ДИПЛОМНА РОБОТА
БАКАЛАВРА
ДРБ. ІП - 27.00.00.000 ПЗ
Група ІП-20-2
Косован Максим

2024

Міністерство освіти і науки України
Івано-Франківський національний технічний університет нафти і газу

Інститут інформаційних технологій

Кафедра інженерії програмного забезпечення

УДК 004.4

ДИПЛОМНА РОБОТА БАКАЛАВРА

Тема: “ Розробка алгоритмічного та програмного забезпечення надання персоналізованих рекомендацій ”

(назва згідно з наказом ректора)

спеціальність – 121 “Інженерія програмного забезпечення”

ПОЯСНЮВАЛЬНА ЗАПИСКА

ДРБ.ІП - 27.00.00.000 ПЗ

(позначення)

Консультанти:

з питань нормоконтролю

асист. /М.М. Піх/
(посада) (підпис) (дата)(розшифровка підпису)

Студент

ІП-20-2 /М.Ю.Косован/
(шифр групи) (підпис) (дата) (розшифровка підпису)

Керівник роботи

д.т.н., доц. /В.М.Юрчишин/
(посада) (підпис) (дата) (розшифровка підпису)

Допускається до захисту
Завідувач кафедри

к.т.н., доц. /В.В.Бандура/
(посада) (підпис) (дата) (розшифровка підпису)

2024
(рік)

Івано-Франківський національний технічний університет нафти і газу

Інститут, факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітньо-кваліфікаційний рівень бакалавр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**
доц. В.В. Бандура
“ ____ ” _____2024 р.

ЗАВДАННЯ

НА ДИПЛОМНУ РОБОТУ БАКАЛАВРА СТУДЕНТОВІ

Косовану Максиму Юрійовичу

(прізвище, ім’я, по-батькові)

1. Тема проекту (роботи) **“ Розробка алгоритмічного та програмного забезпечення надання персоналізованих рекомендацій”**

керівник проекту (роботи) Юрчишин Володимир Миколайович, проф.

затверджені наказом закладу вищої освіти від **“** ____ ” травня 2024 р. № _____

2. Строк подання студентом проекту (роботи) 17 червня 2024 р.

3. Вихідні дані до проекту (роботи) Результати і матеріали отримані під час проходження переддипломної практики

4. Зміст розрахунково - пояснювальної записки (перелік питань, які потрібно розробити)

1. Аналіз предметної області побудови рекомендаційних систем
2. Аналіз вимог та моделювання програмного забезпечення системи надання рекомендацій
3. Опис бізнес-процесів розроблюваної системи
4. Розробка архітектури та реалізація системи надання персоналізованих рекомендацій
5. Розгортання системи надання рекомендацій

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Алгоритм фільтрування вмісту (рис. 1.1)
2. Алгоритм колаборативної фільтрації (рис. 1.2)
3. Колаборативна фільтрація на основі користувачів (рис. 1.3)
4. Колаборативна фільтрація на основі вмісту (рис. 1.4)
5. Гібридна модель рекомендаційної системи (рис. 1.5)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання **22 травня 2024 р.**

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту	Примітка
1	Аналіз предметної області побудови рекомендаційних систем	26.05.2024	виконано
2	Аналіз вимог та моделювання програмного забезпечення системи надання рекомендацій	29.05.2024	виконано
3	Опис бізнес-процесів розроблюваної системи	30.05.2024	виконано
4	Розробка архітектури та реалізація системи надання персоналізованих рекомендацій	04.06.2024	виконано
5	Розгортання системи надання рекомендацій	10.06.2024	виконано
6	Оформлення пояснювальної записки дипломної роботи завідувачем кафедри	17.06.2024	виконано

Студент – дипломник _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Виконана дипломна робота присвячена розробці програмного забезпечення для системи, яка надає персоналізовані рекомендації щодо вибору комп'ютерних ігор. Використовуючи алгоритми аналізу даних, система здатна адаптуватися до індивідуальних вподобань та стилів гри користувачів. Розроблене програмне забезпечення враховує різноманітні фактори, такі як жанр гри, графічні налаштування, історія гри користувача та відгуки інших гравців. Це дозволяє створювати індивідуальні рекомендації, що відповідають унікальним потребам кожного гравця.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, РЕЙТИНГ, ВІДЕОІГРИ, ПЕРСОНАЛІЗОВАНІ РЕКОМЕНДАЦІЇ, КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, ОЦІНКА, РЕКОМЕНДАЦІЇ

ANNOTATION

The completed thesis is devoted to the development of software for a system that provides personalized recommendations for the selection of computer games. Using data analysis algorithms, the system is able to adapt to the individual preferences and playing styles of users. The developed software takes into account various factors such as game genre, graphic settings, user's game history and feedback from other players. This allows you to create personalized recommendations that meet the unique needs of each player.

KEYWORDS: WEB APPLICATION, RATING, VIDEO GAMES, PERSONALIZED RECOMMENDATIONS, COLLABORATIVE FILTERING, SCORE, RECOMMENDATIONS

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 8
ВСТУП 9

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ПОБУДОВИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ 11
1.1. Особливості рекомендаційних систем 11
1.2. Алгоритми щодо побудови рекомендацій 15
1.2.1. Фільтрування вмісту 16
1.2.2. Колаборативна фільтрація 17
1.2.3. Гібридні методи фільтрації 20
1.3. Загальний підхід до розробки рекомендаційних систем 22
1.4. Дослідження проблем які виникають під час надання рекомендацій 25
1.5. Аналіз проблеми холодного старту в системах надання рекомендацій 28
1.6. Аналіз методів побудови рекомендацій в ситуації холодного старту 31
1.7. Основні вимоги до розроблюваної системи надання рекомендацій 33

2. АНАЛІЗ ВИМОГ ТА МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ НАДАННЯ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ 35
2.1. Аналіз відомих програмних продуктів надання рекомендацій 35
2.2. Аналіз вимог до розробки. Розробка діаграм використання 39
2.3. Опис бізнес-процесів розроблюваної системи 44
2.4. Розробка структури проекту 47
2.5. Розробка діаграми сутностей шару даних та їх опис 48

2.6. Реалізація шару бізнес логіки 52

3. РОЗРОБКА АРХІТЕКТУРИ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ НАДАННЯ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ 57

3.1. Вибір архітектури веб-застосунку 57

3.2. Розробка інтерфейсу входу в систему 62

3.3. Реалізація основних функцій системи надання персоналізованих рекомендацій 64

3.4. Розгортання системи надання рекомендацій 70

ВИСНОВКИ 74

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ 76

ДОДАТКИ

БІБЛІОГРАФІЧНА ДОВІДКА

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

REST – Representational State Transfer – передача репрезентативного стану;

SVD – Singular Value Decomposition – сингулярне розкладання величини;

RS – Recommender System – Рекомендаційна система

ВСТУП

Сучасний світ досить важко собі уявити без такої технології, як Інтернет. З його допомогою компанії можуть продавати що завгодно, починаючи з продуктів і ліків й закінчуючи виробами мистецтва. Проте не існує досконалих технологій і відповідно у всього є і недоліки, а саме, надлишок товарів і складність вибору для користувача необхідного товару саме для нього, так як виробники намагаються зібрати навколо себе якомога більшу цільову групу.

Різноманітні рекомендаційні системи з'явилися в інтернеті досить давно, близько 20 років тому. Однак справжній прорив в цій області трапився приблизно 10-15 років тому, коли відбулося змагання Netflix Prize.

Тоді компанія Netflix надавала в прокат VHS-касети і DVD, а не цифрові копії. Вже в той час для них було дуже важливо підвищити якість рекомендацій. Чим краще компанія рекомендує користувачам свої фільми, тим більше касет або дисків вони беруть в прокат. Відповідно, підвищується і прибуток. У 2006 році Netflix запустили змагання Netflix Prize. Вони поширили у відкритому доступі зібрані дані: близько 100 мільйонів оцінок за п'ятибальною шкалою з ідентифікаторами користувачів, які поставили оцінку. Учасникам цього змагання необхідно було якнайкраще передбачати оцінку користувача, яку він поставить тому чи іншому фільму.

Звіт про світовий ринок ігор показав, що на кінець 2022 року кількість активних геймерів мала становити 3,2 мільярда гравців. За прогнозами, кількість гравців з кожним роком буде лише збільшуватися. Прибуток світового ринку складе 196,8 мільярда доларів, що більше на 2,1%, ніж у минулому році. Очікується його подальше зростання, попри складні економічні умови.

Беручи до уваги попит до галузі відеоігор, метою є полегшення доступу до потенційно цікавих відеоігор, а запланований до розробки в

даному проєкті інформаційний сервіс із рекомендаційною підсистемою стане засобом досягнення цієї мети.

В результаті наразі існує велика кількість різноманітних алгоритмів, від дуже простих і тривіальних до дуже складних, що враховують величезну кількість факторів. Методи, які використовують, найчастіше можливо поділити на три групи – колаборативна фільтрація, контентна фільтрація та експертні системи. Звичайно, у кожної групи є свої недоліки, тому пізніше з'явилися ще четверта група – гібридні, які поєднують в собі декілька підходів з усіх трьох груп.

Окрім цього, рекомендаційні системи часто використовуються в контексті штучного інтелекту. Можливість видачі рекомендацій, прогнозування подій і підкреслення кореляцій – все це результати застосування ШІ. З іншого боку, при імплементації рекомендаційних систем часто використовуються прийоми машинного навчання. Нейронні мережі дозволяють швидко вирішувати складні завдання і з легкістю маніпулювати великими даними. Взявши в якості вхідних даних список книжок, і зіставивши висновок з одними оцінками, мережа може засвоїти правила і потім з їх допомогою прогнозувати подальші рейтинги, які може проставити конкретний користувач.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ПОБУДОВИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

1.1. Особливості рекомендаційних систем

Рекомендаційні системи націлені на те, щоб передбачити інтереси користувачів, які, швидше за все, їм цікаві. Це одні з найпотужніших систем машинного навчання.

Рекомендації зазвичай прискорюють пошук і полегшують користувачам доступ до цікавого для них контенту, а також дивують їх пропозиціями, які вони ніколи б не шукали.

Більш того, компанії можуть залучати й утримувати клієнтів, розсилаючи електронні листи з посиланнями на нові пропозиції, що відповідають інтересам одержувачів. Користувач починає відчувати, що його знають і розуміють, і він з більшою ймовірністю споживе більше контенту. Знаючи, чого хоче користувач, компанія отримує конкурентну перевагу, і загроза втрати клієнта на користь конкурента зменшується. Забезпечення цієї додаткової цінності для користувачів шляхом включення рекомендацій в системи і продукти є привабливим. Крім того, це дозволяє компаніям випереджати своїх конкурентів і в кінцевому підсумку збільшувати свої доходи.

Більш того рекомендаційні системи - корисна альтернатива пошуковим алгоритмам, оскільки вони допомагають користувачам знаходити елементи, які інакше вони могли б не знайти. Слід зазначити, що рекомендаційні системи часто реалізуються з використанням пошукових систем, індексуючих нетрадиційні дані.

Рекомендаційні системи були вперше згадані в технічному звіті як «цифрова книжкова полиця» в 1990 році Юссі Карлгрену з Колумбійського університету, а з 1994 року збільшувалися в масштабах і опрацьовувалися в

технічних звітах і публікаціях Юссі Карлгрену, потім в SICS, і дослідницькими групами на чолі з Патті Мейс з Массачусетського технологічного інституту, Уіллом Хіллом з Bellcore, і Полом Резником з Массачусетського технологічного інституту, чия робота з GroupLens була удостоєна нагороди ACM Software Systems Award 2010 року.

М. Монтанер, Беатріс Лопес, Естеве в «Таксономія рекомендаційних агентів в Інтернеті» 2003 року подали перший огляд рекомендаційних систем з точки зору інтелектуального агента. А. Тужилін, Г. Адомавічус в своїй роботі «До наступного покоління рекомендаційних систем: огляд сучасного стану та можливих розширень» представили новий альтернативний огляд рекомендаційних систем. Дж. Л. Херлокер в 2004 році надав додатковий огляд методів оцінки рекомендаційних систем.

Однією з подій, які стимулювали дослідження в області рекомендаційних систем, стала премія Netflix. З 2006 по 2009 рік Netflix спонсорував конкурс, пропонуючи головний приз в розмірі 1 млн доларів команді, яка могла взяти запропонований набір даних з більш ніж 100 мільйонів оцінок фільмів і видати рекомендації, які були на 10% точніше, ніж ті, які пропонує існуюча система рекомендацій компанії. Цей конкурс стимулював пошук нових і більш точних алгоритмів. 21 вересня 2009 року головний приз був присуджений команді BellKor Pragmatic Chaos за правилами тай-брейка.

Найточніший алгоритм 2007 року використовував метод ансамблю з 107 різних алгоритмічних підходів, об'єднаних в один прогноз. Завдяки проекту Netflix інтернет отримав безліч переваг. Деякі команди взяли свої технології і застосували їх на інших ринках. Деякі члени команди, що зайняла друге місце, заснували Gravity R&D, механізм рекомендацій, який бере активну участь в співтоваристві RecSys. 4-Tell, Inc. створила рішення на основі проекту Netflix для веб-сайтів електронної комерції.

У зв'язку з набором даних, запропонованим Netflix для конкурсу Netflix Prize, виник ряд проблем з конфіденційністю. Хоча набори даних були анонімними, щоб зберегти конфіденційність клієнтів, в 2007 році два дослідники з Техаського університету змогли ідентифікувати окремих користувачів, зіставивши набори даних з рейтингами фільмів в базі даних Internet Movie Database. В результаті в грудні 2009 року анонімний користувач Netflix подав до суду на Netflix в справі Доу проти Netflix, стверджуючи, що Netflix порушила закони США про справедливу торгівлю і Закон про захист конфіденційності відео, опублікувавши набори даних. Це, а також побоювання Федеральної торгової комісії, привели до скасування другого конкурсу Netflix Prize у 2010 році.

Зазвичай дослідження рекомендаційних систем спрямовані на пошук найбільш точних алгоритмів рекомендацій. Однак є ряд чинників, які також важливі.

Різноманітність - користувачі, як правило, більше задовольняються рекомендаціями, коли існує більша різноманітність всередині списку.

Наполегливість рекомендацій - в деяких ситуаціях більш ефективно повторно показати рекомендації або дозволити користувачам повторно оцінити елементи, ніж показувати нові елементи. На це є кілька причин. Користувачі можуть ігнорувати елементи, коли вони відображаються вперше, наприклад, тому що у них не було часу уважно вивчити рекомендації.

Конфіденційність - рекомендаційним системам зазвичай доводиться мати справу з проблемами конфіденційності, тому що користувачі повинні розкривати конфіденційну інформацію. Створення профілів користувачів з використанням спільної фільтрації може бути проблематичним з точки зору конфіденційності.

У багатьох європейських країнах існує сильна культура конфіденційності даних, і кожна спроба ввести будь-який рівень

профілювання користувачів може привести до негативної реакції клієнтів. У цій області було проведено багато досліджень з поточних питань конфіденційності. Рамакришнан і ін. провели широкий огляд компромісів між персоналізацією і конфіденційністю і виявили, що поєднання слабких зв'язків (несподіване поєднання, яке дає випадкові рекомендації) та інших джерел даних можна використовувати для розкриття особистості користувачів в анонімізовуваном наборі даних.

Демографічні дані користувачів - Джоран Біл та ін. виявили, що демографічні дані користувачів можуть впливати на те, наскільки користувачі задоволені своїми рекомендаціями. У своїй статті вони показують, що літні користувачі, як правило, більше зацікавлені в рекомендаціях, ніж молоді.

Надійність - коли користувачі можуть брати участь в рекомендаційної системі, необхідно вирішити проблему шахрайства.

Інтуїція - це міра того, «наскільки дивні рекомендації». Наприклад, система рекомендацій, яка рекомендує молоко покупцеві в продуктовому магазині, може бути абсолютно точною, але це не дуже хороша рекомендація, тому що це очевидний товар для покупця. «Інтуїція» служить двом цілям: по-перше, зменшується ймовірність того, що користувачі втратять інтерес через занадто однакового набору варіантів. По-друге, ці елементи необхідні для того, щоб алгоритми вчилися і удосконалювалися.

Довіра - рекомендаційна система не має великого значення для користувача, якщо він не довіряє системі. Довіру можна створити за допомогою рекомендаційної системи, пояснивши, як вона генерує рекомендації і чому рекомендує будь-який елемент.

Маркування - на задоволеність користувачів рекомендаціями може впливати маркування рекомендацій. Так, Біл, Джоран, і ін. в «Рекламні і органічні (дослідні роботи) рекомендації і вплив маркування» в роботі кліків (CTR) для рекомендацій помічені як «рекламні» були нижче (CTR = 5,93%),

ніж CTR для ідентичних рекомендацій позначені як «органічний» (CTR = 8,86%).

1.2. Алгоритми щодо побудови рекомендацій

Система рекомендацій працює з великим обсягом інформації, фільтруючи найважливішу інформацію на основі даних, наданих користувачем, та інших факторів, які враховують уподобання та інтереси користувача. Він визначає відповідність між користувачем і елементом і визначає схожість між користувачами та елементами для рекомендації. І користувачі, і послуги, що надаються, отримали вигоду від таких систем. Завдяки таким системам якість і процес прийняття рішень також покращилися.

Системи рекомендацій є ефективними інструментами для фільтрації онлайн-інформації, яка широко поширена через зміну звичок користувачів комп'ютерів, тенденції персоналізації та нові доступи до Інтернету. Навіть незважаючи на те, що останні рекомендаційні системи видатні у наданні точних рекомендацій, вони страждають від різноманітних обмежень і проблем, таких як масштабованість, холодний запуск, розрідженість тощо.

Через існування різноманітних методів вибір методів стає складною роботою під час створення програми - орієнтовані системи рекомендацій. Крім того, кожна техніка має власний набір особливостей, переваг і недоліків, що викликає ще більше питань, на які слід звернути увагу. Ця стаття спрямована на систематичний огляд різноманітних нещодавніх внесків у область рекомендаційних систем, зосереджуючись на різноманітних додатках, таких як книги, фільми, продукти тощо. Спочатку аналізуються різні застосування кожної рекомендаційної системи. Потім виконується алгоритмічний аналіз різних систем рекомендацій і складається таксономія, яка враховує різні компоненти, необхідні для розробки

ефективної системи рекомендацій. Крім того, оцінюються та зазначаються зібрані набори даних, платформа моделювання та показники продуктивності, зосереджені на кожному внеску.

Нарешті, цей огляд надає вкрай необхідний огляд поточного стану досліджень у цій галузі та вказує на існуючі прогалини та проблеми, щоб допомогти нащадкам розробити ефективну систему рекомендацій.

Існує три основні підходи до рекомендацій:

1. Фільтрування вмісту (item-based).

2. Колаборативна фільтрація (user-based).

3. Гібридний підхід.

1.2.1. Фільтрування вмісту

Алгоритми фільтрації на основі вмісту призначені для рекомендації продуктів на основі накопичених знань користувачів. Ця техніка спрямована на порівняння інтересів користувача з характеристиками продукту, тому важливо надати значущу характеристику продуктів у системі.

Перед розробкою системи першочерговим завданням має бути вибір улюблених функцій кожного покупця. Ці дві стратегії можна застосовувати у можливій комбінації. По-перше, користувачеві надається список функцій, щоб вибрати найбільш цікаві. По-друге, алгоритми зберігають облік усіх продуктів, вибраних користувачем у минулому, і формують дані про поведінку клієнта.

Профіль покупця змінюється навколо вибору, смаків і вподобань покупця та формує рейтинг покупця. Він включає в себе, скільки разів окремий покупець натискає на зацікавлені продукти або скільки разів подобається ці продукти в списку бажань. Фільтрування на основі вмісту полягає в схожості між елементами. Близькість і схожість продукту вимірюються на основі подібного вмісту товару. Коли ми говоримо про

контент, він включає жанр, категорію предмета тощо. Візьмемо для прикладу системи рекомендацій у фільмах.

Припустімо, є чотири фільми, у яких користувачеві спочатку подобаються лише два фільми. Проте 3-й фільм схожий на 1-й за жанром, тому система автоматично запропонує 3-й фільм. Це те, що автоматично генерується системою рекомендацій на основі вмісту на основі подібності вмісту.

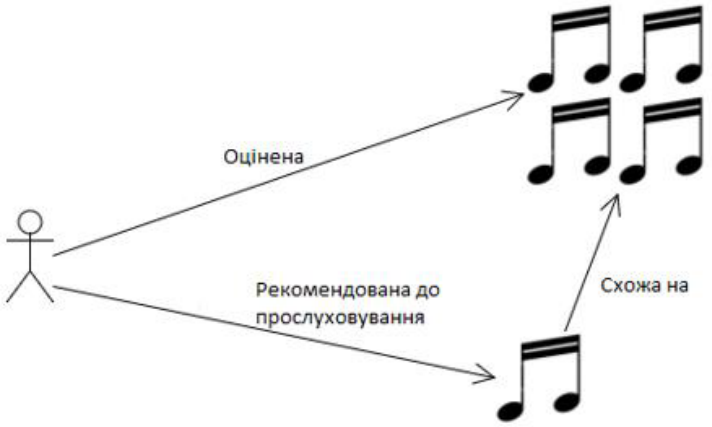


Рисунок 1.1 - Алгоритм фільтрування вмісту

Система фільтрації на основі вмісту не вимагає жодної інформації про покупця, оскільки пропозиція стосується лише покупця та полегшує масштабування для багатьох покупців. Ця система вловлює інтереси користувачів і пропонує товари, якими користується небагато покупців.

1.2.2. Колаборативна фільтрація

Колаборативна фільтрація базується на том, що існують зв’язки між продуктами та інтересами людей. Багато систем рекомендацій використовують спільну фільтрацію, щоб знайти ці зв’язки та дати точну рекомендацію продукту, який може сподобатися користувачеві чи зацікавити його.

Демонстрація алгоритму представлена на рисунку 1.2.

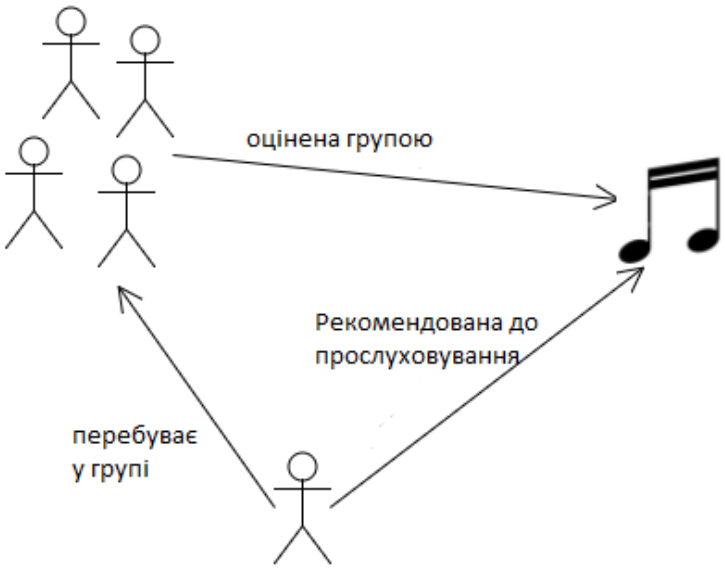


Рисунок 1.2 - Алгоритм колаборативної фільтрації

Колаборативна фільтрація має два підходи: на основі користувачів і на основі змісту. Колаборативна фільтрація на основі користувачів базується на схожості користувачів або сусідстві. Спільна фільтрація на основі елементів базується на подібності між елементами.

У колаборативній фільтрації на основі користувачів у нас є активний користувач, для якого націлена рекомендація. Механізм спільної фільтрації спочатку шукає схожих користувачів. Це користувачі, які поділяють моделі рейтингу активних користувачів. Спільна фільтрація ґрунтується на цій подібності на таких речах, як історія, уподобання та вибір, який користувачі роблять під час покупки, перегляду або задоволення від чогось. Наприклад, музичні композиції, які подібні користувачі оцінили високо. Потім він використовує оцінки цих схожих користувачів, щоб передбачити можливі оцінки активного користувача для муз. композиції, які він раніше не слухав.

Наприклад, якщо 3 користувачі схожі або є сусідами за цікавими музичними композиціями, ми можемо порекомендувати активному користувачеві музичну композицію, яку вже бачили її сусіди, що показано на рисунку 1.3.

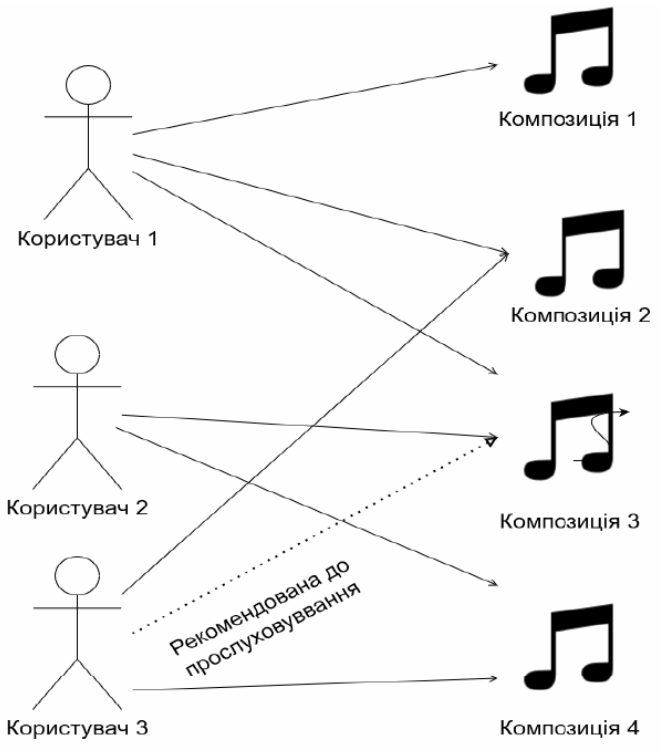


Рисунок 1.3 - Колаборативна фільтрація на основі користувачів

У підході, орієнтованому на користувача, рекомендація ґрунтується на користувачах тої групи, з якими користувач має спільні переваги. Наприклад, оскільки Користувачу 1 і Користувачу 3 сподобалися Елемент 3 і Елемент 4, ми вважаємо їх схожими або сусідніми користувачами та рекомендуємо Елемент 1, який Користувач 1 позитивно оцінив для Користувача 3. У підході на основі елементів подібні елементи будувати околиці на поведінці користувачів. Однак зауважте, що він не базується на їхньому вмісті. Наприклад, Елемент 1 і Елемент 3 вважаються сусідами, оскільки вони були позитивно оцінені як Користувачем 1, так і Користувачем 2. Отже, Елемент 1 можна рекомендувати Користувачеві 3, оскільки він уже виявив інтерес до Елементу 3. Тому рекомендації тут такі: на основі елементів у сусідстві, які користувач може віддати перевагу.

Колаборативна фільтрація на основі вмісту дає рекомендації на основі взаємодії користувача з продуктом у минулому. Припущення, що лежить в

основі алгоритму, полягає в тому, що користувачам подобаються подібні продукти, яким він поставив гарний відгук та не подобаються схожі продукти з тими, де в нього поганий відгук, тому вони дають однакові оцінки подібним продуктам.

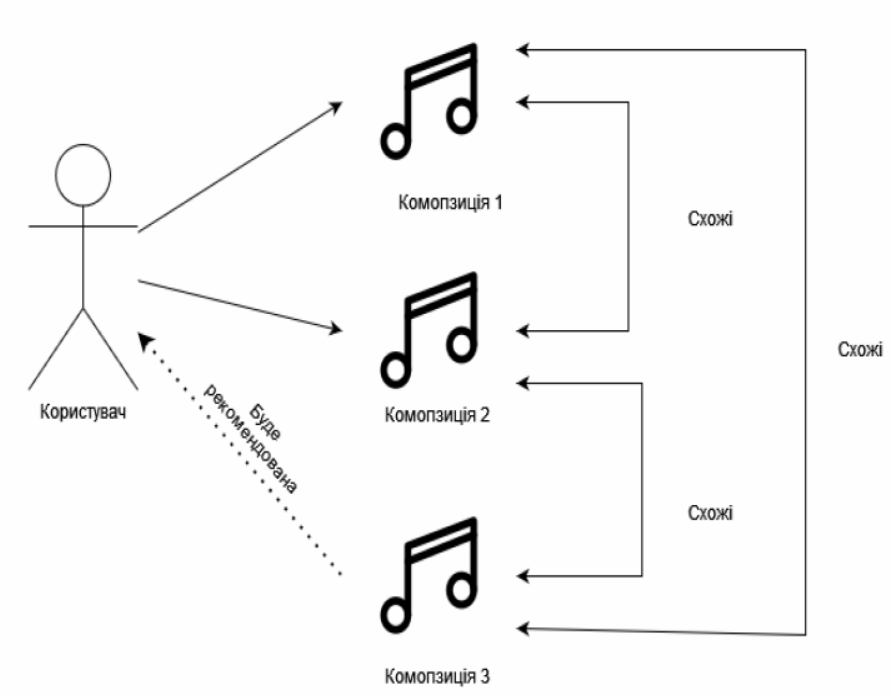


Рисунок 1.4 – Колаборативна фільтрація на основі вмісту

1.2.3. Гібридні методи фільтрації

Більшість систем рекомендацій зараз використовують гібридний підхід (рис. 1.5), поєднуючи спільну фільтрацію, фільтрацію на основі вмісту та інші підходи. Немає жодних причин, чому кілька різних технік одного типу не можуть бути змішані. Гібридні підходи можна реалізувати декількома способами: створюючи прогнози на основі контенту та на основі співпраці окремо, а потім об’єднуючи їх; шляхом додавання можливостей на основі контенту до підходу на основі співпраці (і навпаки); або шляхом уніфікації підходів в одну модель. Декілька досліджень, які емпірично порівнюють ефективність гібридних методів із суто спільними методами та методами, заснованими на вмісті, продемонстрували, що гібридні методи можуть

надати більш точні рекомендації, ніж чисті підходи. Ці методи також можна використовувати для подолання деяких загальних проблем у системах рекомендацій, таких як холодний запуск і проблема розрідженості, а також вузьке місце інженерії знань у підходах, заснованих на знаннях. Netflix є хорошим прикладом використання гібридних систем рекомендацій. Веб-сайт дає рекомендації, порівнюючи звички перегляду та пошуку подібних користувачів (тобто спільне фільтрування), а також пропонуючи фільми, які мають спільні характеристики з фільмами, які користувачі високо оцінили (фільтрування на основі вмісту).

Є декілька основних варіантів комбінування різних методів у рекомендаційних системах гібридного типу:

1. Введення єдиної моделі, яка використовує характеристики обох методів.

2. Інтегрування item-based характеристик в user-based системи.

3. Інтегрування user-based характеристик у item-based системи.

4. Інтегрування item-based та user-based підходів окремо.

Спрощено цей метод показано на рисунку 1.5.



Рисунок 1.5 - Гібридна модель рекомендаційної системи

1.3. Загальний підхід до розробки рекомендаційних систем

Останнім часом популярність рекомендаційних систем стала дуже великою. Це пояснюється прагненням компаній спростити вибір користувача у тій чи іншій ситуації та отримати від цього прибуток. З огляду на велику затребуваність систем, які автоматично формують пропозиції користувачам з урахуванням їх інтересів, робляться спроби розробки універсальних рекомендаційних систем. Загальна схема роботи з рекомендаційною системою показано на рис. 1.6. Пропонується інтегрувати систему з основним сервісом компанії, що пропонує товари чи послуги, налаштувавши її на відповідну область через створення моделі предметної області на основі даних, які отримують від основного сервісу у певному форматі.

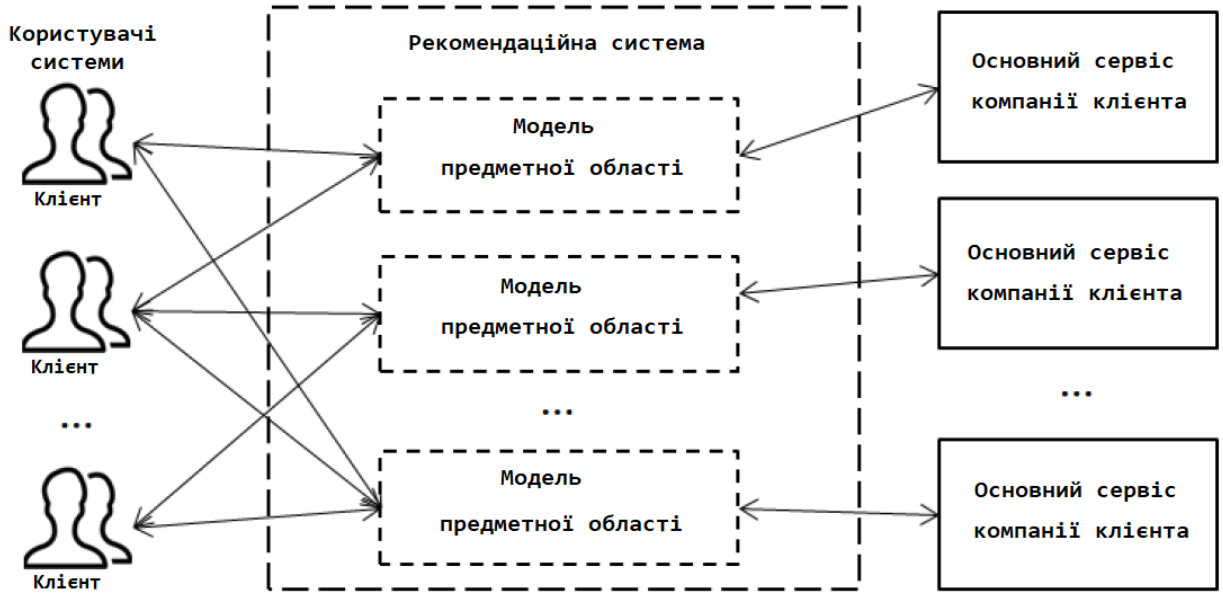


Рисунок 1.6 – Схема інтеграції рекомендаційної системи з основними сервісами компаній-клієнтів

Універсальна рекомендаційна система включає дві підсистеми, що взаємодіють з основним сервісом: власне рекомендаційна система, яка

забезпечує збір даних та вироблення рекомендацій; модуль обробки та інтелектуального аналізу даних.

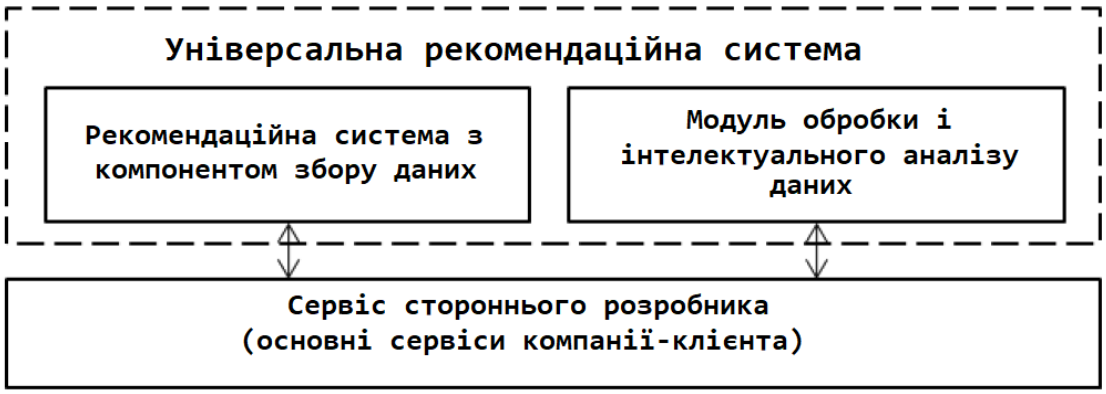


Рисунок 1.7 – Підсистеми універсальної рекомендаційної системи

Ці дві підсистеми можуть працювати незалежно одна від одної. Однак доповнення рекомендаційної системи засобами аналізу даних суттєво підвищує якість її роботи.

Під час проектування архітектури системи, що взаємодіє з сервісом та

Користувачами часто обирають обрано мікросервісний підхід. Мікросервісний підхід – це особливий спосіб розробки програмних систем, який знаходить широке застосування останніми роками. Мікросервісна архітектура передбачає розробку програмних додатків як набору незалежних невеликих сервісів, що розгортаються, кожен з яких запускає унікальний процес і забезпечує комунікацію за допомогою чітко визначеного, легкого механізму. Завдяки гнучкості та масштабованості ці архітектурні рішення є найбільш підходящими, коли потрібно включити підтримку для цілого ряду платформ та пристроїв різних типів.

Загальна архітектура рекомендаційної системи на основі мікросервісів представлена рис. 1.8. Як основний кластер для зберігання та обробки даних використовується програмна платформа Hadoop – система, яка складається з безлічі різних компонентів, що у сукупності створюють єдину платформу.

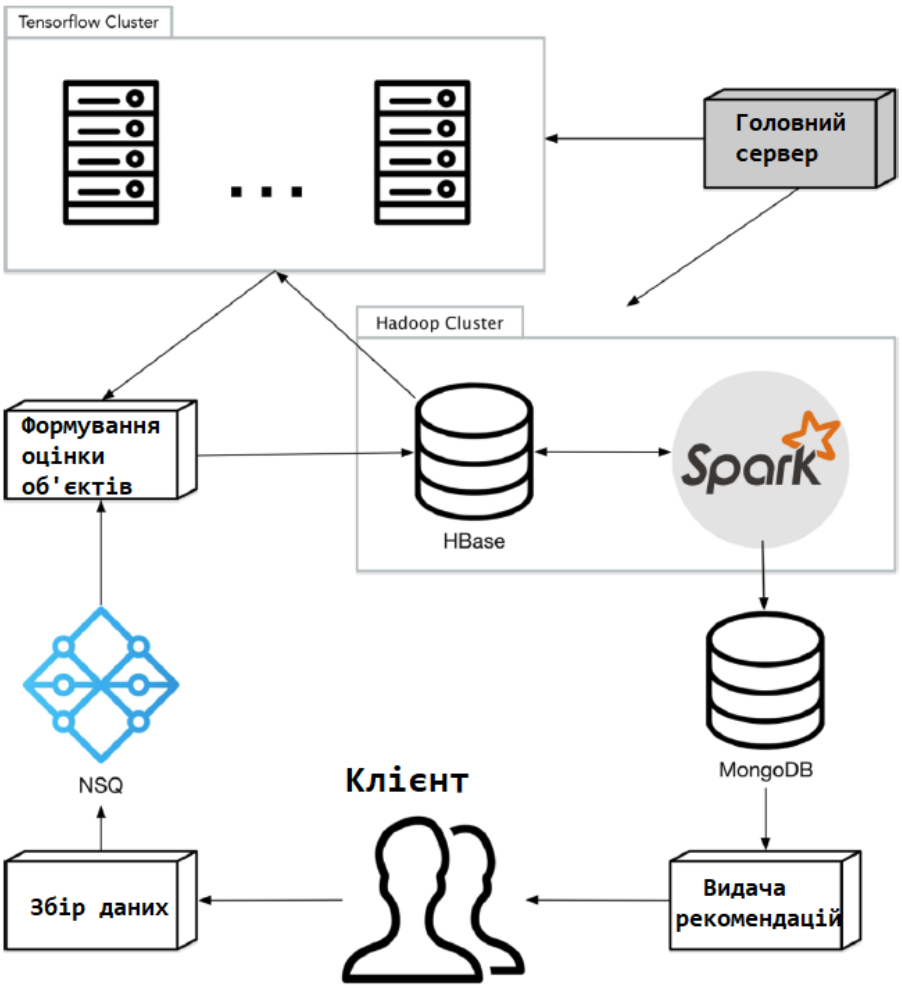


Рисунок 1.8 – Спрощена архітектура рекомендаційної системи на основі мікросервісів

Ця платформа має відкритий вихідний код і призначена для зберігання та обробки даних, які надто великі для одного конкретного пристрою або сервера. Сила Hadoop полягає в її здатності масштабуватись по тисячах товарних серверів, які не поділяють пам'ять або дисковий простір.

На кластері Hadoop як база даних використовується Apache Hbase. Дана база даних є колонковою, працює на HDFS. Вона масштабується зберігання великих обсягів розріджених даних. У схемі Big Data вона відповідає категорії сховища та є альтернативним або додатковим варіантом сховища даних.

Для обробки даних на кластері використовується Apache Spark. Spark – це середовище виконання, яке працює з файловою системою, щоб

розподіляти дані по кластеру та обробляти ці дані паралельно. Spark також приймає набір інструкцій із програми, написаної розробником. Це середовище підтримує різні мови програмування (Java, Python та Scala).

1.4. Дослідження проблем які виникають під час надання рекомендацій

Підходи до спільної фільтрації часто страждають від трьох проблем: холодний старт, рідкість та масштабованість.

Холодний старт – проблема нового користувача або елемента з недостатньою кількістю даних для точних рекомендацій.

Масштабованість – мільйони користувачів та продуктів в системах, що надають рекомендації досить сильно навантажують її. Таким чином, для надання рекомендацій часто потрібна велика кількість обчислювальної потужності.

Розрідженість – кількість предметів, що продаються на основних сайтах електронної комерції, надзвичайно велика. Найактивніші користувачі оцінюють лише невелику підмножину загальної бази даних.

Таким чином, навіть найпопулярніші предмети мають дуже мало оцінок. Одним з найвідоміших прикладів спільної фільтрації є спільне фільтрування по предметах (люди, які купують x, також купують y), алгоритм, популяризований системою рекомендацій Amazon.

Багато соціальних мереж спочатку використовували спільну фільтрацію, щоб рекомендувати нових друзів, груп та інших соціальних зв’язків, вивчаючи подібності між користувачем та його друзями. Спільна фільтрація все ще використовується як частина гібридних систем.

Проблема холодного старту в рекомендаційних системах може значно ускладнити процес надання релевантних рекомендацій новим користувачам або новим елементам каталогу. Основна причина цієї проблеми полягає в

тому, що система не має достатньої кількості інформації про користувача або елемент, щоб зробити точні рекомендації. Це може призвести до того, що нові користувачі не отримують релевантних рекомендацій, що може вплинути на їх задоволеність використанням системи та зменшити їх бажання повернутися на платформу. Нові елементи також можуть не отримувати достатньої уваги в системі, що може призвести до зниження попиту на них та їх помітність серед користувачів.

Для подолання проблеми холодного старту в рекомендаційних системах, можна використовувати різні підходи. Один з таких підходів – це використання колаборативного фільтрування на основі контенту. Цей підхід базується на тому, що якщо користувач зробив вибір певного елемента, то система може запропонувати інші елементи зі схожим контентом, що сподівається, що буде релевантними для користувача. Наприклад, якщо користувач обрав певну гру, то система може запропонувати інші ігри зі схожою тематикою або жанром.

Рекомендаційні системи є важливим інструментом для покращення користувальницького досвіду в багатьох сферах, включаючи електронну комерцію, соціальні мережі, медіа, культуру та багато іншого. Одним з головних завдань рекомендаційних систем є знаходження найбільш відповідних або релевантних елементів для користувача, основуючись на їхніх діях, інтересах, поведінці та інших факторах.

Побудова рекомендацій включає в себе такі етапи, як збір та аналіз даних, побудову моделей рекомендацій, відбір та ранжування рекомендацій (рисунок 1.9). Залежно від характеру завдання та типу даних, можуть використовуватись різноманітні методи та алгоритми, такі як колаборативна фільтрація, контент-базована фільтрація, глибинне навчання та інші.

У процесі побудови рекомендацій необхідно вирішувати багато важливих завдань, таких як підвищення точності та релевантності рекомендацій, зменшення ризику спаму та підробок, забезпечення

ефективності та масштабованості системи, захист персональної інформації користувачів тощо.



Рисунок 1.9 - Процес побудови рекомендацій

Одним з важливих аспектів різноманітності завдань побудови рекомендацій є збір та обробка даних. Для досягнення якісних результатів необхідно забезпечити якість даних, а також уникнути недостатньої кількості даних для побудови моделі.

Для забезпечення якісної роботи рекомендаційної системи, необхідно звернути особливу увагу на збір та обробку даних. Недостатня кількість даних або неправильний їх формат можуть значно погіршити якість рекомендацій.

Збір даних може бути здійснений різними способами, включаючи збір даних з веб-сторінок, використання даних соціальних мереж, даних з транзакцій та інші. Після збору даних необхідно їх обробити, щоб використовувати їх для побудови моделі рекомендацій.

Одним з ключових завдань обробки даних є очистка та препроцесинг. Це означає видалення непотрібної або дубльованої інформації, нормалізацію та векторизацію даних, щоб зробити їх придатними для подальшої обробки.

Для досягнення якісних результатів та забезпечення точності рекомендаційної системи необхідно використовувати різноманітні методи машинного навчання та статистичного аналізу даних. До таких методів можуть відноситися алгоритми колаборативної фільтрації, контент-базовані методи та гібридні рекомендаційні системи.

Узагальнюючи, рекомендаційні системи є важливим інструментом в сучасному світі, де інформаційна надмірність стає все більшою проблемою. Для забезпечення якісної роботи рекомендаційної системи, необхідно звернути особливу увагу на збір та обробку даних, використання різноманітних методів машинного навчання та статистичного аналізу даних, а також використання різноманітних метрик оцінки як точності, так і ефективності рекомендаційної системи.

1.5. Аналіз проблеми холодного старту в системах надання рекомендацій

Холодний старт – це ситуація, коли рекомендаційна система не може рекомендувати продукти користувачеві в зв'язку з тим, що недостатньо інформації про користувача та його взаємодії з системою (рисунок 1.10).



Рисунок 1.10 – Характеристика холодного старту

Ми можемо мати три різні підпроблеми в області проблем холодного запуску. Перша проблема виникає, коли ми не маємо жодної інформації про нового користувача, який входить у систему. це відбувається лише тоді, коли користувач приєднується до системи вперше. Прикладом може бути, якщо ви приєднуєтеся до Amazon або Flipkart вперше, коли виникає така проблема. Ця проблема відома як проблема холодного запуску нового користувача. Друга проблема виникає, коли ми вводимо в систему новий елемент. цей предмет дуже оригінальний у своєму роді.

Система спільної фільтрації, якій потрібна матриця оцінки елементів користувача, щоб надати рекомендацію, не може запуститися, і проблема відома як проблема холодного запуску. Третя проблема виникла, коли ми вперше запустили систему. У цьому випадку ми не маємо ні інформації про користувача, ні інформації про предмети. Іншими словами, ми не маємо жодної матриці оцінок за елементами користувача, яка потрібна для належної роботи спільних систем рекомендацій. Ця проблема відома як проблема системи холодного запуску. У проблемах холодного запуску можна застосувати відомі рішення на основі вмісту для вирішення проблеми. Інші рішення використовують комбінацію різних методів машинного навчання.

Якщо говорити більш конкретно про основні проблеми, пов'язані з холодним стартом, включають:

- Недостатньо інформації про користувача: у випадку нового користувача, система не має достатньо даних про його інтереси, поведінку, відгуки тощо. Це зменшує точність рекомендацій та знижує задоволеність користувача.

- Обмежені можливості фільтрації: у випадку нового продукту, система не може використовувати інформацію про попередні взаємодії з користувачами, оскільки немає жодного відгуку або дій користувача щодо цього продукту.

- Проблема розвитку: у випадку малої кількості користувачів або продуктів, система може мати проблеми зі зростанням та розвитком, оскільки не вистачає даних для покращення алгоритмів рекомендацій.

Холодний старт є надзвичайно важливою проблемою для рекомендаційних систем, оскільки він може позначитися на якості рекомендацій та задоволеності користувача. Однак, є кілька можливих шляхів для розв'язання проблем холодного старту.

По-перше, можна використовувати різноманітні методи аналізу та обробки даних, щоб зібрати більше інформації про користувачів. Наприклад, можна використовувати дані з соціальних мереж або попередніх взаємодій користувача зі схожими продуктами, щоб дізнатися про їх інтереси.

По-друге, можна використовувати методи активного навчання, які дозволяють зібрати більше даних про користувачів. Наприклад, можна використовувати експерименти, де користувачі запрошуються відповісти на запитання про свої вподобання та інтереси.

По-третє, можна використовувати генеративні моделі, які дозволяють генерувати синтетичні дані про користувачів та продукти на основі вже наявних даних. Це допомагає збільшити обсяг даних та забезпечити більш точні рекомендації для нових користувачів або продуктів.

По-четверте, можна використовувати гібридні моделі, які комбінують кілька методів рекомендацій в одній системі. Наприклад, можна використовувати колаборативний та контентний підхід для рекомендацій.

Загалом, холодний старт є важкою проблемою для рекомендаційних систем, але застосування вищезгаданих методів може допомогти зменшити його вплив на якість рекомендацій та задоволеність користувачів. Важливо розуміти, що кожен підхід має свої переваги та недоліки, і вибір підходу для конкретної рекомендаційної системи залежить від різних чинників, таких як тип продукту, масштаб системи та доступність даних. Однак, якщо використовувати комбінацію декількох методів та постійно вдосконалювати

рекомендаційну систему, можна досягти більшої точності та ефективності в роботі з користувачами.

1.6. Аналіз методів побудови рекомендацій в ситуації холодного старту

Холодний старт є однією з найважливіших проблем в області рекомендаційних систем. Це стосується ситуації, коли система звертається до нового користувача, для якого немає жодної інформації про його вподобання та інтереси, або до нового продукту, про який немає достатньої кількості даних. У таких випадках стандартні алгоритми рекомендаційних систем не можуть впоратися з поставленою задачею, оскільки не мають достатньої кількості даних для побудови зв'язків між користувачами та продуктами.

Для розв'язання проблеми холодного старту, існує кілька методів побудови рекомендацій, які дозволяють зменшити вплив цієї проблеми на ефективність рекомендаційних систем.



Рисунок 1.11 - Методи побудови рекомендацій в ситуації холодного старту

Першим методом є використання контентних факторів. Контентні фактори включають інформацію про характеристики продуктів, такі як автор, рік видання, жанр, опис тощо, а також профіль користувача, який містить інформацію про його вік, стать, місцезнаходження, звички та інтереси. Використання контентних факторів дозволяє знайти схожі продукти за їхніми характеристиками та зв'язати їх з новим користувачем на основі його профілю. Цей метод дозволяє створювати рекомендації для нових користувачів та продуктів, що не мають достатньої кількості історичних даних.

Другим методом є використання гібридних рекомендаційних систем, які комбінують різні методи рекомендацій в одній системі. Наприклад, можна використовувати контентні фактори разом з колаборативним фільтруванням, яке базується на історичних даних про взаємодії користувачів з продуктами. Гібридні системи дозволяють знизити вплив проблеми холодного старту, коли в системі недостатньо історичних даних для рекомендацій.

Третім методом є використання ранжування рекомендацій. Ранжування рекомендацій дозволяє присвоювати вагу різним факторам, які впливають на рівень релевантності рекомендації. Наприклад, якщо два продукти мають схожі контентні характеристики, то рекомендація може бути ранжована в залежності від рейтингу, який користувачі присвоюють цим продуктам.

Четвертим методом є використання аналізу зв'язків між продуктами та користувачами. Цей метод використовує аналіз графа взаємодій міжкористувачами та продуктами, щоб знайти спільні зв'язки та встановити взаємозв'язки між ними. Наприклад, якщо користувачі, які мають подібний профіль, взаємодіють з одними й тими ж продуктами, то можна припустити, що ці продукти є взаємозамінними. Цей метод дозволяє знаходити неочевидні зв'язки та забезпечує більш точні рекомендації.

П'ятим методом є використання інтерактивних рекомендаційних систем, які дозволяють користувачам активно взаємодіяти з системою та надавати зворотний зв'язок.

Цей метод передбачає, що рекомендації не створюються тільки на основі вхідних даних, але й на основі дій користувачів, які взаємодіють з системою. Інтерактивні рекомендаційні системи забезпечують користувачам можливість вибирати, чи вони хочуть отримувати рекомендації, а також надавати системі зворотний зв'язок про рекомендації, які їм були надані.

Один з простих прикладів інтерактивних рекомендаційних систем – це система, що пропонує підписатися на різні категорії новинних розсилок, коли користувач реєструється на новинний сайт. Користувач може обрати, які новинні категорії він хоче отримувати і як часто він хоче отримувати ці розсилки.

Інтерактивні рекомендаційні системи можуть бути використані і в інших галузях, таких як онлайн-магазини, де користувачі можуть брати участь у опитуваннях, розглядати товари та надавати зворотний зв'язок про продукти, які їм сподобалися, або не сподобалися.

У підсумку, існує багато різних методів створення рекомендацій, і краще використовувати їх у поєднанні. Завдяки поєднанню різних методів, можна створити більш точну та персоналізовану систему рекомендацій, яка допоможе користувачам знайти те, що їм дійсно цікаво і корисно.

1.7. Основні вимоги до розроблюваної системи надання рекомендацій

Для досягнення поставленої мети необхідно вирішити наступний перелік задач:

– реєстрація та авторизація користувачів;

– забезпечення взаємодії користувачів з відеоіграми: пошук відеоігор, перегляд детальної інформації, додавання відеоігор у власну бібліотеку, створення на її основі власних колекцій, оцінка відеоігор;

– створення рекомендацій на основі існуючих оцінок;

– надання можливості користувачам обговорювати відеоігри між собою;

– можливість додавання нових предметних областей.

Отже, виконано аналіз предметної області, де виділено наступні сутності, котрі є невід’ємною частиною будь-якого сервісу, що надає користувачам інформацію про ігри. Проведено аналіз відомих алгоритмічних рішень. З поміж різновидів рекомендаційних моделей обрана модель основана на схожості користувачів. Сформульована постановка завдання для розробки рекомендаційної системи.

2. АНАЛІЗ ВИМОГ ТА МОДЕЛЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ НАДАННЯ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ

2.1. Аналіз відомих програмних продуктів надання рекомендацій

Існує безліч із системами рекомендацій. Наведемо деякі з них, які мають значний вплив і популярність.

Netflix використовує систему рекомендацій, щоб персоналізувати рекомендації щодо фільмів і серіалів для своїх користувачів. Він заснований на спільній фільтрації, а також використовує інформацію про історію веб-перегляду, оцінки користувачів та інші фактори.

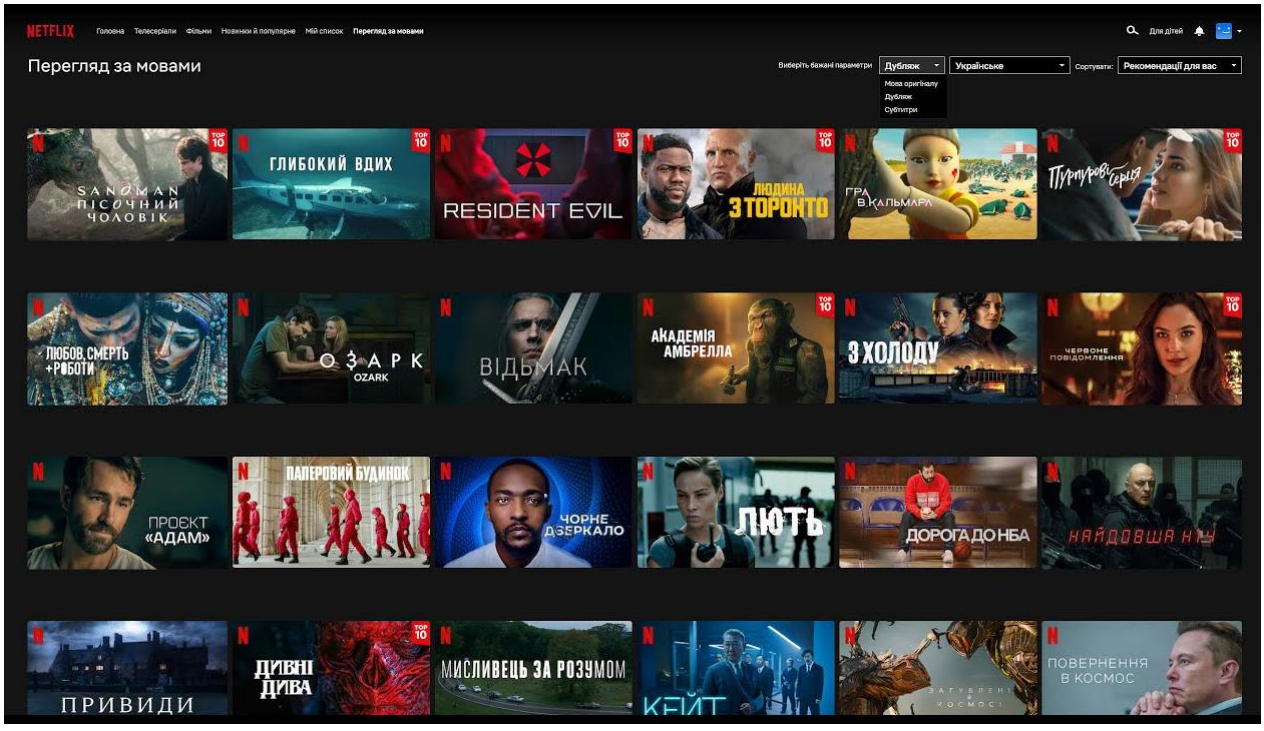


Рисунок 2.1 – Приклад рекомендацій на платформі Netflix

Недоліки. Одним із недоліків Netflix є те, що рекомендації можуть бути обмежені сферою інтересів користувача. Крім того, він може страждати від проблеми «бульбашки фільтрів»(“filter bubble”), коли користувачам

рекомендовано лише схожий вміст, що може обмежити їхній доступ до нових і різноманітних фільмів і телешоу.

Amazon використовує систему рекомендацій, щоб персоналізувати рекомендації продуктів для своїх користувачів. Він враховує попередні покупки, оцінки, перегляди товарів, а також використовує спільну фільтрацію та аналіз звичок покупців.

Amazon.com використовує рекомендації спільної фільтрації по елементах на більшості сторінок свого веб-сайту та в кампаніях електронної пошти. За даними McKinsey, 35% покупок Amazon здійснюються завдяки системам рекомендацій. Кілька прикладів, коли Amazon використовує системи рекомендацій.

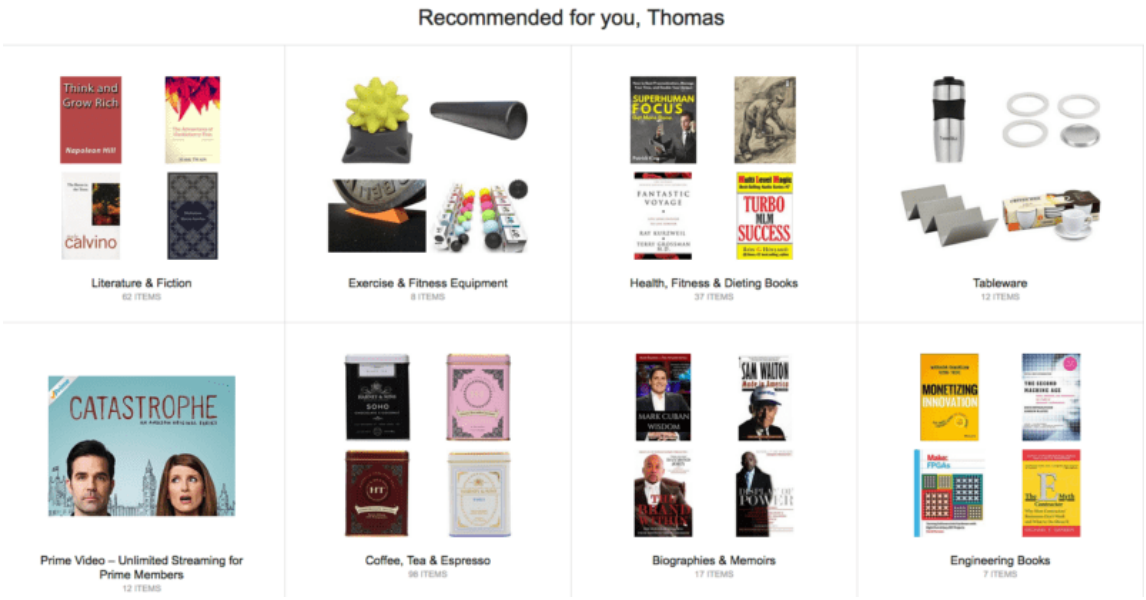


Рисунок 2.2 - Приклад рекомендаційної системи Amazon.com

Одним із недоліків системи Amazon є її висока залежність від вхідних даних клієнтів. Якщо покупець має обмежену кількість попередніх покупок або оцінок, система може недостатньо точно рекомендувати товари.

Як і будь-який інший канал соціальних мереж, LinkedIn також використовує рекомендації типу «Ви також можете знати» або «Вам також може сподобатися».

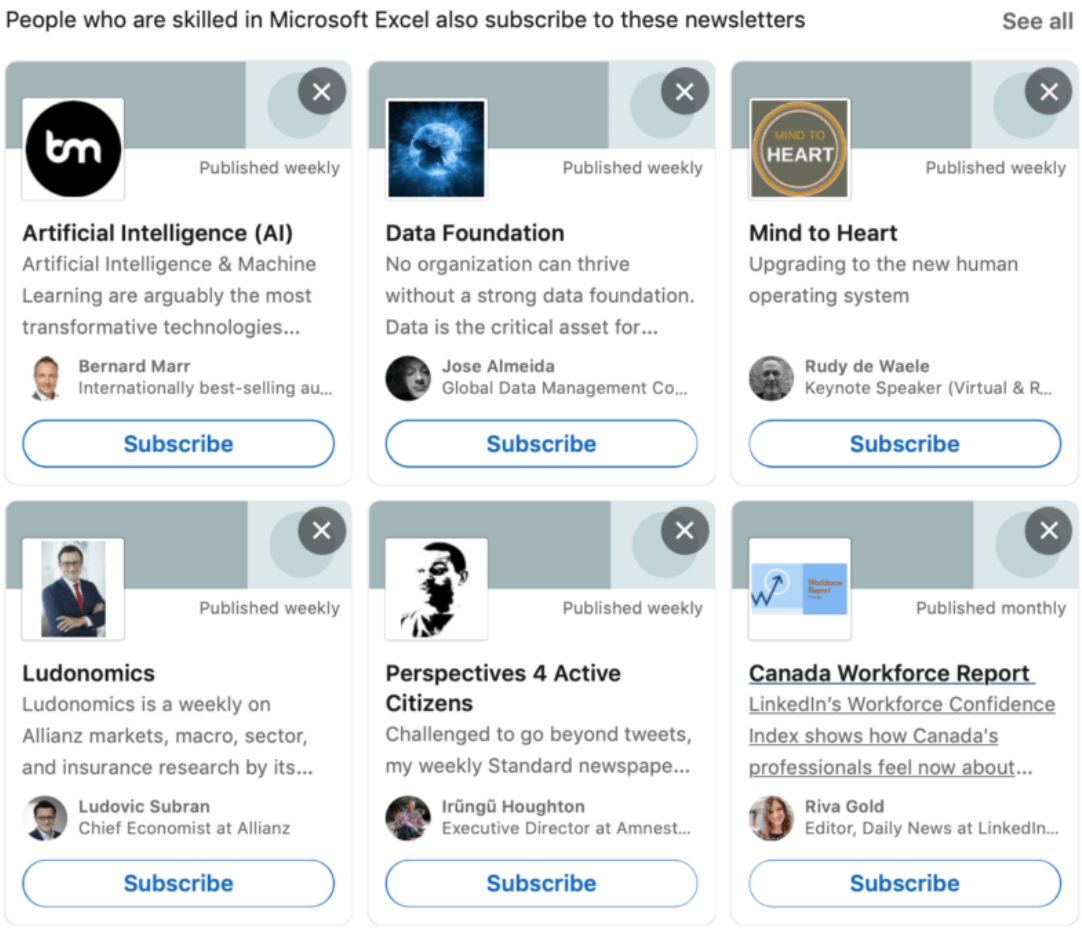


Рисунок 2.3 - Приклад рекомендаційної системи Linkedin

Spotify використовує систему рекомендацій, щоб персоналізувати музичні рекомендації для своїх користувачів. Він аналізує музичні вподобання, історію прослуховування, створює списки відтворення та рекомендує нову музику на основі цих даних.

Щотижня Spotify створює новий налаштований список відтворення для кожного підписника під назвою «Discover Weekly». Це персоналізований список із 30 пісень на основі унікальних музичних смаків користувачів.

Придбання Echo Nest, музичного інтелектуального стартапу та аналітики даних, дозволило їм створити систему музичних рекомендацій, яка використовує три різні типи моделі рекомендацій:

• Спільна фільтрація: фільтрація пісень шляхом порівняння даних про прослуховування користувачами з історією прослуховування інших користувачів.

• Обробка природної мови: пошук в Інтернеті інформації про певних виконавців і пісні. Потім кожному виконавцю чи пісні призначається динамічний список найпопулярніших термінів, який змінюється щодня та зважується за релевантністю. Потім механізм визначає, чи схожі два музичні твори або виконавці.

• Аналіз аудіофайлу: алгоритм аналізує характеристики кожного окремого аудіофайлу, включаючи темп, гучність, тональність і тактовий розмір, і дає відповідні рекомендації.

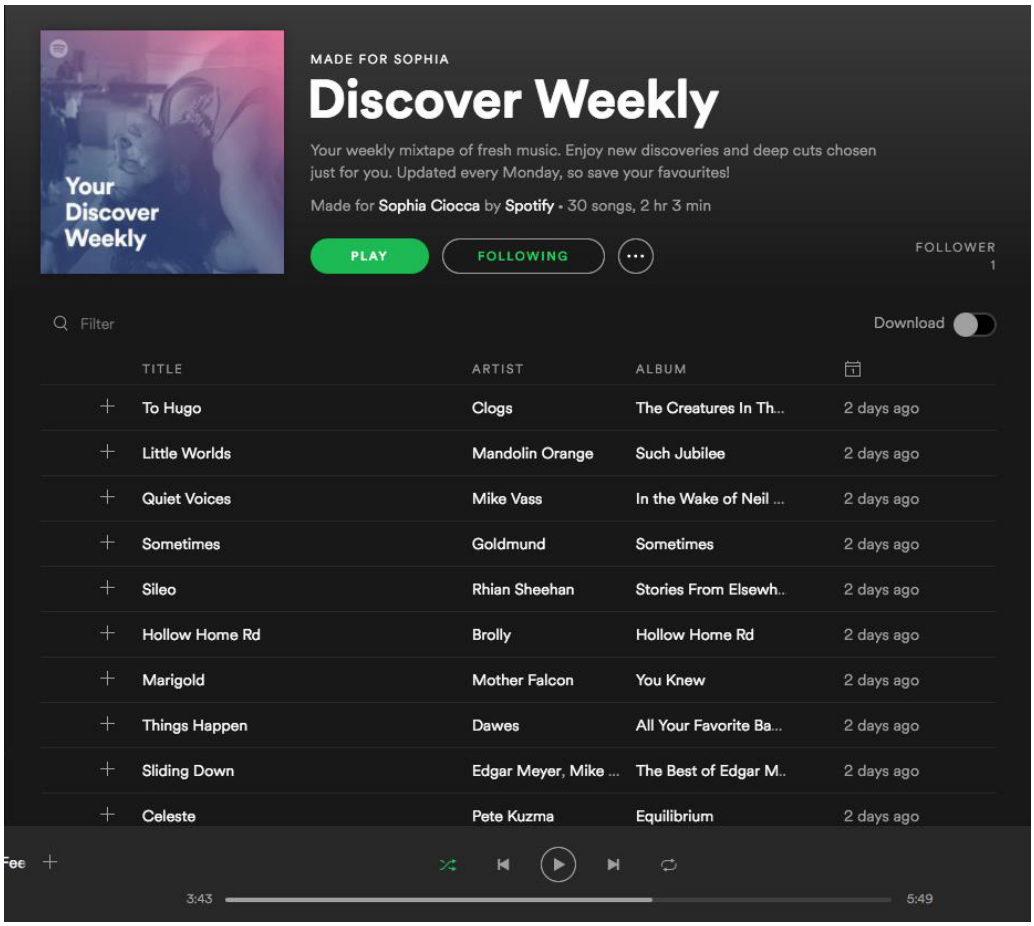


Рисунок 2.4 - Приклад рекомендаційної системи Spotify

Недолік Spotify - також проблема "бульбашки фільтрів". Користувачам часто рекомендується лише подібна музика, що може обмежити їх зустріч із новими жанрами та виконавцями.

Moregameslike.com – веб-сайт із системою рекомендацій відеоігор на основі схожості. Сервіс надає користувачам можливість знайти нові ігри, що відповідають їх вподобанням та побачити стислу інформацію про кожну. Для цього в пошукове поле необхідно ввести назву гри, яка сподобалася, або улюблені жанри. Після чого користувач отримує в якості рекомендацій багато ігор, що мають подібні елементи геймплею.

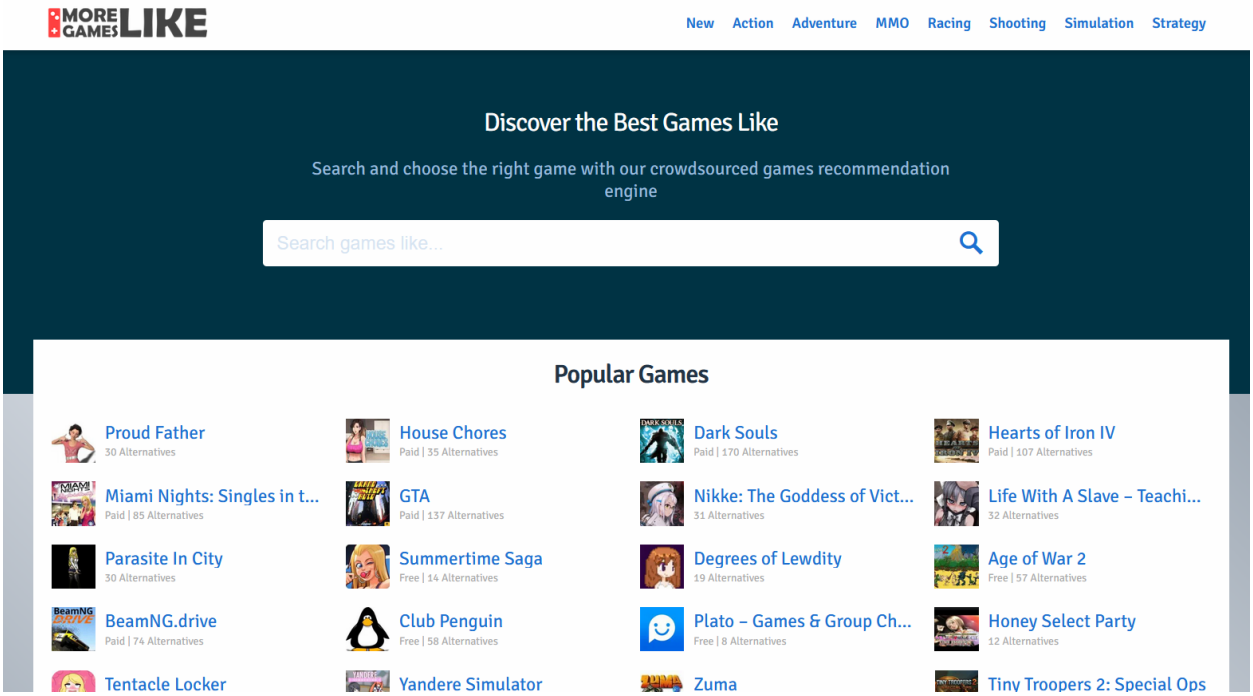


Рисунок 2.5 – Платформа надання рекомендацій Moregameslike.com

2.2. Аналіз вимог до розробки. Розробка діаграм використання

Система повинна передбачити дві явних ролі: користувач, адміністратор; і одну неявну – відвідувач.

– Користувач може повноцінно взаємодіяти з системою: працювати з власною бібліотекою, залишати коментарі, виставляти оцінки іграм, отримувати рекомендації.

– Адміністратор має доступ до всього функціоналу, передбаченого системою: заповнення бази даних, редагування та видалення даних, що стосуються жанрів та ігор.

– Відвідувач має право тільки на перегляд наявної інформації та пошук ігор в бібліотеці.

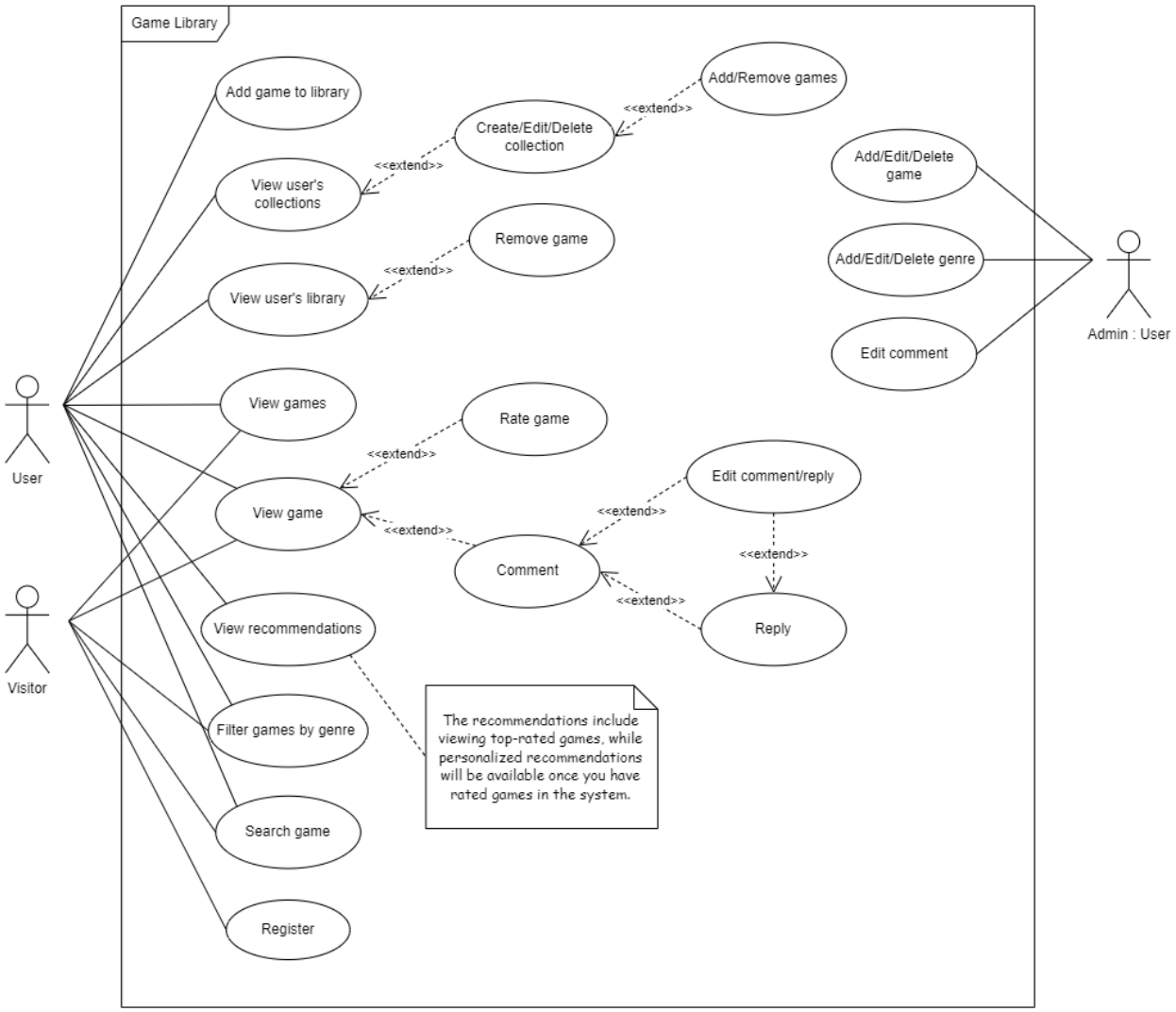


Рисунок 2.1 – Діаграма варіантів використання системи надання рекомендацій

Діаграма варіантів використання програмного забезпечення наведено на рисунку 2.6, опис основних елементів наведено нижче:

Use case #1. Реєстрація нового користувача.

Дія. Не зареєстрований користувач потрапляє на сторінку реєстрації. Вводить відповідні дані: електронна пошта користувача, нікнейм, пароль, його повтор для підтвердження. Після заповнення даних користувач натискає кнопку реєстрації і перенаправляється на головну сторінку.

Результат. Якщо якесь конкретне поле введено некоректно, то зверху з'являється інформаційний надпис.

Use case #2. Додавання, редагування, видалення ігор.

Дія. Адміністратор через панель керування відкриває сторінку для додавання відеогри. В поля для вводу вводяться відповідні дані: назва гри, опис, зображення, дата релізу, дозволений віковий рейтинг, трейлер, розробник, жанри. Далі адміністратор підтверджує дію. Після цього він перенаправляється на головну сторінку.

В межах кожної картки з грою для адміністратора відображається кнопка, яка містить випадаючий список, що складається з дії редагування та дії видалення. Якщо вибрати редагування, то відкривається форма, ідентична формі додавання, де необхідно виправити дані та підтвердити дію. Якщо вибрати видалення, то гра видаляється з системи.

Результат. У випадку невідповідності типу даних поля та типу введених даних, виникає повідомлення з попередженням.

Use case #3. Отримання детальної інформації про гру.

Дія. Користувач переходить на головну сторінку, вибирає відеогру та натискає на її картку. Після цього він перенаправляється на сторінку детальної інформації про гру.

Результат. Перехід на сторінку детальної інформації про гру

Use case #4. Перегляд, додавання, редагування, видалення жанрів.

Дія. Адміністратор через панель керування відкриває сторінку для додавання жанрів. В поля для входу вводяться відповідні дані: назва жанру,

опис. Після заповнення даних адміністратор натискає кнопку підтвердження. Після цього він перенаправляється на сторінку з переліком жанрів, де поруч з кожним відображається кнопка редагування та видалення. Вибрав редагування, відкривається форма, ідентична формі додавання, де необхідно виправити дані та підтвердити дію. Вибрав видалення, виникає вікно з підтвердженням дії. Після підтвердження жанр видаляється з системи.

Результат. При додаванні нового жанру, у випадку пропуску заповнення полів, виникає повідомлення з попередженням. Якщо все успішно то відбувається оновлення даних жанрів.

Use case #5. Створити, видалити, редагувати колекцію ігор.

Дія. Користувач переходить на сторінку керування колекціями. Щоб створити нову колекцію, користувач натискає відповідну кнопку. Після цього він перенаправляється на сторінку створення нової колекції, де вводить у відповідне поле назву колекції. Підтвердивши дію, користувач перенаправляється на сторінку керування колекціями.

Щоб видалити колекцію, користувач на сторінці керування колекціями натискає кнопку видалення, що належить відповідній колекції. З'являється модальне вікно, де необхідно підтвердити дію. Після підтвердження колекція зникає з переліку.

Щоб редагувати колекцію, користувач на сторінці керування колекціями натискає кнопку редагування, що належить відповідній колекції. Після цього він перенаправляється на сторінку редагування колекції, де змінює назву колекції. Підтвердивши дію, користувач перенаправляється на сторінку керування колекціями.

Щоб додати в колекцію одну або декілька ігор, користувач натискає відповідну кнопку, після чого перенаправляється на сторінку, де потрібно вибрати ігри для додавання. Після підтвердження дії користувач перенаправляється на сторінку змісту колекції.

Щоб вилучити гру з колекції, користувач на сторінці змісту колекції натискає відповідну кнопку на картці гри.

Результат. Оновлення даних колекцій

Use case #6. Здійснення пошуку гри.

Дія. З головної сторінки користувач вводить у поле для пошуку повну або часткову назву відеогри. Після підтвердження пошуку, користувач отримує результати. Для здійснення навігації натискаються кнопки “Next” і “Previous”. При досягненні останньої/першої сторінки кнопка “Next”/“Previous” стає неактивною.

Результат. Відображення результатів, співпадаючих із запитом пошуку. При здійснені навігації змінюється сторінка результатів

Use case #7. Додавання коментаря до гри.

Дія. Користувач переходить на сторінку детальної інформації про гру. У відповідне поле вводить текст та підтверджує дію.

Результат. Додається новий коментар до гри

Use case #8. Надання оцінки певній грі, покращення надання рекомендацій.

Дія. Користувач переходить на сторінку детальної інформації про гру. У відповідне поле вводиться оцінка від 1 до 5. Далі користувач підтверджує дію. Якщо надане значення не відповідає вимогам системи, до з’являється попереджувальне повідомлення.

Результат. Вплив на середній рейтинг гри, вплив на формування персоналізованих рекомендацій.

Use case #9. Перегляд рекомендованих ігор

Дія. Користувач відкриває головну сторінку, де зверху відображається перелік рекомендованих ігор.

На рисунку 2.7. представлено модель вимог у загальному вигляді.

#	Name	Priority	Risk
FR-1	Реєстрація користувача	High	High
FR-2	Авторизація і автентиф корист.	High	High
FR-3	Дод., ред., вид. гри	High	High
FR-4	Перегляд відеоігор	High	Low
FR-5	Перегляд детальної інформації про гру	High	Low
FR-6	Дод., ред., вид. жанру	High	Medium
FR-7	Перегляд даних про жанри	Medium	Low
FR-8	Створ., ред., вид. корист. колекції ігор	Low	Low
FR-9	Перегляд корист. колекцій	Low	Low
FR-10	Дод., вилуч. ігор із колекції	Low	Low
FR-11	Дод. та вилуч. гри з бібліотеки	Low	Low
FR-12	Перегляд вмісту корист. бібліотеки	Low	Low
FR-13	Пошук відеоігор	High	Medium
FR-14	Фільтрація ігор за жанром	Medium	Low
FR-15	Перегляд коментарів до гри	Medium	Low
FR-16	Дод., ред. коментарів до гри	Medium	Medium
FR-17	Оцінювання відеоігор	High	Medium
FR-18	Отримання рекомендованих ігор	High	Low
FR-19	Пагінація	Medium	Low

Рисунок 2.7 – Модель вимог

2.3. Опис бізнес-процесів розроблюваної системи

Для опису бізнес процесів програмного забезпечення може використовуватися BPMN модель та опис послідовності дій процесу.

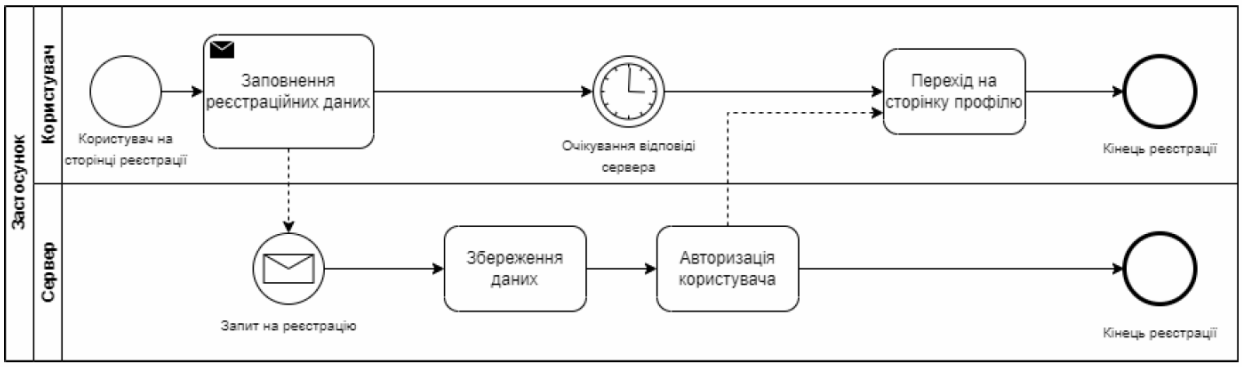


Рисунок 2.8 – Приклад BPMN схеми для процесу реєстрації користувача

Опис послідовності реєстрації користувача:

− користувач потрапляє на форму реєстрації;

− користувач здійснює заповнення необхідних полів;

− у випадку не вірності введених даних згідно шаблону то на клієнтській стороні, біля відповідних полів зазначається помилка;

− користувач натискає кнопку реєстрації;

− дані облікового запису користувача зберігаються у БД;

− відбувається автоматична авторизація користувача та перехід на сторінку профілю.

Опис послідовності авторизації користувача:

− користувач переходить на сторінку авторизації;

− користувач заповнює поля авторизації;

− якщо дані що були внесені в поля порушують шаблон, то користувачу зазначається помилка;

− користувач натискає кнопку авторизації;

− якщо введені дані не відповідають значенням з БД, користувач залишається на сторінці авторизації для повторного введення даних;

− після успішної перевірки даних, користувач авторизований у системі та переходить на сторінку профілю.

Опис послідовності пошуку елементів за ключовим словом (рис. 2.9):

− користувач переходить на одну із сторінок зі списками елементів;

− користувач вводить ключове слово у відповідне поле;

− користувач натискає кнопку пошуку;

− система виконує пошук у базі даних за ключовим словом у назві та опису елементів, після чого складає список знайдених елементів в системі рекомендацій;

− список елементів відображається користувачу в залежності від номера сторінки.

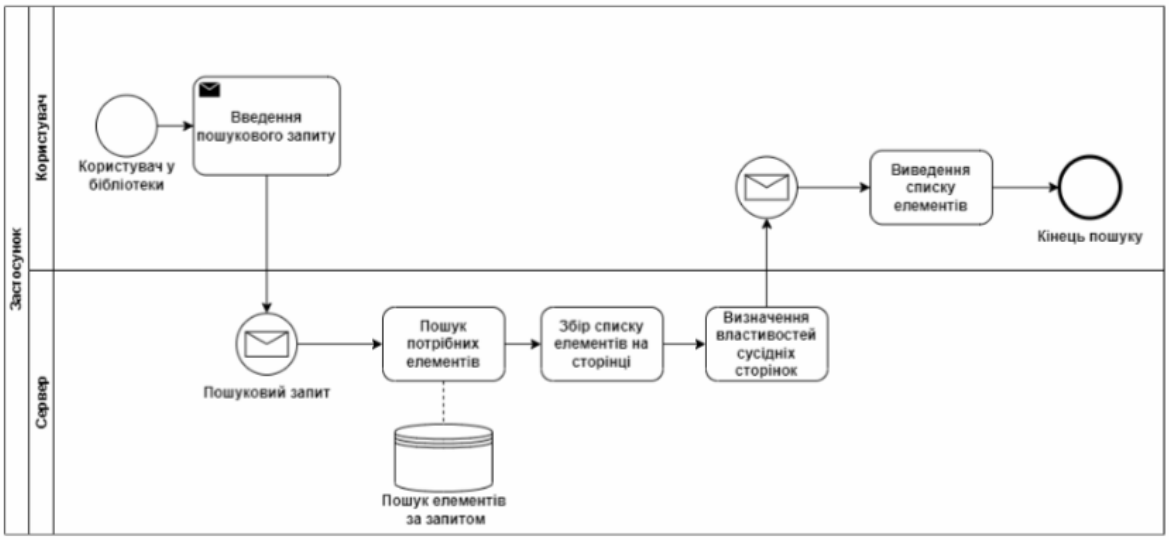


Рисунок 2.9 – Приклад BPMN схеми для пошуку елементів за ключовим словом

Опис процесу отримання персональних рекомендацій (рисунок 2.10):

– користувач переглядає наявні відеоігри;

– обирає гру якій хоче поставити оцінку;

– здійснює оцінювання;

– якщо даних достатньо для формування персональних рекомендацій, то після повернення на головну сторінку застосунку відобразяться рекомендовані відеоігри, інакше потрібно здійснити оцінювання більшої кількості відеоігор.

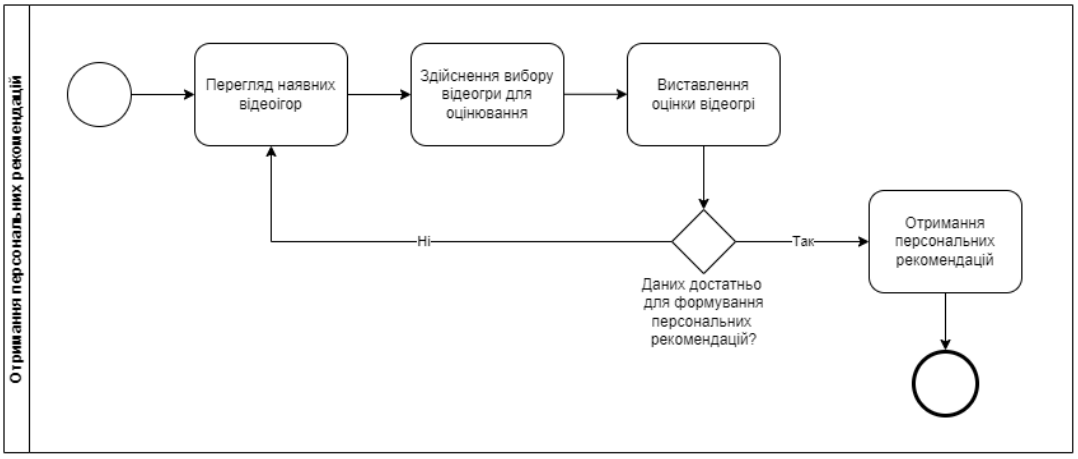


Рисунок 2.10 – Схема бізнес-процесу отримання персональних рекомендацій

2.4. Розробка структури проекту

Конструювання програмного забезпечення (Software Construction) - це процес реалізації програмного продукту на основі вимог до нього. Це один з ключових етапів життєвого циклу розробки програмного забезпечення, який передує тестуванню та впровадженню програми.

Під час конструювання програмного забезпечення програмісти перетворюють вимоги до програми, що були сформульовані на етапі аналізу та проектування, у функціональний код.

У межах конструювання програмного забезпечення багатошарова архітектура була реалізована за допомогою семи підпроєктів, що знаходяться в одному рішенні.

Всього рішення містить у собі 8 проектів (рисунок 2.11), включаючи 3 проєкти для шару бізнес-логіки, 3 проєкти для шару доступу до даних, 1 проєкт для рівня презентації та 1 проєкт, що містить модульні тести.

Опис проектів:

– DAL.Entities – містить сутності які відповідають таблицям в базі даних;

– DAL.Infrastructure – реалізація поведінки роботи із СКБД за допомогою ORM Entity Framework Core та підходу Code First;

– DAL.Interfaces – абстракції для взаємодії з шаром даних;

– BLL.DTO – моделі шару бізнес-логіки, що використовуються для передачі даних між компонентами системи;

– BLL.Core – реалізація поведінки системи, що включає бізнес-логіку, виключення та інші компоненти, які забезпечують основну функціональність додатку.

– BLL.Interfaces – абстракції для взаємодії з бізнес-логікою/

– UnitTests – модульні тести.

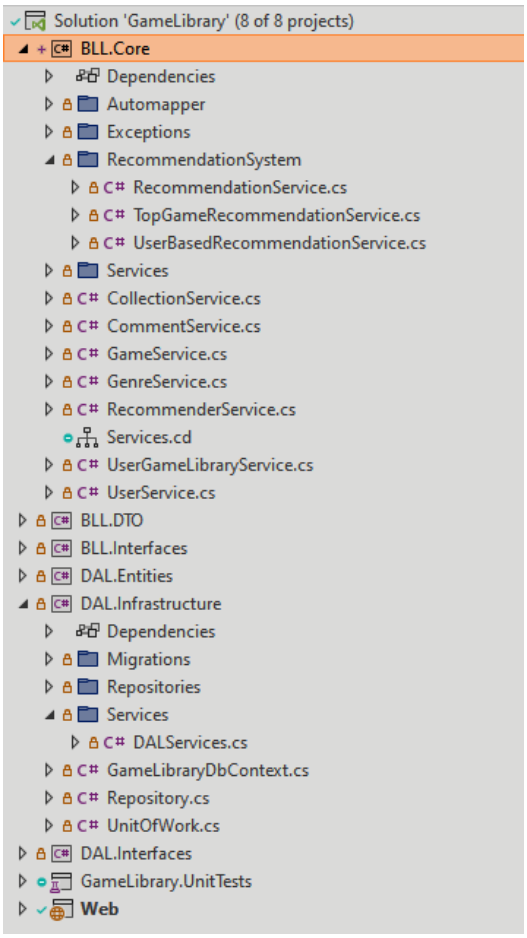


Рисунок 2.11 – Структура проекту

– Web – ASP.NET Core застосунок, що містить контролери та представлення. Виступає в ролі шару презентації, який забезпечує взаємодію з користувачем, обробку запитів та відображення даних.

2.5. Розробка діаграми сутностей шару даних та їх опис

Шар доступу до даних визначає усі необхідні класи та інтерфейси для взаємодії з базою даних. Шари, що визначають бізнес-логіку, взаємодіють з цим шаром, що дозволяє виконувати необхідні завдання та запити до бази даних.

Для зручності розробки та підтримки додатку використовується підхід Code First, який дозволяє спочатку визначити основні класи, а потім автоматично створити таблиці бази даних за допомогою ORM (Object-

Relational Mapping). Це зменшує час розробки та полегшує підтримку додатку в майбутньому, оскільки потрібно працювати з об’єктами та моделями, а не SQL-запитами.

Зв’язки між сутностями та їх властивості зображено на діаграмі класів (рисунок 2.12).

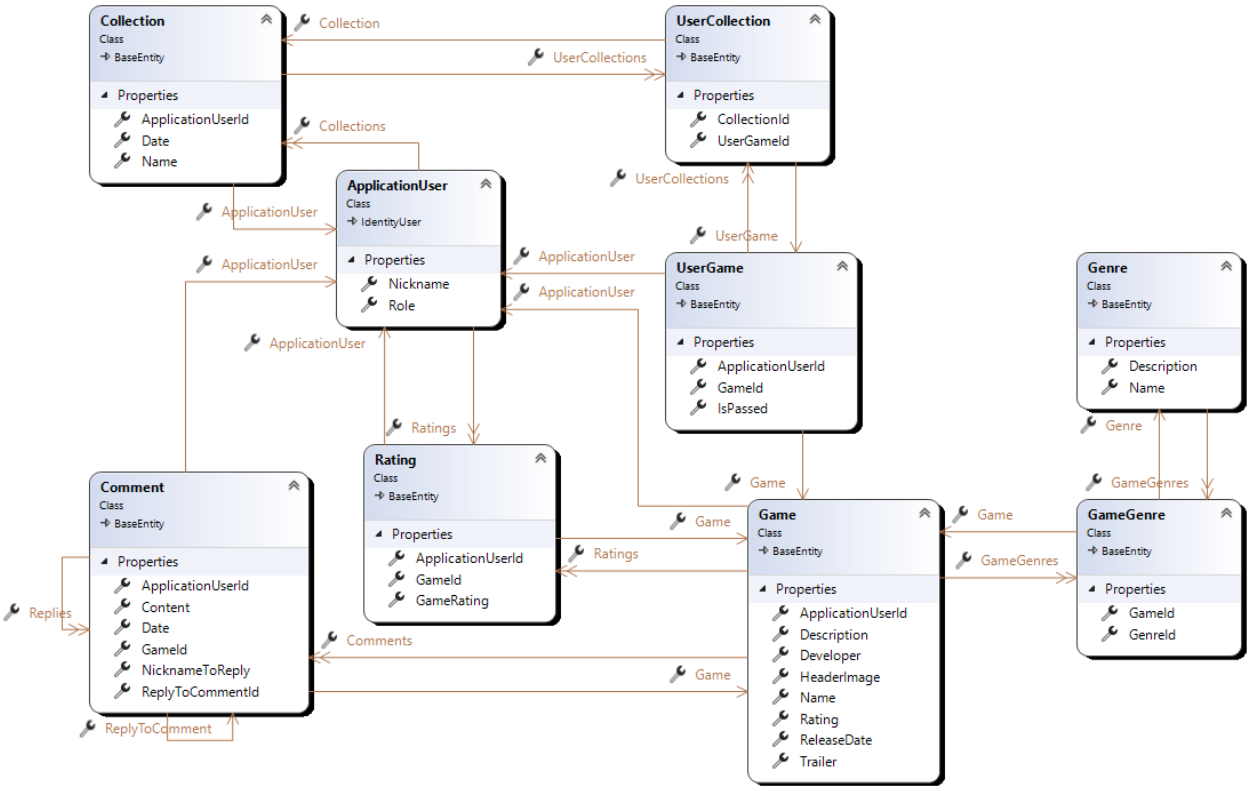


Рисунок 2.12 – Діаграма сутностей шару даних

Таким чином, для предметної області ігрової бібліотеки створюються класи-сутності, що відображають об’єкти цієї області. Детальний опис деяких сутностей наведено нижче у таблицях.

Таблиця 2.1 - Опис сутності Genre

Сутність	Назва поля	Тип даних
Genre	Name	string
	Description	string
	GameGenres	List<GameGenre>

Таблиця 2.2 – Опис сутності ApplicationUser

Сутність	Назва поля	Тип даних
ApplicationUser	Nickname	string
	Role	string
	Collections	List<Collection>
	Rating	List<Rating>

Таблиця 2.3 – Опис сутності Rating

Таблиця	Назва поля	Тип даних
Rating	ApplicationUserId	string
	ApplicationUser	ApplicationUser
	GameId	int
	Game	Game
	GameRating	int

Таблиця 2.4 – Опис сутності UserGame

Сутність	Назва поля	Тип даних
UserGame	Gameld	int
	Game	Game
	ApplicationUser	ApplicationUser
	ApplicationUserld	string
	UserCollections	List<UserCollection>

Таблиця 2.5 – Опис сутності GameGenre

Сутність	Назва поля	Тип даних
GameGenre	Gameld	int
	Game	Game
	Genreld	int
	Genre	Genre

Таблиця 2.6 – Опис сутності Game

Сутність	Назва поля	Тип даних
Game	Name	string
	Description	string
	HeaderImage	string
	ReleaseDate	DateOnly
	AgeRating	string
	Trailer	string
	Ratings	List<Rating>

Таблиця 2.7 – Опис сутності Comment

Сутність	Назва поля	Тип даних
Comment	Content	string
	Date	DateTime
	ReplyT oComment	Comment
	ReplyToCommentId	int
	Replies	ICollection<Comment>
	NicknameToReply	string
	Game	Game
	GameId	int
	ApplicationUser	ApplicationUser
	ApplicationUserId	string

Крім основних сутностей були додані допоміжні класи. Клас BaseEntity регламентує наявність поля Id у всіх класів-нащадків. Клас ApplicationUser реалізує і розширює сутність IdentityUser. Це необхідно для можливості здійснення процесів реєстрації, автентифікації та авторизації користувачів. Для взаємодії з базою даних використовується Entity Framework Core.

Для використання функціоналу EF Core необхідно створити власну реалізацію класу DbContext. Оскільки у застосунку потрібна наявність

реєстрації, автентифікації та авторизації, то використовується IdentityDbContext – клас, що спадкується від DbContext та надає властивості DbSet, необхідні для керування таблицями Identity. Екземпляр IdentityDbContext представляє собою сеанс роботи з базою даних і використовується для запитів і зберігання екземплярів сутностей.

Щоб позбутися залежності від конкретного фреймворку – Entity Framework Core, і в майбутньому мати можливість використовувати інше ORM, без внесення змін у логіку додатку, було реалізовано шаблони Repository та Unit of Work. Крім того, використання цих шаблонів має мету полегшити автоматизоване модульне тестування.

У рішенні, клас Repository реалізує інтерфейс загального репозиторію IRepository<TEntity>. Саме цей інтерфейс визначає базові операції, такі як читання, оновлення, створення, видалення. Конкретні репозиторії успадковують ці операції та реалізують додаткові – що необхідні для конкретного типу сутності.

2.6. Реалізація шару бізнес логіки

Шар бізнес логіки забезпечує функціонування програмного застосунку та відповідає за всі обчислення та рішення, пов’язані з бізнес процесами. Цей шар взаємодіє з шаром даних і шаром презентації, що представляє собою Web проєкт у рішенні. Відповідно до вибраної архітектури та підходів створені класи-моделі або DTOs (Data Transfer Objects) для передачі даних між різними частинами програми: сервісами, компонентами та методами.

Також тут знаходяться класи-моделі, які необхідні для реєстрації та входу користувача – UserRegistrationModel, UserLoginModel, відповідно. Вони містять необхідні властивості, за якими сервіс, що відповідає за роботу з користувачами за допомогою Identity, дозволяє здійснювати вхід та реєстрацію користувачів у системі.

Ще одна модель – ComparedUserModel. Вона потрібна для зберігання коефіцієнту схожості між користувачами та розрахунку рекомендацій. Класи-моделі зіставляються з класами-сутностями шару доступу до даних за допомогою класу-маперу, який перетворює вхідний об’єкт одного типу в вихідний об’єкт іншого типу. У застосунку використовується бібліотека AutoMapper, яка є реалізацією шаблону маперу.

Структура системи рекомендацій побудована відповідно до патерну «Фасад». Класи UserBasedRecommendationService та TopGames RecommendationService реалізують різні алгоритми надання рекомендацій. Подібних класів може бути більше, якщо буде вирішено в майбутньому розширити варіанти надання рекомендацій. RecommendationService виступає в ролі інтерфейсу (або фасаду) для роботи з цими сервісами. Це надає можливість представити клієнту, яким в даному випадку є контролер, лише необхідний функціонал.

Таким чином, можна надавати як агреговані види рекомендацій, так і конкретні види. В рекомендаціях основаних на схожості користувачів після знаходження k-найближчих сусідів, вибираються серед них ті ігри, в які не грав цільовий користувач, і що мають середній рейтинг $\bar{r} \geq 3{,}50$.

Під час конструювання рекомендацій на основі користувачів, довелося вирішити дві проблеми. Перша, це випадки упередженості користувачів. Порівнюючи, наприклад, вектори {5, 5, 5} та {1, 1, 1} схожість буде дорівнювати 1. Це рідкісна ситуація, але вона може впливати на коректність отриманих значень схожості. Користувачі між собою не схожі, тому що їх оцінки протилежні, а коефіцієнт схожості при цьому набуває максимального значення.

Коли користувач послідовно оцінює об’єкти в один і той самий бал, то це не сприяє розумінню ні користувача, ні об’єкта. Щоб запобігти впливу низьких оцінок було прийнято рішення не брати до розгляду користувачів, середнє значення рейтингів яких нижче порогового значення t, де t = 3.

Друга проблема, це кількість або відсутність рекомендацій. Якщо користувач не проявляє активності, виставляючи оцінки, або тільки зареєструвався в системі, рекомендацій не буде, або буде недостатньо. Тож у цьому випадку, було вирішено рекомендувати випадкові ігри з високим рейтингом, що не оцінив поточний користувач.

Як і у шарі даних, усі сервіси шару бізнес логіки реалізують власні інтерфейси. Це забезпечує взаємодію шара презентації з ним через абстракції, а не через конкретні реалізації сервісів.

Нижче на рисунку 2.13 представлена діаграма класів-сервісів і наведено опис класів шару бізнес логіки:

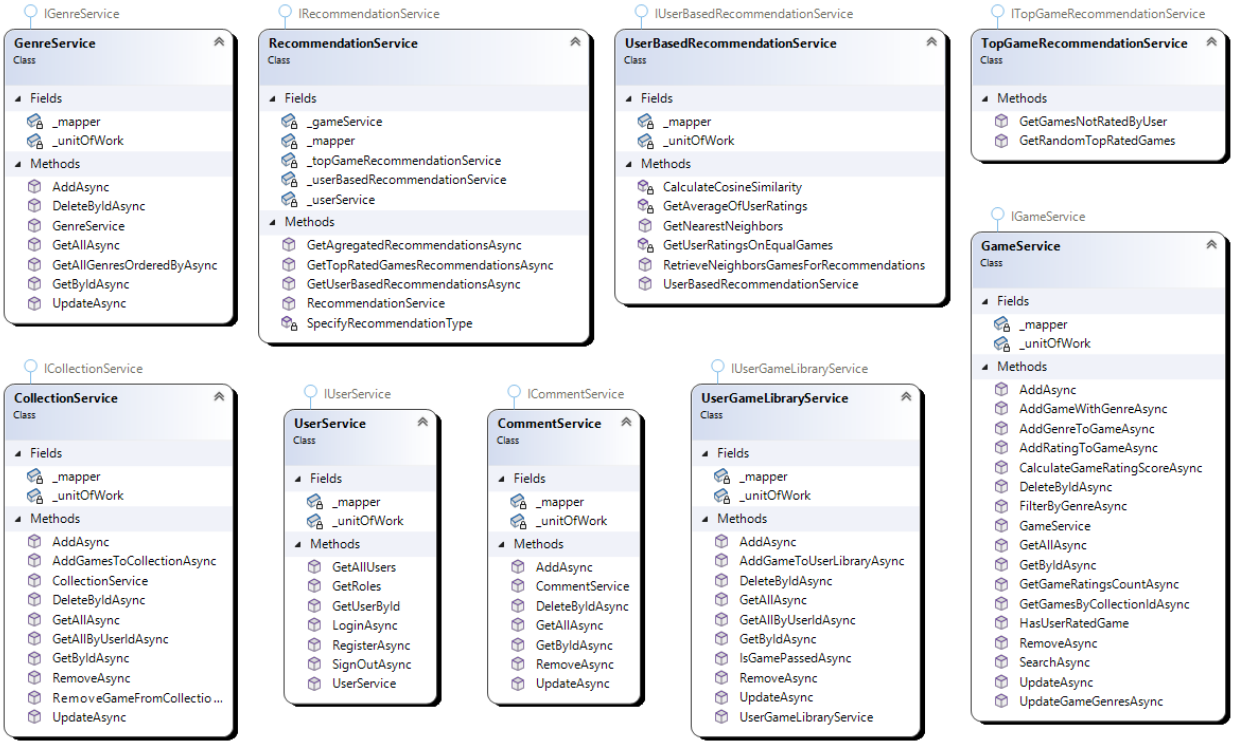


Рисунок 2.13 – Діаграма класів-сервісів

- AutomapperProfile - Клас з налаштуваннями зіставлення класів що представляють Entities та DTO.

- RecommendationOptions - Містить властивості, необхідні для налаштування рекомендацій на основі користувача.

- GameDTO - Модель даних для передачі інформації про гру.

- RecommendedGameDTO - Модель даних для передачі інформації про рекомендовану гру.

- GenreDTO - Модель даних для передачі інформації про жанр.

- CommentDTO - Модель даних для передачі інформації про коментар.

- RatingDTO - Модель даних для передачі інформації про рейтинг.

- CollectionDTO - Модель даних для передачі інформації про колекцію.

- GameGenreDTO - Модель даних для передачі інформації про жанри гри.

- UserGameDTO - Модель даних для передачі інформації про ігри користувача.

- ApplicationUserDTO - Модель даних для передачі інформації про користувача.

- UserCollectionDTO - Модель даних для передачі інформації про колекції користувача.

- IGameService - Інтерфейс описує контракт для роботи з сервісом ігор.

- IGenreService - Інтерфейс описує контракт для роботи з сервісом жанрів.

- IRecommendationService - Інтерфейс описує контракт для роботи з сервісом рекомендацій.

- IUserBasedRecommendationService - Інтерфейс описує контракт для роботи з сервісом відповідальним за рекомендації на основі користувачів.

- ITopGamesRecommendationService - Інтерфейс описує контракт для роботи з сервісом відповідальним за рекомендації на основі високооцінених ігор.

- ICollectionService - Інтерфейс описує контракт для роботи з сервісом колекцій користувача.

- IUserLibraryService- Інтерфейс описує контракт для роботи з сервісом бібліотеки користувача.

- IUserService - Інтерфейс описує контракт для роботи з сервісом користувача.

- ICommentService - Інтерфейс описує контракт для роботи з сервісом коментарів.

- BLLServices - Клас для реєстрації залежностей шару бізнес логіки.

Отже, було спроєктовано структуру застосунку та описано у вигляді діаграми класів, зазначені нюанси взаємодії її елементів. Також було детально описано роботу основного алгоритму – створення рекомендацій, а також виділено особливості його реалізації. Разом із застосунком було спроєктовано й базу даних. Її структуру також відображено графічно у вигляді схеми бази даних, а також описано кожну її таблицю.

3. РОЗРОБКА АРХІТЕКТУРИ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ НАДАННЯ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ

3.1. Вибір архітектури веб-застосунку

Для розробки веб-застосунку була обрана монолітна архітектура. Це традиційна модель програмного забезпечення, яка представляє з себе єдиний модуль, що працює автономно і незалежно від інших додатків. У монолітному застосунку об’єднані всі бізнес-задачі.

На рішення обрати монолітну архітектуру вплинули наступні чинники:

– розробка застосунку виконується однією людиною;

– відсутність складної бізнес-логіки у веб-додатку;

– необхідність у швидкому старті проєкту;

– просте розгортання;

– спрощене налагодження та тестування коду.

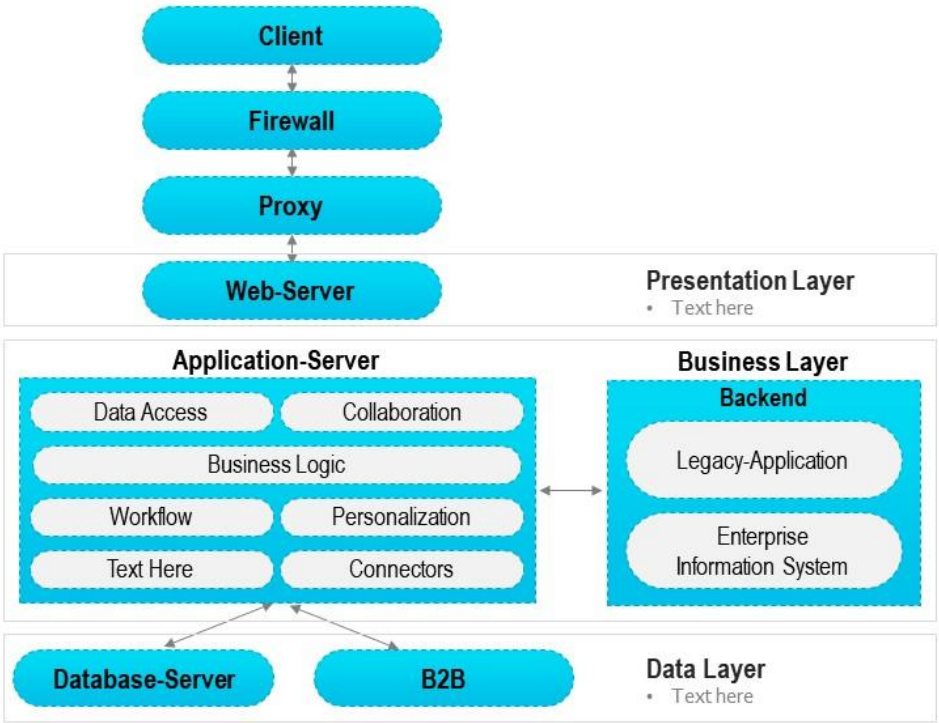


Рисунок 3.1 – Багатошарова архітектура веб-застосунку

Моноліт поділений на логічні шари за принципами багатошарової архітектури (рисунок 3.1). Фундаментальними шарами є: шар представлення, бізнес-логіки і доступу до даних. Користувачі посилають запити за допомогою користувацького інтерфейсу, який взаємодіє лише із шаром бізнес логіки. Шар бізнес-логіки обробляє дані, які йому передає користувач, або залучає до роботи шар доступу до даних за необхідності роботи із базою даних. Шар доступу до даних, відповідно, працює з базою даних. Таким чином, кожен шар має одну конкретну відповідальність.

Для зменшення залежності між шарами застосовується принцип інверсії залежностей. Впровадження цього принципу допомагає досягти більшої гнучкості та розширюваності системи, дозволяючи шарам взаємодіяти через абстракції, замість прямої залежності від конкретних реалізацій. Це полегшує тестування, сприяє внесенню змін та підтримці системи.

Моноліт вважається традиційним підходом до архітектури програмного забезпечення. При його застосуванні весь додаток розробляється як ціла система. У цій архітектурі всі компоненти та модулі застосунку взаємодіють безпосередньо один з одним у межах одного виконуваного процесу.

Для монолітної архітектури характерно розділяти додаток на компоненти які в результаті об’єднані в одну систему. Із основних компонентів що присутні майже в кожному додатку є користувацький інтерфейс, рівень бізнес логіки та рівень управління базами даних. Також часто виділяють окремо такі компоненти як безпека та логування. Безпека даних зазвичай включає в себе захист додатку від несанкціонованого доступу, забезпечення автентифікації та авторизації користувачів, контроль доступу до ресурсів. А логування або моніторинг відповідає за надання інформації про стан додатку.

Користувацький інтерфейс (UI) відповідає за всю взаємодію між користувачем і додатком. Він включає в себе компоненти для відображення

інформації користувачеві, збору введення та обробки дій, які користувач здійснює на стороні клієнта. Користувацький інтерфейс може включати в себе такі елементи, як форми, кнопки, меню, таблиці, графіки тощо. Його основна мета - забезпечити зручну та ефективну взаємодію користувача з додатком.

Бізнес-логіка містить основну функціональність додатку і відповідає за виконання бізнес-правил та обробку даних. Він включає в себе логіку, яка визначає, як додаток повинен опрацьовувати дані, виконувати розрахунки, здійснювати перевірки та приймати рішення. Цей модуль може включати класи, функції, модулі або компоненти, які реалізують різні аспекти бізнес-логіки. Він забезпечує коректну роботу додатку згідно з вимогами та бізнес-правилами, включеними в його функціональність.

Управління даними відповідає за доступ до даних та їх управління. Він включає в себе функції для збереження, отримання, оновлення та видалення інформації з бази даних. Управління даними може включати реляційну або нереляційну базу даних, залежно від потреб проекту. Цей модуль забезпечує правильне збереження та доступ до даних, включаючи оптимізацію запитів, кешування та забезпечення цілісності даних. Він є важливим елементом для забезпечення надійності та ефективності додатку при роботі зі збереженими даними.

Ці модулі взаємодіють між собою для забезпечення повноцінного функціонування монолітної архітектури. Користувацький інтерфейс взаємодіє з користувачем, передаючи його введення та запити до бізнес-логіки. Бізнес-логіка обробляє ці запити, звертається до управління даними для доступу до необхідної інформації та виконання розрахунків, а потім повертає результати назад до користувацького інтерфейсу для відображення.

Управління даними виконує завдання збереження, отримання та оновлення даних у відповідності до логіки додатку і запитів, які надходять з бізнес-логіки.

Отже, у монолітній архітектурі весь додаток розгортається як єдина технологічна система – це є ключовою характеристикою в моноліті. Якщо наприклад відокремити користувацький інтерфейс від серверної частини (бізнес логіки), то така система вже не буде монолітом, а клієнт-серверною архітектурою, що є розподіленою системою. Але варто зазначити що кожна частина розподіленої архітектури може бути окремим монолітом.

Підсумовуючи основні характеристики та перевагимонолітної архітектури.

Основні характеристики монолітної архітектури:

– один виконуваний процес – весь код застосунку компілюється в один виконуваний файл або пакет, який запускається на сервері;

– централізоване управління – всі компоненти та логіка застосунку розміщені в одному місці, що спрощує розробку, внесення змін та управління;

– пряма взаємодія – компоненти взаємодіють між собою безпосередньо через функційні виклики або спільне використання ресурсів бази даних;

– монолітний код – весь код застосунку зазвичай розташовується в одній кодовій базі.

Переваги монолітної архітектури:

– простота розробки – монолітна архітектура дозволяє розробникам швидко створювати та розгортати застосунки без складних конфігурацій;

– простота тестування – у монолітній архітектурі тестування може бути виконане широко та комплексно, охоплюючи всі компоненти застосунку;

– швидка відповідь – монолітні застосунки можуть мати швидку відповідь на запити, оскільки взаємодія між компонентами відбувається локально.

Монолітна архітектура підходить для менших і середніх проектів з обмеженою функціональністю.

В якості сховища даних була обрана об’єктно-реляційна СКБД – PostgreSQL. На її вибір посприяв факт відсутності особливих вимог для рішення. Наразі всі популярні реляційні бази даних можуть задовольнити потреби невимогливих проєктів. Компанії, що віддають перевагу гнучкості, економічній ефективності та інноваціям, зазвичай обирають рішення з відкритим вихідним кодом, що як раз характерно для PostgreSQL. У разі необхідності СКБД можна замінити будь-якою іншою, завдяки архітектурі додатку, що враховує дану можливість.

Щоби реалізувати програмне забезпечення, архітектура якого передбачає взаємодію між різними компонентами, необхідно мати такий фреймворк і мову програмування, які дозволять достатньо швидко розробляти та підтримувати програмне забезпечення з урахуванням змінних вимог проєкту. Таким чином було обрано мову програмування C# та фреймворк .NET Core, що дозволяють задовольнити ці вимоги.

.NET Core є сучасним та ефективним інструментом для розробки. Його особливістю є кросплатформність. Це означає, що він працює у Windows, Mac OS X і деяких дистрибутивах Linux.

Для побудови фронтенд частини застосунку було вирішено використовувати популярний фреймворк Bootstrap. Bootstrap –це відкрите програмне забезпечення для розробки веб-інтерфейсів із використанням HTML, CSS і JavaScript. Він містить набір готових стилів, компонентів та інструментів для швидкої і простої розробки адаптивних та зручних веб-сторінок і веб-додатків.

Популярність Bootstrap пов'язана з тим, що він дозволяє верстати сайти в кілька разів швидше, ніж використовуючи CSS і JavaScript у чистому вигляді.

Ще один аспект – доступність. Вона зводиться до того, що надає можливість навіть початківцю веб-розробнику (без глибоких знань і достатньої практики) створювати якісні макети.

3.2. Розробка інтерфейсу входу в систему

Для того щоб мати можливість повноцінно взаємодіяти з бібліотекою ігор необхідно виконати реєстрацію. Для цього треба натиснути кнопку “Register” у правому верхньому куту сторінки (рисунок 3.2).

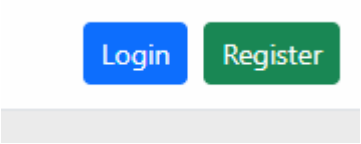


Рисунок 3.2 – Кнопки для входу та реєстрації

Після цього відкриється сторінка з формою для реєстрації (рисунок 3.3), де необхідно вказати свою електронну адресу, нікнейм, пароль і повторити його знову. Далі необхідно підтвердити дію.

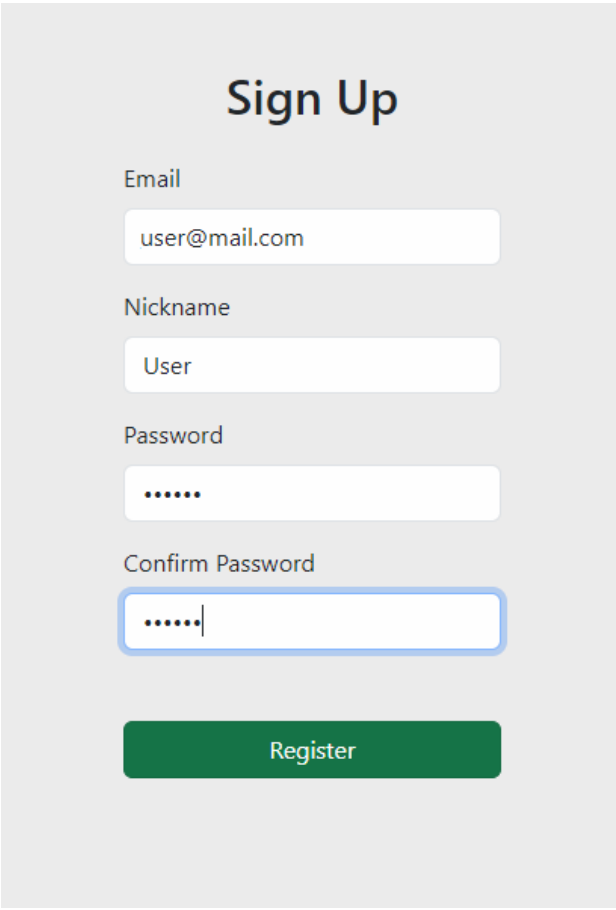


Рисунок 3.3 – Форма реєстрації

Якщо реєстрація не пройшла успішно, то з’явиться повідомлення з інформацією про помилку (рисунок 3.4). Потім система надасть інформацію щодо подальших дій користувача.

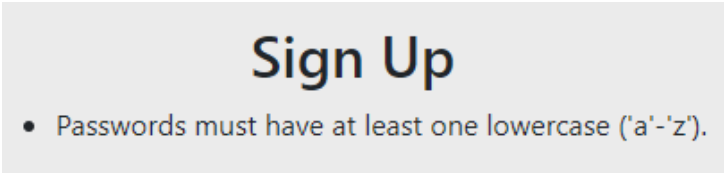


Рисунок 3.4 – Приклад можливої помилки

Якщо реєстрація в системі надання персоналізованих рекомендацій пройшла успішно, то відкриється головна сторінка (рисунок 3.6) з додатковими елементами інтерфейсу, що стають доступними після успішної авторизації.

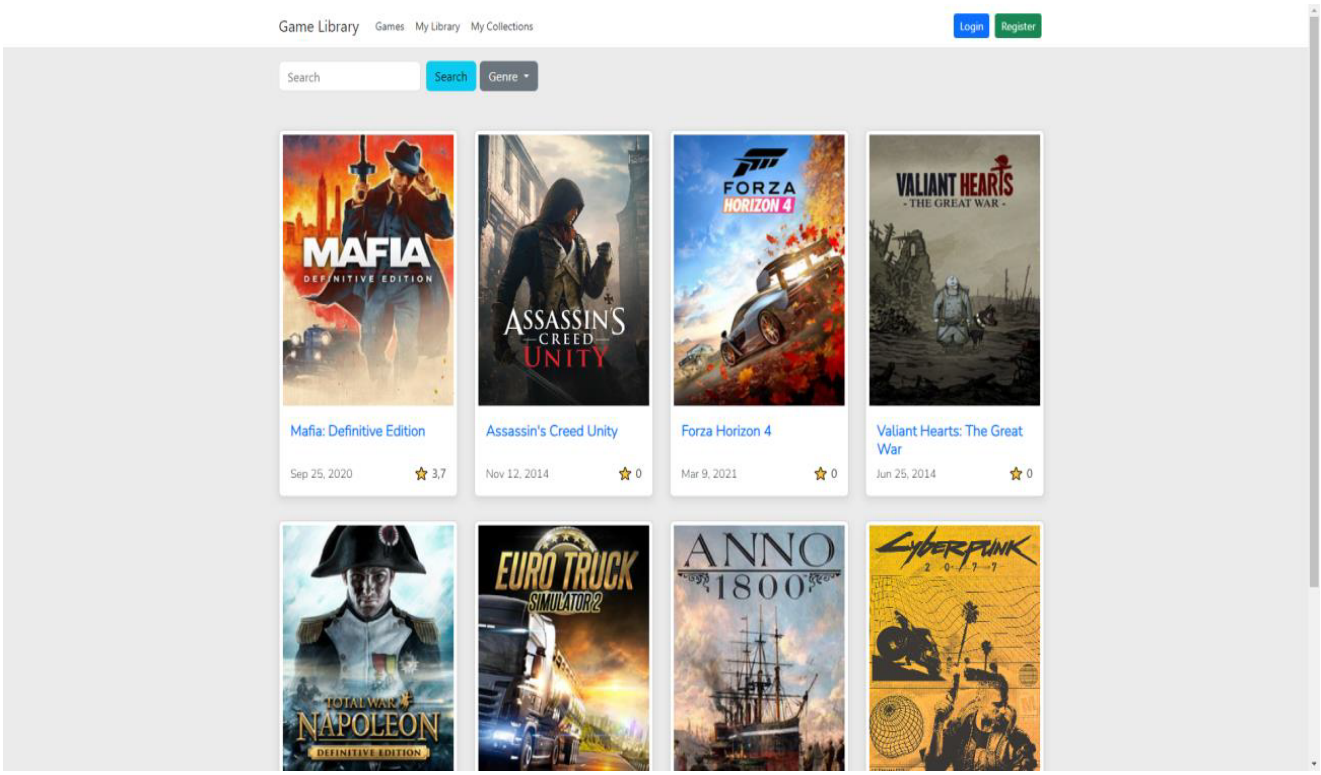


Рисунок 3.5 – Вигляд головної сторінки системи незареєстрованого користувача

Після успішної реєстрації відображається головна сторінка, яка містить зверху карусель рекомендацій з випадковими іграми, які мають високий рейтинг (рисунок 3.6).

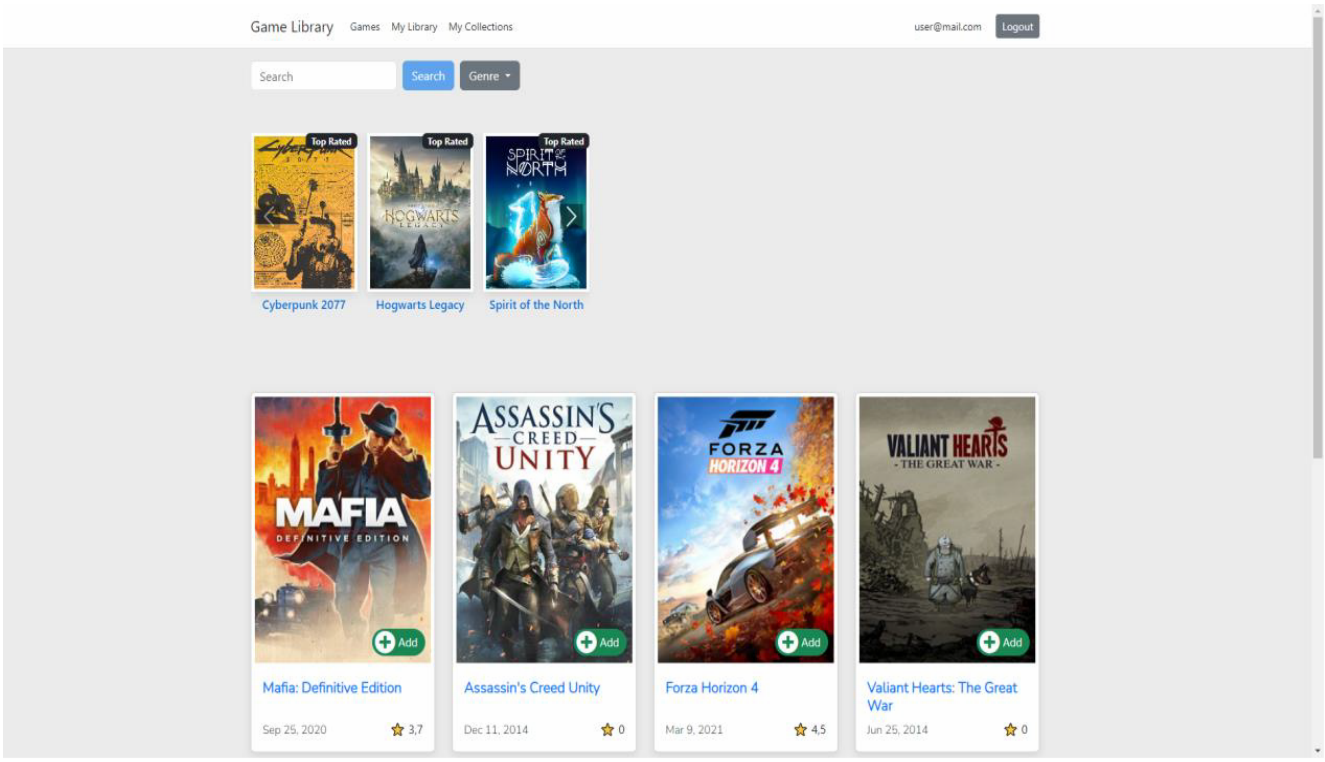


Рисунок 3.6 – Вигляд головної сторінки системи для зареєстрованого користувача

3.3. Реалізація основних функцій системи надання персоналізованих рекомендацій

Після входу в систему користувачу доступна карусель із рекомендаціями. В каруселі рекомендацій (рисунок 3.7) можна побачити випадкові ігри з високим середнім рейтингом. Рейтинг формують усі зареєстровані в системі користувачі надання рекомендацій, що виставляють оцінки іграм.

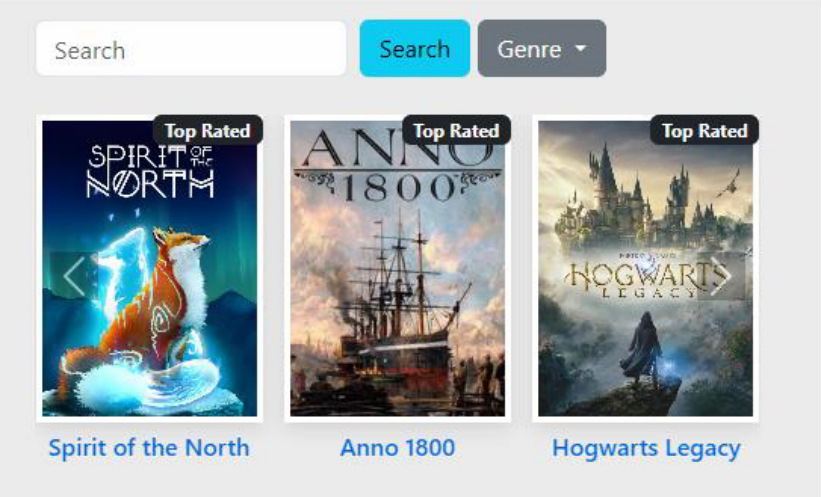


Рисунок 3.7 – Карусель рекомендацій, що містить ігри з високим рейтингом

Для того щоб отримувати персональні рекомендації, необхідно активно приймати участь в оцінюванні ігор. Це сприятиме зростанню якості рекомендацій не тільки для поточного користувача, а і для кожного користувача в системі.

Щоб виставити оцінку необхідно відкрити сторінку з грою, натиснувши на неї, та у відповідне поле вписати рейтинг від 1 до 5. Після цього треба підтвердити дію, натиснувши кнопку із зіркою. На рисунку 3.8 показано приклад виставляння оцінки 4 для певної гри.

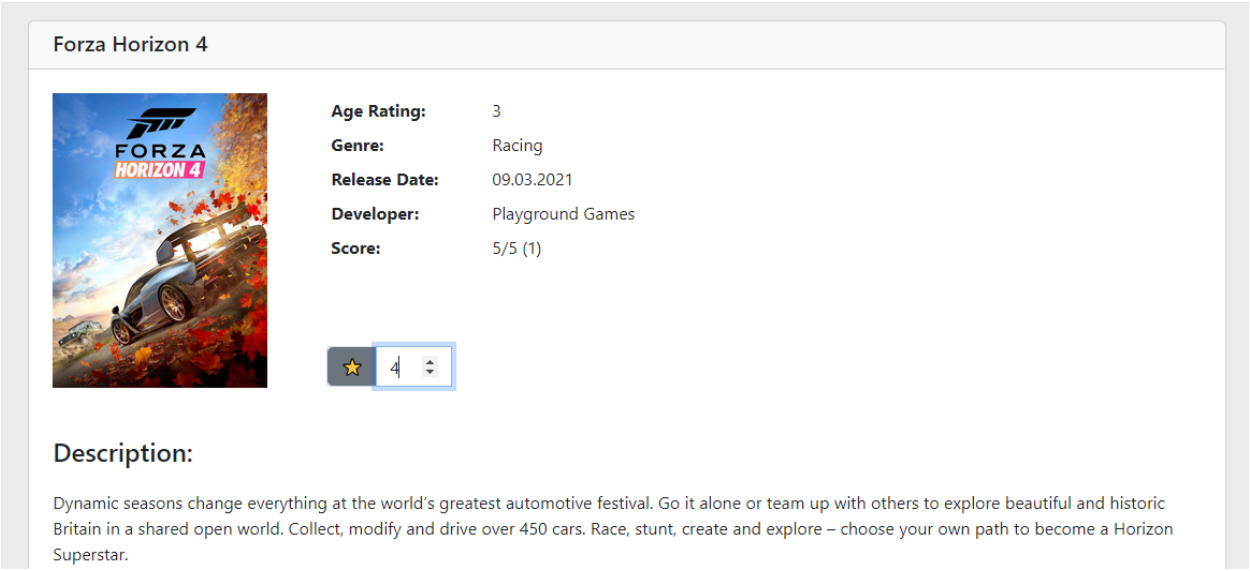


Рисунок 3.8 – Приклад оцінювання гри

Після повернення на головну сторінку, можна буде побачити, що рекомендації змінились і тепер містять не тільки ігри з високим середнім рейтингом, а персоналізовані рекомендації з написом “For You” (рисунок 3.9).

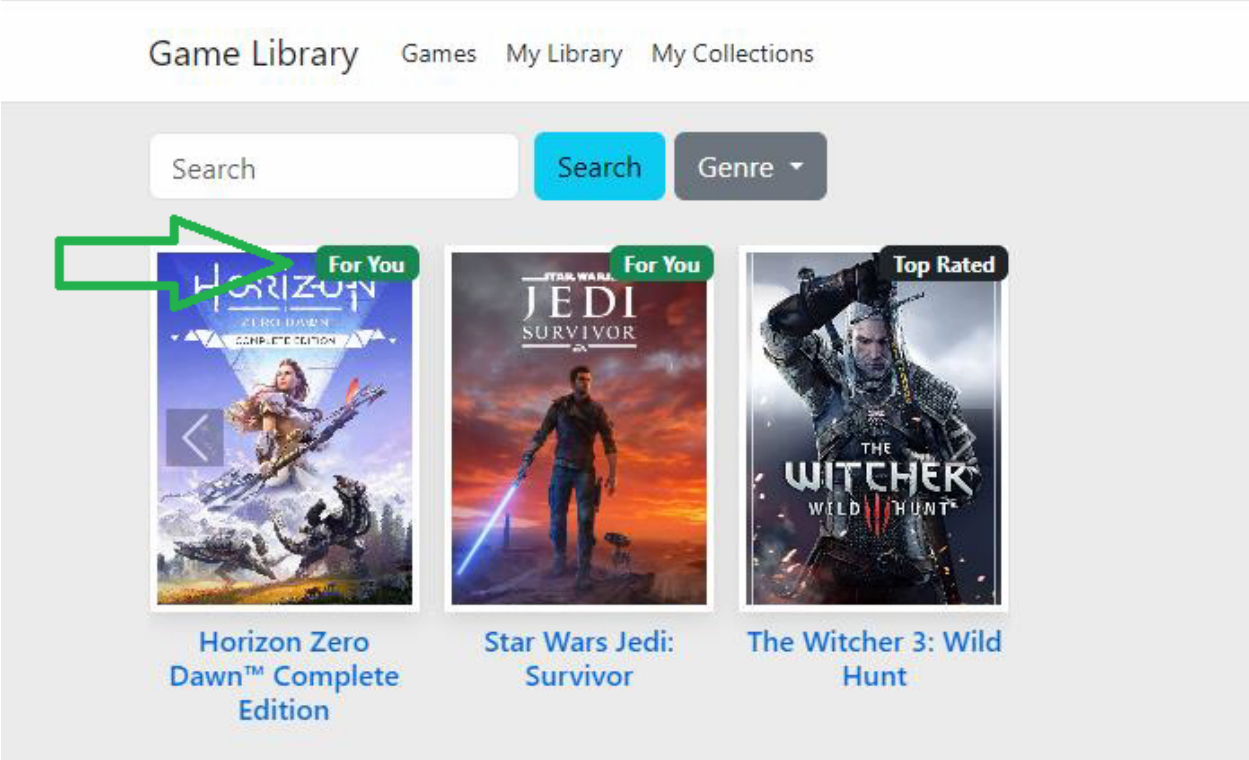


Рисунок 3.9 – Приклад надання персоналізованих рекомендацій

Персоналізовані рекомендації формуються через позитивний рейтинг: тільки так система розуміє вподобання користувача. Тому якщо цей тип рекомендацій не з’явився, то варто оцінити більше ігор, які подобаються користувачу.

Для того щоб додати гру у власну бібліотеку необхідно натиснути кнопку “Add”, яка знаходиться на картці гри (рисунок 3.10). Кнопка доступна тільки під час перегляду ігор на головній сторінці, сторінці пошуку та сторінці фільтрації.

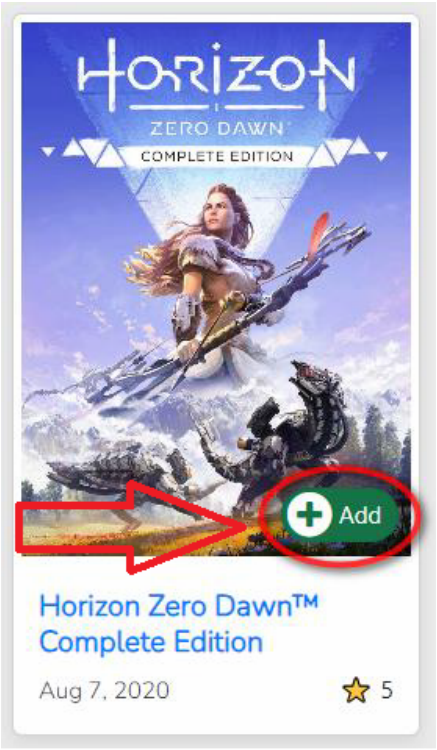


Рисунок 3.10 – Приклад додавання гри у власну бібліотеку

Знайти додані ігри можна в розділі “My Library” основного меню системи (рисунок 3.11).



Рисунок 3.11 – Розділ “My Library”

У цьому ж розділі на ігрових картках є кнопка для вилучення гри з власної бібліотеки з використанням конопки “Remove”, що показано на рисунку 3.12.

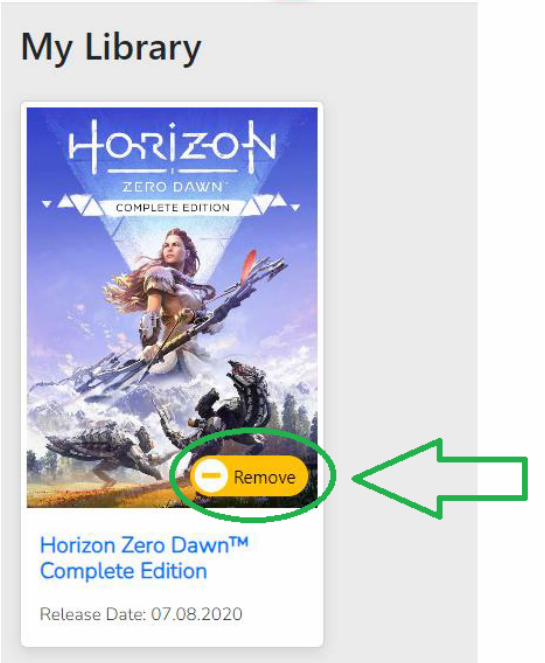


Рисунок 3.12 – Приклад видалення гри з бібліотеки користувача

У розділі “My Collections” (рисунок 3.13) можна створювати нові колекції за допомогою кнопки “Create Collection”.

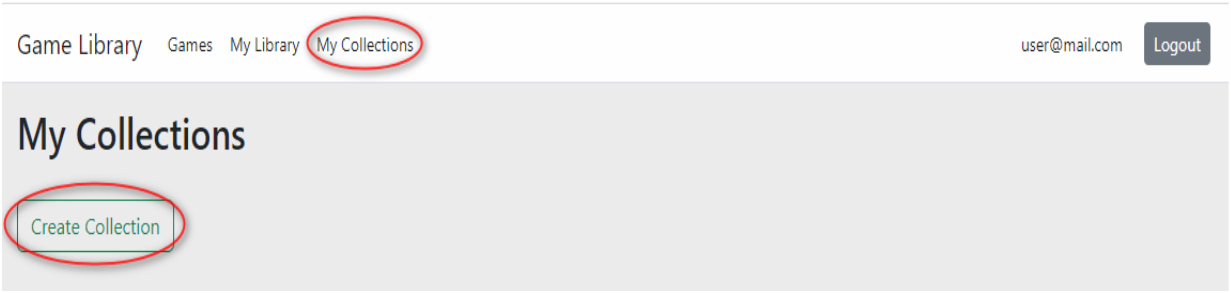


Рисунок 3.13 – Меню створення нової колекції

В створену колекцію можна потім додавати ігри з власної бібліотеки за допомогою кнопки “Add to”. Праворуч від цієї кнопки знаходиться кнопка для редагування інформації про колекцію та кнопка видалення колекції, відповідно (рис. 3.14).

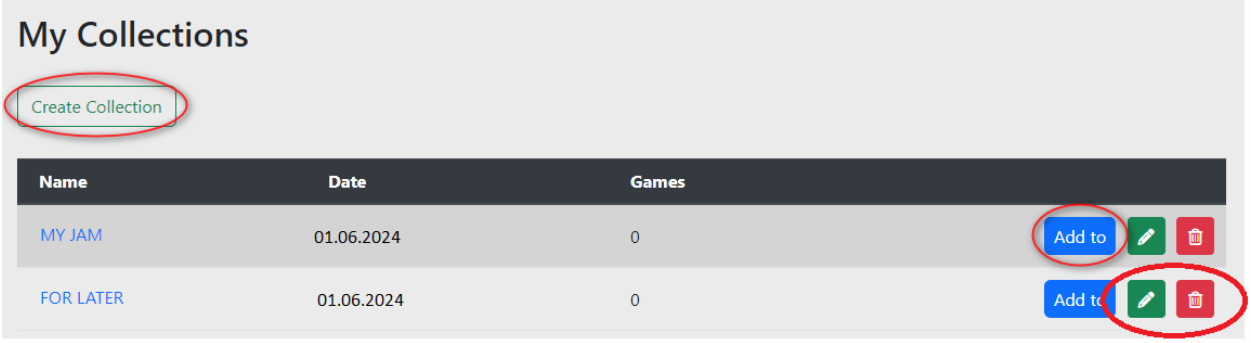


Рисунок 3.14 – Вигляд кнопок щодо управління колекціями

Для того щоб залишити відгук до гри або долучитися до її обговорення з іншими користувачами, необхідно прокрутити сторінку з детальною інформацію про гру донизу.

Щоб залишити коментар треба заповнити текстове поле та натиснути кнопку “Send”. Для редагування коментаря та відповіді на інші коментарі справа в блоці коментаря знаходяться відповідні кнопки.

На рисунку 3.15 зображено інтерфейс для роботи з коментарями та візуальними поясненнями ролей його складових елементів.

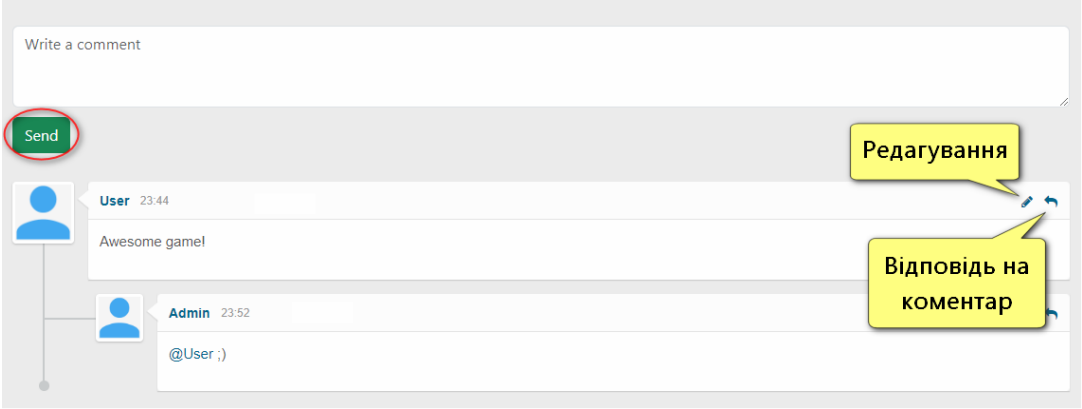


Рисунок 3.15 – Вигляд інтерфейсу написання коментарів

Отже, в даному підрозділі описано приклад роботи веб-застосунку надання персоналізованих рекомендацій від початку реєстрації нового користувача до отримання рекомендацій користувачем.

3.4. Розгортання системи надання рекомендацій

Для розгортання системи було обрано платформу Render, оскільки це одне з не багатьох можливих рішень на сьогодні, що підтримує безкоштовний план.

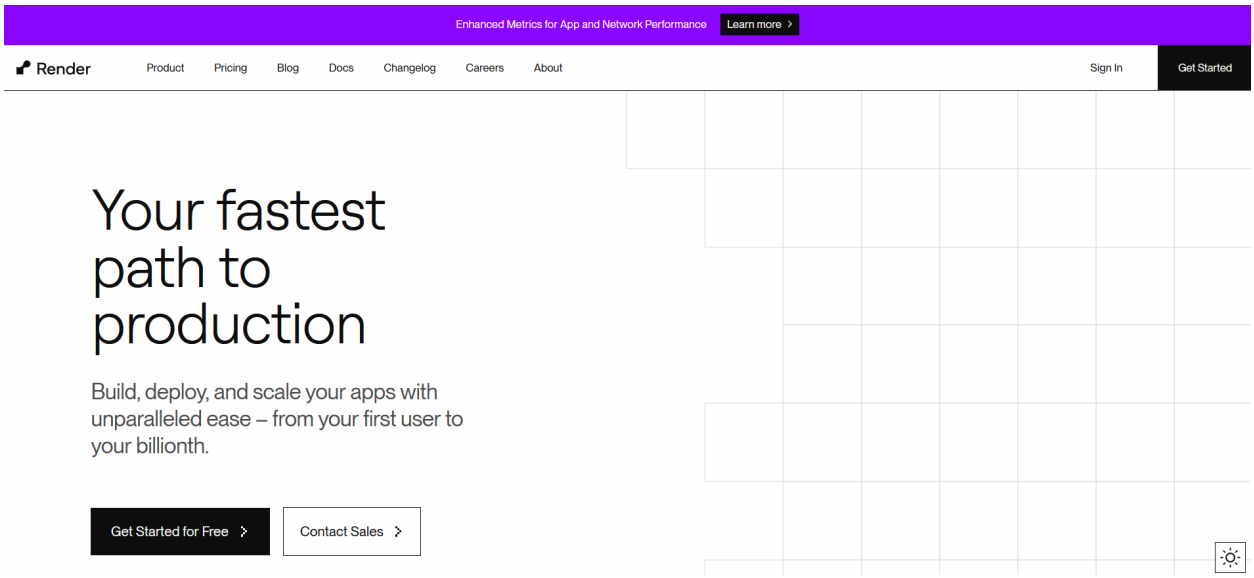


Рисунок 3.16 – Головна сторінка платформи Render

Render є платформою для розгортання та керування веб-додатками і статичними сайтами. Ця платформа надає хмарні обчислювальні ресурси, включаючи сервери, бази даних та інші служби, які допомагають розміщувати і робити доступними веб-додатки в Інтернеті. Render пропонує простий інтерфейс для керування ресурсами, автоматичне масштабування та інші зручні функції для розробників. Він підтримує різні мови програмування і фреймворки, такі як Node.js, Python, Ruby, Go та інші, що дозволяє розгортати різноманітні веб-додатки.

Безоплатний план Render включає можливість створення бази даних PostgreSQL. Для цього необхідно відкрити меню створення нової служби та вибрати з переліку “PostgreSQL” (рисунок 3.17).

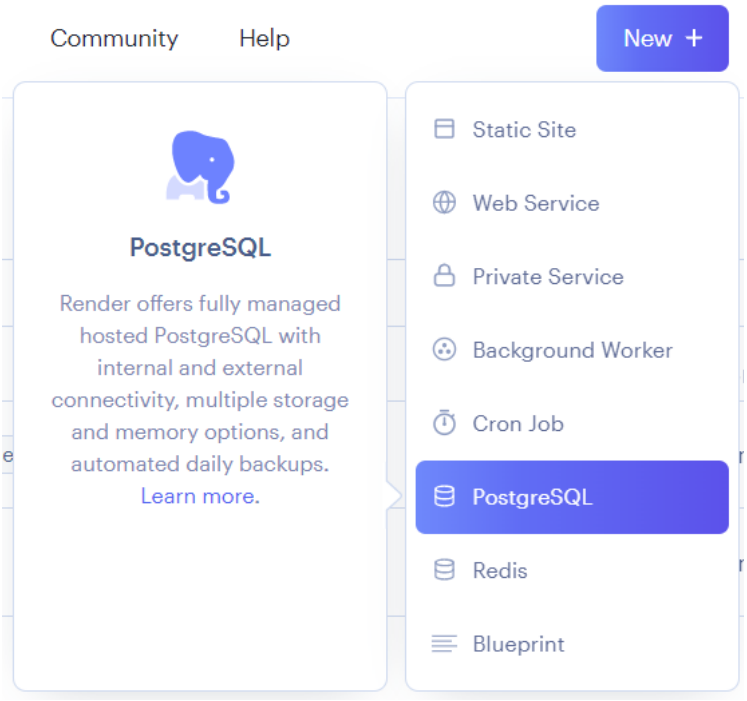


Рисунок 3.17 – Вибір бази даних PostgreSQL на платформі Render

Після того, як екземпляр PostgreSQL створено, необхідно додати новий веб-сервіс (рисунок 3.18).

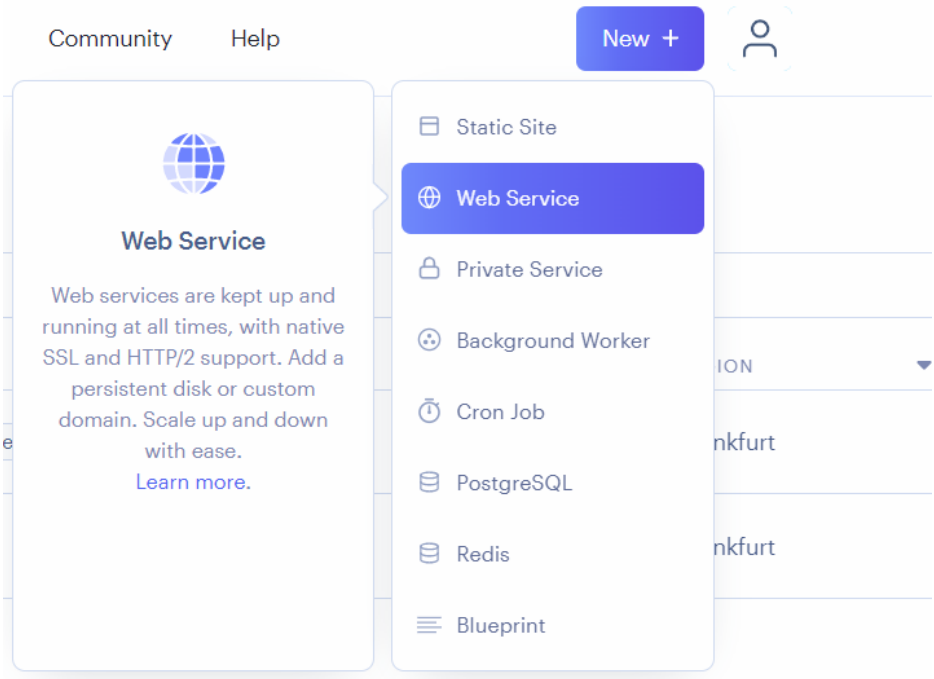


Рисунок 3.18 – Створення нового веб-сервісу

Потім потрібно підключити репозиторій з GitHub, що містить пропоноване рішення, що буде використовуватися цією веб-службою (рисунок 3.19).

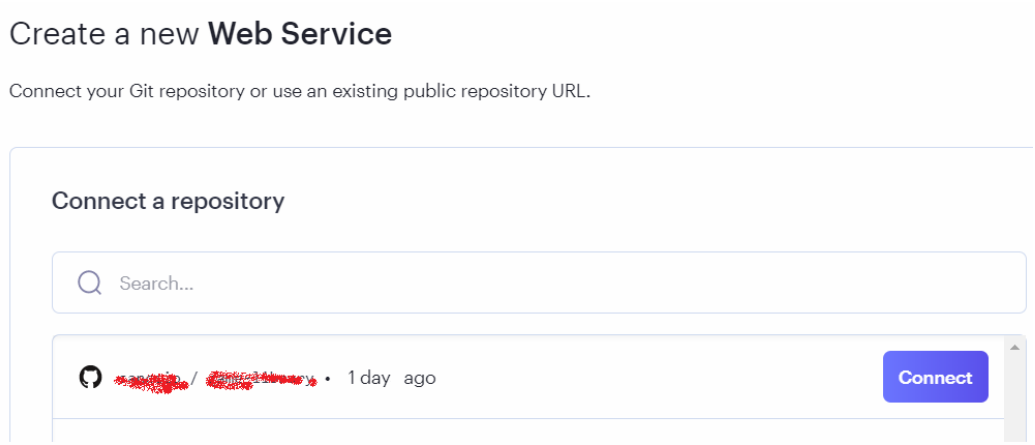


Рисунок 3.19 – Підключення до репозиторію

На сторінці створення нового веб-сервісу потрібно вказати назву та зазначити найближчий регіон (рисунок 3.20).

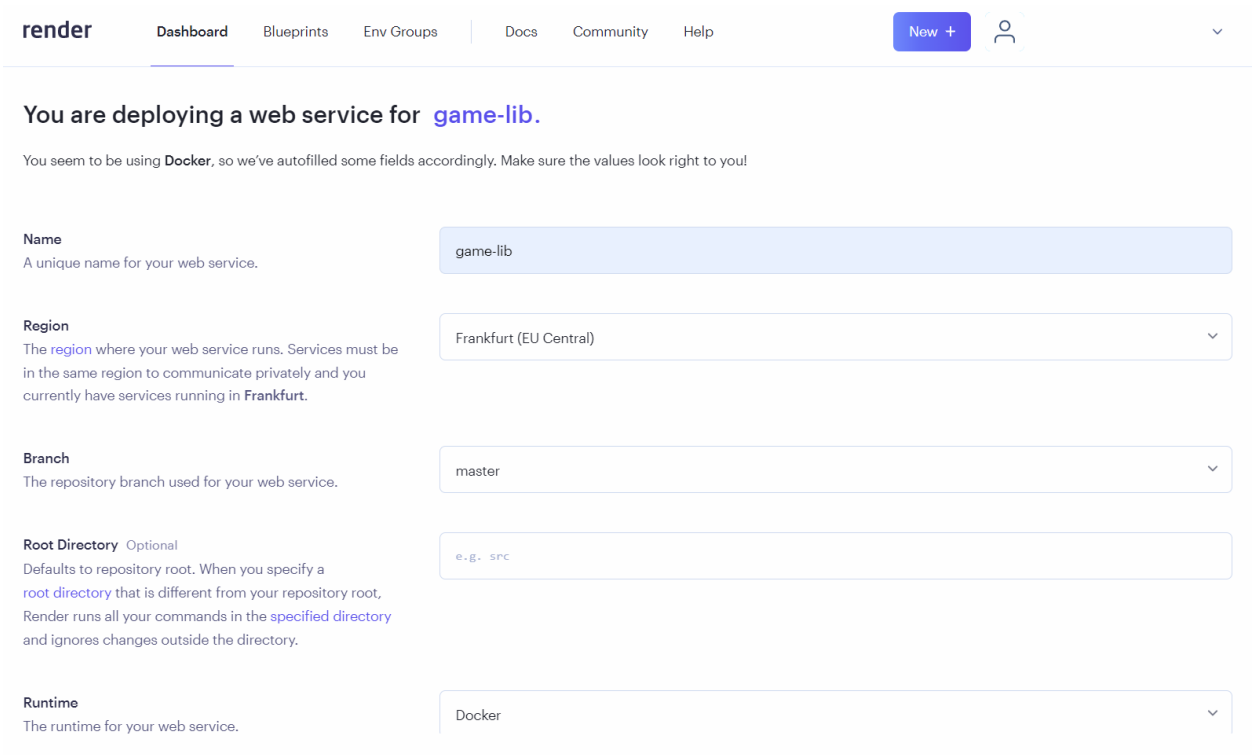


Рисунок 3.20 – Додавання характеристик до сервісу

Розгортання виконується за допомогою Docker файлу, який знаходиться в корені веб-проєкту. Render автоматично виявляє файли Docker та пропонує відповідне середовище.

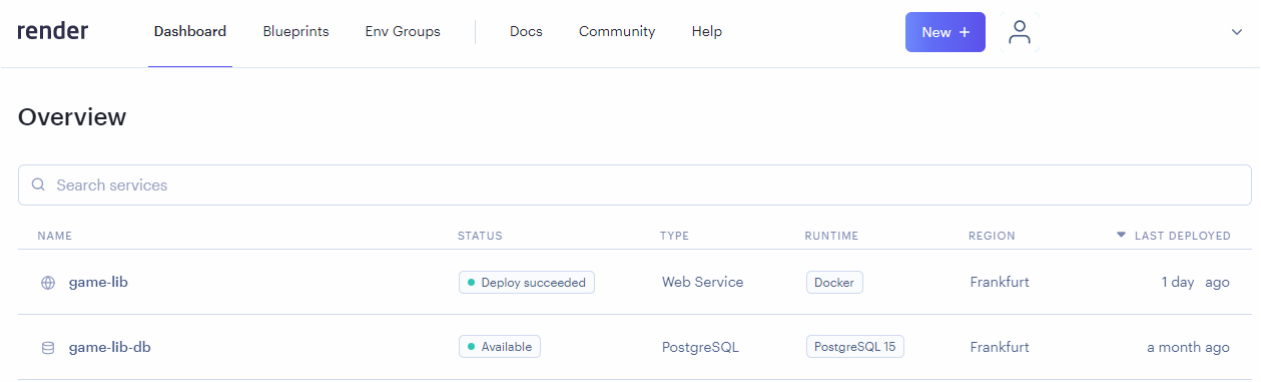


Рисунок 3.21 – Перелік створених служб на платформі

Результатом розгортання є дві служби: веб-сервіс та база даних PostgreSQL, які взаємодіють між собою, що показано на рисунку 3.21.

Оновлення веб-застосунку, розгорнутого на платформі Render, відбувається автоматично після внесення змін до гілки, яка прив’язана до веб-сервісу системи надання рекомекндацій.

ВИСНОВКИ

В дипломній роботі здійснена розробка алгоритмічного та програмного забезпечення надання персоналізованих рекомендацій в межах інформаційного сервісу бібліотеки відеоігор.

Було проведено аналіз існуючих методів та визначено, що найбільш ефективними є гібридні методи, які поєднують різні підходи до формування рекомендацій.

Виконано аналіз вимог до розроблюваної системи надання рекомендацій, розглянуто предметну область, проаналізовано відомі алгоритмічні рішення, засоби розробки, відомі програмні продукти, вимоги до програмного забезпечення; виконана постановка задачі.

У результаті розробки було створено застосунок, який дозволяє користувачам шукати ігрові застосунки, але, що найголовніше – отримати рекомендації на основі оцінених елементів, тобто на основі інтересів користувачів. Слід виділити той факт, що створений застосунок, на відміну від популярних платформ із рекомендаційними системами, більшою мірою базується саме на базі відгуків щодо елементів та схожості користувачів у системі, а не жанровій подібності.

Здійснено моделювання та конструювання програмного забезпечення, обґрунтовано вибір архітектури, проведений аналіз безпеки даних. Здійснено аналізу якості та тестування програмного забезпечення, розглянуто метрики якості коду, описані процеси тестування та тести для перевірки основного функціоналу системи за допомогою наскрізного тестування; наведений опис контрольного прикладу. Також описано процес розгортання застосунку на платформі і зазначено спосіб його підтримки.

В результаті роботи було розроблено систему надання персоналізованих рекомендацій, яка використовує найбільш ефективні

методи формування рекомендацій та забезпечує високу точність та різноманітність наданих рекомендацій.

Готовий застосунок системи надання персоналізованих рекомендацій було успішно проаналізовано на якість реалізації та повною мірою перевірено на коректність роботи. Це дозволяє стверджувати про достатній рівень розробки, надійність роботи застосунку, а також його відповідність усім затвердженим вимогам. Після цього, додаток було розгорнуто для використання користувачами.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Aggarwal C. C. An Introduction to Recommender Systems. Recommender Systems. Cham, Springer International Publishing, 2016. P. 1–28.
2. Мелешко Е. В., Семенов С. Г., Хох В. Д. Дослідження методів побудови рекомендаційних систем в мережі інтернет. 2018. Т. 1, № 47. С. 131–136.
3. Gaming Your Mental Health: A Narrative Review on Mitigating Symptoms of Depression and Anxiety Using Commercial Video Games / M. Kowal та ін. 2021. DOI: 10.2196/26575.
4. Charu C. Aggarwal – Recommender systems: the textbook, 2016 – Springer. P. 33–38.
5. Newzoo Global Games Market Report 2022. URL: https://newzoo.com/resources/trend-reports/newzoo-global-games-marketreport-2022-free-version.
6. Ricci F., Rokach L., Shapira B. Introduction to Recommender Systems
7. Handbook. Recommender Systems Handbook. Boston, MA: Springer, 2010. P. 1– 35.
8. Sharma P., Yadav L. Movie recommendation system using item based collaborative filtering. International Journal of Innovative Research in Computer Science & Technology. 2020. Vol. 8, No. 4. P. 1–10.
9. Raghuwanshi S. K., Pateriya R. K. Movie Recommendation System Content-Based and Collaborative Filtering. International Journal of Computer Sciences and Engineering. 2018. Vol. 6, No. 4. P. 476–481.
10. Saini V. K., Singh J. R. Recommendation System. International Journal for Modern Trends in Science and Technology. 2021. Vol. 6, No. 12. P. 484–492.

11. Tout K., Evans D. J., Yakan A. Collaborative filtering: special case in predictive analysis. International Journal of Computer Mathematics. 2005. Vol. 82, No. 1. P. 1–11.
12. Мелешко Ю. Методи оцінки якості роботи рекомендаційних систем. Системи управління, навігації та зв'язку. Збірник наукових праць. 2018. Т. 5, № 51. С. 92–97.
13. Visual Studio. URL: https://visualstudio.microsoft.com/
14. Sarwar B., Karypis G., Konstan J., Reidl J. Item-based collaborative filtering recommendation algorithms. Proceedings of the 10th international conference on World Wide Web; ACM, 2001. 295 c
15. Carousel. URL: https://getbootstrap.com/docs/5.3/components/carousel/#how-it-works
16. Rider. URL: https://www.jetbrains.com/rider/
17. IGDB. URL: https://www.igdb.com/
18. Collaborative filtering: URL: https://www.analyticsvidhya.com/blog/2022/02/introduction-to-collaborativefiltering/
19. Twitch Acquires Gaming Database Website IGDB. 2019. URL: https://www.pcmag.com/news/twitch-acquires-gaming-database-ebsiteigdb.
20. BPMN. 2022. URL: https://www.ibm.com/cloud/blog/bpmn
21. Object-Relational Mapping. URL: https://www.techopedia.com/definition/24200/object-relational-mapping--orm.
22. Devika P. V. Book Recommendation System. International Conference on Computing Communication and Networking Technologies (ICCCNT): Proceedings of 12th conference, Kharagpur, India, 6–8 July, 2021. P. 3–9.
23. Cacheda F. Comparison of collaborative filtering algorithms. ACM Transactions on the Web. 2011. Vol. 5, No. 1. P. 1–33.
24. Dahiya P., Duhan N. Comparative Analysis of Various Collaborative Filtering Algorithms. International Journal of Computer Sciences and Engineering. 2019. Vol. 7, No. 8. P. 347–351.

25. Herzog D., Wörndl W. Extending Content-Boosted Collaborative Filtering for Context-aware, Mobile Event Recommendations. International Conference on Web Information Systems and Technologies: Proceedings of 12th conference, Rome, Italy, 23–25 April, 2016. P. 293–303
26. Projection. URL: https://docs.automapper.org/en/stable/Projection.html
27. Code metrics – Maintainability index range and meaning. 2022. URL: https://learn.microsoft.com/en-us/visualstudio/code-quality/code-metricsmaintainability-index-range-and-meaning?view=vs-2022

ДОДАТКИ

Додаток А

Частина лістингів коду

UserBasedRecommendationService

```
namespace BLL.Core.RecommendationSystem
{
public class UserBasedRecommendationService :
IUserBasedRecommendationService
{
readonly IUnitOfWork _unitOfWork;
readonly IMapper _mapper;
public UserBasedRecommendationService(IUnitOfWork unitOfWork, IMapper
mapper)
{
_unitOfWork = unitOfWork;
_mapper = mapper;
}
private static double CalculateCosineSimilarity(ApplicationUserDTO
targetUserDTO, ApplicationUserDTO userDTO)
{
double numeratorSum = 0;
double firstDenumeratorSum = 0;
double secondDenumeratorSum = 0;
GetUserRatingsOnEqualGames(targetUserDTO, userDTO, out List<RatingDTO>
targetUserRatingsDTO, out List<RatingDTO> userRatingsToCompareDTO);
for (int i = 0; i < targetUserRatingsDTO.Count; i++)
{
numeratorSum += targetUserRatingsDTO[i].GameRating *
userRatingsToCompareDTO[i].GameRating;
firstDenumeratorSum += Math.Pow(targetUserRatingsDTO[i].GameRating,
2);
secondDenumeratorSum +=
Math.Pow(userRatingsToCompareDTO[i].GameRating, 2);
}
return numeratorSum / (Math.Sqrt(firstDenumeratorSum) *
Math.Sqrt(secondDenumeratorSum));
}
private static void GetUserRatingsOnEqualGames(ApplicationUserDTO
targetUserDTO, ApplicationUserDTO userDTO, out List<RatingDTO>
targetUserRatingsDTO,
out List<RatingDTO> userRatingsToCompareDTO)
{
targetUserRatingsDTO = new();
userRatingsToCompareDTO = new();
for (int i = 0; i < targetUserDTO.Ratings.Count; i++)
{
for (int j = 0; j < userDTO.Ratings.Count; j++)
{
if (targetUserDTO.Ratings[i].GameId == userDTO.Ratings[j].GameId)
{
targetUserRatingsDTO.Add(targetUserDTO.Ratings[i]);
userRatingsToCompareDTO.Add(userDTO.Ratings[j]);
}
```

```
}
}
}
public List<ComparedUserModel> GetNearestNeighbors(ApplicationUserDTO
targetUserDTO, IEnumerable<ApplicationUserDTO> usersDTO, double
minAverageOfUserRatings, int userCount)
{
List<ComparedUserModel> neighbors = new();
if (GetAverageOfUserRatings(targetUserDTO) <= minAverageOfUserRatings)
{
return neighbors;
}
foreach (var user in usersDTO)
{
if (targetUserDTO.Id != user.Id && GetAverageOfUserRatings(user) >=
minAverageOfUserRatings)
{
neighbors.Add(new ComparedUserModel
{
ComparedUserId = user.Id,
SimilarityScore = CalculateCosineSimilarity(targetUserDTO,
user)
});
}
}
var similarUsers = neighbors.OrderByDescending(c =>
c.SimilarityScore).Take(userCount).Where(c =>
!double.IsNaN(c.SimilarityScore)).ToList();
return similarUsers;
}
private static double GetAverageOfUserRatings(ApplicationUserDTO
userDTO)
{
return userDTO.Ratings.Select(x =>
x.GameRating).DefaultIfEmpty().Average();
}
public async Task<List<GameDTO>>
RetrieveNeighborsGamesForRecommendations(string currentUserId,
List<ComparedUserModel> neighbors, double minAverageRating)
{
var ratings = _unitOfWork.RatingRepository.GetAllWithIncludes();
var gamesRatedByTargetUser = ratings
.Where(r => r.ApplicationUserId == currentUserId)
.Select(r => r.Game);
var gamesRatedByComparedUsers = ratings
.Where(r => neighbors.Select(cu => cu.ComparedUserId)
.Contains(r.ApplicationUserId) & r.GameRating >= minAverageRating)
.Select(r => r.Game);
var gamesToRecommend = await
gamesRatedByComparedUsers.Except(gamesRatedByTargetUser).ToListAsync()
;
return _mapper.Map<List<Game>, List<GameDTO>>(gamesToRecommend);
}
}
}
```

GameService.cs

```
namespace BLL.Core
{
public class GameService : IGameService
{
readonly IUnitOfWork _unitOfWork;
readonly IMapper _mapper;
public GameService(IUnitOfWork unitOfWork, IMapper mapper)
{
_unitOfWork = unitOfWork;
_mapper = mapper;
}
public async Task AddAsync(GameDTO gameDTO)
{
var game = _mapper.Map<GameDTO, Game>(gameDTO);
await _unitOfWork.GameRepository.InsertAsync(game);
await _unitOfWork.SaveChangesAsync();
}
public async Task DeleteByIdAsync(int id)
{
var game = await _unitOfWork.GameRepository.GetByIdAsync(id);
_unitOfWork.GameRepository.Delete(game);
await _unitOfWork.SaveChangesAsync();
}
public async Task RemoveAsync(GameDTO gameDTO)
{
var game = _mapper.Map<GameDTO, Game>(gameDTO);
_unitOfWork.GameRepository.Delete(game);
await _unitOfWork.SaveChangesAsync();
}
public async Task<IEnumerable<GameDTO>> GetAllAsync()
{
var games = await
_unitOfWork.GameRepository.GetAllWithIncludesAsync();
return _mapper.Map<IEnumerable<Game>, IEnumerable<GameDTO>>(games);
}
public async Task<GameDTO> GetByIdAsync(int id)
{
var game = await
_unitOfWork.GameRepository.GetByIdWithIncludesAsync(id)
?? throw new ObjectNotFoundException(nameof(Game), id);
return _mapper.Map<Game, GameDTO>(game);
}
public async Task UpdateAsync(GameDTO gameDTO)
{
var game = _mapper.Map<GameDTO, Game>(gameDTO);
_unitOfWork.GameRepository.Update(game);
await _unitOfWork.SaveChangesAsync();
}
public async Task<IEnumerable<GameDTO>> SearchAsync(string
searchString)
{
var games = await GetAllAsync();
return games.Where(g =>
g.Name.ToUpper().Contains(searchString.ToUpper()));
}
```

```
public async Task<IEnumerable<GameDTO>> FilterByGenreAsync(int
gameGenreId)
{
var gameGenres = await _unitOfWork.GameGenresRepository.GetAllAsync();
var gameGenresDTO = _mapper.Map<IEnumerable<GameGenre>,
IEnumerable<GameGenreDTO>>(gameGenres);
ICollection<GameDTO> gamesDTO = new List<GameDTO>();
var result = gameGenresDTO.Where(x => x.GenreId == gameGenreId);
foreach (var gameGenreDTO in result)
{
var game = await GetByIdAsync(gameGenreDTO.GameId);
gamesDTO.Add(game);
}
return gamesDTO;
}
public async Task AddGameWithGenreAsync(GameDTO gameDTO, string[]
selectedGenres)
{
var game = _mapper.Map<GameDTO, Game>(gameDTO);
await _unitOfWork.GameRepository.InsertAsync(game);
await _unitOfWork.SaveChangesAsync();
for (int i = 0; i < selectedGenres.Length; i++)
{
int genreId = int.Parse(selectedGenres[i]);
await AddGenreToGameAsync(game.Id, genreId);
}
await _unitOfWork.SaveChangesAsync();
}
public async Task AddGenreToGameAsync(int gameId, int genreId)
{
GameGenreDTO gameGenreDTO = new()
{
GameId = gameId,
GenreId = genreId
};
var gameGenre = _mapper.Map<GameGenreDTO, GameGenre>(gameGenreDTO);
await _unitOfWork.GameGenresRepository.InsertAsync(gameGenre);
}
public async Task<IEnumerable<UserCollectionDTO>>
GetGamesByCollectionIdAsync(int collectionId)
{
var userCollections = await
_unitOfWork.UserCollectionRepository.GetAllWithIncludes().ToListAsync(
);
var userCollectionGames = userCollections.Where(g => g.CollectionId ==
collectionId).ToList();
return _mapper.Map<IEnumerable<UserCollection>,
IEnumerable<UserCollectionDTO>>(userCollectionGames);
}
public async Task AddRatingToGameAsync(string userId, int gameId, int
ratingScore)
{
if (ratingScore < 1 || ratingScore > 5)
{
throw new ArgumentOutOfRangeException(nameof(ratingScore));
}
bool isRatingExist = await HasUserRatedGame(gameId, userId);
```

```
if (!isRatingExist)
{
RatingDTO ratingDTO = new()
{
ApplicationUserId = userId,
GameId = gameId,
GameRating = ratingScore
};
await _unitOfWork.RatingRepository.InsertAsync(_mapper.Map<RatingDTO,
Rating>(ratingDTO));
await _unitOfWork.SaveChangesAsync();
}
}
public async Task<int> GetGameRatingsCountAsync(int gameId)
{
var ratings = await _unitOfWork.RatingRepository.GetAllAsync();
var ratingsCount = ratings.Where(r => r.GameId == gameId).Count();
return ratingsCount;
}
public async Task<double> CalculateGameRatingScoreAsync(int gameId)
{
var ratings = await _unitOfWork.RatingRepository.GetAllAsync();
var gameRatings = ratings.Where(r => r.GameId == gameId);
if (!gameRatings.Any())
{
return 0;
}
return gameRatings.Average(r => r.GameRating);
}
public async Task UpdateGameGenresAsync(int gameId, string[]
selectedGenres)
{
var game = await
_unitOfWork.GameRepository.GetByIdWithIncludesAsync(gameId)
?? throw new ObjectNotFoundException(nameof(Game), gameId);
foreach (var gameGenre in game.GameGenres)
{
_unitOfWork.GameGenresRepository.Delete(gameGenre);
}
foreach (string selectedGenre in selectedGenres)
{
int genreId = int.Parse(selectedGenre);
await AddGenreToGameAsync(game.Id, genreId);
}
await _unitOfWork.SaveChangesAsync();
}
public async Task<bool> HasUserRatedGame(int gameId, string userId)
{
var game = await _unitOfWork.GameRepository.GetByIdAsync(gameId);
if (game.Ratings == null)
{
return false;
}
var hasUserRated = game.Ratings
.Where(r => r.ApplicationUserId == userId && r.GameId == gameId)
.Any();
return hasUserRated;
```

```
}
}
}
```

CommentService.cs

```
namespace BLL.Core
{
public class CommentService : ICommentService
{
readonly IUnitOfWork _unitOfWork;
readonly IMapper _mapper;
public CommentService(IUnitOfWork unitOfWork, IMapper mapper)
{
_unitOfWork = unitOfWork;
_mapper = mapper;
}
public async Task AddAsync(CommentDTO commentDTO)
{
commentDTO.Date = DateTime.UtcNow;
var comment = _mapper.Map<CommentDTO, Comment>(commentDTO);
await _unitOfWork.CommentRepository.InsertAsync(comment);
await _unitOfWork.SaveChangesAsync();
}
public async Task DeleteByIdAsync(int id)
{
var comment = await _unitOfWork.CommentRepository.GetByIdAsync(id);
_unitOfWork.CommentRepository.Delete(comment);
await _unitOfWork.SaveChangesAsync();
}
public async Task RemoveAsync(CommentDTO commentDTO)
{
var comment = _mapper.Map<CommentDTO, Comment>(commentDTO);
_unitOfWork.CommentRepository.Delete(comment);
await _unitOfWork.SaveChangesAsync();
}
public async Task<IEnumerable<CommentDTO>> GetAllAsync()
{
var comments = await _unitOfWork.CommentRepository.GetAllAsync();
return _mapper.Map<IEnumerable<Comment>,
IEnumerable<CommentDTO>>(comments);
}
public async Task<CommentDTO> GetByIdAsync(int id)
{
var comment = await _unitOfWork.CommentRepository.GetByIdAsync(id)
?? throw new ObjectNotFoundException(nameof(Comment), id);
return _mapper.Map<Comment, CommentDTO>(comment);
}
public async Task UpdateAsync(CommentDTO commentDTO)
{
var comment = _mapper.Map<CommentDTO, Comment>(commentDTO);
_unitOfWork.CommentRepository.Update(comment);

await _unitOfWork.SaveChangesAsync();
}
}
```

БІБЛІОГРАФІЧНА ДОВІДКА

Тема дипломної роботи: “ Розробка алгоритмічного та програмного забезпечення надання персоналізованих рекомендацій ”

Обсяг пояснювальної записки: 78 аркушів.

Дата завершення роботи: 17 червня 2024 р.

Підпис _____