізації штучного інтелекту підбору рекомендацій було обрано згорткову нейронну мережу, на основі архітектури якої буде побудована така архітектура мережі, що дозволить з достатньою точністю надавати доречні рекомендації.

Запропоновано структуру інформаційної технології для надання рекомендацій книг, побудованої на основі вдосконаленої архітектури згорткової нейронної мережі.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ СИСТЕМИ НАДАННЯ РЕКОМЕНДАЦІЙ ДЛЯ ПЛАТФОРМИ ОГЛЯДУ КНИГ

3.1 Розробка алгоритму роботи програмних засобів рекомендаційної системи

Для функціонування рекомендаційного модуля було вирішено використати один із алгоритмів колаборативної фільтрації, які, як правило, можна розділити на два класи: фільтрація на основі користувачів або на основі елементів. У будь-якому випадку будується матриця подібності. Для спільної фільтрації на основі користувачів матриця схожості користувачів складатиметься з певної метрики відстані, яка вимірює схожість між будь-якими двома парами користувачів [26].

Загальною метрикою відстані є подібність косинусів. Метрику можна сприймати геометрично, якщо трактувати рядок (стовпець) даного користувача (елемента) матриці оцінок як вектор. Для спільної фільтрації на основі користувачів подібність двох користувачів вимірюється як косинус кута між векторами двох користувачів. Для користувачів u та u′ косинусова подібність дорівнює:

$$sim(u,u') = \cos(\theta) = \frac{r_u r_{u\prime}}{|r_u||r_{u'}|} = \sum_i \frac{r_{ui} r_{u\prime i}}{\sqrt{\sum_i r_{ui}^2}\sqrt{\sum_i r_{u\prime i}^2}}$$

(3.1)

Дана формула може бути реалізована у вигляді циклу. Відштовхуючись від неї, можна побудувати алгоритм роботи рекомендаційного модуля:

Рисунок 3.1 – Схема загального алгоритму функціонування системи

Розглянемо алгоритм роботи програмного модуля надання рекомендацій, який наведений на рисунку 3.2:

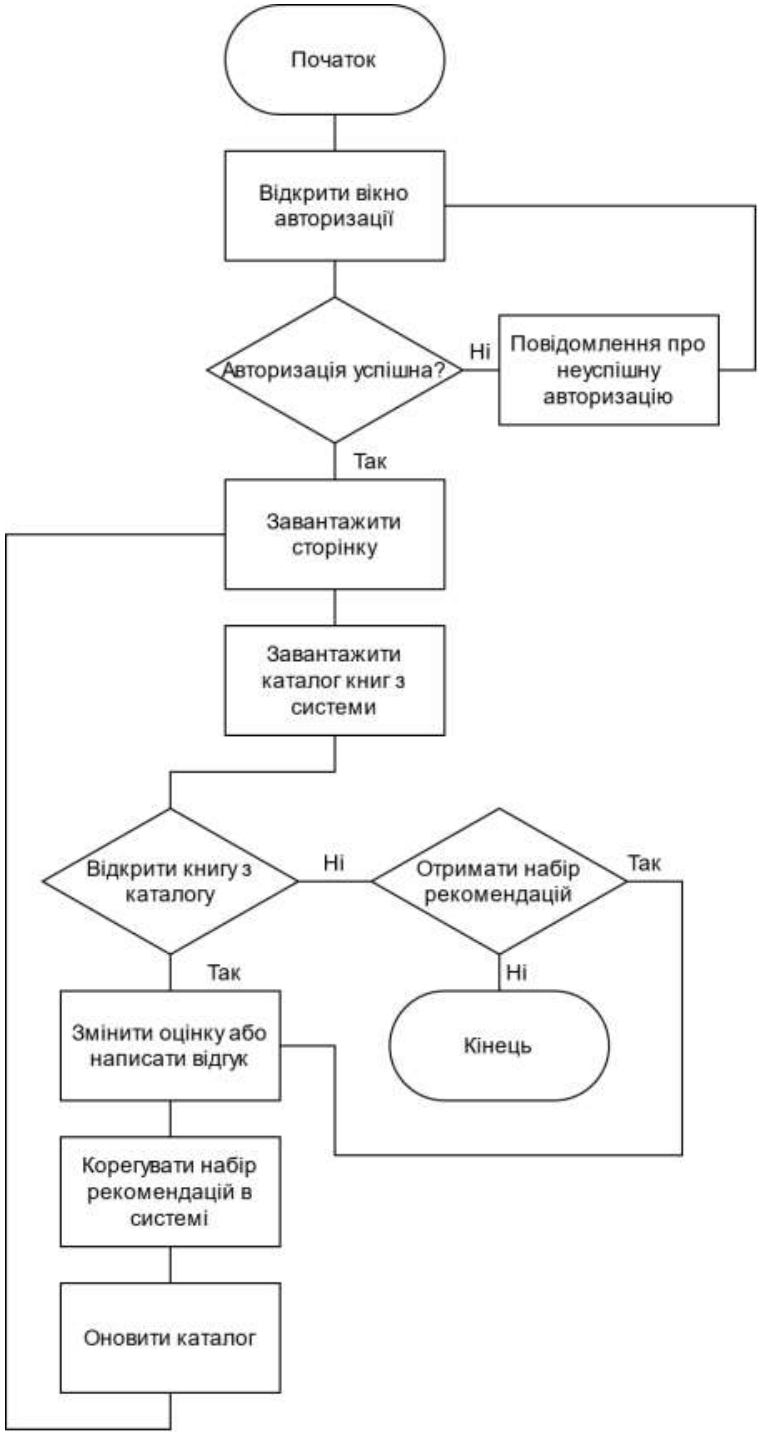


Рисунок 3.2 – Схема загального алгоритму функціонування системи

Алгоритм складається з таких кроків:

Крок 1. Перший етап – етап авторизації. Користувач повинен ввести свої деталі логіну, для того, щоб авторизуватись в системі.

Крок 2. Вибір користувача для авторизації/реєстрації.

Крок 2.1. Відкриття вікна для реєстрації нового користувача.

Крок 2.2. Перевірка правильності введених даних.

Крок 3. Спроба авторизації користувача.

Крок 3.1. Якщо авторизація успішна, тоді перехід до пункту 1.

Крок 4. Показ головної сторінки.

Крок 5. Завантажити каталог книг з бази даних.

Крок 5.1. Обрати книгу з каталогу.

Крок 5.2. Змінити оцінку обраної книги.

Крок 5.3. Коригувати набір рекомендацій в системі.

Крок 5.3. Оновити каталог на головній сторінці.

Крок 6. Завантажити список рекомендацій з системи

Крок 6.1. Перейти до кроку 5.

3.2 Обгрунтування вибору та розробка архітектури програмної реалізації

Монолітна архітектура додатку – це традиційний підхід до розробки програмного забезпечення, в якому вся функціональність системи базується на одній програмі як єдиному автономному блоці. У розробці програмного забезпечення цей єдиний блок буде означати єдину платформу [18].

У монолітному додатку всі функції керуються та обслуговуються в одному місці. Звичайно, додаток має свою внутрішню структуру, що складається з бази даних, клієнтського інтерфейсу, бізнес-логіки, але вона все ще залишається нерозривною одиницею. Його компоненти не потребують API для зв’язку. В архітектурі мікросервісів бізнес-логіка розбивається на легкі, самодостатні послуги. Кожна служба в рамках цього типу архітектури відповідає за певну бізнес-ціль. По суті, мікросервісна архітектура виглядає як конструкція Lego, яку можна

розкласти на ряд модулів. Взаємодія між компонентами системи забезпечується за допомогою API.

Значною перевагою монолітної архітектури є те, що її легше реалізувати, тоді як мікросервіси вимагають набагато більше зусиль. Простіше реалізувати бізнес-логіку із однією точкою входу, ніж турбуватись про її розбиття на декілька систем. Також монолітні програми зазвичай легше розгортати, оскільки вони не потребують додаткового налаштування зв'язків та спілкування сервісів.

Проте останні декілька років все більш поширеною є мікросервісна архітектура додатків, яка має декілька дуже суттєвих переваг. Перш за все, мікросервісні системи є дуже автономними один від одного. Можна створювати незалежні сервіси для кожної бізнес-цілі та розгортати їх окремо [27].

Таким чином можна розпаралелити роботу над цілісною системою, оскільки кожна команда буде розробляти власний сервіс для реалізації конкретної мети. Якщо станеться так, що один із мікросервісів буде недоступний чи збоїтиме – це не на стільки суттєво вплине на роботу всієї системи, як збій в монолітному додатку. Монолітний додаток не може функціонувати коли певна його частина не робоча, а мікросервіси, налаштовані певним чином, можуть обробляти такі критичні ситуації. Ще однією суттєвою перевагою мікросервісів є маштабованість. Мікросервіси можуть бути розподілені на різних серверах як і навантаження на них. Мікросервіси можуть бути маштабовані як горизонтально (додаючи більше ресурсів самому серверу), так і вертикально (збільшуючи кількість запущених екземплярів мікросервісу).

Проте проектування такої розподіленої системи може бути складним завданням і багато часу і зусиль витрачатиметься для розробки з'єднань всередині системи.

Перш за все порівняємо мікросервісну та монолітну архітектури за певними суттєвими характеристиками.

Таблиця 3.1 – Порівняння монолітної та мікросервісної архітектури

Критерій порівняння	Монолітна	Мікросервісна
Надійність	Один збій може вивести з ладу всю систему	Збій одного мікросервісу не пливає на інші
Розгортання	Розгортається вся система один раз і налаштовується за потреби	Можливість розгортання та відкату кожного мікросервісу окремо
Маштабованість	Тільки вертикальна	Вертикальна та горизонтально
Можливість застосовувати різні технології	Неможливо впровадити нові технології, мови програмування	Можливість використовувати різні мови, технології для різних бізнес-потреб
Розробка	Команди залучаються до процесу розробки одночасно	Різні команди можуть працювати над різними елементами рішення
Оновлення системи	Оновлення можуть зайняти деякий час через внутрішні залежності в архітектурі та інших розробників, які працюють одночасно	Швидкі оновлення через мінімалістичний характер модулів через автономний характер послуг
Тестування	Необхідність end-to-end тестування	Кожен компонент потрібно тестувати окремо
Безпека	Безпечна обробка та передача даних легше на системному рівні	Зв'язок між мікросервісами через API викликає проблеми безпеки

Порівнявши два архітектурних підходи було обрано мікросервісний для реалізації задачі даного проекту, оскільки такий підхід дозволить розділити основну функціональність платформи огляду із модулем для надання рекомендацій, а також, написати модуль мовою програмування JAVA та з легкістю інтегрувати його з основним додатком за допомогою API.

Модель–вигляд–контролер (Model-view-controller, MVC) — архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення.

Даний шаблон проектування має на основі поділ свієї системи на три логічні взаємопов’язані частини: модель даних, вигляд та модуль керування. Шаблон практично має застосування коли необхідне відокремлення даних користувача від роботи з даними, оскільки це зменшить кількість змін у моделях користувача та його програмному інтерфейсі. Зміни моделі даних такоє можуть змінюватись без змін інтерфейсу користувача.

Цей архітектурний шаблон проектування є одним із найпоширеніших шаблонів проектування, оскільки він дозволяє легко інкапсулювати логіку збереження, обробки та представлення даних:

- Модель - модель представляє об'єкт або JAVA POJO (plain old java object), що несе дані. Він також може мати логіку для оновлення контролера, якщо його дані змінюються.
- Вигляд - подання представляє візуалізацію даних, що містять модель.
- Контролер - діє як на модель, так і на вигляд. Він контролює потік даних в об'єкт моделі та оновлює подання, коли дані змінюються. Це тримає огляд та модель окремо.

Основними перевагами даного архітетурного паттерну є:

- Традиційно використовується для графічних інтерфейсів користувача (GUI).
- Популярний у веб-додатках.

- Обов'язки MVC розподілені між клієнтом та сервером, сумісно з архітектурою веб-додатків..
- MVC є корисним шаблоном проектування на етапі планування
- Поділ проблем: цей код ділиться залежно від функції на модель, вигляд або сегмент контролера.
- Усуває потребу в лишніх залежностях.
- Код можна перевикористовувати без змін.
- MVC робить класи моделей багаторазовими без змін.
- Розширюваний код.
- Висока згуртованість.
- Кожен сегмент можна протестувати незалежно (модель, вигляд, контролер)

Сегмент «Модель» відомий як найнижчий рівень, що означає, що він відповідає за збереження даних. Модель насправді підключена до бази даних, тому все, контролює усе, що відбувається з даними. Додавання або отримання даних здійснюється в компоненті моделі. Він відповідає на запити контролера, оскільки контролер ніколи не розмовляє з базою даних сам. Модель звертається до бази даних або навпаки, а потім передає необхідні дані контролеру. Примітка: модель ніколи не має безпосереднього зв’язку з видом [26].

Основною метою шаблону є отримати гнучкий та маштабований дизайн програмного забезпечення, який матиме можливість полегшувати та гнучко інтегрувати в існуючу систему подільші зміни чи розширення фнукціоналу програми, а також такий підхід полегшує можливість повторного використання модулів та шаблонів реалізованого програмного збезпечення. Використання цього шаблону у великих ентерпрайзних системах також полегшує розуміння та функціонування самої системи, адже робить її менш складними і більш зрозумілими. У рамках архітектурного шаблону модель–вигляд–контролер (MVC) програма поділяється на три окремі, але взаємопов'язані частини з розподілом функцій між компонентами. Модель (Model) відповідає за зберігання даних та їх

структуру. Вигляд (View) відповідальний за представлення цих даних користувачеві, тобто інтерфейс програми. Контролер (Controller) керує компонентами, отримує сигнали у вигляді реакції на дії користувача (зміна положення курсора миші, натискання кнопки, ввід даних в текстове поле) і передає дані у модель [24]. Центральним компоментом шаблону є модель, оскільки вона відображає поведінку додатку, а тобто інтерфейс, який є незалежним від користувача. Цей компонент працює безпосередньо із керуванням даними користувача, логікою компонентів та правилами. Він є найбільш чутливим до хакерських атак а також має справу із вразливими та айбільш важливими даними проекту.

Вигляд – це будь-яке представлення інформації, одержуване на виході, наприклад графік чи діаграма, також інтерфейс користувача чи інтерфейс додакту. Одночасно можуть співіснувати кілька представлень однієї і тієї ж інформації, наприклад різні екрани додатку для менеджера, для звчиайного користувача, для користувача із адмінськими правами, для дитина тощо. Контролер одержує вхідні дані й перетворює їх на команди для моделі чи вигляду. Модель інкапсулює ядро даних і основний функціонал їхньої обробки і не залежить від процесу вводу чи виводу даних. Вигляд може мати декілька взаємопов'язаних областей, наприклад різні таблиці і поля форм, в яких відображаються дані. Контролер має на меті реакцію та відстеження певних подій, які приходять в результаті користування користувачем додатком. Ці вхідні точки можуть бути структуровані за схожістю запитів користувача, які можна згрупувати в окремі класи [27].

Наприклад у шаблонному MVC-проекті може бути користувацький контролер, що містить групу методів, пов'язаних з управлінням обліковим записом користувача, таких як реєстрація, авторизація, редагування профілю та зміна пароля, а також контролер для менеджера чи адміністратора, який матиме функції видалення користувача, надання йому певних прав тощо. Зареєстровані події транслюються в різні запити, що спрямовуються компонентам моделі або об'єктам,

відповідальним за відображення даних. Відокремлення моделі від вигляду даних дозволяє незалежно використовувати різні компоненти для відображення інформації.

Архітектура шаблону MVC для загального проекту зображена на Рисунку 3.3.

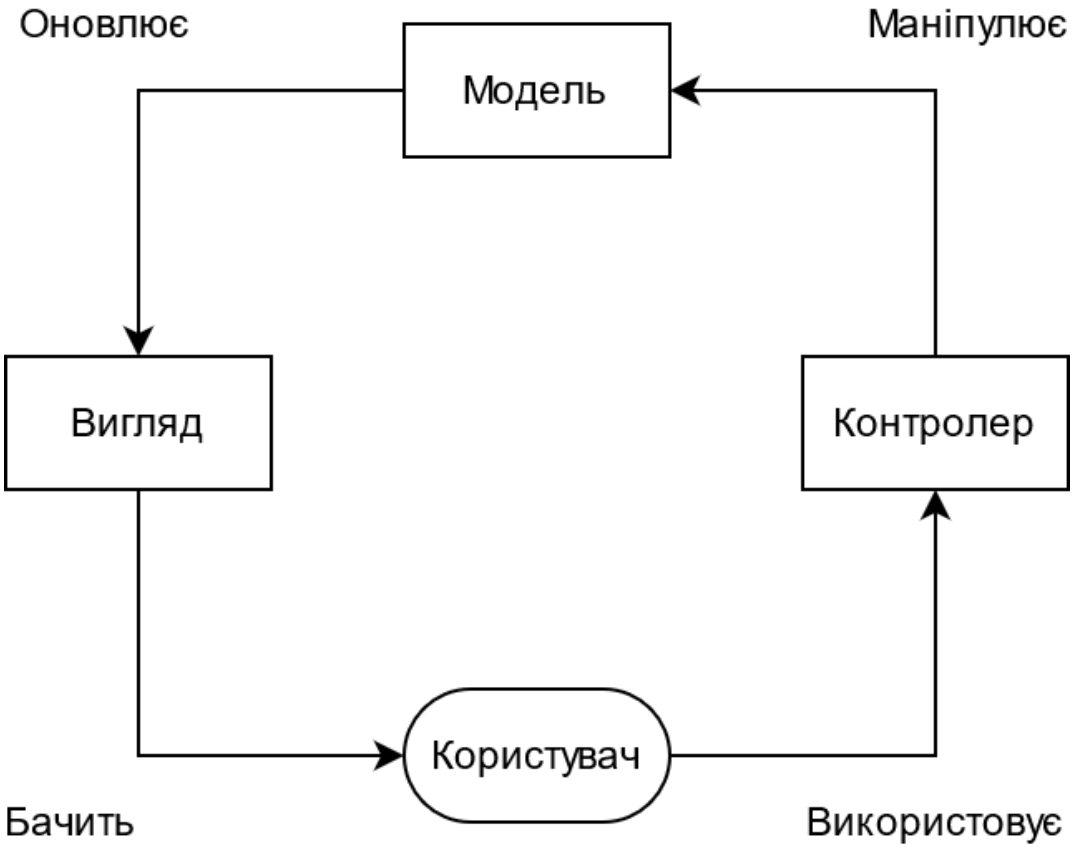


Рисунок 3.3 –Архітектура шаблону MVC для загального проекту

Проект реалізує модель MVP для зручності реалізації клієнтської частини. Проект має MainView, MainPresenter та MainModel класи, які безпосередньо спілкуються один із одним. У даному випадку MainView є графічним інтерфейсом, із яким буде взаємодіяти користувач. Наприклад, користувач натискає на книгу папку, очікуючи побачити інформацію про неї. Дане натиснення на папку обробляє MainView та надсилає у MainPresenter клас, який надсилає запит до MainModel.

MainModel клас відповідальний за формування та надсилання запитів до серверу, використовуючи серверні API, та отримання відповіді. Безпосередньо MainModel клас не відповідає за обробку чи валідацію даних. Після отримання відповіді від сервера дані надсилаються знову у MainPresenter, який відповідає за їх обробку. У даному прикладі відповідь із серверу прийшла у форматі JSON. Для того, щоб мати можливість показати дані користувачу, їх необхідно конвертувати до необхідного вигляду. MainPresenter конвертує отриману відповідь у необхідний для відображення користувачу формат та надсилає валідні конвертовані дані до MainView. MainView відображає повернені дані та очікує наступні дії користувача [27].

На Рисунку 2.6 зображено розроблену архітектуру шаблону MVP для даного проекту.

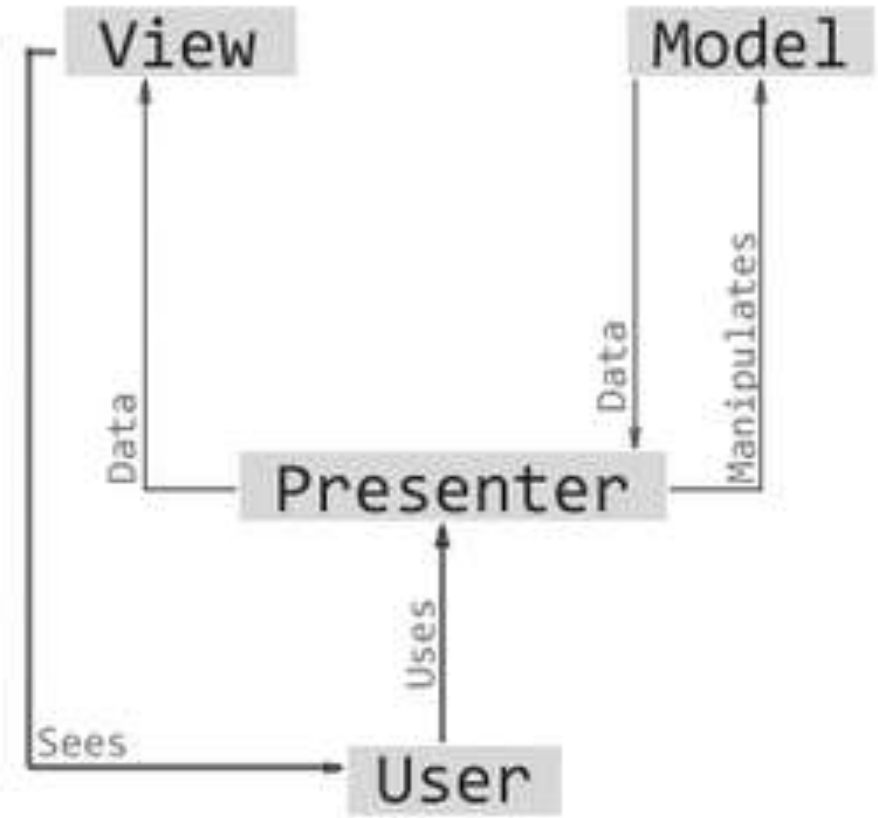


Рисунок 3.4 – Архітектура шаблону MVP

Мікросервіс реалізує шаблон MVC для зручного спілкування клієнтської та серверної частин. Мікросервіс має два основних контролери:

- BookController;
- ReviewController.

Запити, надіслані із метою отримати інформацію про рецензію чи додавання нової надіслані до ReviewController, який має /review мапінг та всі необхідні методи щодо роботи із рецензіями. Запити, надіслані із метою отримати чи відредагувати інформацію про книгу будуть надіслані до BookController, який має /book мапінг та всі необхідні методи щодо роботи із книгою.

На рівні контролера перевіряється валідність введених користувачем ідентифікаторів та правильний формат рецензії чи інформації про книгу. Далі введені дані передаються у сервіс.

Проект має два сервіси:

- BookService;
- ReviewService.

Кожен із них реалізує власний інтерфейс сервісу із необхідними методами для взаємодії із книгами чи рецензією, відповідно.

На рівні сервісу перевіряється можливість збереження введених користувачем даних. Наприклад, при виконанні запиту на збереження нової рецензії для вказаної книги на рівні сервісу перевіряється чи є уже рецензія від такого користувача для такої самої книги. Рецензія має бути унікальною для забезпечення більш простої взаємодії із ними. В тому випадку, якщо рецензія уже існує, на рівень контролера передається помилка, яка сповіщає про наявність такої рецензії в проекті. Таким чином, запит на створення нової рецензії не буде створений, доки фото не матиме необхідні валідні параметри [28].

Якщо усі вказані дані правильні, вони надсилаються на рівень репозиторію. У проекті є два репозиторії:

- BookRepository;
- ReviewRepository.

Вони реалізують інтерфейс JpaRepository, який значно спрощує роботу із базою даних. Після виконання запитів до бази даних, отримані дані повертаються до сервісу та перевіряються на валідність. Наприклад, якщо рецензія не знайдена для вказаної книги, репозиторій не поверне нічого, а сервіс на основі цього поверне відповідну помилку. Ці дані надсилаються знову до контролера, звідки передаються клієнтській частині.

Мікросервіс для надання рекомендацій не матиме спілкування із клієнтською частиною додатку. Його основною функцією буде надання списку рекомендованих книг для певного користувача.

Сервіс має один ендпоінт, який має /recomend мапінг та надсилає користувачу ідентифікатори книг. Основний клас проекту має назву Recomender та він відповідає за обрахування і надання ідентифікаторам. Книги ніяким чином не будуть валідуватись, оскільки очікується що передані дані є вже правильними та консистентними.

Після отримання фото через API воно передається у інший клас – BookAnalyzer, який відповідає за аналіз книг та збереження їх архетипу. У цьому класі книга аналізується за допомогою нейромережі, яка є у загальнодоступній бібліотеці RONX. Для цього необхідно під’єднати певні додаткові модулі у проект. Методи бібліотеки конфігуруються необхідними для даного проекту значеннями та у бібліотеку передається книга. Метод classify() даного класу повертає лише текстові значення класифікацій книги.

Щоб сервіс працював правильно, потрібно бути авторизованим у системі. Авторизація використовується для того, щоб закріпити результати оцінок за певним користувачем, адже потім ці дані будуть використовуватись для закріплення за ними оцінок огляду книг та подальшому використання цих оцінок в наданні рекомендацій.

Метод recommend_books() викликається після того, як книги проаналізоані. Йому передається самі теги та результат класифікації стрічкового типу. При успішному наданні рекомендаціх повертається 200 статус код та список ідентифікаторів книг. При помилці повернуться 500 статус код із текстом “Internal server error”.

Загальна архітектура описаного мікросервісу, зображена на Рисунку 2.3.

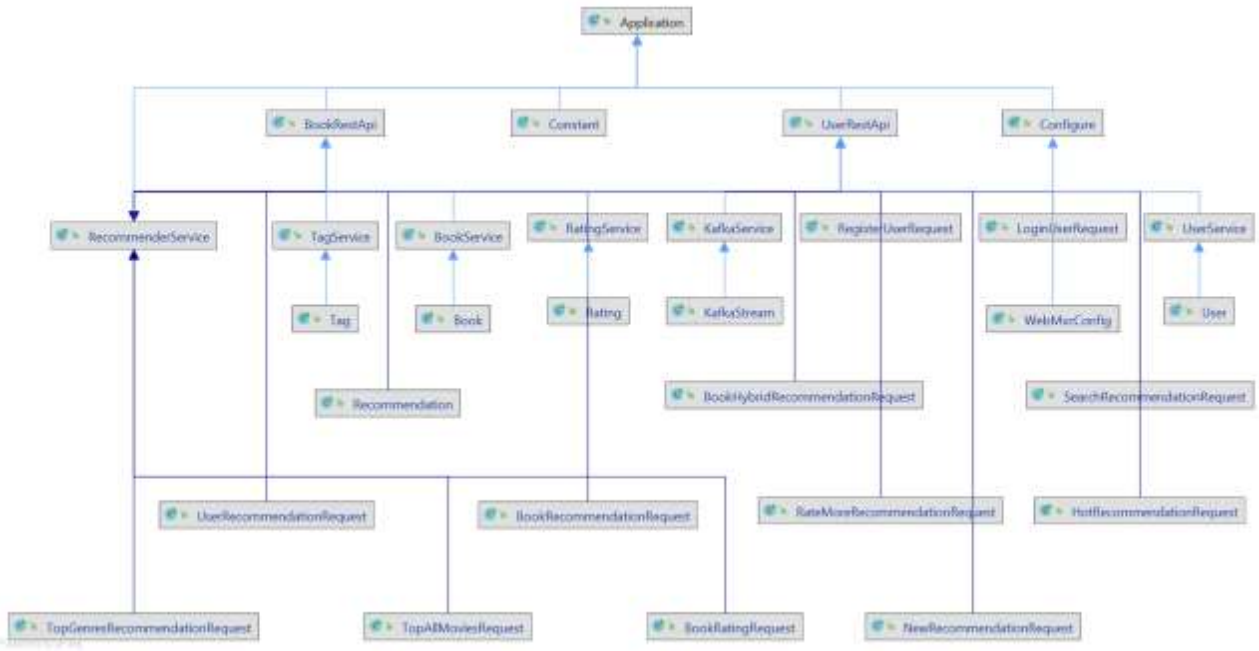


Рисунок 3.5 – Загальна архітектура мікросервісу

Для обміну інформацією між клієнтською та серверною частинами буде використано технологію API. Прикладний програмний інтерфейс (Application Programming Interface, API) — набір визначень підпрограм, протоколів взаємодії та засобів для створення програмного забезпечення.

Це набір чітко визначених методів для взаємодії різних компонентів. Інтерфейс API надає засоби для швидкої розробки програмного забезпечення. API може бути розроблений для веб-систем, операційних систем, баз даних, програмних бібліотек, тощо.

Розроблено класи: User, Book, Rating, Tag, Recommendation, BookService, RatingService, RecommenderService, TagService, UserService.

Класи User, Book, Rating, Tag, Recommendation використовуються для маніпуляції даними, що надходять з бази даних. Ці класи є реалізацією специфікації JPA – Java Persistence API. Ця специфікація мови програмування Java використовується як реалізація ORM фреймворку. ORM (Object-Relational mapping) фреймворк використовується для того, щоб вирішити невідповідність між об'єктно-орієнтованою моделлю мови програмування та реляційною моделлю бази даних. Ці класи є реалізацією сегменту Model з архітектурного паттерну MVC, який було обрано під час етапу планування розробки програмного модуля [15].

Для кожного з класів з сегменту Model існує відповідний клас, що реалізовує шаблон DAO (Data Access Object), що використовується для реалізації методів доступу до бази даних. Наприклад, метод findUserById використовується для пошуку користувачів у базі даних за допомогою відповідного ідентифікатора. Метод saveBook із класу BookDAO використовується для збереження об'єкту користувача (User) як поле у базі даних. Тобто, дані класи використовуються для базових CRUD (create, read, update, delete) операцій.

Класи RatingService, UserService, RecommenderService, TagService та BookService служить для реалізації сегменту «контролер» з архітектурного паттерну MVC. Ці класи містять в собі безпосередню логіку надання рекомендацій коритувачеві та логіку обробки отриманих даних.

Класи BookRestApi, UserRestApi використовуються для прийому запитів та їх зв'язку з безпосередньою логікою, що надсиліються з інтерфейсу користувача.

Основні API, які матиме серверна частина даного проекту розробки додатку для збереження фото користувача [17].

Для того, щоб працювати з додатком, спочатку необхідно в ньому зареєструватися. Для реєстрації необхідні наступні дані: нікнейм, пароль, дата народження, пошта.

Діаграма роботи сервісу під час проходження реєстрації зображено на рисунку 3.1.

Рисунок 3.6 − Діаграма діяльності сервера для проведення реєстрації

Програмно даний алгоритм реалізується наступним чином:

```
@RequestMapping("/rest/users")
@CrossOrigin
@Controller
public class UserRestApi extends Application {

   @Autowired
   private UserService userService;

   @RequestMapping(value = "/register", produces = "application/json", method =
RequestMethod.GET)
   @ResponseBody
      model.addAttribute("success",userService.registerUser(new
RegisterUserRequest(username,password)));
      return model;
   }
   @Autowired
   private UserService userService;

   @RequestMapping(value = "/register", produces = "application/json", method =
RequestMethod.GET)
   @ResponseBody
      model.addAttribute1("success",userService.registerUser(new
RegisterUserRequest(username,password)));
      return model;
      model.addAttribute2("success",userService.registerUser(new
RegisterUserRequest(username,password)));
      return model;
      model.addAttribute("success",userService.registerUser(new
RegisterUserRequest(username,password)));
      return model;

   }
```

Діяаграма роботи сервісу під час авторизації зображено на рисунку 3.2.

Рисунок 3.7 − Діаграма діяльності сервера для авторизації

Щоб сервіс працював правильно, потрібно бути авторизованим у системі. Авторизація використовується для того, щоб закріпити результати оцінок за певним користувачем, адже потім ці дані будуть використовуватись для закріплення за ними оцінок огляду книг та подальшому використання цих оцінок в наданні рекомендацій [21].

У вигляді коду даний алгоритм має наступне представлення:

```
@RequestMapping("/rest/users")
@CrossOrigin
@Controller
public class UserRestApi extends Application {

   @Autowired
   private UserService userService;

   @RequestMapping(value = "/login", produces = "application/json", method = RequestMethod.GET )
   @ResponseBody
   public Model login(@RequestParam("username") String username, @RequestParam("password") String password, Model model) {
      User user  =userService.loginUser(new LoginUserRequest(username,password));
      model.addAttribute("success",user != null);
      model.addAttribute("user",user);
      return model;
   }
```

Таблиця 3.2 – Опис основних API проекту

Опис API	URI	Параметри запиту	Відповідь
Внутрішнє API мікросервісу			

Таблиця 3.2 – Продовження

Отримати інформацію про книгу	GET /book/{id}		
Додати інформацію про нову книгу	POST /book/		
Видалити інформацію про книгу	DELETE /book/{id}		
Оновити інформацію про існуючу книгу	UPDATE /book/{id}		
Отримати інформацію про рецензію на обрану книгу	GET /review/{bookId}		
Додати інформацію про нову рецензію на обрану книгу	POST /review/{bookId}		
Видалити рецензію для обраної книги	DELETE /review/{id}		

Таблиця 3.2 – Продовження

Оновити рецензію для обраної книги	UPDATE /review/{id}		
Отримати рекомендації книг	GET /recommend/{userId}		

3.3 Обґрунтування вибору мови програмування

Наразі є досить багато сучасних, популярних та потужних мов програмування, які підходять для виконання завдань даного проекту. При виборі мови програмування було звернено увагу лише на ті, які реалізують об’єктно-орієнтовану методологію написання коду (ООП). Ця методологія користується найбільшою популярністю та легко вирішує багато питань, які виникають при розробці. ООП - одна з парадигм програмування, яка розглядає програму як множину «об'єктів», що взаємодіють між собою. Основу ООП складають чотири основні концепції: інкапсуляція, успадкування, поліморфізм та абстракція.

Однією з переваг ООП є краща модульність програмного забезпечення (тисячу функцій процедурної мови, в ООП можна замінити кількома десятками класів із своїми методами) [33]. При виборі мови програмування було порівняно мови Java та C# та обрано мову Java для реалізації задач даного проекту.

Ці дві мови програмування є дуже схожими між собою, адже вони реалізують модель ООП, мають достатню кількість фреймфорків для полегшення розробки, реалізують найбільш поширені зараз шаблони проектування та легко інтегруються із найбільш відомими базами даних. Найбільш суттєва різниця є в тому, що C# найкраще підходить для розробки програмного забезпечення під операційну

систему Windows, в той час як Java є більш поширеною для розробки веб-систем та Android-додатків, ніж C# [21].

Java була випущена у 1995 році та з тих часів зазнала багатьох змін. З 2009 року мовою займається компанія «Oracle». Головною проблемою, яку мала реалізувати мова програмування Java була потреба в мові, яка б не залежала від платформи і яку водночас можна було б використовувати для створення програмного забезпечення, що вбудовується в різноманітні побутові електронні прилади, такі як мобільні засоби зв'язку, пристрої дистанційного керування тощо

У створенні мови програмування Java було п'ять початкових цілей:

- мова повинна мати простий, зрозумілий, схожий на звичайну людську мовиу синтаксис;
- реалізація має бути безвідмовною, надійною, стабільною та безпечною;
- повинна зберегтися незалежність від апаратного забезпечення та легка переносність на іншу систему;
- висока продуктивність виконання;
- мова має бути інтерпретованою із динамічним зв'язуванням модулів.

В офіційній реалізації Java-програми компілюються у байт-код, який при виконанні інтерпретується віртуальною машиною для конкретної платформи, що збільшує швидкість роботи програмного забезпечення, на відміну від мов програмування як C++. Java розроблялась як платформо-незалежна мова, тому вона має менше низькорівневих можливостей для роботи з апаратним забезпеченням. Java використовує автоматичний збирач сміття для керування пам'яттю під час життєвого циклу об'єкта. Програміст вирішує, коли створювати об'єкти, а віртуальна машина відповідальна за звільнення пам'яті після того, як об'єкт стає непотрібним. Коли до певного об'єкта вже не залишається посилань, збирач сміття може автоматично прибирати його із пам'яті. Проте, витік пам'яті все ж може статися, якщо код, написаний програмістом, має посилання на вже непотрібні об'єкти, наприклад на об'єкти, що зберігаються у діючих контейнерах.

Отже, для виконання завдань даного мікросервісу і для розробки клієнтської буде використана мова програмування Java [21].

Spring Framework — це програмний фреймворк з відкритим кодом та контейнери з підтримкою інверсії управління для платформи Java [39]. Основні особливості Spring Framework можуть бути використані будьяким додатком Java, але є розширення для створення веб-додатків на платформі Java EE. Незважаючи на це, Spring Framework не нав'язує якоїсь конкретної моделі програмування, Spring Framework став популярним в спільноті Java як альтернатива, або навіть доповнення моделі Enterprise JavaBean (EJB).

Spring Framework складається з кількох модулів, які надають широкий спектр послуг:

- Контейнер Інверсії управління: Конфігурація компонентів додатків і управління життєвим циклом об'єктів Java, здійснюється головним чином через Інверсію управління;
- Аспектно-орієнтоване програмування: дозволяє реалізувати наскрізні процедури;
- Доступ до даних: робота з реляційною системою управління базами даних на платформі Java з використанням JDBC і об'єктно-реляційні відображення та інструментів з NoSQL баз даних;
- Управління транзакціями: об'єднує кілька API, управління транзакціями та координує операції для Java-об'єктів;
- Модель-Вигляд-Управління (Model-View-Controller): програмний каркас на основі HTTP сервлета, що забезпечує створення веб-додатків і веб-служб RESTful;
- Аутентифікація і авторизація: налаштовувані процеси безпеки, які підтримують цілий ряд стандартів, протоколів, інструментів і практик за допомогою підпроекту Spring Security (колишня система безпеки AcerI для Spring);

- Віддалене керування: конфігураційний вплив і управління Java об'єктами для місцевої (локальної) або віддаленої конфігурації через JMX;
- Тестування: підтримка класів для написання юніт-тестів та інтеграційних тестів.

Spring Framework був обраний для реалізації задач даного проекту оскільки суттєво полегшує роботу розробника. Робота з базою даних, написання API та можливість додавання нового функціоналу досить спрощене за рахунок контейнеру Inversion of Control та шаблону проектування Dependency Injection, які реалізовані даним фреймворком [22].

3.3 Обгрунтування вибору бази даних

Для реалізації задач даного проекту підходить реляційна модель бази даних, адже кількість таблиць та зв’язків між ними не є великою. Було проведено огляд та порівняння двох баз даних MySQL та PostgreSQL, як найбільш поширених баз даних у невеликих проектах, який наведений у таблиці 2.4.

Таблиця 3.3 – Порівняння MySQL та PostgreSQL баз даних

Назва бази даних	Переваги	Недоліки
MySQL	– популярність та легкість у використанні; – досить висока безпека; – добре інтегрована із Spring Data; – досить висока швидкість доступу до даних.	– невисока швидкість виконання операцій Full Join.

Таблиця 3.3 – Продовження

PostgreSQL	– легкість у використанні; – досить висока швидкість доступу до даних.	– менша за MySQL популярність; – менша за MySQL інтегрованість із Spring Data.

Для реалізації задачі даного проекту було обрано MySQL базу даних. Вона є однією із найбільш поширених баз даних, яка використовується як у великих проектах, так і для малих. MySQL є добре інтегрованою із фреймворком Spring Data, який широко використовується у даному проекті. Він забезпечує полегшення та більший комфорт роботи із базою даних.

3.4 Обгрунтування вибору бази даних

Для розробки інформаціної технології надання рекомендацій для платформи огляду книг було вирішено обрати середовище розробки програмного забезпечення IntelliJ IDEA від компанії JetBrains. Дане середовище розробки програмного забезпечення є передовим у індустрії та містить безліч переваг серед інших середовищ.

Кожен компонент IntelliJ IDEA створений для того, щоб максимально підвищити продуктивність розробки. Розумний редактор коду в поєднанні з ергономічним дизайном роблять розробку не тільки ефективною, а й приємною. Після індексування вихідного коду IntelliJ IDEA надає масу можливостей для швидкої і ефективної розробки: розумне автодоповнення, аналіз коду в реальному часі і надійні рефакторинги [17].

Всі необхідні інструменти вже під рукою, а значить не потрібно шукати і встановлювати плагіни для інтеграції з системами контролю версій і підтримки

популярних мов і фреймворків. У той час як базове автодоповнення пропонує імена класів, методів, полів і ключових слів в області видимості, розумне автодоповнення пропонує тільки ті елементи коду, які актуальні в поточному контексті.

IntelliJ IDEA аналізує одноманітні завдання в процесі розробки і автоматизує їх, щоб ви могли зосередитися на загальній картині.

IntelliJ IDEA аналізує код в пошуках зв'язків між символами у всіх файлах і на всіх мовах, що використовуються в проекті. На основі цього аналізу IntelliJ IDEA надає допомогу при написанні коду, зручну навігацію, перевірку помилок в коді і, звичайно, рефакторингом. Всі розумні функції допомоги при написанні коду працюють не тільки для вибраної мови, а й для виразів і строкових літералів іншою мовою. Наприклад, ви можете вставляти в рядкові літерали Java фрагменти коду на SQL, XPath, HTML, CSS або JavaScript. Знаючи все про використання символів, IntelliJ IDEA пропонує ефективні і точні рефакторингом. Наприклад, при перейменуванні класу в JPA-вираженні, IntelliJ IDEA оновить все, що необхідно: від класу суті до кожного виразу JPA, в якому він використовується.

IDE вміє миттєво знаходити дублікати в коді. Якщо ви збираєтеся отримати змінну, константу або метод, IntelliJ IDEA повідомить вам, що існує аналогічний фрагмент коду, який також слід замінити.

Коли IntelliJ IDEA виявляє потенційну помилку, в редакторі з'являється значок лампочки. Натисніть на нього або використовуйте поєднання клавіш Alt + Enter, щоб подивитися список можливих виправлень.

Під час налагодження IntelliJ IDEA показує значення змінних прямо в вихідному коді. Не потрібно наводити курсор на змінну і отримати доступ до панелі змінних у вікні налагоджувача. Кожен раз, коли змінна змінює значення, IDE виділяє її іншим кольором, щоб було зрозуміліше, як стан змінюється в коді.

У IntelliJ IDEA реалізований універсальний інтерфейс для роботи з основними системами контролю версій, включаючи Git, SVN, Mercurial, CVS,

Perforce і TFS. Середовище розробки дозволяє переглядати історію змін, керувати гілками і вирішувати конфлікти злиття.

3.4 Тестування розробленого програмного засобу системи рекомендацій для платформи огляду книг та аналіз результатів його роботи

Для розробленого програмного модуля надання рекомендацій для платформи огляду книг було проведено тестування.

Тестування полягало у перевірці правильності результатів роботи рекомендаційного модуля шляхом додавання в систему тестових даних та ручній перевірці цих даних. Також було проведено тестування стабільності роботи програмного модуля шляхом наповнення бази даних великою кількістю даних. Наразі додаток не знаходиться в фазі продакшену, тому результати, що отримуються від модуля надання рекомендацій можуть відображати не найкращі результати, бо якість наданих рекомендацій напряму залежить від кількості користувачів, що користуються системою, а також від кількості оглядів, що було створено цими користувачами.

В першу чергу для роботи з платформою рекомендацій необхідно авторизуватись в систему. Якщо користувач не зареєстрований, то у нього є можливість реєстрації у цій системі. Для цього потрібно ввести нікнейм, електронну пошту, та пароль у необхідні для цього поля. Усі поля проходять перевірку на правильність даних.

Після авторизації користувача перед ним відкривається головна сторінка застосунку (рисунок 3.3).

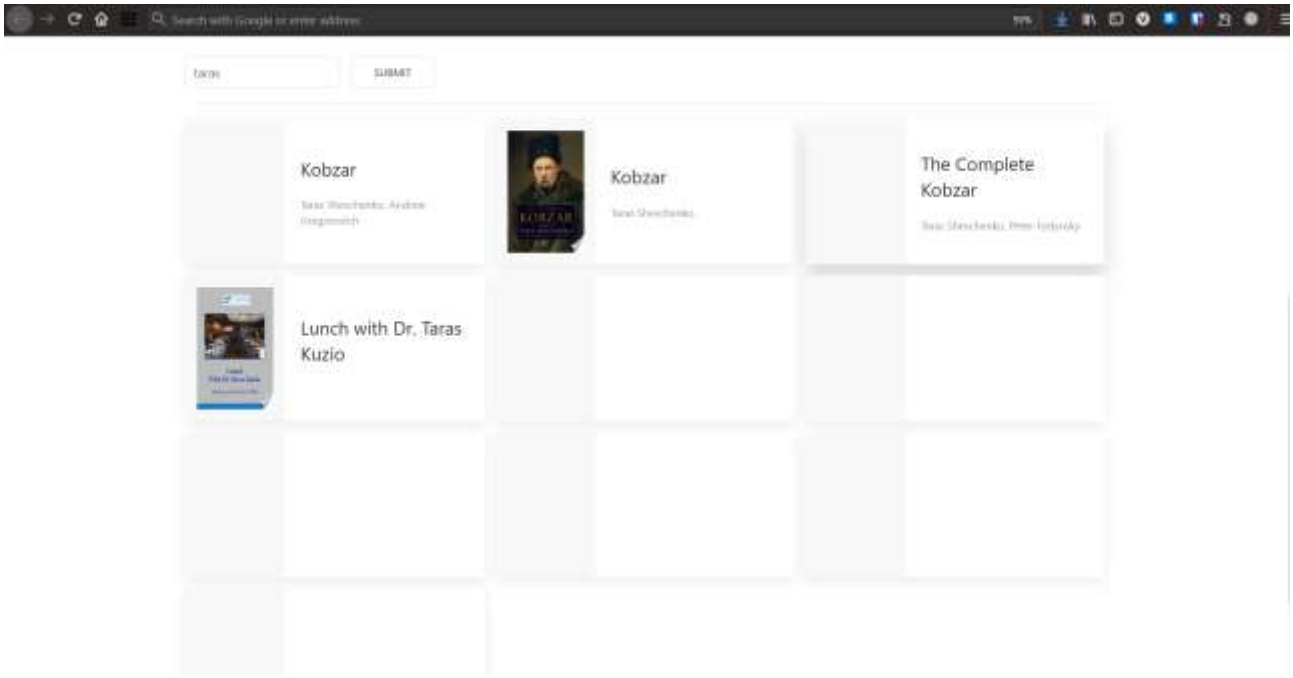


Рисунок 3.3 − Головна сторінка додатку

Він може виконати наступні дії: переглянути каталог доступних книг, обрати книгу з каталогу, переглянути список рекомендацій.

Перейшовши до обраної з каталогу книги перед користувачем відкривається вікно редагування огляду для цієї книги (рисунок 3.4).

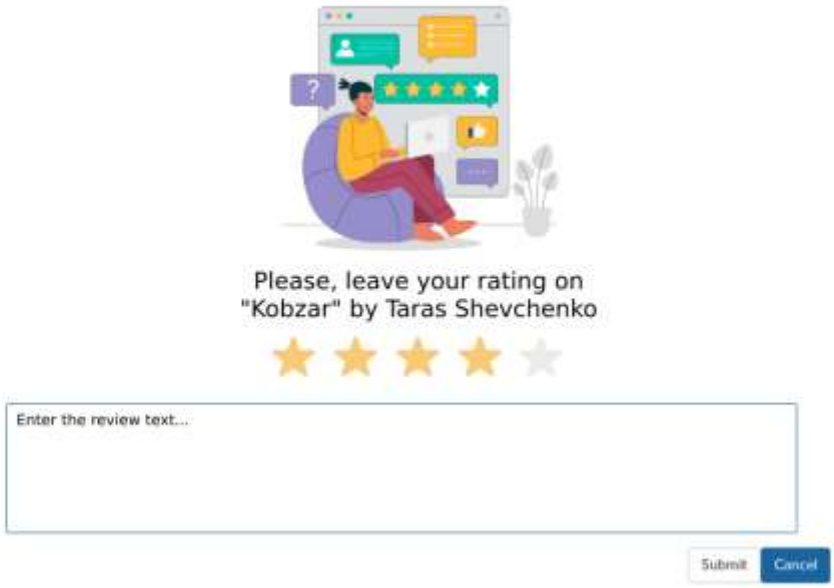


Рисунок 3.4 – Вікно редагування огляду

Після того, як користувач завершить написання огляду, він буде збережений до бази даних.

Для проведення тестування роботи програмного модуля надання рекомендацій у систему було додано тестові дані та коректність наданя рекомендацій була перевірена вручну.

У прикладі було створено користувача «A», «B», «C», «D», «E». Також додано двох тестових користувачів.

В залежності від існуючих оцінок модуль має вирішити, додавати книгу до списку рекомендацій, чи ні.

У таблиці 3.1 наведено порівняння результатів тестування якості рекомендацій.

Таблиця 3.1 − Порівняння результатів тестування

Оцінки/ Користувачі	Beowulf	Sauron Defeated	The War Of the Ring	Kobzar	Harry Potter and philosophers stone
Корист. «A»	4,5	1	5	3	5
Корист. «B»	1	2	3	5	5
Корист. «C»	3,5	5	5	1	2
Корист. «D»	1	5	3	5	2
Корист. «E»	4,5	2	4,5	2	1
Тестовий користувач	**4**	*Не рекоменд.*	**4**	**3**	*Не рекоменд.*

Тестовий користувач 2	**1**	**4,5**	*Рекоменд.*	**3**	**3**

У програмному додатку розроблений функціонал, що було описано в попередньому розділі. На зображеннях можна побачити сторінки авторизації, реєстрації, головну сторінку, каталог, панель рекомендацій а також вікно редагування відгуку.

Отже, провівши тестування розробленого програмного модуля, можна зробити висновок, що він працює досить швидко, помилок не виявлено; порівняно з програмами-аналогами показано хороші результати за характеристиками.

3.5 Висновки з розділу

У даному розділі було проаналізовано та обрано технології розробки програмного забезпечення надання рекомендацій для платформи огляду книг. Насамперед обґрунтовано вибір мови програмування для реалізації клієнтської та мікросервісів серверної частини додатку. Було обрано мову програмування Java для реалізації клієнтської частини та мікросервісу управління рецензіями і книгами, а також мову Python для реалізації мікросервісу надання рекомендацій.

Проаналізовано життєвий цикл програмного забезпечення та моделі його розробки. Було розроблено архітектури двох мікросервісів – для управління рецензіями та наданні рекомендацій, описано їх структуру, основні класи та методи.

Було оглянуто основні API проекту – внутрішні для спілкування мікросервісу та клієнтської частини, а також API для спілкування із мікросервісом, який відповідає за аналіз книг. У ході виконання аналізу було обрано архітектури MVC та MVP для реалізації серверної та клієнтської частин відповідно та

збереження інформаці про рецензії у базі даних та базу даних MySQL для реалізації завдань даного проекту.

Здійснено тестування створеного програмного забезпечення, яке показало, що додаток працює коректно та відповідає усім поставленим вимогам. Зокрема виконано перевірку додатку на якість наданих рекомендацій. Критерій оцінки якості рекомендацій базувався на тестових даних і на відгуках користувачів. Якість надання рекомендацій влаштовувала більшість користувачів.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Оцінювання комерційного потенціалу розробки

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки. Для проведення технологічного аудиту було залучено двох незалежних експертів. Такими експертами будуть Арсенюк І.Р. та Сілагін О.В.

Оцінювання комерційного потенціалу розробки буде здійснено за 12-ма критеріями за 5-ти бальною шкалою. Результати оцінювання комерційного потенціалу розробки наведено в таблиці 4.1.

Таблиця 4.1 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта	
	Арсенюк І.Р.	Кабачій В.В.
	Бали, виставлені експертами:	
1	4	4
2	3	3
3	3	4
4	4	3
5	3	4
6	4	3
7	3	3
8	4	4
9	4	3
10	3	4
11	3	4
12	3	4
Сума балів	$СБ_1 = 42$	$СБ_2 = 44$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{2} = 43$	

Отже, з отриманих даних таблиці 4.1 видно, що нова розробка має високий рівень комерційного потенціалу.

4.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько–технологічної роботи.

Для розробки нового програмного продукту необхідні такі витрати.

Основна заробітна плата для розробників визначається за формулою (4.1):

$$З_о = \frac{M}{T_р} \cdot t, \qquad (4.1)$$

де M – місячний посадовий оклад конкретного розробника;

$T_р$ – кількість робочих днів у місяці, $T_р$ = 22 дні;

t – число днів роботи розробника, t = 50 днів.

Розрахунки заробітних плат для керівника і програміста наведені в таблиці 4.2.

Таблиця 4.2 – Розрахунки основної заробітної плати

Працівник	Оклад М, грн.	Оплата за робочий день, грн.	Число днів роботи, t	Витрати на оплату праці, грн.
Науковий керівник	15000	750	5	4500
Інженер-програміст	10007	500,35	50	25017,5

Всього:	29517,5

Розрахуємо додаткову заробітну плату:

$$З_{дод} = 0,1 \cdot 29517,5 = 2952,75 \text{ (грн.)}$$

Нарахування на заробітну плату операторів НЗП розраховується як 22% від суми їхньої основної та додаткової заробітної плати:

$$Н_{зп} = (З_о + З_р) \cdot \frac{\beta}{100}, \quad (4.2)$$

$$Н_{зп} = (29517,5 + 2952,75) \cdot \frac{22}{100} = 7143,24 \text{(грн.)}.$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за такою формулою:

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12}, \quad (4.3)$$

де Ц – балансова вартість обладнання, грн;

$Н_а$ – річна норма амортизаційних відрахувань % (для програмного забезпечення 25%);

Т – Термін використання (Т=3 міс.).

Таблиця 4.3 – Розрахунок амортизаційних відрахувань

Найменування програмного забезпечення	Балансова вартість, грн.	Норма амортизації, %	Термін використання, міс.	Величина амортизаційних відрахувань,грн

Таблиця 4.3 – Продовження

Персональний комп’ютер	10000	25	3	625
Всього:				625

Розрахуємо витрати на комплектуючі. Витрати на комплектуючі розрахуємо за формулою:

$$K = \sum_1^n \text{Н}_\text{і} \cdot \text{Ц}_\text{і} \cdot \text{К}_\text{і}, \quad (4.4)$$

де *n* – кількість комплектуючих;

Ні - кількість комплектуючих і-го виду;

Ці – покупна ціна комплектуючих і-го виду, грн;

Кі – коефіцієнт транспортних витрат (приймемо Кі = 1,1).

Таблиця 4.4 - Витрати на комплектуючі, що були використані для розробки ПЗ.

Найменування матеріалу	Одиниці виміру	Ціна, грн.	Витрачено	Вартість витрачених матеріалів, грн.
Флешка	шт.	180	1	180
Пачка паперу	уп.	130	1	130
Ручка	шт.	10	1	10
Всього з урахуванням транспортних витрат				352

Витрати на силову електроенергію розраховуються за формулою:

$$B_e = B \cdot П \cdot Ф \cdot K_n \,, \tag{4.5}$$

де B – вартість 1кВт-години електроенергії (В=1,68 грн/кВт);

$П$ – установлена потужність комп'ютера (П=0,6кВт);

$Ф$ – фактична кількість годин роботи комп'ютера (Ф=200 год.);

$К_п$ – коефіцієнт використання потужності ($К_п < 1$, $К_п = 0{,}7$).

$$B_e = 1{,}68 \cdot 0{,}6 \cdot 200 \cdot 0{,}7 = 141{,}12 \text{ (грн.)}$$

Розрахуємо інші витрати $В_{ін}$.

Інші витрати $І_в$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$В_{ін} = (1..3) \cdot (З_о + З_р). \tag{4.6}$$

Отже, розрахуємо інші витрати:

$$В_{ін} = 1 * (15909{,}11+1590{,}91) = 17500{,}02 \text{ (грн).}$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи:

$$В = З_о + З_д + Н_{зп} + А + К + В_е + І_в$$

$$В = 29517{,}5 + 2952{,}75 + 7143{,}24 + 625 + 352 + 141{,}12 + 17500{,}02 = 58231{,}63 \text{ (грн.)}$$

Розрахуємо загальну вартість наукової роботи $B_{заг}$ за формулою:

$$В_{заг} = \frac{В_{ін}}{\alpha}, \quad (4.7)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях = 1.

$$В_{заг} = \frac{58231,63}{1} = 58231,63.$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta}, \quad (4.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Отже, розрахуємо загальні витрати:

$$ЗВ = \frac{58231,63}{0,9} = 64701,82 \text{ (грн.)}$$

4.3 Прогнозування комерційних ефектів від реалізації результатів розробки інформаційної технології

Спрогнозуємо отримання прибутку від реалізації результатів нашої розробки. Зростання чистого прибутку можна оцінити у теперішній вартості грошей. Це забезпечить підприємству (організації) надходження додаткових коштів, які дозволять покращити фінансові результати діяльності .

Оцінка зростання чистого прибутку підприємства від впровадження результатів наукової розробки. У цьому випадку збільшення чистого прибутку підприємства $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_1^n (\Delta П_я \cdot N + П_я \Delta N)_i \quad (4.9)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки витрати на виготовлення інформаційної технології зменшаться на 25 грн (що автоматично спричинить збільшення чистого прибутку підприємства на 25 грн), а кількість користувачів, які будуть користуватись збільшиться: протягом першого року – на 150 користувачів, протягом другого року – на 125 користувачів, протягом третього року – 100 користувачів. Реалізація інформаційної технології до впровадження результатів наукової розробки складала 500 користувачів, а прибуток, що отримував розробник до впровадження результатів наукової розробки – 400 грн.

Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Отже, збільшення чистого продукту $\Delta П_1$ протягом першого року складатиме:

$$\Delta П_1 = 25 \cdot 500 + (400 + 25) \cdot 150 = 76250 \text{ грн.}$$

Протягом другого року:

$$\Delta П_2 = 25 \cdot 500 + (400 + 25) \cdot (150 + 125) = 129375 \text{ грн.}$$

Протягом третього року:

$$\Delta П_3 = 25 \cdot 500 + (400 + 25) \cdot (150 + 125 + 100) = 171875 \text{ грн.}$$

4.4 Розрахунок ефективності вкладених інвестицій та визначення періоду їх окупності

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність $Е_{абс}$вкладених інвестицій розраховується так:

$$Е_{абс} = (ПП - PV), \quad (4.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, 3 роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до т. 2, 3,4.

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, зображений на рисунку 4.1.

Рисунок 4.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

Розрахуємо вартість чистих прибутків за формулою:

$$ПП = \sum_1^m \frac{\Delta П_i}{(1+\tau)^t} \quad (4.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

τ – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки.

Отже, розрахуємо вартість чистого прибутку:

$$ПП = \frac{64701,82}{(1+0,1)^0} + \frac{76250}{(1+0,1)^2} + \frac{129375}{(1+0,1)^3} + \frac{171875}{(1+0,1)^4} = 342312,64 \text{ (грн.)}$$

Тоді розрахуємо $E_{абс}$:

$$Е_{абс} = 342312,64 - 64701,82 = 277610,82 \text{ грн.}$$

Оскільки $E_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$ за формулою:

$$E_в = \sqrt[T]{1 + \frac{E_{абс}}{PV}} - 1, \quad (4.12)$$

де $E_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$T_ж$ – життєвий цикл наукової розробки, роки.

Тоді будемо мати:

$$E_в = \sqrt[3]{1 + \frac{277610,82}{64701,82}} - 1 = 1,62 \text{ або } 162\ \%.$$

Далі, розраховану величина $E_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau = d + f,$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2019 році в Україні d = 0,2;

f – показник, що характеризує ризикованість вкладень, величина f = 0,1.

$$\tau = 0,2 + 0,1 = 0,3$$

Оскільки $E_в$ = 169% $> \tau_{мін}$ = 0,3 = 30%, то у інвестор буде зацікавлений вкладати гроші в дану наукову розробку.

Термін окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ розраховується за формулою:

$$T_{ок} = \frac{1}{E_в},$$

$$T_{ок} = \frac{1}{1,62} = 0,62 \text{ року}.$$

Обрахувавши термін окупності даної наукової розробки, можна зробити висновок, що фінансування даної наукової розробки буде доцільним.

4.5 Висновок до розділу 4

У розділі було виконано розрахунок витрат на розробку та виготовлення нового технічного рішення, сума яких складає 64701,82 гривень. Було спрогнозовано орієнтовану величину витрат по кожній з статей витрат. Також розаховано чистий прибуток, який може отримати виробник від реалізації нового технічного рішення, знайдено термін окупності витрат для виробника та економічний ефект для споживача при використанні даної розробки. В результаті аналізу розрахунків можна зробити висновок, що розробка у виробництві та використання дешевша за аналог і є висококонкурентоспроможним. Період окупності складе близько 0,62 роки.

ВИСНОВКИ

У ході виконання магістерської кваліфікаційної роботи розроблено інформаційну технологію надання рекомендацій для платформи огляду книг на основі графової згорткової нейронної мережі з вдосконаленим методом підрахунку рекомендацій.

Здійснено аналіз предметної області та досліджено основні методи розпізнавання емоцій. У ході аналізу було визначено, що використання графа під час операції формування ймовірних рекомендацій є більш оптимізованим а також дає змогу давати більш точні та глибокі рекомендації.

Виконано огляд та аналіз існуючих програмних засобів, що надають рекомендацію щодо вибору книг. Визначено, що більшість розглянутих програмних рішень не можуть забезпечити достатньо точне надання рекомендацій для окремих демографічних регіонів.

Доведено доцільність розробки інформаційної системи, представленої у вигляді програмного додатку. Наведено та обґрунтовано вимоги до розроблюваного програмного продукту.

Після проведеного аналітичного огляду існуючих методів основних етапів технології рекомендаційних систем було доведено: для задачі генерації оцінок слід використовувати метод гібридної колаборативної фільтрації; для генерації оцінок елементів було вирішено використовувати графовий метод в купі з згортковою нейронною мережею.

Запропоновано математичну модель інформаційної технології надання рекомендацій щодо вибору книг, на основі якої базується згорткова нейронна мережа, та шари, які входять до складу її архітектури.

Запропоновано структуру інформаційної технології для надання рекомендацій для вибору книг, побудованої на основі згорткової нейронної мережі. На основі структури інформаційної технології визначено алгоритм роботи програмного

засобу для надання рекомендацій щодо вибору книг, який складається з трьох модулів: навчання, розпізнавання та вадідації, а також розроблено алгоритм роботи кожного з модулів.

Обґрунтовано доцільність застосування мови програмування Java та Python для розробки програмного засобу. Для програмної реалізації згорткової нейронної мережі обрано нейромережеву бібліотеку TensorFlow. Програмна реалізація програмного додатку виконана у середовищі програмування IntellijIDEA.

На основі здійснених досліджень розроблено програмне забезпечення, що дозволяє практично реалізувати технологію надання рекомендацій щодо вибору книг.

Здійснено тестування розробленого програмного додатку, яке підтвердило коректність його роботи.

Розроблено автоматизовану систему тестування додатку. Під час тестування точність рекомендацій становила близько 97% для підготованого набору даних про користувачів та книг, а під час валідації у відеопотоці – 95% (найближчий, за характеристиками аналог, додаток LoveReading – 92%, проте без урахування вподобань певних демографічних груп, тобто точність зросла в середньому на 2 – 3%). У результаті тестування суттєвого збільшення навантаження на апаратні ресурси комп’ютера та втрати кадрів не виявлено. Отже, можна констатувати, що програма перевершує можливості своїх аналогів.

У результаті виконання даної кваліфікаційної роботи поставлені задачі було виконано у повному обсязі. Отже, мета магістерської роботи досягнута.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Зайчик В. О. Дослідження методів створення рекомендаційних систем для платформи огляду книг. / В. О. Зайчик, І. Р. Арсенюк // Тези доповідей науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії. – Вінниця: ВНТУ, 2021. Електронний ресурс (режим доступу https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2021/paper/view/12207/10159)

2. Breese J., Heckerman D. Empirical Analysis of Predictive Algorithms for Collaborative Filtering // Archived. 2013. URL: http://uai.sis.pitt.edu/displayArticleDetails.jsp?mmnu=1&smnu=2&article_id=231&proceeding_id=14 (дата звернення: 26.04.2021).

3. Xiaoyuan S. A survey of collaborative filtering techniques // Advances in Artificial Intelligence archive. 2009. URL: http://www.hindawi.com/journals/aai/2009/421425/ (дата звернення: 15.04.2021).

4. Recommender Systems - The Textbook | Charu C. Aggarwal | Springer. Springer: ISBN 9783319296579, 2016. 67 с.

5. Ghazanfar M. A. Kernel-Mapping Recommender system algorithms // Information Sciences. URL: https://doi.org/10.1016%2Fj.ins.2012.04.012 (дата звернення: 25.04.2021).

6. Abhinandan D. Google news personalization // Proceedings of the 16th international conference on World Wide Web. 2007. URL: https://doi.org/10.1145%2F1242572.1242610 (дата звернення: 14.04.2021).

7. Айвазян С.А. Прикладная статистика и основы эконометрики. Москва: Юнити. 2001. − 260 с.

8. AnsTester для Windows [Електронний ресурс]. – Режим доступу: https://www.softportal.com/software-8009-anstester.html. – Назва з екрану.

9. Goldberg D. Using collaborative filtering to weave an information tapestry / D. Goldberg, D. Nichols, B. M. Oki., 1992. – 72 с.

10. Chen L. A user-centric evaluation framework for recommender systems. In: Proceedings of the fifth ACM conference on Recommender Systems / L. Chen, P. Pu, R. Hu. – Нью Йорк: ACM, 2011.

11. Breese J. S. Емпіричний аналіз прогнозних алгоритмів робочий фільтр. В: Матеріали чотирнадцятої щорічної конференції з питань невизначеності в ШІ [Електронний ресурс] / J. S. Breese, D. Heckerman, C. Kadie. – 1998.

12. Чебанюк О. В. Методика розпізнавання рукописного тексту на основі аналізу векторів руху за допомогою сенсорних пристроїв / О. В. Чебанюк, Д. А. Долотов – Вісник східноукраїнського національного університету імені Володимира Даля 2015, С. 230 – 236.

13. Claypool O. Combining content-based and collaborative filters in an online newspaper./ O. Claypool, / ACM SIGIR Workshop on Recommender Systems: algorithms and evaluation. – 2011. – 6 с.

14. Гудфеллоу Я. Методы фильтрации / Я. Гудфеллоу, И. Бенджио, А. Курвилль пер. с анг. А. А. Слинкина – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.

15. Методичні вказівки до виконання студентами-магістрантами економічної частини магістерських кваліфікаційних робіт / Уклад. В. О. Козловський – Вінниця: ВНТУ, 2012. – 22 с.

16. Rokach L. Recommender Systems Handbook / Lior Rokach. – Springer US: Bracha, 2015. – 226 с.

17. Pankaj G. WTF: The who-to-follow system at Twitter [Електронний ресурс] / G. Pankaj, G. Ashish, L. Jimmy // Proceedings of the 22nd international conference on World Wide Web – Режим доступу до ресурсу: http://dl.acm.org/citation.cfm?id=2488433.

18. Fleder D. Blockbuster Culture's Next Rise or Fall: The Impact of Recommender Systems on Sales Diversity / D. Fleder, K. Hosanagar. // Management Science. – 2009. – С. 697–712.

19. Adamopoulos P. On Unexpectedness in Recommender Systems: Or How to Better Expect the Unexpected / P. Adamopoulos, A. Tuzhilin. // ACM Transactions on Intelligent Systems and Technology. – 2015. – С. 1–32.

20. Hofmann T. Proceedings of the 2007 ACM conference on Recommender systems [Електронний ресурс] / Thomas Hofmann // Portal.acm.org.. – 2007. – Режим доступу до ресурсу: https://doi.org/10.1145%2F1297231.1297240.

21. Andrew H. Why batch and user evaluations do not give the same results / P. Adamopoulos, A. Tuzhilin. // Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval – С. 21–92.

22. Jannach D. Lecture Notes in Computer Science / P. Adamopoulos, A. Tuzhilin. // Springer Berlin Heidelberg – С. 25–37.

23. Naz S., Ziauddin S., Shahid A.R. Driver fatigue detection using mean intensity, SVM, and SIFT. International Journal of Interactive Multimedia and Artificial Intelligence, 2019, vol. 5, no. 4, pp. 86–93.

24. Talegaonkar I., Joshi K., Valunj S., Kohok R., Kulkarni A. Real time facial expression recognition using deep learning // Proc. of International Conference on Communication and Information Processing (ICCIP). 2019 [Электронный ресурс]. Режим доступа: https://ssrn. com/abstract=3421486.

25. Гудфеллоу Я., Бенджио И., Курвилль А. Г93 Глубокое обучение / пер. с анг. А. А. Слинкина. – 2-е изд., испр. – М.: ДМК Пресс, 2018. – 652 с.: цв. ил.

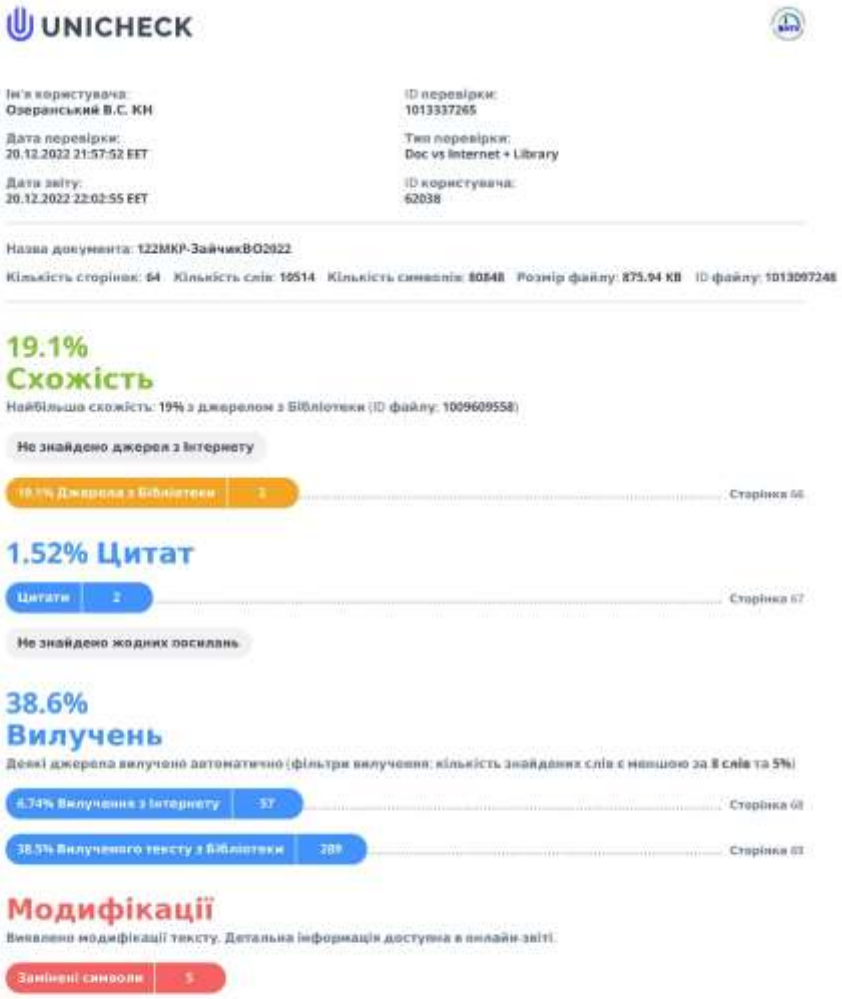
26. A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images, 2009.

27. A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

28. A. Gudi. Recognizing semantic features in faces using deep learning. arXiv preprint arXiv:1512.00743, 2015.

ДОДАТКИ

Додаток А



UNICHECK

Ім'я користувача:
Озеранський В.С. КН

ID перевірки:
1013337265

Дата перевірки:
20.12.2022 21:57:52 EET

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
20.12.2022 22:02:55 EET

ID користувача:
62038

Назва документа: 122МКР-ЗайчикВО2022

Кількість сторінок: 64 Кількість слів: 10514 Кількість символів: 80848 Розмір файлу: 875.94 KB ID файлу: 1013097248

19.1%
Схожість

Найбільша схожість: 19% з джерелом з Бібліотеки (ID файлу: 1009609558)

Не знайдено джерел з Інтернету

19.1% Джерела з Бібліотеки 2 Сторінка 66

1.52% Цитат

Цитати 2 Сторінка 67

Не знайдено жодних посилань

38.6%
Вилучень

Деякі джерела вилучено автоматично (фільтри вилучення: кількість знайдених слів є меншою за 8 слів та 5%)

6.74% Вилучення з Інтернету 57 Сторінка 68

38.5% Вилученого тексту з Бібліотеки 289 Сторінка 63

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи 5

Додаток Б

Фрагмент лістингу роботи програми

Клас BookService

```
@Service
public class BookService extends BookRestApi {

    @Autowired
    private MongoClient mongoClient;

    @Autowired
    private ObjectMapper objectMapper;

    private MongoCollection<Document> movieCollection;
    private MongoCollection<Document> averageMoviesScoreCollection;
    private MongoCollection<Document> rateCollection;

    private MongoCollection<Document> getBookCollection() {
        if (null == movieCollection)
            movieCollection =
mongoClient.getDatabase(Constant.MONGODB_DATABASE).getCollection(Constant.MONGODB
_MOVIE_COLLECTION);
        return movieCollection;
    }

    private MongoCollection<Document> getAverageBooksScoreCollection() {
        if (null == averageMoviesScoreCollection)
            averageMoviesScoreCollection =
mongoClient.getDatabase(Constant.MONGODB_DATABASE).getCollection(Constant.MONGODB
_AVERAGE_MOVIES_SCORE_COLLECTION);
        return averageMoviesScoreCollection;
```

```
    }

    private MongoCollection<Document> getRateCollection() {
        if (null == rateCollection)
            rateCollection = mongoClient.getDatabase(Constant.MONGODB_DATABASE).getCollection(Constant.MONGODB_RATING_COLLECTION);
        return rateCollection;
    }

    public List<Book> getRecommendeMovies(List<Recommendation> recommendations) {
        List<Integer> ids = new ArrayList<>();
        for (Recommendation rec : recommendations) {
            ids.add(rec.getMid());
        }
        return getBooks(ids);
    }

    public List<Book> getHybirdRecommendeMovies(List<Recommendation> recommendations) {
        List<Integer> ids = new ArrayList<>();
        for (Recommendation rec : recommendations) {
            ids.add(rec.getMid());
        }
        return getBooks(ids);
    }

    public List<Book> getBooks(List<Integer> mids) {
        FindIterable<Document> documents = getBookCollection().find(Filters.in("mid", mids));
        List<Book> books = new ArrayList<>();
        for (Document document : documents) {
            Book m = documentToBook(document);
            books.add(m);
        }
```

```
        books.sort((book, t1) -> {
            double val = book.getScore() - t1.getScore();
            if (val < 0) {
                return 1;
            } else if (val == 0)
                return 0;
            else
                return -1;
        });
        return books;
    }

    private Book documentToBook(Document document) {
        Book book = null;
        try {
            book = objectMapper.readValue(JSON.serialize(document), Book.class);
            Document score = getAverageBooksScoreCollection().find(Filters.eq("mid",
book.getMid())).first();
            if (null == score || score.isEmpty())
                book.setScore(0D);
            else
                book.setScore((Double) score.get("avg", 0D));
        } catch (IOException e) {
            e.printStackTrace();
        }
        return book;
    }

    private Rating documentToRating(Document document) {
        Rating rating = null;
        try {
            rating = objectMapper.readValue(JSON.serialize(document), Rating.class);
        } catch (IOException e) {
```

```
            e.printStackTrace();
        }
        return rating;
    }

    public boolean movieExist(int mid) {
        return null != findByMID(mid);
    }

    public Book findByMID(int mid) {
        Document document = getBookCollection().find(new Document("mid", mid)).first();
        if (document == null || document.isEmpty())
            return null;
        return documentToBook(document);
    }

    public void removeMovie(int mid) {
        getBookCollection().deleteOne(new Document("mid", mid));
    }

    public List<Book> getMyRateBooks(int uid) {
        FindIterable<Document> documents = getRateCollection().find(Filters.eq("uid", uid));
        List<Integer> ids = new ArrayList<>();
        Map<Integer, Double> scores = new HashMap<>();
        for (Document document : documents) {
            Rating rating = documentToRating(document);
            ids.add(rating.getMid());
            scores.put(rating.getMid(), rating.getScore());
        }
        List<Book> books = getBooks(ids);
        for (Book book : books) {
            book.setScore(scores.getOrDefault(book.getMid(), book.getScore()));
        }
```

```
        return books;
    }

    public List<Book> getNewBooks(NewRecommendationRequest request) {
        FindIterable<Document> documents =
getBookCollection().find().sort(Sorts.descending("issue")).limit(request.getSum());
        List<Book> books = new ArrayList<>();
        for (Document document : documents) {
            books.add(documentToBook(document));
        }
        return books;
    }

}
```

Клас RatingService

```
@Service
public class RatingService extends BookRestApi {

    @Autowired
    private MongoClient mongoClient;
    @Autowired
    private ObjectMapper objectMapper;
    @Autowired
    private Jedis jedis;

    private MongoCollection<Document> ratingCollection;

    private MongoCollection<Document> getRatingCollection() {
        if (null == ratingCollection)
```

```
        ratingCollection = mongoClient.getDatabase(Constant.MONGODB_DATABASE).getCollection(Constant.MONGODB_RATING_COLLECTION);
        return ratingCollection;
    }

    private Rating documentToRating(Document document) {
        Rating rating = null;
        try {
            rating = objectMapper.readValue(JSON.serialize(document), Rating.class);
        } catch (IOException e) {
            e.printStackTrace();
        }
        return rating;

    }

    public boolean movieRating(BookRatingRequest request) {
        Rating rating = new Rating(request.getUid(), request.getMid(), request.getScore());
        updateRedis(rating);
        if (ratingExist(rating.getUid(), rating.getMid())) {
            return updateRating(rating);
        } else {
            return newRating(rating);
        }
    }

    private void updateRedis(Rating rating) {
        if (jedis.exists("uid:" + rating.getUid()) && jedis.llen("uid:" + rating.getUid()) >= Constant.REDIS_MOVIE_RATING_QUEUE_SIZE) {
            jedis.rpop("uid:" + rating.getUid());
        }
        jedis.lpush("uid:" + rating.getUid(), rating.getMid() + ":" + rating.getScore());
```

```
}

public boolean newRating(Rating rating) {
    try {
        getRatingCollection().insertOne(Document.parse(objectMapper.writeValueAsString(rating)));
        return true;
    } catch (JsonProcessingException e) {
        e.printStackTrace();
        return false;
    }
}

public boolean ratingExist(int uid, int mid) {
    return null != findRating(uid, mid);
}

public boolean updateRating(Rating rating) {
    BasicDBObject basicDBObject = new BasicDBObject();
    basicDBObject.append("uid", rating.getUid());
    basicDBObject.append("mid", rating.getMid());
    getRatingCollection().updateOne(basicDBObject,
            new Document().append("$set", new Document("score", rating.getScore())));
    return true;
}


public Rating findRating(int uid, int mid) {
    BasicDBObject basicDBObject = new BasicDBObject();
    basicDBObject.append("uid", uid);
    basicDBObject.append("mid", mid);
    FindIterable<Document> documents = getRatingCollection().find(basicDBObject);
    if (documents.first() == null)
        return null;
```

```
        return documentToRating(documents.first());
    }

    public void removeRating(int uid, int mid) {
        BasicDBObject basicDBObject = new BasicDBObject();
        basicDBObject.append("uid", uid);
        basicDBObject.append("mid", mid);
        getRatingCollection().deleteOne(basicDBObject);
    }

    public int[] getMyRatingStat(User user) {
        FindIterable<Document> documents = getRatingCollection().find(new Document("uid",
user.getUid()));
        int[] stats = new int[10];
        for (Document document : documents) {
            Rating rating = documentToRating(document);
            Long index = Math.round(rating.getScore() / 0.5);
            stats[index.intValue()] = stats[index.intValue()] + 1;
        }
        return stats;
    }

}
```

Клас RecomenderService

```
@Service
public class RecommenderService extends Application {

    private static Double CF_RATING_FACTOR = 0.3;
    private static Double CB_RATING_FACTOR = 0.3;
    private static Double SR_RATING_FACTOR = 0.4;
```

```
    @Autowired
    private MongoClient mongoClient;
    @Autowired
    private TransportClient esClient;

    private List<Recommendation> findMovieCFRecs( int mid, int maxItems ) {
        MongoCollection<Document> movieRecsCollection =
mongoClient.getDatabase(Constant.MONGODB_DATABASE).getCollection(Constant.MONGODB
_MOVIE_RECS_COLLECTION);
        Document movieRecs = movieRecsCollection.find(new Document("mid", mid)).first();
        return parseRecs(movieRecs, maxItems);
    }

    private List<Recommendation> findUserCFRecs( int uid, int maxItems ) {
        MongoCollection<Document> movieRecsCollection =
mongoClient.getDatabase(Constant.MONGODB_DATABASE).getCollection(Constant.MONGODB
_USER_RECS_COLLECTION);
        Document userRecs = movieRecsCollection.find(new Document("uid", uid)).first();
        return parseRecs(userRecs, maxItems);
    }

    private List<Recommendation> findContentByMongoDb(int mid, int maxItems) {
        MongoCollection<Document> contentCollection =
mongoClient.getDatabase(Constant.MONGODB_DATABASE).getCollection(Constant.MONGODB
_CONTENT_MOVIE_RECS);
        Document contentRecs = contentCollection.find(new Document("mid", mid)).first();
        return parseRecs(contentRecs, maxItems);
    }


    private List<Recommendation> findContentBasedMoreLikeThisRecommendations( int mid, int
maxItems ) {
```

```
        MoreLikeThisQueryBuilder query = QueryBuilders.moreLikeThisQuery(/*new String[]{"name",
"descri", "genres", "actors", "directors", "tags"},*/
                new                                                    MoreLikeThisQueryBuilder.Item[]{new
MoreLikeThisQueryBuilder.Item(Constant.ES_INDEX,                    Constant.ES_MOVIE_TYPE,
String.valueOf(mid))});

        return
parseESResponse(esClient.prepareSearch().setQuery(query).setSize(maxItems).execute().actionGet());
    }

    public List<Recommendation> findStreamRecs( int uid, int maxItems ) {
        MongoCollection<Document>                    streamRecsCollection                    =
mongoClient.getDatabase(Constant.MONGODB_DATABASE).getCollection(Constant.MONGODB
_STREAM_RECS_COLLECTION);
        Document streamRecs = streamRecsCollection.find(new Document("uid", uid)).first();
        return parseRecs(streamRecs, maxItems);
    }



    private List<Recommendation> parseRecs( Document document, int maxItems ) {
        List<Recommendation> recommendations = new ArrayList<>();
        if (null == document || document.isEmpty())
            return recommendations;
        ArrayList<Document> recs = document.get("recs", ArrayList.class);
        for (Document recDoc : recs) {
            recommendations.add(new                    Recommendation(recDoc.getInteger("mid"),
recDoc.getDouble("score")));
        }
        Collections.sort(recommendations, new Comparator<Recommendation>() {
            @Override
            public int compare( Recommendation o1, Recommendation o2 ) {
                return o1.getScore() > o2.getScore() ? -1 : 1;
            }
```

```
        });
        return recommendations.subList(0, maxItems > recommendations.size() ? recommendations.size()
: maxItems);
    }

    private List<Recommendation> findContentBasedSearchRecommendations( String text, int
maxItems ) {
        MultiMatchQueryBuilder query = QueryBuilders.multiMatchQuery(text, "name", "descri");
        return
parseESResponse(esClient.prepareSearch().setIndices(Constant.ES_INDEX).setTypes(Constant.ES_
MOVIE_TYPE).setQuery(query).setSize(maxItems).execute().actionGet());
    }

    private List<Recommendation> parseESResponse( SearchResponse response ) {
        List<Recommendation> recommendations = new ArrayList<>();
        for (SearchHit hit : response.getHits()) {
            recommendations.add(new Recommendation((int) hit.getSourceAsMap().get("mid"), (double)
hit.getScore()));
        }
        return recommendations;
    }

    private List<Recommendation> findHybridRecommendations( int productId, int maxItems ) {
        List<Recommendation> hybridRecommendations = new ArrayList<>();

        List<Recommendation> cfRecs = findMovieCFRecs(productId, maxItems);
        for (Recommendation recommendation : cfRecs) {
            hybridRecommendations.add(new Recommendation(recommendation.getMid(),
recommendation.getScore() * CF_RATING_FACTOR));
        }

//      List<Recommendation> cbRecs = findContentBasedMoreLikeThisRecommendations(productId,
maxItems);
```

```
List<Recommendation> cbRecs = findContentByMongoDb(productId, maxItems);
System.out.println(cbRecs);
for (Recommendation recommendation : cbRecs) {
   hybridRecommendations.add(new Recommendation(recommendation.getMid(),
recommendation.getScore() * CB_RATING_FACTOR));
}
```

Клас User

```
public class User extends Entity{
   private String login;
   private String password;
   private String name;
   private int questionCount;
   private int questionCorrectCount;
   public String getLogin() {
      return login;
   }
   public void setLogin(String login) {
      this.login = login;
   }
   public String getPassword() {
      return password;
   }

   public void setPassword(String password) {
      this.password = password;
   }
   public String getName() {
      return name;
   }
   public void setName(String name) {
      this.name = name;
   }
```

```
    public int getQuestionCount() {
        return questionCount;
    }
    public void setQuestionCount(int questionCount) {
        this.questionCount = questionCount;
    }
    public int getQuestionCorrectCount() {
        return questionCorrectCount;
    }
    public void setQuestionCorrectCount(int questionCorrectCount) {
        this.questionCorrectCount = questionCorrectCount;
    }
}
```

Клас Servlet

```
org.apache.log4j.Logger;
import ozamkovyi.db.dao.UserDao;
import ozamkovyi.db.entity.User;
import javax.servlet.ServletException;
import javax.servlet.http.*;
import java.io.*;
public class LoginServlet extends HttpServlet {
    private static final Logger logger = Logger.getLogger(LoginServlet.class);
    private final int COOKIE_MAX_AGE = 10 * 60;
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        req.getRequestDispatcher("/html/login.html").forward(req, resp);
    }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws
ServletException, IOException {
        HttpSession session = req.getSession();
        if (req.getParameter("loginButton") != null) {
```

```
            // if login button is pressed set login and password cookies
            logger.debug("Login button is pressed");
            String login = req.getParameter("loginLabel");
            String password = req.getParameter("passwordLabel");
            Cookie cookieLogin = new Cookie("login", login);
            Cookie cookiePassword = new Cookie("password", password);
            cookieLogin.setMaxAge(COOKIE_MAX_AGE);
            cookiePassword.setMaxAge(COOKIE_MAX_AGE);
            //check for admin with current login and password
            //check for client with current login and password
            User client = new UserDao().findClientByLoginAndPassword(login, password);
            if (client != null) {
                //if client is logged in then add cookies and redirect to /clientHomepage
                logger.info("The client is logged in");
                resp.addCookie(cookieLogin);
                resp.addCookie(cookiePassword);
                session.setAttribute("currentUser", client);
                logger.info("Redirect to /clientHomepage");
                resp.sendRedirect("/clientHomepage");
            } else {
                // if wrong login or password then show message
                logger.debug("Wrong login or password");
                session.setAttribute("wrongLogin", "true");
                logger.info("Forward to /jsp/login.jsp");
                req.getRequestDispatcher("/jsp/login.jsp").forward(req, resp);
            }
        }
        if (req.getParameter("RegistrationButton") != null) {
            // if registration button is pressed then redirect to /registration
            logger.info("Redirect to /registration");
            resp.sendRedirect("/registration");
        }
    }
```

```
        }
```

Клас RegistrationServlet

```
import org.apache.log4j.Logger;

import ozamkovyi.db.dao.UserDao;

import ozamkovyi.db.entity.User;

import javax.servlet.ServletException;

import javax.servlet.http.*;

import java.io.IOException;

public class RegistrationServlet extends HttpServlet {

   private static final Logger logger = Logger.getLogger(RegistrationServlet.class);

   private final int COOKIE_MAX_AGE = 10 * 60;

   @Override

   protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {

      req.getRequestDispatcher("/html/registration.html").forward(req, resp);

   }

   @Override

   protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {

      HttpSession session = req.getSession();

      User client = new User();

      req.setCharacterEncoding("UTF-8");

      User client1 = new UserDao().findClientByLoginAndPassword(client.getLogin(),
client.getPassword());

      if (client1 == null) {

         logger.trace("New client");

         logger.debug("Add new client to bd");

         client.setLogin(req.getParameter("login"));
```

```
            client.setPassword(req.getParameter("password"));
            client.setName(req.getParameter("name"));
            client.setQuestionCount(0);
            client.setQuestionCorrectCount(0);
            session.setAttribute("currentUser", client);
            Cookie cookieLogin = new Cookie("login", req.getParameter("loginLabel"));
            Cookie cookiePassword = new Cookie("password", req.getParameter("passwordLabel"
            cookieLogin.setMaxAge(COOKIE_MAX_AGE);
            cookiePassword.setMaxAge(COOKIE_MAX_AGE);
            resp.addCookie(cookieLogin);
            resp.addCookie(cookiePassword);
            logger.info("Forward to /jsp/clientHomepage.jsp");
            req.getRequestDispatcher("/jsp/clientHomepage.jsp").forward(req, resp);
        } else {
            logger.debug("combination of login and password is already exists");
            session.setAttribute("wrongRegistrationDate", "true");
            logger.info("Forward to /jsp/registration.jsp");
            req.getRequestDispatcher("/jsp/registration.jsp").forward(req, resp);
        }
    }
}
```

Клас TagService

```
@Service
public class TagService extends BookRestApi {

    @Autowired
```

```
    private MongoClient mongoClient;

    @Autowired

    private ObjectMapper objectMapper;

    @Autowired

    private TransportClient esClient;


    private MongoCollection<Document> tagCollection;


    private MongoCollection<Document> getTagCollection(){

        if(null == tagCollection)

            tagCollection                                                                    =
mongoClient.getDatabase(Constant.MONGODB_DATABASE).getCollection(Constant.MONGODB
_TAG_COLLECTION);

        return tagCollection;

    }


    private Tag documentToTag(Document document){

        try{

            return objectMapper.readValue(JSON.serialize(document),Tag.class);

        } catch (JsonParseException e) {

            e.printStackTrace();

            return null;

        } catch (JsonMappingException e) {

            e.printStackTrace();

            return null;

        } catch (IOException e) {

            e.printStackTrace();
```

```
            return null;
        }
    }

    public void newTag(Tag tag){
        try{
            getTagCollection().insertOne(Document.parse(objectMapper.writeValueAsString(tag)));
            updateElasticSearchIndex(tag);
        }catch (JsonProcessingException e) {
            e.printStackTrace();
        }
    }
    private void updateElasticSearchIndex(Tag tag){
        GetResponse getResponse = esClient.prepareGet(Constant.ES_INDEX,Constant.ES_MOVIE_TYPE,String.valueOf(tag.getMid()))
.get();
        Object value = getResponse.getSourceAsMap().get("tags");
        UpdateRequest updateRequest = new UpdateRequest(Constant.ES_INDEX,Constant.ES_MOVIE_TYPE,String.valueOf(tag.getMid()));
        try{
            if(value == null){

updateRequest.doc(XContentFactory.jsonBuilder().startObject().field("tags",tag.getTag()).endObject()
);
            }else{

updateRequest.doc(XContentFactory.jsonBuilder().startObject().field("tags",value+"|"+tag.getTag()).e
ndObject());
            }
```

```
            esClient.update(updateRequest).get();
        } catch (InterruptedException | IOException | ExecutionException e) {
            e.printStackTrace();
        }
    }

    public List<Tag> findMovieTags(int mid){
        List<Tag> tags = new ArrayList<>();
        FindIterable<Document> documents = getTagCollection().find(new Document("mid",mid));
        for (Document document: documents) {
            tags.add(documentToTag(document));
        }
        return tags;
    }
    public List<Tag> findMyMovieTags(int uid, int mid){
        List<Tag> tags = new ArrayList<>();
        BasicDBObject basicDBObject = new BasicDBObject();
        basicDBObject.append("uid",uid);
        basicDBObject.append("mid",mid);
        FindIterable<Document> documents = getTagCollection().find(basicDBObject);
        for (Document document: documents) {
            tags.add(documentToTag(document));
        }
        return tags;
    }
```

```
    public void removeTag(int eid){
        getTagCollection().deleteOne(new Document("eid",eid));
    }

}
```

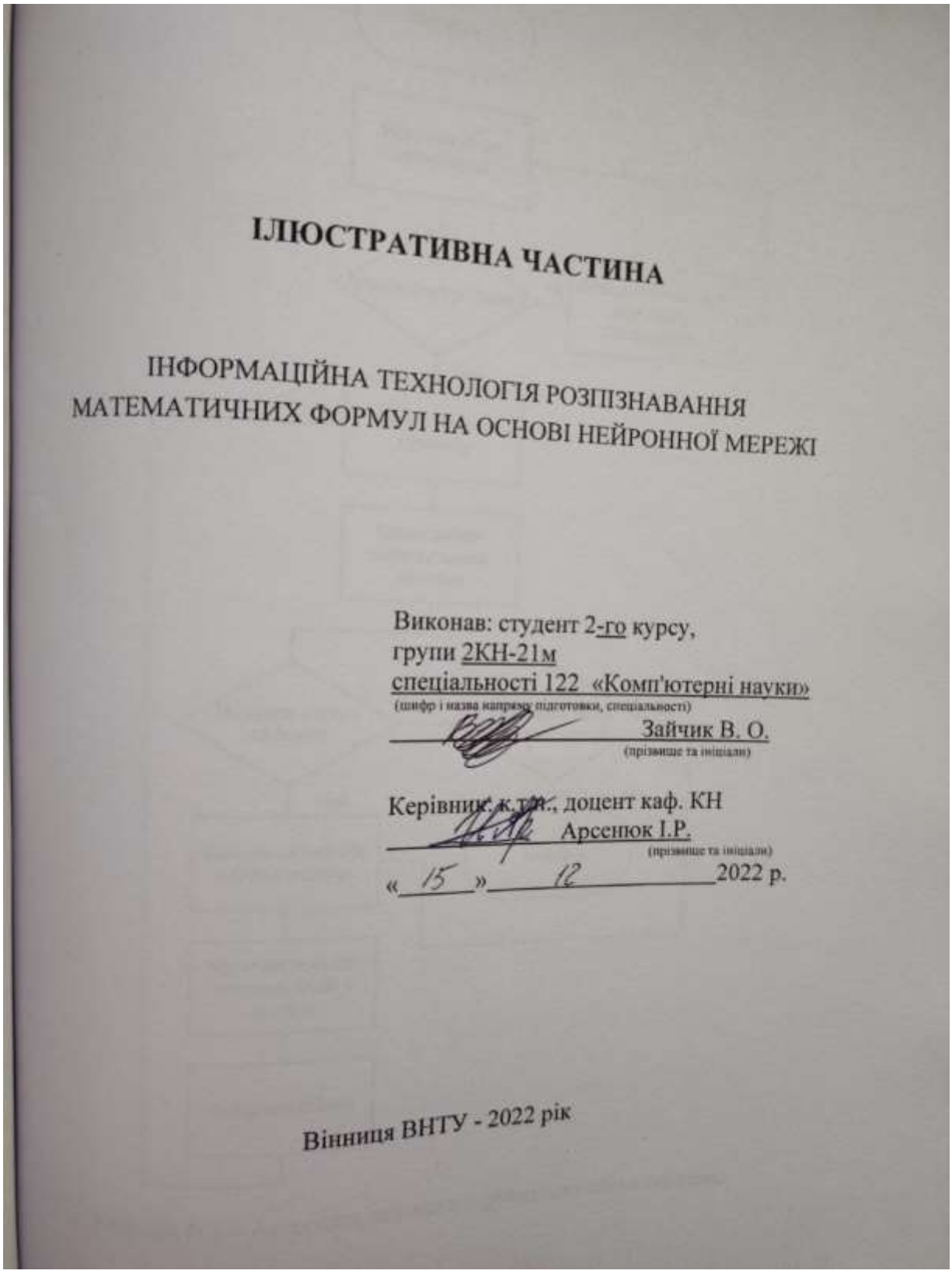
Клас UserHomepageServlet

```
import org.apache.log4j.Logger;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
public class UserHomepageServlet extends HttpServlet {
    private static final Logger logger = Logger.getLogger(UserHomepageServlet.class);
    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
            req.getRequestDispatcher("/html/UserHomepage.html").forward(req, resp);
    }
    @Override
    protected void doPost(HttpServletRequest req, HttpServletResponse resp) throws ServletException,
IOException {
        if (req.getParameter("buttonTraining") != null) {
            logger.debug("buttonTraining is pressed");
            logger.info("Redirect to /clientCardMenu");
            resp.sendRedirect("/userTraining");
        }
```

```
if (req.getParameter("buttonExam") != null) {
    logger.debug("buttonExam is pressed");
    logger.info("Redirect to /clientAccountMenu");
    resp.sendRedirect("/userExam");
}
if (req.getParameter("buttonMistake") != null) {
    logger.debug("buttonMistake is pressed");
    logger.info("Redirect to /clientPaymentMenu");
    resp.sendRedirect("/userMistake");
}
if (req.getParameter("buttonDirectory") != null) {
    logger.debug("buttonDirectory is pressed");
    logger.info("Redirect to /clientPaymentMenu");
    resp.sendRedirect("/userDirectory");
}
```

Додаток В

ІЛЮСТРАТИВНА ЧАСТИНА



ІЛЮСТРАТИВНА ЧАСТИНА

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ РОЗПІЗНАВАННЯ
МАТЕМАТИЧНИХ ФОРМУЛ НА ОСНОВІ НЕЙРОННОЇ МЕРЕЖІ

Виконав: студент 2-го курсу,
групи 2КН-21м
спеціальності 122 «Комп'ютерні науки»
(шифр і назва напряму підготовки, спеціальності)
Зайчик В. О.
(прізвище та ініціали)

Керівник: к.т.н., доцент каф. КН
Арсенюк І.Р.
(прізвище та ініціали)
«15» 12 2022 р.

Вінниця ВНТУ - 2022 рік

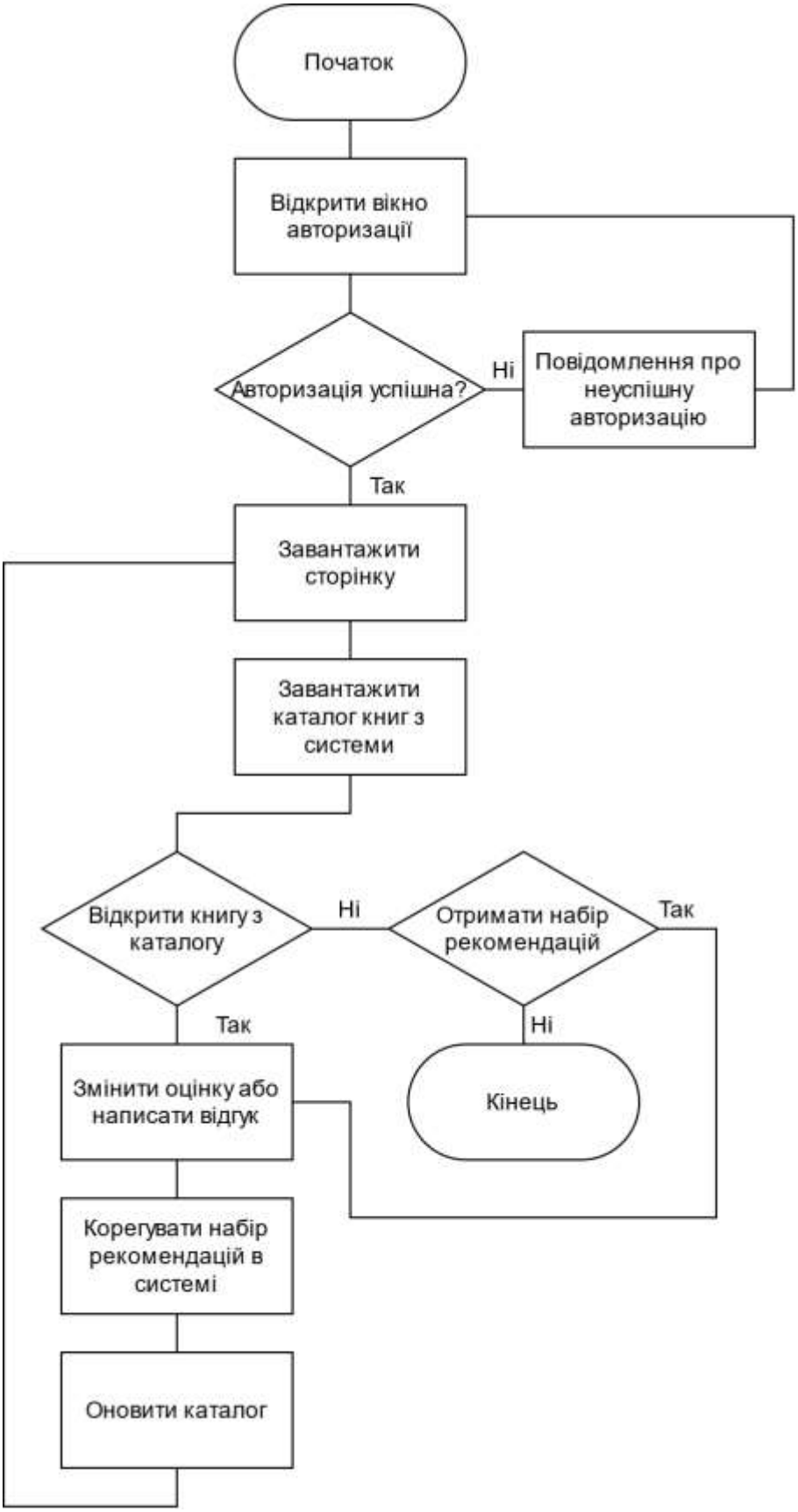


Рисунок В.1 – Алгоритм загального функціонування системи

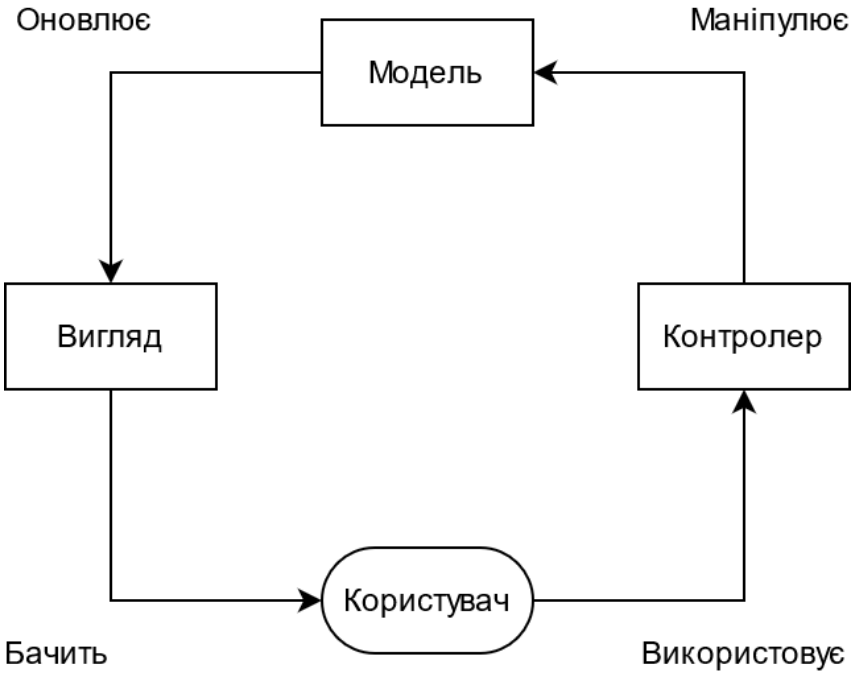


Рисунок В.2 – Загальна структурна схема

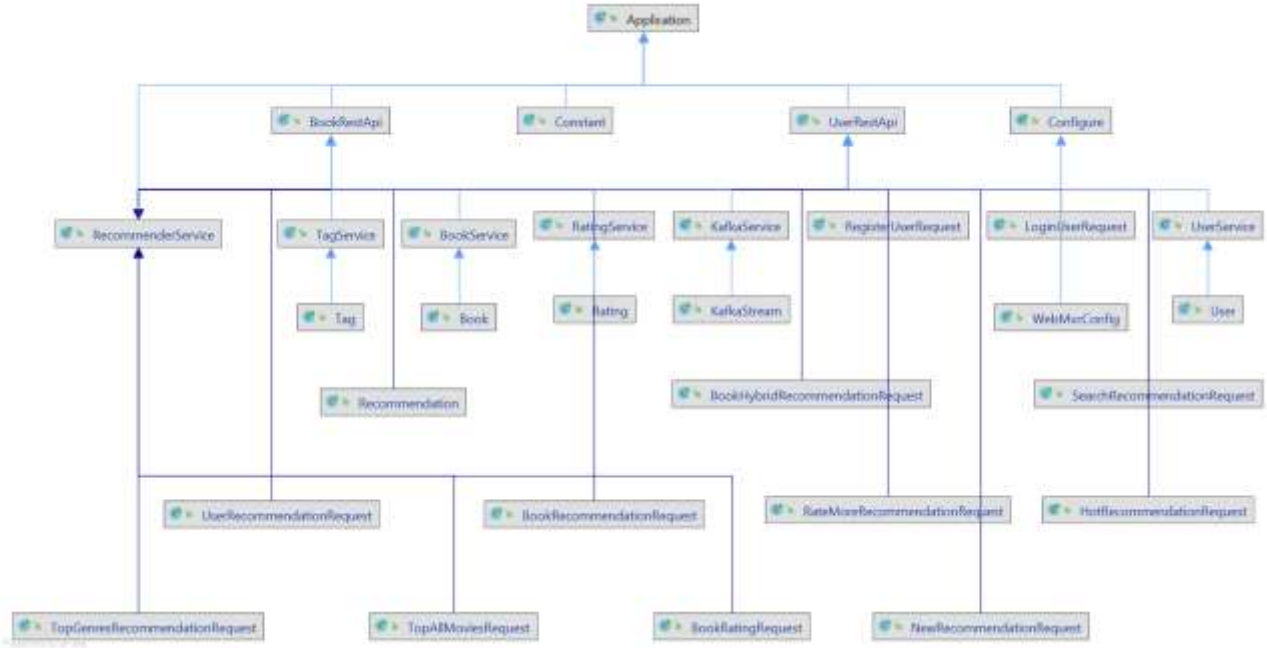


Рисунок В.3 – UML-діаграма класів

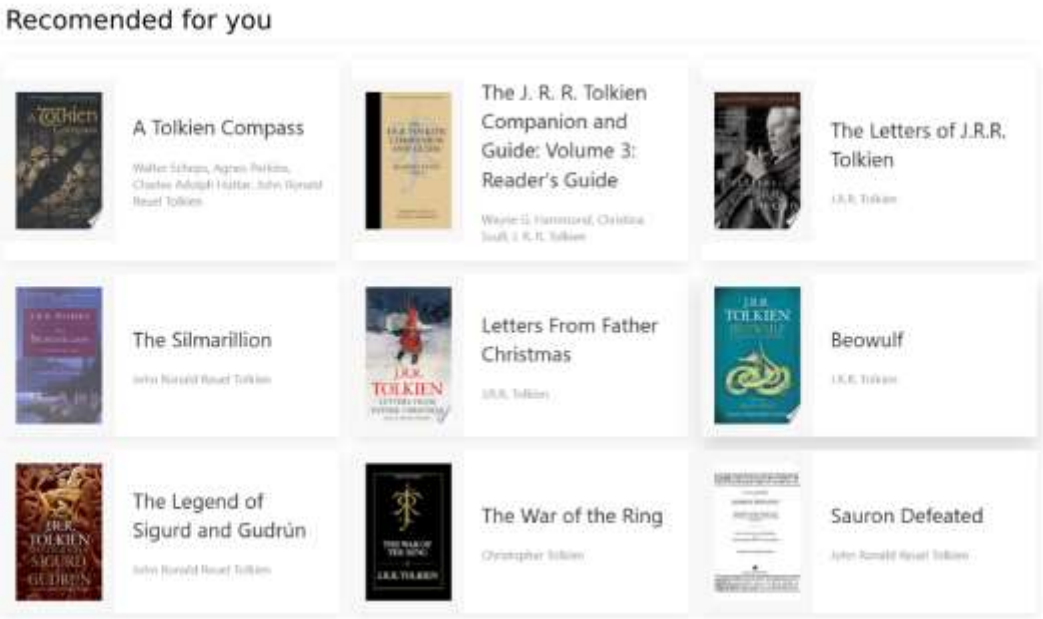


Рисунок В.4 – Вигляд інтерфейсу користувача

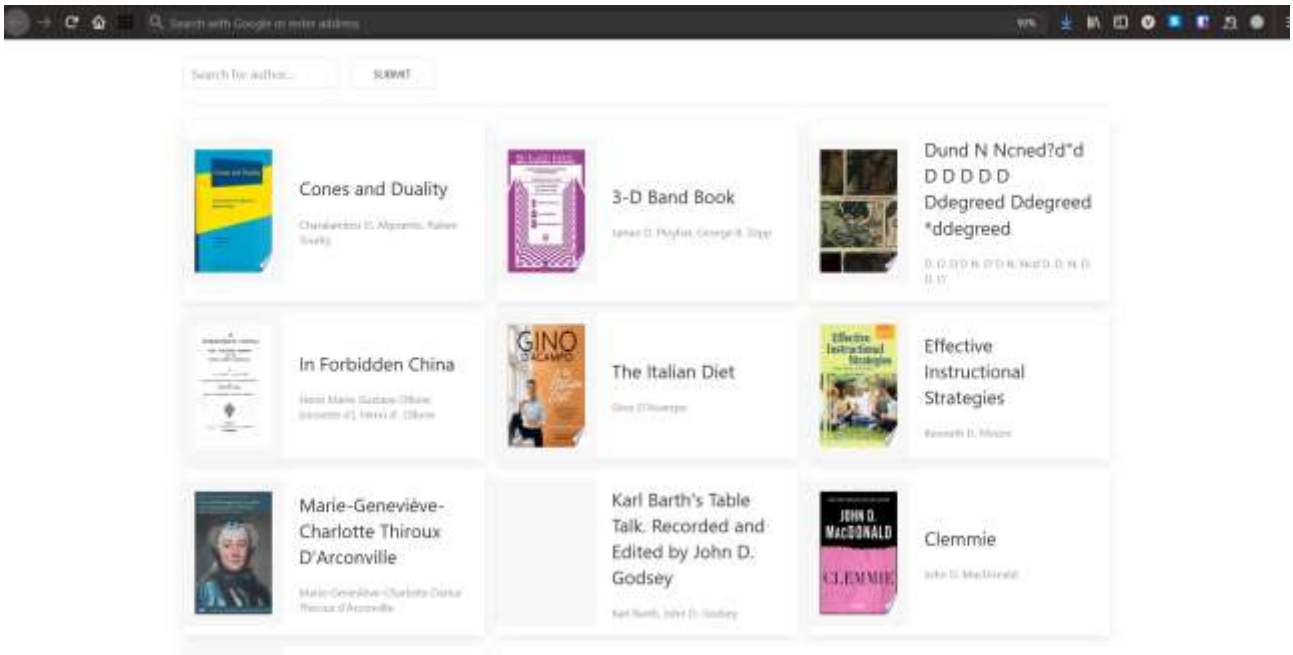


Рисунок В.5 – Вигляд інтерфейсу користувача

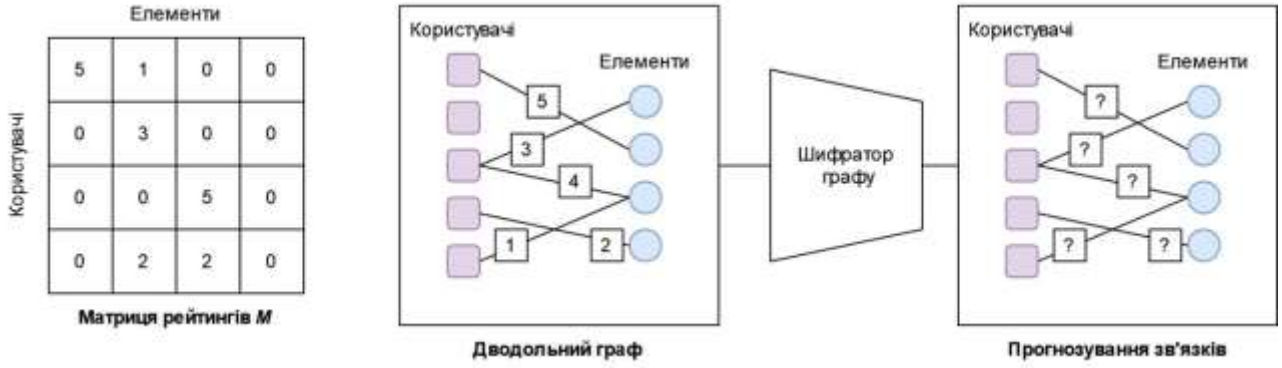


Рисунок В.6 – Загальна схема роботи методу колаборативної фільтрації на графі

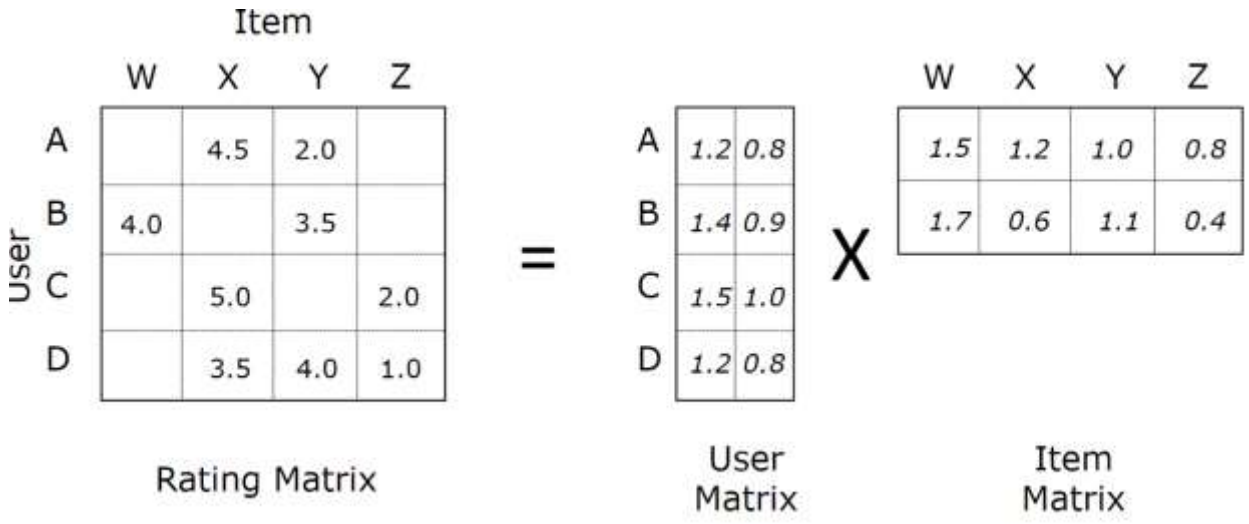


Рисунок В.7 – Матрична факторизація

Додаток Г

Інструкція користувача

Після запуску сервісу відкривається вікно авторизації користувача (рисунок А.1). Користувач повинен ввести дані свого облікового запису.

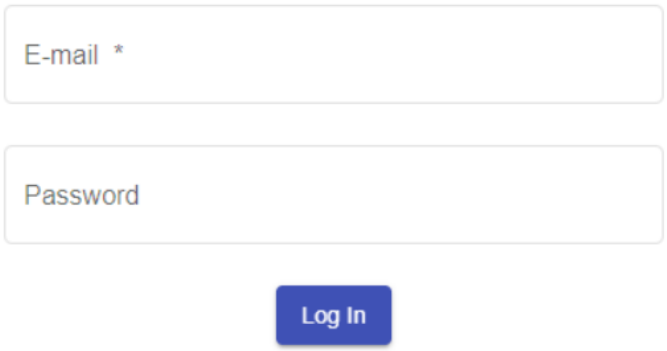


Рисунок А.1 − Авторизація користувача

Якщо такого користувача не було знайдено в системі, тоді програма зарекомендує йому зареєструватись. (рисунок А.2).

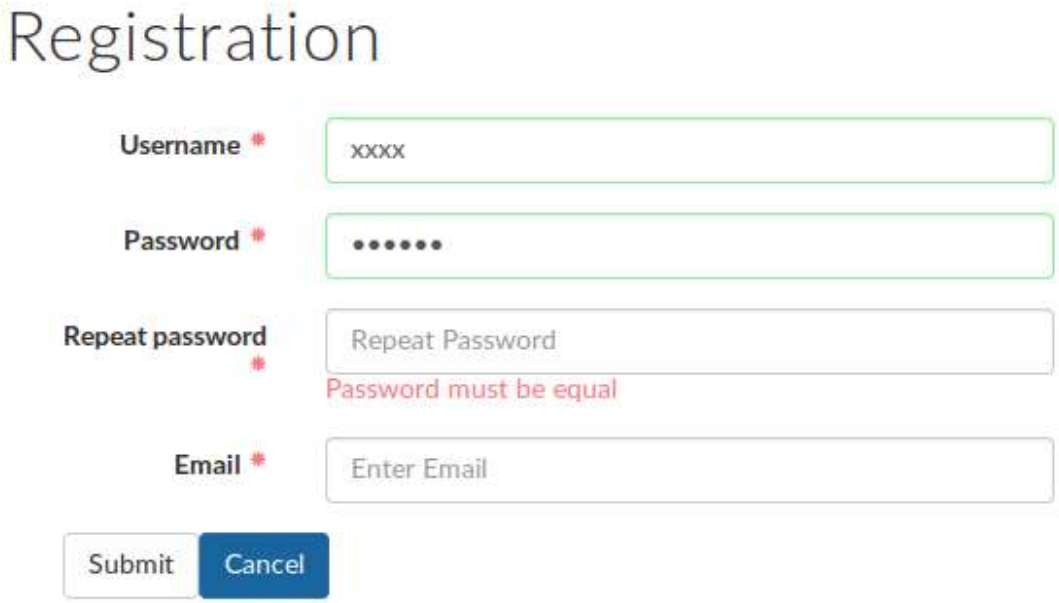


Рисунок А.2 − Реєстрація користувача

Як тільки користувач авторизувався, перед ним відкривається титульна сторінка платформи для огляду книг (рисунок А.3). Далі користувач має можливість обрати книгу.

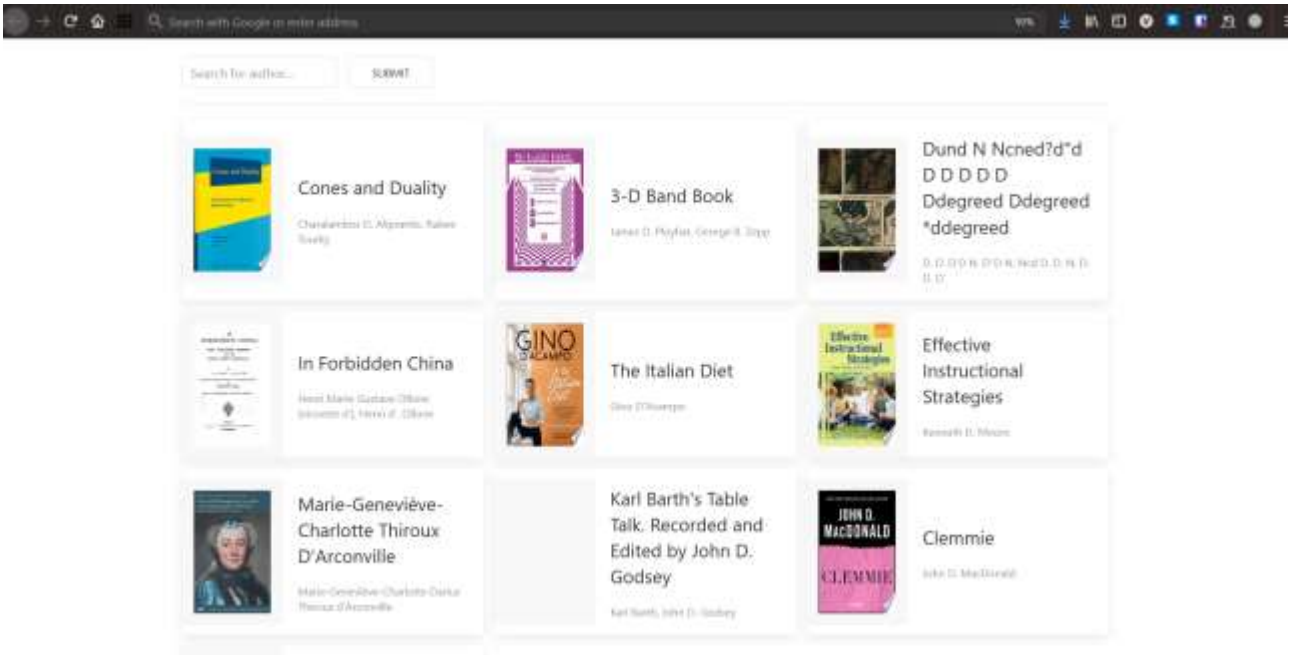


Рисунок А.3 − Головна сторінка програми

Для того щоб додати огляд на певну книгу потрібно натиснути на книгу з каталогу (рисунок А.5).

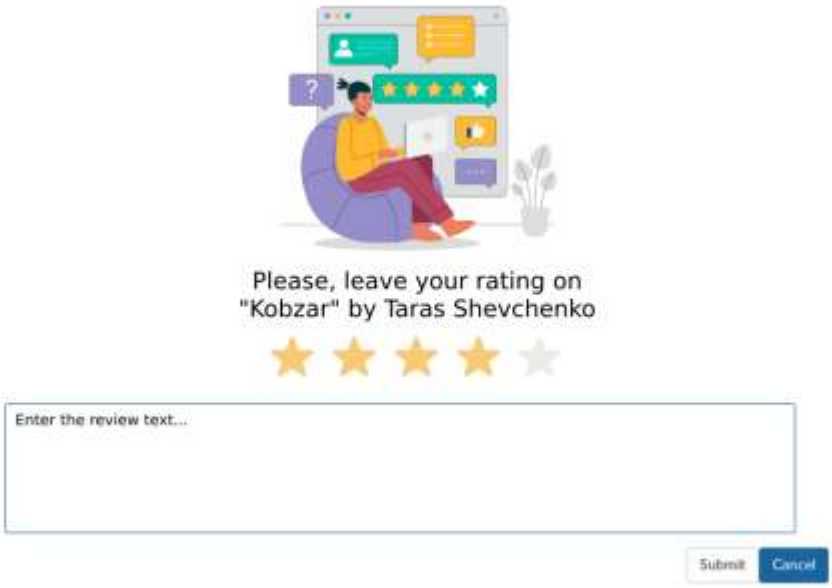


Рисунок А.4 − Проходження тестування

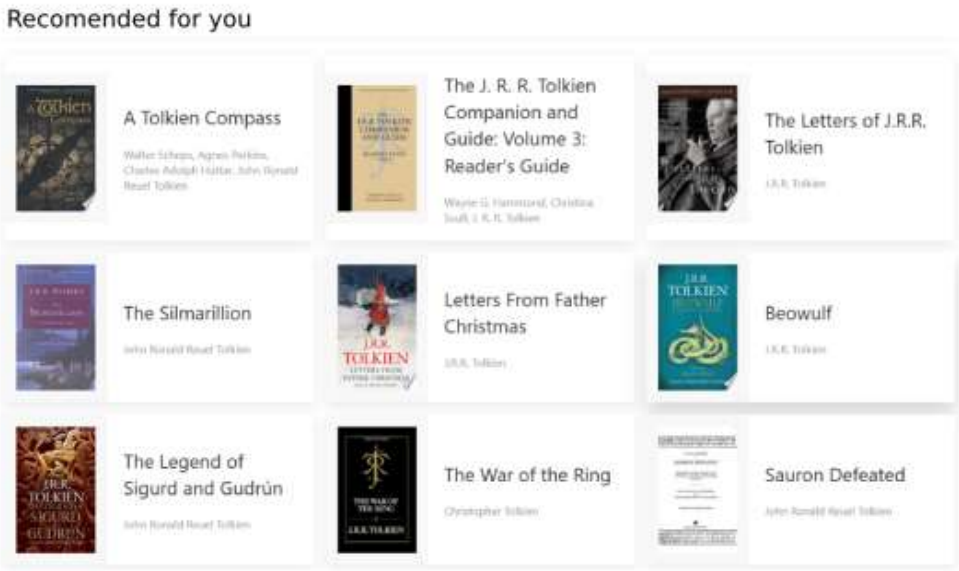


Рисунок А.5 − Режим довідника