Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної Інженерії
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження використання рекомендаційних систем в веб-системі у сфері освіти
(тема)

Виконав:
студент (ка) 2 курсу, групи ІПЗм-22-3

Агатін Є. Л.
(прізвище, ініціали)

Спеціальність 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)
Тип програми освітньо-наукова

Керівник к.т.н., доц. Назаров О.С.
(посада, прізвище, ініціали)

Допускається до захисту
Зав. кафедри _____ З.В.Дудар
(підпис) (прізвище, ініціали)

2024 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
Кафедра Програмної Інженерії
Рівень вищої освіти другий (магістерський)
Спеціальність 121 – Інженерія програмного забезпечення
Тип програми освітньо-наукова програма
Освітня програма Інженерія програмного забезпечення
(шифр і назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«___» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Агатін Євген Леонідович
(прізвище, ім’я, по батькові)

1. Тема роботи «Дослідження використання рекомендаційних систем в веб-системі у сфері освіти»

Затверджена наказом по університету від 29.03. 2024р. № 250 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 18.06.2024

3. Вихідні дані до роботи опис методів дослідження, опис технологій, вимоги до дослідження, вимоги до розробки схеми бази даних для проведення дослідження, мови програмування C#, технології .NET 8.0, СУБД Ms SQL Server, середовища розробки Visual Studio 2022 та Visual Studio Code

4. Перелік питань, що потрібно опрацювати в роботі

вступ, постановка задачі, аналіз предметної галузі, підготовка до проведення дослідження, алгоритми рекомендацій, розробка застосунку для проведення дослідження, аналіз результатів дослідження.

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	Аналіз предметної галузі та постановка задачі	29.03 – 15.04.24	*виконано*
4	Розробка веб-системи	15.04 – 20.0.24	*виконано*
5	Програмна реалізація кожного алгоритмів рекомендаційних систем	20.04 – 25.04.24	*виконано*
6	Експериментальні дослідження	25.04 – 30.04.24	*виконано*
7	Аналіз результатів експериментальних досліджень	30.04 – 01.05.24	*виконано*
8	Написання та оформлення статті та тез доповіді	01.05 – 31.05.24	*виконано*
9	Підготовка пояснювальної записки	01.04 – 6.06.24	*виконано*
11	Нормоконтроль	6.06 – 08.06.24	*виконано*
12	Рецензування	08.06 – 13.06.24	*виконано*
13	Занесення диплома в електронний архів	15.06.2024	*виконано*
14	Попередній захист	15.06.2024	*виконано*
15	Допуск до захисту у зав. кафедри	16.06.2024	*виконано*

Дата видачі завдання 29 березня 2024р.

Студент (ка) _____ Агатін Є.Л.
(підпис)

Керівник роботи _____ к.т.н., доц. Назаров О.С.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка містить: 76 с., 28 рис., 7 таблиці, 12 джерел, 5 додатків.

АЛГОРИТМИ, ВЕБ-ДОДАТОК, МОВА ПРОГРАМУВАННЯ C#, РЕКОМЕНДАЦІЙНІ МОДЕЛІ, ANGULAR, TYPESCRIPT.

Об'єктом дослідження є рекомендаційні моделі які можна використовувати у сфері освіти.

Метою роботи є аналіз різних рекомендаційних моделей, їх ефективність та вплив на продуктивність системи у сфері освіти.

Робота передбачає дослідження та порівняння різних рекомендаційних моделей, на їх результат у сфері освіти та вплив на продуктивність системи.

ALGORITHMS, WEB APPLICATION, C# PROGRAMMING LANGUAGE, RECOMMENDATION MODELS, ANGULAR, TYPESCRIPT.

The object of research is recommendation models that can be used in the field of education.

The purpose of the work is the analysis of various recommendation models, their effectiveness and impact on the productivity of the system in the field of education.

The work involves the study and comparison of various recommendation models, their results in the field of education and the impact on the productivity of the system.

Заява щодо самостійного виконання кваліфікаційної роботи та можливості її публікації в електронному архіві відкритого доступу ElArKhNURE.

Я, Агатін Євген Леонідович, студент гр. ІПЗм-22-3, здобувач вищої освіти на другому (магістерському) рівні кафедри «Програмна інженерія», заявляю: моя кваліфікаційна робота на тему «Дослідження використання рекомендаційних систем в веб-системі у сфері освіти», що буде представлена в екзаменаційну комісію для публічного захисту, виконана самостійно, в ній не містяться елементи плагіату і вона може бути опублікована в електронному архіві відкритого доступу ElArKhNURE. Всі запозичення з друкованих та електронних джерел мають відповідні посилання.

Я ознайомлений з діючим положенням «Про протидію академічному плагіату в ХНУРЕ», згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування дисциплінарних заходів.

ЗМІСТ

ВСТУП

Ще з давніх часів люди навчилися зберігати, розповсюджувати, отримувати інформацію. Інформації ставало занадто багато, що на ознайомлення з усім майже не вистачає часу, та потрібні були з’явитися засоби які допоможуть знаходити, сортувати, фільтрувати інформацію, одним з таких рішень стали – рекомендаційні системи.

Рекомендаційні системи стали неодмінною частиною нашого цифрового світу, використовуючись у різних сферах для полегшення вибору та покращення користувацького досвіду. В сучасному освітньому середовищі, де важливо забезпечити індивідуалізований підхід до навчання, розробка та впровадження рекомендаційних систем набуває особливої актуальності.

Значущі зміни в сфері освіти викликають необхідність впровадження сучасних технологій для оптимізації процесів вибору навчального контенту, індивідуальної траєкторії навчання та підвищення мотивації користувачів. Рекомендаційні системи можуть забезпечити ефективний вибір навчальних ресурсів, враховуючи особливості кожного студента та його академічні потреби.

Метою даного дослідження є вивчення можливостей та перспектив використання рекомендаційних систем у сфері освіти. Робота спрямована на розробку та впровадження системи, яка забезпечить ефективне навчання, враховуючи індивідуальні особливості користувачів та сприятиме їхньому успішному розвитку.

Актуальність дослідження визначається необхідністю вдосконалення освітнього процесу та створення умов для ефективного навчання в умовах сучасного інформаційного суспільства. Розробка рекомендаційної системи для освітнього середовища має великий потенціал у підвищенні якості освіти та задоволення потреб користувачів.

За результатами роботи було підготовлено презентацію (див. додаток Б), а матеріали з тезами доповіді до XXVIII Міжнародного молодіжного форуму “Радіоелектроніка та молодь у XXI столітті”[1] винесені в якості додатку В, на тему “ Дослідження рекомендаційних систем у сфері освіти”.

Для дослідження будуть використовуватись останні, найпопулярніші методи реалізації методів рекомендаційних систем, проаналізована їх робота в цілому в системі та на зібраному датасеті[2].

Таким чином, впровадження рекомендаційних систем у сферу освіти не тільки покращує процес навчання, але й робить його більш індивідуалізованим та адаптованим до потреб кожного студента. Ці системи здатні аналізувати великі обсяги даних, враховувати різноманітні параметри.

Отже, проведене дослідження сприятиме не лише покращенню якості освіти, але й стане основою для подальших розробок у цій галузі. Сподіваємось, що результати нашої роботи будуть корисні як для наукової спільноти, так і для практиків у сфері освіти.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Виявлення проблематики

У світлі стрімкого розвитку сучасних технологій у сфері освіти виникає необхідність впровадження та оптимізації рекомендаційних систем. Ця проблематика обумовлена рядом важливих аспектів, які визначають актуальні завдання дослідження:

– індивідуалізація навчання – зростання різноманітності освітніх потреб користувачів ставить завдання перед системами освіти в індивідуалізації підходу до навчання. Виявлення та врахування унікальних відомостей, навичок та інтересів кожного користувача є ключовим для створення ефективної рекомендаційної системи;

– ефективність освітнього процесу – забезпечення високої якості освіти вимагає ефективного використання ресурсів та визначення оптимальних шляхів розвитку для користувачів. Виявлення та вирішення проблеми покращення освітнього процесу через рекомендаційні системи стає пріоритетним завданням;

– адаптація до змін – швидкі темпи розвитку педагогічних методик та технологій вимагають від рекомендаційних систем гнучкості та здатності швидко адаптуватися до нових вимог та відкриттів у галузі освіти;

– оцінка ефективності – визначення успішності та ефективності рекомендаційних систем у контексті навчання вимагає ретельного аналізу та виявлення факторів, які впливають на їхню дію.

Виявлення цих проблем створює передумови для дослідження та впровадження високоефективних рекомендаційних систем у сфері освіти. Наше дослідження спрямоване на розробку стратегій, які забезпечать не лише інноваційний підхід до навчання, але й покращать якість освітнього процесу, забезпечуючи індивідуалізований підхід для кожного користувача.

2.2 Постановка задачі

Метою кваліфікаційного проекту є підготовка до майбутнього магістерського дослідження. З цією метою визначено конкретні завдання для роботи «Дослідження використання рекомендаційних систем в веб-системі у сфері освіти». Структурована постановка задачі включає наступні етапи:

– аналіз поточного стану рекомендаційних систем в освітній галузі – провести огляд існуючих рекомендаційних систем, визначити їхні переваги та недоліки, ідентифікувати основні виклики та тенденції в цьому напрямку;

– вивчення особливостей сфери освіти та визначення вимог – розглянути основні аспекти освітнього процесу, визначити особливості та вимоги, що визначатимуть ефективність рекомендаційних систем у галузі;

– дослідження можливостей та обмежень рекомендаційних систем в освіті – проаналізувати, як рекомендаційні системи можуть впливати на індивідуальний підхід до навчання, а також вивчити обмеження, які можуть виникнути при їх впровадженні;

– аналіз різних методів створення рекомендаційних систем – розглянути різні підходи та алгоритми, які використовуються в рекомендаційних системах, оцінити їхню ефективність та можливості в контексті освітнього середовища;

– встановлення критеріїв для порівняння алгоритмів рекомендаційних систем – визначити ключові критерії, за якими будуть оцінюватися різні алгоритми рекомендаційних систем у сфері освіти;

– формулювання висновків та узагальнення результатів – підсумувати отримані результати дослідження, сформулювати висновки та рекомендації для подальших кроків у розробці та впровадженні рекомендаційних систем у сфері освіти;

– проаналізувати використання та «підводні камені» використання Angular, C# 12[3] та SQL для створення рекомендаційних моделей.

Ця структурована постановка задачі визначає напрямки подальшого дослідження та забезпечує чітку орієнтацію в процесі виконання роботи.

Зазначені завдання становлять важливий крок у розробці імплементації рекомендаційних систем, спрямованих на вдосконалення освітнього процесу та задоволення потреб користувачів.

2 АНАЛІЗ ПРЕДМЕТНОЇ ГАЛУЗІ

2.1 Рекомендаційні системи у сфері освіти

Рекомендаційні системи – це програмні або алгоритмічні рішення, які використовують аналіз даних та інтелектуальні алгоритми для надання користувачам персоналізованих рекомендацій чи пропозицій. Зазвичай вони застосовуються у великому обсязі галузей, таких як електронна комерція, розваги, медіа, та, зокрема, у сфері освіти.

Існує багато різних видів рекомендаційних систем, основний перелік цих різновидностей:

- фільтр на основі контенту – рекомендації базуються на характеристиках або властивостях елементів та користувачів. Наприклад, в освітній сфері, це може бути рекомендація навчальних матеріалів на основі змісту чи тематики;
- фільтр на основі спільноти (колаборативний фільтр) – враховує спільність інтересів користувачів. Це може включати рекомендації на основі дій та вподобань інших користувачів з подібними інтересами;
- гібридні системи – комбінують підходи фільтра на основі контенту і фільтра на основі спільноти для отримання більш точних та персоналізованих рекомендацій.

Сфера освіти не стала в стороні та також може використовувати рекомендаційні моделі для покращення процесів навчання та засвоєння інформації.

Основі переваги використання:

- персоналізоване навчання – рекомендаційні системи можуть пропонувати індивідуальні навчальні матеріали, курси або завдання, враховуючи стиль навчання, рівень знань та інтереси користувача;
- підтримка вибору курсів – для студентів, які обирають курси, системи можуть надавати рекомендації щодо предметів, які допоможуть

розширити їхні знання або відповідають їхнім майбутнім професійним цілям;

- оптимізація навчального процесу – рекомендаційні системи можуть допомагати вчителям і адміністраторам оптимізувати навчальні програми, визначаючи ефективні методи навчання та підходи до конкретних груп користувачів;
- особистий розвиток – системи можуть рекомендувати додаткові ресурси, літературу чи активності для самостійного вивчення та розвитку;
- ефективне використання освітніх платформ – використання рекомендацій для просування користувачів через різні функції та можливості освітніх платформ.

Рекомендаційні системи в сфері освіти відкривають нові можливості для персоналізованого навчання та підтримки користувачів у досягненні їхніх освітніх цілей. Вони стають невід'ємною частиною інноваційного підходу до навчання, сприяючи покращенню якості освіти та розвитку індивідуального потенціалу користувачів.

2.2 Історія розвитку рекомендаційних систем

Спроби автоматизації процесу рекомендацій виникали ще в 90-х роках. Однак, справжній прорив на цьому полі став можливим завдяки дослідженням в галузі штучного інтелекту та інформаційних технологій. Перші рекомендаційні системи засновані на фільтрах на основі контенту і фільтрах на основі спільноти дозволили користувачам отримувати рекомендації на основі їхніх власних дій або характеристик товарів.

З появою Інтернету та збільшенням обсягів доступної інформації, розширення і вдосконалення методів колаборативного фільтрування стали ключовими. Методи, такі як Singular Value Decomposition та Latent Semantic Analysis, взяли свій старт, дозволяючи системам здійснювати більш точні та персоналізовані рекомендації.

Колаборативне фільтрування[4] – це один із ключових підходів в рекомендаційних системах, який ґрунтується на аналізі взаємодії між користувачами та предметами (товарами, ресурсами, інформацією). Мета полягає у прогнозуванні чи фільтрації інтересів користувачів на основі їхніх попередніх взаємодій чи вподобань. Його можна побачити на рисунку 2.1.

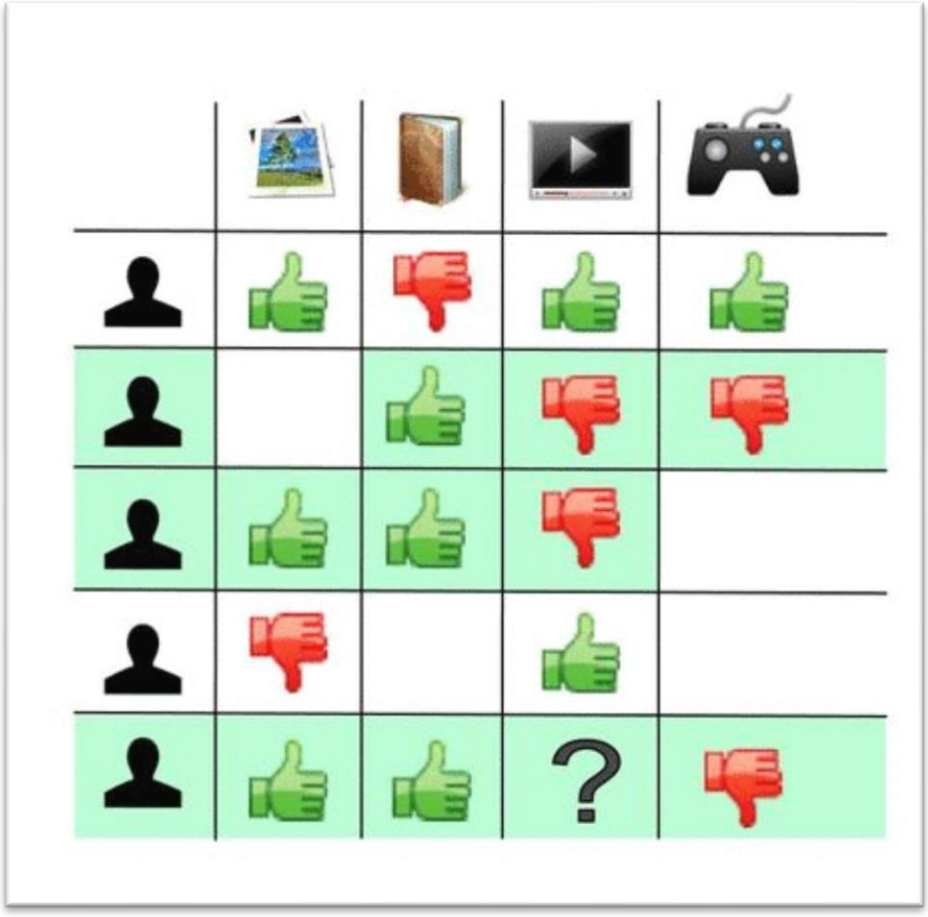

Рисунок 2.1 – Матриця колаборативного фільтрування (з даних[5])

Останнє десятиліття призначене глибоким дослідженням і вдосконаленням рекомендаційних систем. Гібридні системи, що комбінують підходи різних методів (фільтр на основі контенту та колаборативне фільтрування), набули популярності. Також, методи глибокого навчання виявили свою ефективність у вдосконаленні рекомендацій, здатних адаптуватися до складних патернів поведінки користувачів.

В останні роки, рекомендаційні системи входять в різні сфери, включаючи освіту. Вони стали важливим інструментом для персоналізованого навчання,

підтримки вибору курсів, а також розвитку особистих та професійних навичок. Застосування рекомендацій в сфері освіти сприяє створенню ефективних навчальних середовищ та забезпечує доступ студентам до індивідуально підібраних ресурсів.

Усі ці етапи свідчать про постійний розвиток та вдосконалення рекомендаційних систем, які стають все більш важливими в сучасному світі інформації та освіти.

2.3 Колаборативне фільтрування

Основна частина опису колаборативного фільтрування[4] була описана у підрозділі 2.2.

Є 2 види колаборативного фільтрування:

- user-based (Фільтрування на основі користувачів) – цей підхід базується на схожості між користувачами. Якщо два користувачі мають схожі історії взаємодій з предметами, то їм рекомендується подібний контент. Недоліком є проблема «холодного старту» для нових користувачів, яким відсутні попередні взаємодії;
- item-based (Фільтрування на основі предметів) – цей підхід визначає схожість між предметами, а не користувачами. Якщо один предмет був популярний у користувачів, то інші подібні предмети рекомендуватимуться користувачам, які спробували перший предмет.

Цей алгоритм створює Матриці Взаємодій (див. рис. 2.1), інформація про взаємодії користувачів із предметами подається у вигляді матриці, де рядки відповідають користувачам, а стовпці – предметам. Визначає схожість між користувачами або предметами визначається за допомогою різних метрик, таких як косинусна схожість або кореляція. Прогнозує рейтинг для користувача, для якого потрібно зробити рекомендацію, розраховуються прогнозовані рейтинги для предметів, які він ще не спробував. Це може бути зроблено, наприклад, шляхом взваженого усереднення рейтингів схожих користувачів або предметів. В заверщення – формує рекомендації.

Колаборативне фільтрування залишається важливою складовою сучасних рекомендаційних систем і є ефективним інструментом для забезпечення персоналізованих рекомендацій у різних областях, включаючи електронну комерцію, медіа та освіту.

2.4 Фільтрування на основі змісту

Фільтрування на основі контенту[6] є одним з ключових підходів у рекомендаційних системах. Цей метод спирається на аналіз властивостей самого контенту (товарів, ресурсів, інформації) та враховує індивідуальні вподобання користувачів для надання персоналізованих рекомендацій.

Алгоритм спочатку аналізує контент, характеристика самого контенту. Це може включати текстовий зміст (ключові слова, теми), атрибути (наприклад, автор, жанр, рік випуску), зображення або інші властивості. Базуючись на взаємодії користувача з контентом (оцінки, перегляди, вибори), формується профіль вподобань користувача. Розраховується схожість між контентом та профілем користувача. Можна побачити на рисунку 2.2.

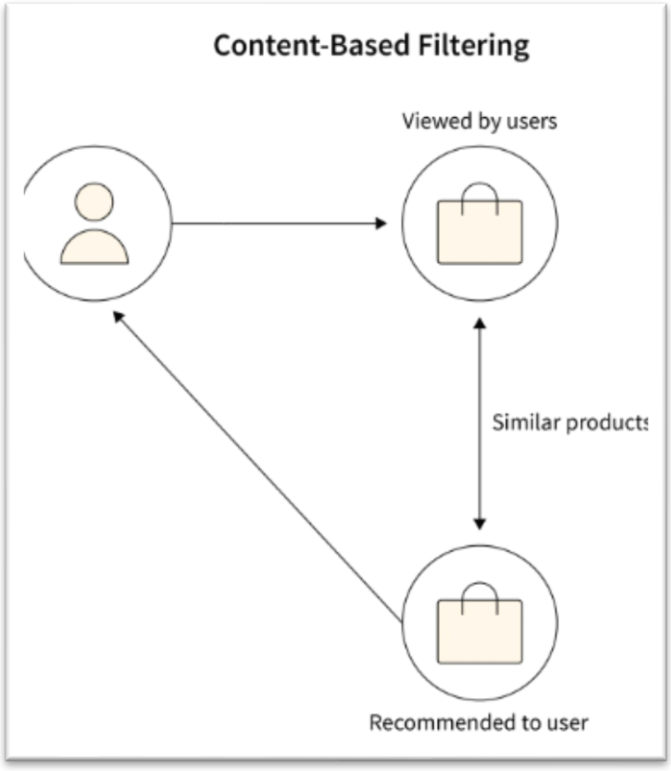

Рисунок 2.2 – Схема фільтрування на основі змісту (з даних[7])

Це може бути здійснено за допомогою різних методів, таких як косинусна схожість для текстового контенту чи евклідова відстань для числових характеристик. Для кожного непереглянутого користувачем контенту розраховується прогнозована оцінка або ймовірність вподобання, використовуючи схожість з відомим контентом. Найвищі прогнозовані оцінки рекомендуються користувачеві для перегляду, покупки або іншої взаємодії.

Фільтрування на основі контенту залишається важливим інструментом для створення ефективних та персоналізованих рекомендацій, особливо в областях з великим обсягом різноманітного контенту.

2.5 Матричні розкладення

Матричні розкладання[8] – це метод у рекомендаційних системах, який базується на розбитті матриці взаємодій користувачів і предметів на декілька менших матриць для визначення схованих властивостей.

Принцип дії матричного розкладання можна побачити на рисунку 2.3.

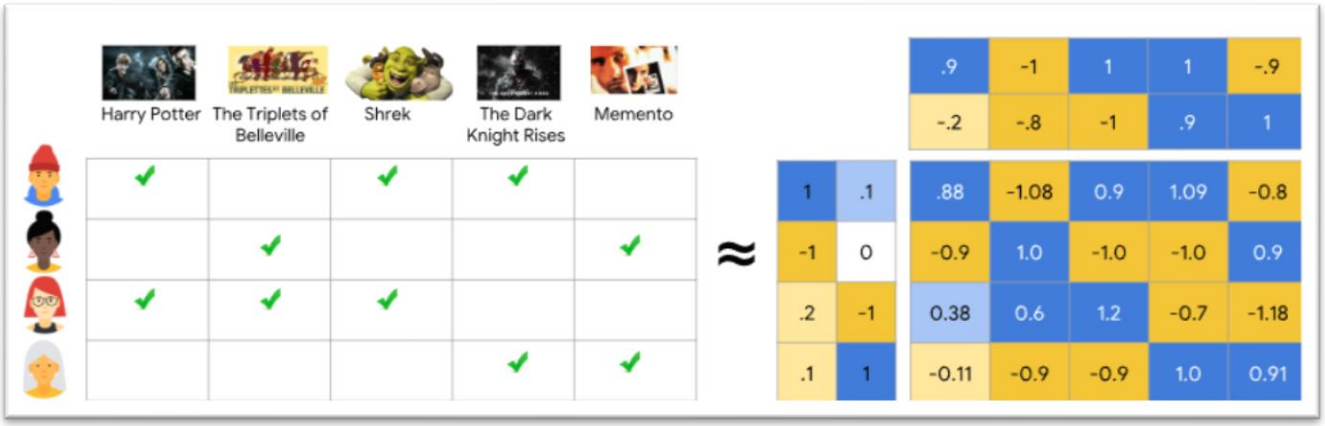

Рисунок 2.3 – Матричні розкладення (з даних[9])

Інформація про взаємодії (наприклад, рейтинги) користувачів з предметами представлена у вигляді матриці. Рядки матриці відповідають користувачам, а стовпці – предметам. Матриця взаємодій розкладається на декілька менших матриць. Зазвичай це робиться з використанням методів оптимізації, таких як градієнтний спуск або стохастичний градієнтний спуск. Кожен розкладений елемент матриці представляє собою деякий «схований фактор». Це може бути

схована властивість, яка асоційована як з користувачем, так і з предметом. Процес розкладання дозволяє отримати прогнозовані рейтинги для тих елементів матриці, які користувач ще не оцінив. Найвищі прогнозовані рейтинги або найбільші схожості використовуються для формування рекомендацій користувачам.

Матричні розкладання залишаються ефективним методом у рекомендаційних системах, особливо при роботі з розрідженими даними та великими обсягами інформації.

2.6 Глибинне навчання

Глибинне навчання[10] – це галузь машинного навчання, яка використовує нейронні мережі з багатьма шарами (глибокими архітектурами) для вирішення завдань, таких як класифікація, регресія, генерація контенту та рекомендації. У рекомендаційних системах глибинне навчання знайшло успішне застосування через здатність автоматично витягувати високорівневі абстракції з складних даних.

Глибинні рекомендаційні системи використовують нейронні мережі з багатьма шарами. Такі мережі можуть включати в себе вступний шар, декілька прихованих шарів і вихідний шар. Дані про взаємодію користувачів із предметами або характеристиками подаються на вхідний шар. Це може бути інформація про перегляди, рейтинги, купівлі тощо. Інформація про взаємодію обробляється внутрішніми шарами мережі, де відбувається витягування абстракцій та формування репрезентацій користувачів та предметів. На виході глибокої мережі отримують прогнозовані оцінки чи ймовірності взаємодії користувачів з предметами. Ці прогнози використовуються для формування персоналізованих рекомендацій.

Глибинне навчання у рекомендаційних системах дозволяє автоматично вивчати складні закономірності в поведінці користувачів та надавати високоефективні персоналізовані рекомендації.

2.7 Застосування NLP

Обробка природної мови (NLP)[10] – це галузь штучного інтелекту, яка займається взаємодією між комп'ютерами та людською мовою. NLP використовується для аналізу, розуміння та генерації тексту на природній мові, що робить його надзвичайно корисним для рекомендаційних систем. У рекомендаційних системах NLP застосовується для обробки текстових даних, таких як відгуки користувачів, описи товарів, статті та інший контент, з метою покращення якості рекомендацій.

Рекомендаційні системи, які використовують NLP, здатні аналізувати текстові дані для витягування важливої інформації та формування релевантних рекомендацій. Це досягається через різні методи, такі як токенізація, стемінг, лематизація, векторизація тексту, а також складніші моделі, такі як вбудовування слів (word embeddings) і трансформери. Застосування цих методів дозволяє системам зрозуміти семантику тексту та знайти приховані закономірності.

3 ПІДГОТОВКА ДО ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ

3.1 Теоретичне порівняння рекомендаційних алгоритмів

Для дослідження потрібно порівняти алгоритми на спільні характеристики, для подальшого розуміння реалізації цих алгоритмів на веб-системі з використанням Angular, C# 12[3] та SQL.

Для цього обираємо такі моделі:

- фільтр заснований на контенті[6] (Content-Based Filtering) – Використовує частоту та важливість термінів для оцінки схожості між ресурсами та інтересами користувача;
- фільтр заснований на спільному використанні[4] (Collaborative Filtering) – Використовує історію взаємодії користувачів для рекомендацій. Використовує моделі для прогнозування взаємодій, такі як матричні розкладення, факторизація матриць та методи машинного навчання;
- матричні розкладення[8] (Matrix Factorization) – Розкладає матриці взаємодій користувачів і ресурсів на менші частини. Використовує ітеративний метод для знаходження матриць факторизації;
- глибинне навчання[10] (Deep Learning) – Використовує глибокі нейронні мережі для з'єднання інформації про користувачів та ресурси. Комбінує глибокий навчання з лінійними моделями для урахування широкого спектру факторів;
- застосування NLP[10] (Natural Language Processing) – Використовується для розуміння семантики текстової інформації та контексту в рекомендаціях.

Далі потрібно обрати критерії за якими можна охарактеризувати та порівняти рекомендаційні моделі. Такими характеристиками будуть:

- точність рекомендацій – Оцінка того, наскільки точні та релевантні рекомендації, надані алгоритмом, відповідають вподобанням та інтересам користувачів;

- повнота рекомендацій – Оцінка того, наскільки алгоритм здатен враховувати всі можливі релевантні ресурси або елементи, які можуть бути цікавими користувачам;
- швидкість прогнозування – Вимірює час, необхідний для генерації рекомендацій. Це важливо для реальних систем, де швидкість відгуку грає ключову роль;
- масштабованість – Оцінка того, як добре алгоритм працює з великим обсягом даних та збільшується при збільшенні кількості користувачів та ресурсів;
- споживач ресурсів – Оцінка кількості обчислювальних та інших ресурсів, які вимагаються для роботи алгоритму.

Тепер характеризуємо обрані рекомендаційні моделі за критеріями в таблицях Д.1-Д.2.

Теоретично оцінюємо кожну характеристику алгоритму за 5ти бальною шкалою, де 0 – найгірша оцінка, а 5 – найкраща. Занесемо дані до таблиці 3.1

Таблиця 3.1 – Оцінки алгоритмів за характеристиками (таблиця виконана самостійно)

	Точність рекомендацій	Повнота рекомендацій	Швидкість прогнозування	Масштабованість	Споживач ресурсів
Фільтр заснований на контенті	5	3	4	3	4
Фільтр заснований на спільному використанні	3	4	3	3	3
Матричні розкладення	4	3	3	3	3
Глибинне навчання	5	4	2	3	2
Застосування NLP	4	3	3	3	3

Наступний крок – будемо використовувати лінійну адитивну згортку з ваговими коефіцієнтами, тому що деякі з окремих критеріїв є більш важливі за інші. Тому надамо кожному критерію свій коефіцієнт значення для підрахунків:

- точність рекомендацій – 0.4;
- повнота рекомендацій – 0.15;
- швидкість прогнозування – 0.3;
- масштабованість – 0.05;
- споживач ресурсів – 0.1.

Найбільший коефіцієнт має критерій – «Точність рекомендацій», адже від нього залежить результат рекомендації, чого у майбутній системі намагаємось достигнути.

Далі розраховуємо найбільш підходящий алгоритм за формулою 3.1 Згорткової оптимізаційної моделі[11] :

$$Z^* = \max_{i=1,m} \sum_{j=1}^{n} \alpha_j \beta_j a_{ij} \tag{3.1}$$

де α_j – нормуючі множники,

β_j – вагові коефіцієнти,

a_{ij} – значення характеристики.

Результати розрахунків заносимо в таблицю 3.2

За результати отримали оцінку кожного алгоритму:

- фільтр заснований на контенті – 0,238;
- фільтр заснований на спільному використанні – 0,182;
- матричні розкладення – 0,193;
- глибинне навчання – 0,194;
- застосування NLP – 0,193.

За результатами було отримано те що, найбільш підходящий алгоритм це – Фільтр заснований на контенті з результатом 0,238.

Потрібно мати програмний продукт пов'язаний з освітою, наприклад якась соц. мережа, де можна використати рекомендаційні системи, де можна проаналізувати та перевірити теоретичні дані.

Таблиця 3.2 – Результати розрахунків (таблиця виконана самостійно)

	Точність рекомендацій	Повнота рекомендацій	Швидкість прогнозування	Масштабованість	Споживач ресурсів	Z*
Фільтр заснований на контенті	5	3	4	3	4	0,238
Фільтр заснований на спільному використанні	3	4	3	3	3	0,182
Матричні розкладення	4	3	3	3	3	0,193
Глибинне навчання	5	4	2	3	2	0,194
Застосування NLP	4	3	3	3	3	0,193
β	0,4	0,15	0,3	0,05	0,1	
α	0,048	0,059	0,067	0,067	0,067	

А для отримання результатів – використовувати такі речі, як: збір відгуків користувачів, аналіз метрик ефективності, тестування на специфічних випадках та використати зіставлення з альтернативами для отримання позитивного або негативного результату використання рекомендаційних систем у сфері освіти, побудова пояснень для рекомендаційних систем на основі часової динаміки користувацьких вподобань[12], за для оцінки динаміки та якості змін результатів рекомендацій на основі останніх користувацьких вподобань.

4 ОПИС ЗАСТОСУНКУ

4.1 Опис функціоналу

Розробляємий веб-застосунок має низку функцій, спрямованих на покращення користувацького досвіду та забезпечення зручності користування.

Перш за все, веб-застосунок дозволяє користувачам створювати нові рівні за допомогою зручного редактора, що дозволяє додавати необхідну інформацію та 3D моделі.

Користувачі можуть фільтрувати рівні за різними критеріями, такими як тема, складність, кількість коментарів та інші.

Система також автоматично пропонує користувачеві рівні на основі їхніх попередніх виборів та результатів проходження.

Користувачі можуть залишати відгуки та реакції на рівні або коментарі інших користувачів, а також обговорювати рівні, залишаючи коментарі під ними.

Після завершення рівня користувачі можуть переглянути свої результати та помилки для подальшого вдосконалення.

Система відстежує рейтинги користувачів та рівнів для стимулювання досягнень та визначення найпопулярніших рівнів.

Користувачі мають обмежену кількість енергії, яка використовується для доступу до рівнів та інших можливостей, а також можуть отримати додаткову енергію безкоштовно за виконання певних завдань.

Застосунок надає можливість придбати підписку, яка дозволяє підвищити рівень енергії та отримати доступ до додаткових можливостей.

Адміністратори мають доступ до панелі управління для контролю за роботою системи, блокування користувачів або видалення некоректного контенту.

Розробка веб-застосунку відображає важливість інноваційних методів навчання та використання рекомендаційних систем у сфері освіти. Цей проект не лише дозволяє користувачам зануритися в історичні події через цікаві рівні та 3D моделі, а й надає можливості для саморозвитку та взаємодії. Завдяки функціональності, що описана вище, користувачі зможуть не лише отримувати

нові знання, але й ділитися ними, створювати власний контент та спілкуватися з іншими учасниками спільноти. Разом з тим, система керування енергією та рейтингова система дозволять забезпечити стійкий інтерес користувачів та створити динамічне середовище для навчання та розвитку.

4.2 Архітектура застосунку

На рисунку 4.1, можна побачити UML діаграму розгортання веб-додатку для підтримки спільноти для спільного вивчення історії.

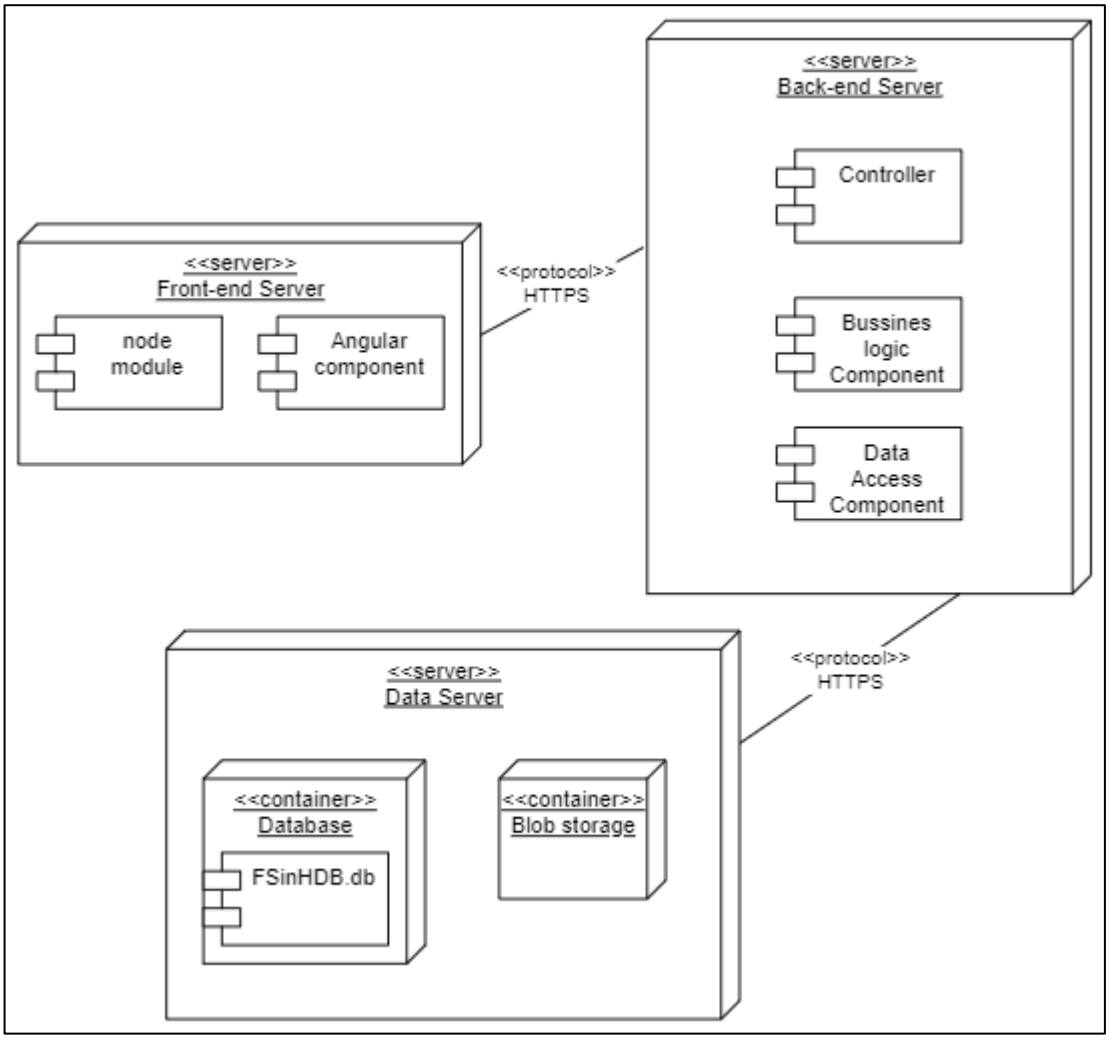

Рисунок 4.1 – Діаграма розгортання (рисунок створено самостійно)

На діаграмі розгортання (див. рис. 4.1) показана взаємодія компонентів системи між собою.

Система складається з таких частин:

– база даних;

– серверна частина;

– клієнтська частина;

– сховище файлів.

Клієнтська частина містить компоненти Angular та додаткові модулі з бібліотек. Вона взаємодіє з серверною частиною через протокол HTTPS, де відбуваються складні операції та робота з даними. Серверна частина комунікується з сервером бази даних також за допомогою протоколу HTTPS. На сервері бази даних розміщено два сховища: одне для даних, інше – для великих файлів, таких як 3D моделі.

На рисунку 4.2, можна побачити UML діаграму компонентів серверної частини додатку.

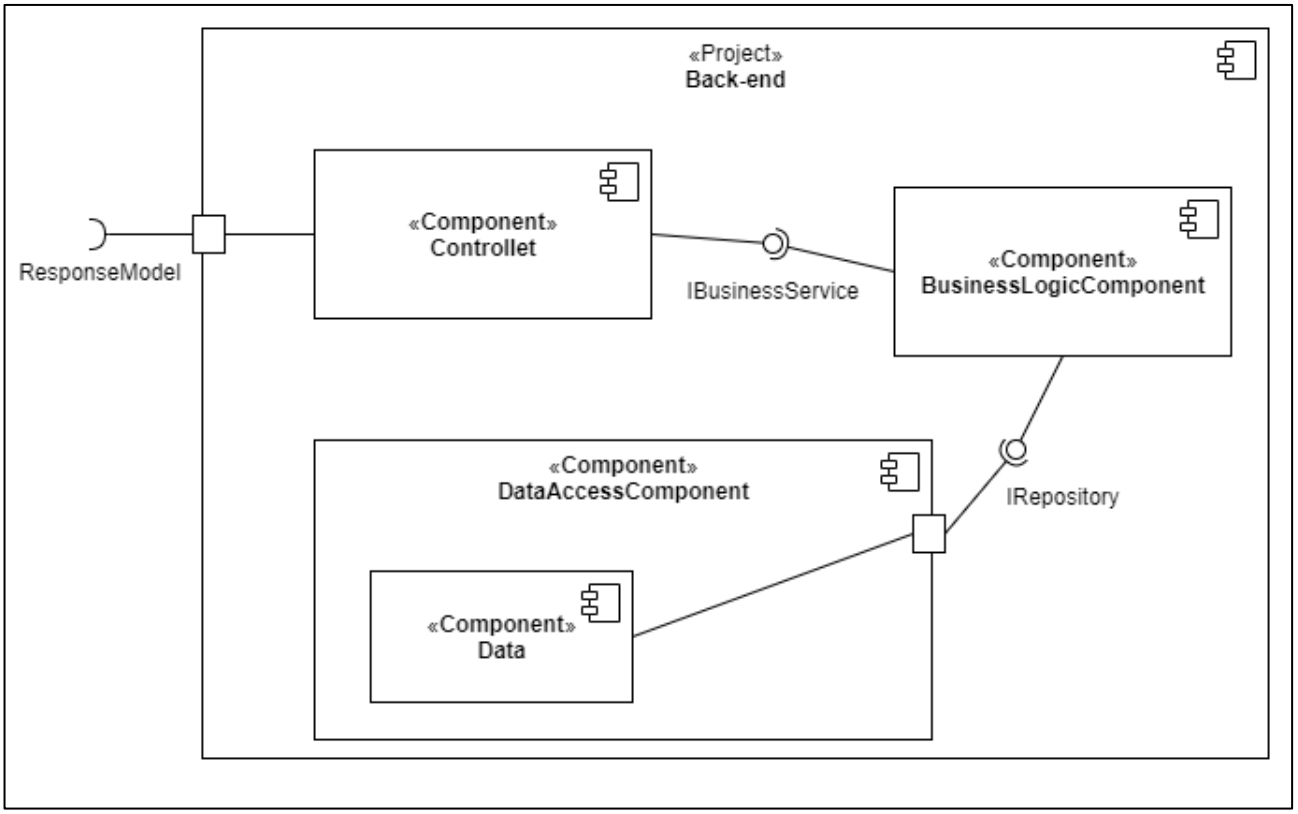

Рисунок 4.2 – Діаграма компонентів серверної частини (рисунок створено самостійно)

На діаграмі компонентів серверної частини (див. рис. 5.2) представлена взаємодія компонентів між собою на рівні сервера.

Компоненти рівня даних виконують інтерфейс IRepository для доступу до даних безпосередньо або через певну обробку, що використовується бізнес-логікою додатку.

Компонент, що відповідає за бізнес-логіку, отримує дані через інтерфейс IRepository та реалізує інтерфейс IBusinessService, який містить необхідні методи для роботи логіки додатку.

Контролер координує запитами з клієнтської частини та для цього використовує сервіси з компоненту бізнес-логіки. Після обробки контролер повертає ResponseModel, який отримує клієнтська частина.

На рисунку 4.3, можна побачити UML діаграму компонентів клієнтської частини додатку.

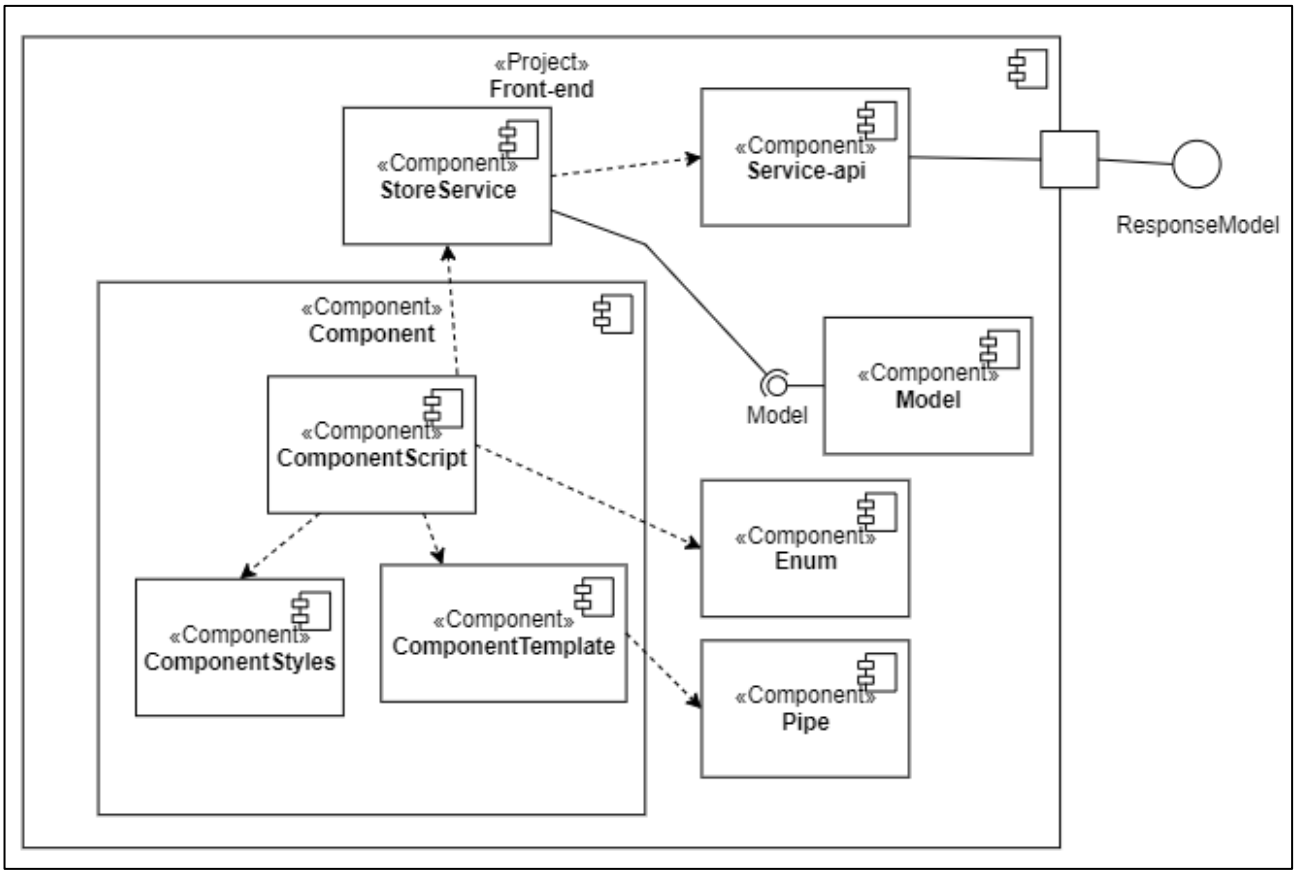

Рисунок 4.3 – Діаграма компонентів клієнтської частини (рисунок створено самостійно)

На діаграмі компонентів клієнтської частини (див. рис. 4.3) представлена взаємодія компонентів між собою на рівні клієнта.

Після отримання відповіді з серверної частини, її обробляє компонент ServiceApi.

Компоненти StoreService викликають або оброблюють відповіді з ServiceApi, керують моделями системи для збереження та кешування їх, а також слідкують за змінами цих моделей в системі та відповідним чином їх обробляють.

Основні компоненти клієнтської частини використовують StoreService для слухання даних з клієнтської частини. Кожен компонент має скрипт, що виконує логіку обробки даних, стилі та власний шаблон зовнішнього вигляду.

На рисунку 4.4, можна побачити UML діаграму пакетів серверної частини додатку.

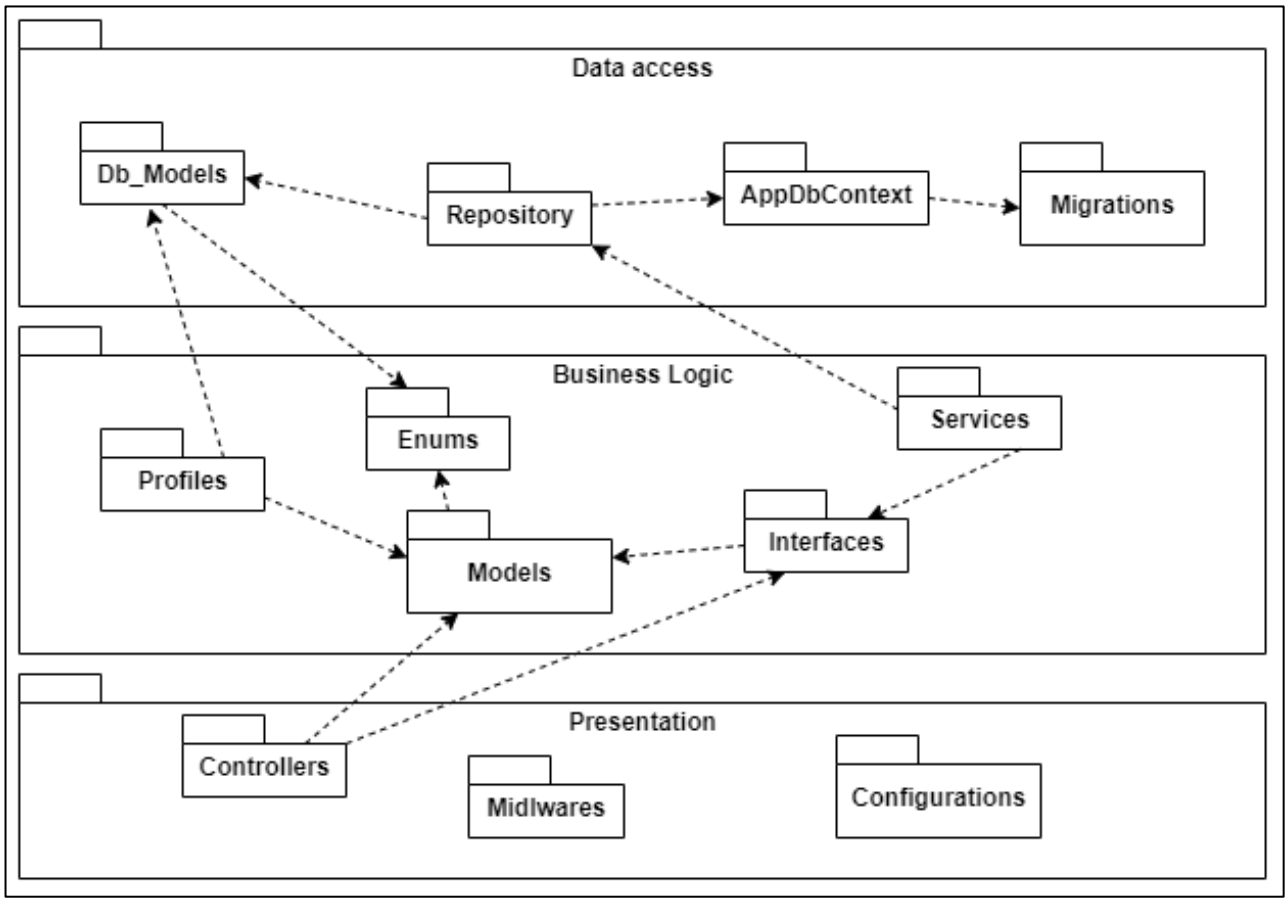

Рисунок 4.4 – Діаграма пакетів серверної частини (рисунок створено самостійно)

На діаграмі пакетів серверної частини (див. рис. 4.4) показано розташування файлів у проекті та їх взаємозв'язки.

Пакети організовані за шарами, починаючи від доступу до даних і до інших, де лише моделі, що змінюються відповідно до профілів маппінгу та репозиторіїв для обробки даних.

На рівні бізнес-логіки розміщені основні та змінені моделі, які використовуються клієнтською частиною, а також інтерфейси логіки, що викликаються у контролерах.

Система розглядається як трьохрівнева.

На першому рівні розташована база даних, яка включає в себе MSSQL Server для зберігання даних та BlobStorage для файлів. Ці компоненти розміщені на окремому сервері.

Другий рівень включає серверну частину, яка відповідає за обчислення, бізнес-логіку та періодичні розрахунки, а також використовує базу даних. Архітектура серверної частини побудована на ASP.NET і реалізує трьохшарову структуру. Перший шар back-end – PresentationLayer, взаємодіє з клієнтською частиною через контролери та активує потрібний функціонал із шару BusinessLogic. Другий шар – BusinessLogicLayer, забезпечує функціональність бізнес-логіки та обробку запитів від інших частин системи, включаючи мапінг моделей та інфраструктуру додатку. Він використовує шар роботи з даними. Третій шар – DataAccessLayer, відповідає за роботу з даними та запити до бази даних MSSQL Server та Blob Storage, використовуючи Entity Framework та паттерн Repository. Цей рівень також відповідає за міграції та постійне підключення до бази даних.

Серверна частина розташована на окремому сервері. На третьому рівні знаходиться web-частина, яка показує користувацький інтерфейс та взаємодіє з серверною частиною для обробки та відображення даних. Web-частина також розміщена на окремому сервері і доступна для користувачів через веб-браузер.

4.3 Структура зберігання даних

На рисунку 4.5 представлена ER-діаграма бази даних, яка є необхідною для роботи всієї логіки програмного застосунку. База даних складається з різних

таблиць, які описують різні аспекти системи. Наприклад, «User» містить інформацію про користувачів системи та їх ролі, «Token» зберігає ключі безпеки користувачів, «SubscribersHistory» фіксує платні операції та підписки, «Category» та «Location» описують категорії та локації подій постів відповідно. Крім того, є таблиці для зберігання даних про самі пости «Post», моделі «Model», реакції користувачів до постів «PassPost» та «CommentReaction», питань «Question» та відповідей «Answer», результатів користувачів «QuestionResult» та коментарів «Comment», а також таблиці «Complaint» для зберігання скарг та «Configuration» для налаштувань додатку. Кожна з цих таблиць має свою власну структуру та взаємозв'язки з іншими таблицями, що забезпечує надійну інтеграцію та зручне управління даними в системі.

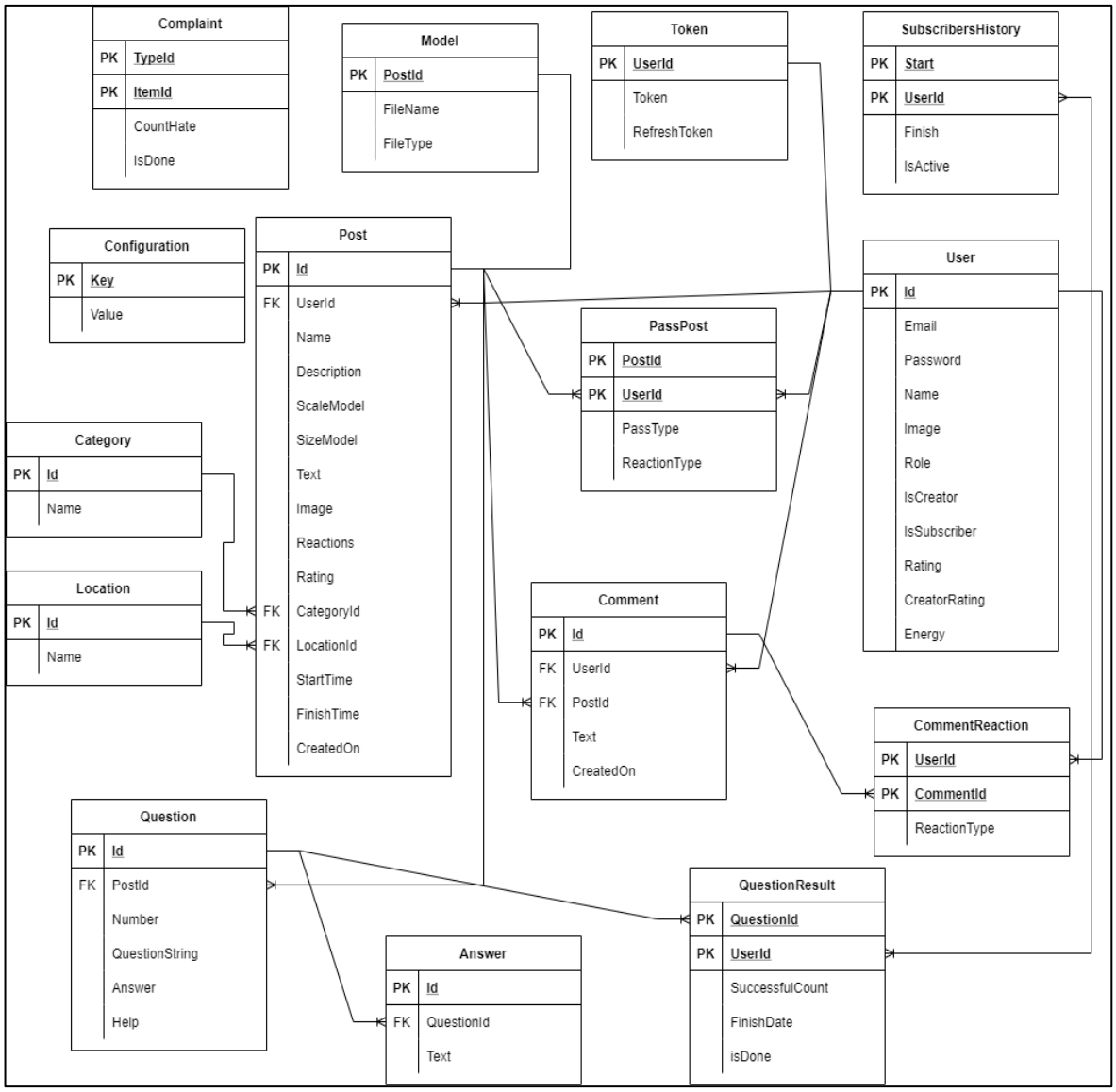

Рисунок 4.5 – ER-діаграма бази даних (рисунок створено самостійно)

Ця база даних має таку організацію, що дозволяє ефективно виконувати стратегічні запити для бізнесу та підтримувати функції спільного вивчення історії в рамках системи. Для такої бази буде зручне інтегрування різних рекомендаційних систем, які будуть рекомендувати пости для користувачів.

4.4 Розробка UI/UX дизайну системи

Для розробки інтерфейсу користувача використовувалися додаткові інструменти проектування UI у Figma.

На рисунку 4.6 можна побачити приклад форми.

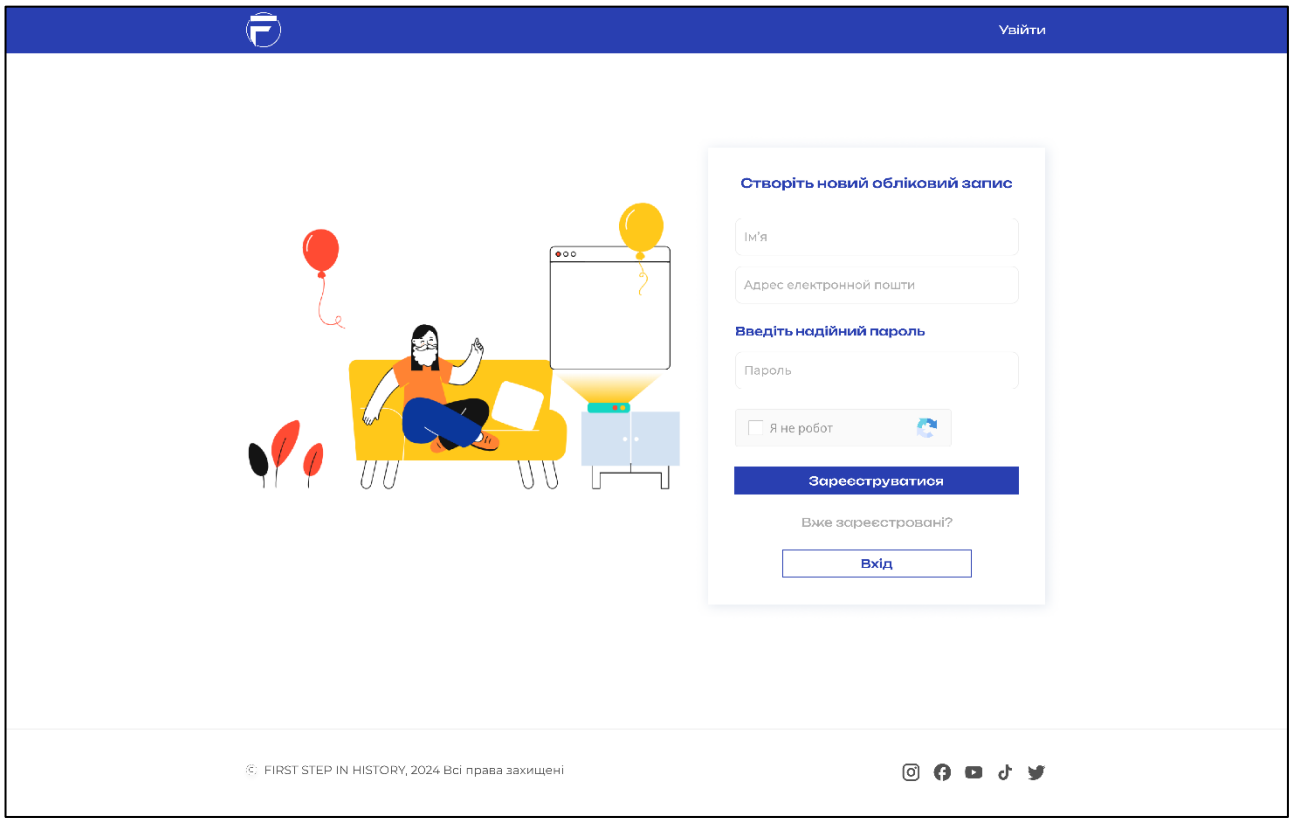

Рисунок 4.6 – Форма реєстрації додатку (рисунок створено самостійно)

Наприклад, на формі (зображення на рис. 4.6) елементи були розміщені один за одним з певним інтервалом. Кожне поле мало валідацію, яка автоматично перевіряла введені дані та повідомляла користувача про помилки. Будь-які проблеми на клієнтській частині або конфлікти з сервером відображалися знизу, щоб користувач мав зрозумілу інформацію про те, що сталося. На головному

інтерфейсі додатку (зображення на рис. 4.7) у хедері був розміщений логотип додатку та основний функціонал, доступний з будь-якої сторінки.

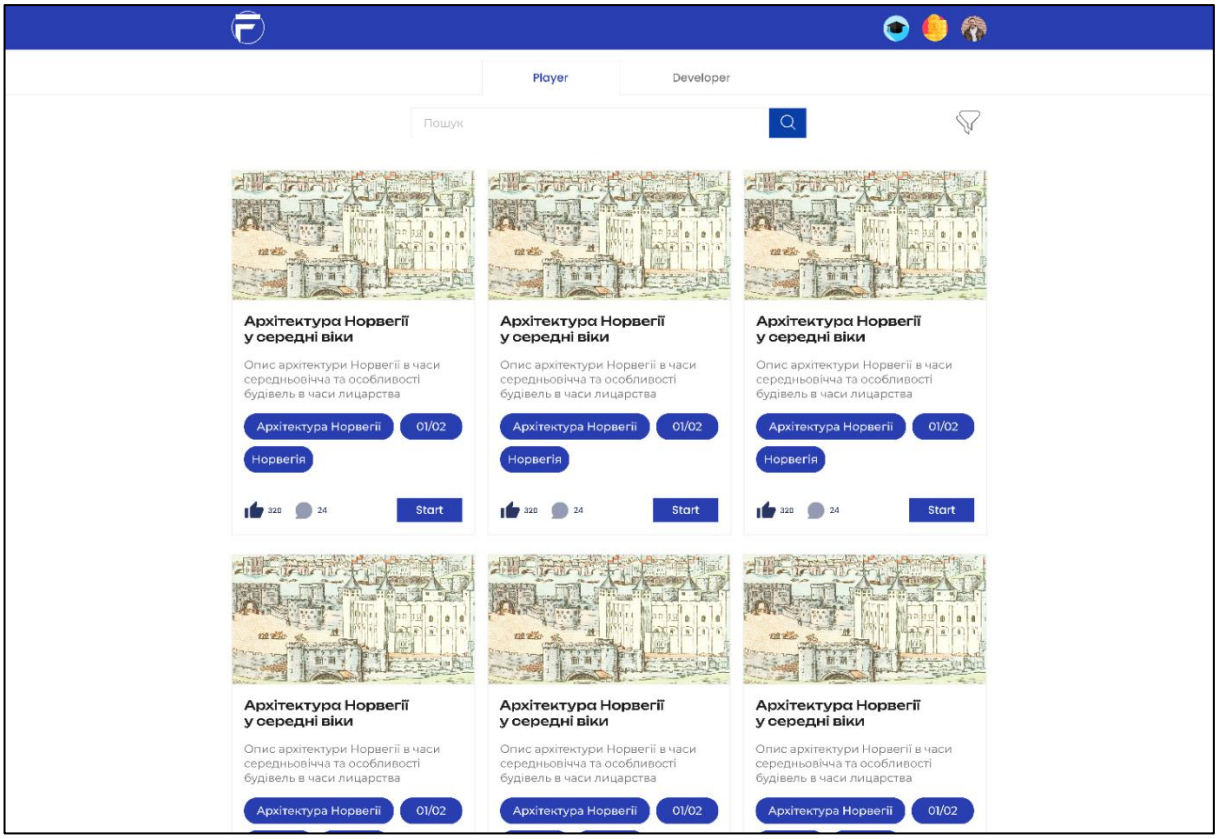

Рисунок 4.7 – Сторінка усіх постів додатку (рисунок створено самостійно)

Таби для переключення між різними функціями, такими як «Користувач» та «Розробник», дозволяли зручно управляти інтерфейсом. На вкладці «Користувач» був доступний функціонал для фільтрації постів, який знаходився у згорнутому вигляді для економії місця. Інформація про користувача, така як ім'я, зображення та рівень енергії, також відображалася у хедері, разом з повідомленнями для користувачів, наприклад, готовими тестами для самоперевірки. На вкладці «Розробник» був доступний функціонал для створення постів.

Ці пости можна використовувати під час дослідження рекомендаційних алгоритмів як основні джерела контенту для аналізу та вибору оптимальних методів рекомендацій.

5 РЕАЛІЗАЦІЯ АЛГОРИТМІВ

5.1 Реалізація колаборативного фільтрування

Першим алгоритмом розберемо колаборативне фільтрування[4]. Для цього на самому початку розберемо які дані будемо використовувати для рекомендацій.

На рисунку 5.1 зображена частина схеми бази даних яку будемо використовувати в алгоритмах.

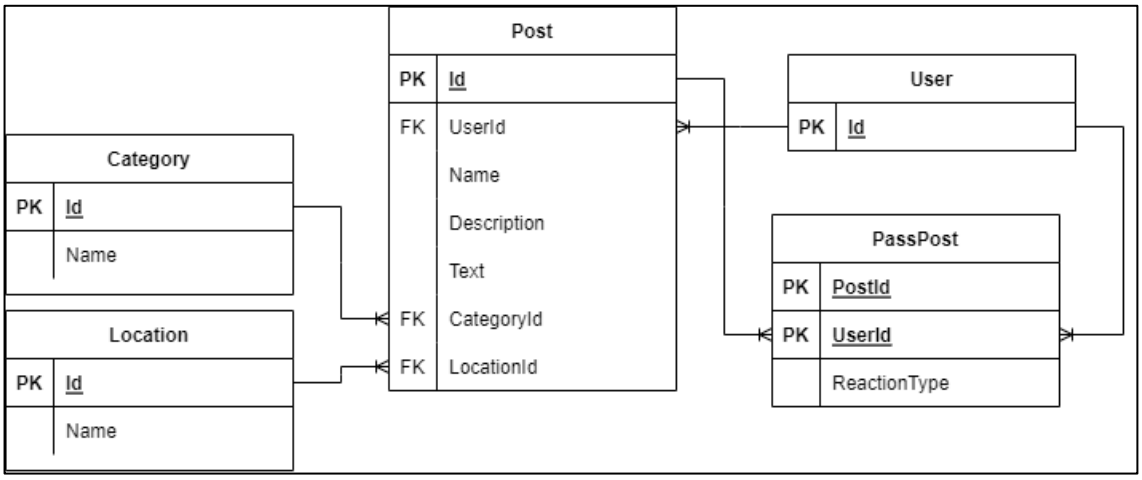

Рисунок 5.1 – Частина схеми бази даних для рекомендацій (рисунок створено самостійно)

Таким чином, для рекомендацій будуть використовуватись користувачі, публікації та реакції користувачів на ці публікації. З даних постів будемо використовувати:

- назва;
- опис;
- текст;
- категорія;
- локація.

Використовуючи такі дані можна реалізовувати алгоритми, які будуть на основі їх створювати рекомендації для певного користувача або користувачів.

Метод колаборативного фільтрування[4] використовується для надання персоналізованих рекомендацій користувачам на основі їхніх попередніх взаємодій з публікаціями В основі цього методу лежить ідея про те, що

користувачі, які мають схожі смаки або вподобання, склонні сподіватися на ті ж самі предмети. Для цього створюємо код (див. рис. 5.2) для колаборативного фільтрування.

```
public async Task<IEnumerable<PostItem>> CollaborativeFiltering(int userId)
{
    Dictionary<int, Dictionary<int, double>> BuildUserSimilarityMatrix(IEnumerable<PassPost> ratings)...

    var passPosts = await _repository.GetRangeAsync<PassPost>(false, x => true);
    var userRatings = passPosts.Where(r => r.UserId == userId).ToDictionary(r => r.PostId, r => r.ReactionType);
    var posts = passPosts.Select(r => r.PostId).Distinct().ToList();
    var userSimilarityMatrix = BuildUserSimilarityMatrix(passPosts);

    var weightedScores = new Dictionary<int, double>();
    var similaritySums = new Dictionary<int, double>();

    foreach (var user2 in userSimilarityMatrix[userId].Keys)
    {
        foreach (var post in posts)
        {
            if (!userRatings.ContainsKey(post))
            {
                if (!weightedScores.ContainsKey(post))
                {
                    weightedScores[post] = 0;
                    similaritySums[post] = 0;
                }

                var rating = GetDoubleFromReactionTypeEnum(
                    passPosts.FirstOrDefault(r => r.UserId == user2 && r.PostId == post)?.ReactionType ?? 0
                );
                weightedScores[post] += userSimilarityMatrix[userId][user2] * rating;
                similaritySums[post] += userSimilarityMatrix[userId][user2];
            }
        }
    }

    var recommendations = weightedScores.Select(ws => new
    {
        PostId = ws.Key,
        Score = similaritySums[ws.Key] > 0 ? ws.Value / similaritySums[ws.Key] : 0
    })
        .OrderByDescending(ws => ws.Score)
        .Take(10)
        .Select(ws => ws.PostId)
        .ToList();

    var allPosts = await _repository.GetRangeAsync<Post>(false, x => recommendations.Contains(x.Id));

    IEnumerable<PostItem> filteredPosts = _mapper.Map<IEnumerable<PostItem>>(allPosts);

    return filteredPosts;
}
```

Рисунок 5.2 – Код колаборативного фільтрування (рисунок створено самостійно)

Для розрахунків рекомендацій на самому початку достаються усі зв’язки користувача та публікації, для взяття відгуку.

Використовуючи набір даних створюється матриця схожості, це відбувається в методі BuildUserSimilarityMatrix на рисунку 5.3.

```
Dictionary<int, Dictionary<int, double>> BuildUserSimilarityMatrix(IEnumerable<PassPost> ratings)
{
    var userRatings = ratings.GroupBy(r => r.UserId)
                             .ToDictionary(g => g.Key, g => g.ToDictionary(r => r.PostId, r => r.ReactionType));
    var similarityMatrix = new Dictionary<int, Dictionary<int, double>>();

    foreach (var user1 in userRatings.Keys)
    {
        similarityMatrix[user1] = new Dictionary<int, double>();
        foreach (var user2 in userRatings.Keys)
        {
            if (user1 != user2)
            {
                similarityMatrix[user1][user2] = CalculateCosineSimilarity(userRatings[user1], userRatings[user2]);
            }
        }
    }

    return similarityMatrix;
}
```

Рисунок 5.3 – Код BuildUserSimilarityMatrix (рисунок створено самостійно)

В цьому методі для кожного користувача розраховується схожість з іншими користувачами на основі їхніх взаємодій з публікаціями. В нашому випадку схожість розраховується завдяки косинусной схожості (див. рис. 5.4).

```
private double CalculateCosineSimilarity(Dictionary<int, ReactionTypeEnum> ratings1, Dictionary<int, ReactionTypeEnum> ratings2)
{
    double dotProduct = 0;
    double magnitude1 = 0;
    double magnitude2 = 0;

    foreach (var item in ratings1.Keys)
    {
        if (ratings2.ContainsKey(item))
        {
            dotProduct += GetDoubleFromReactionTypeEnum(ratings1[item]) * GetDoubleFromReactionTypeEnum(ratings2[item]);
        }
        magnitude1 += Math.Pow(GetDoubleFromReactionTypeEnum(ratings1[item]), 2);
    }

    foreach (var rating in ratings2.Values)
    {
        magnitude2 += Math.Pow(GetDoubleFromReactionTypeEnum(rating), 2);
    }

    magnitude1 = Math.Sqrt(magnitude1);
    magnitude2 = Math.Sqrt(magnitude2);

    if (magnitude1 == 0 || magnitude2 == 0)
    {
        return 0;
    }

    return dotProduct / (magnitude1 * magnitude2);
}
```

Рисунок 5.4 – Код CalculateCosineSimilarity (рисунок створено самостійно)

CalculateCosineSimilarity розраховує схожість двох користувачів за формулою (5.1).

В результаті отримаємо словник з користувачем, та ще одним словником – інший користувач з його схожістю.

Використовуємо користувацьку схожість, бо можлива схожість на основі публікації, та це була б просто обернена матриця. Для нашого застосунку підходить більше схожість між користувачами.

$$\text{similarity} = \frac{\sum_{i=1}^{n} r_{1i} r_{2i}}{\sqrt{\sum_{i=1}^{n} r_{1i}^{2}} * \sqrt{\sum_{i=1}^{n} r_{2i}^{2}}} \quad (5.1)$$

де r_{1i} – оцінки (реакції) користувача 1,

r_{2i} – оцінки (реакції) користувача 2.

Після цього для кожної публікації, з яким користувач ще не взаємодіяв, розраховується прогнозований рейтинг на основі схожості з іншими користувачами, які взаємодіяли з цим постом. Достаємо необхідну кількість публікацій з самим високим розрахованим коефіцієнтом схожості. Достаємо публікації з бази даних та перероблюємо з моделі бази даних на модель клієнтської частини.

5.2 Реалізація фільтрування на основі змісту

Метод фільтрування на основі змісту[6] використовую для своїх рекомендацій різних критерій публікації. В основі цього методу лежить шукання схожості між декількома об’єктами на основі їх схожості один на одній. Для цього створюємо код (див. рис. 5.5) для фільтрування на основі вмісту.

В цьому методі на самому початку також беруться відгуки користувачів, для користувача, для якого ми шукаємо рекомендації створюємо профіль завдяки методу BuildUserProfile, де на основі минулих ревізій користувача створюється профіль з такими плями як:

- CategoryScores – схожість за категоріями;
- LocationScores – схожість за локаціями;
- WordScores – схожість за контентом у тексті публікації.

```
public async Task<IEnumerable<PostItem>> ContentBased(int userId)
{
    ContentBasedUserProfile BuildUserProfile(int userId, IEnumerable<PassPost> passPosts)...

    Dictionary<string, double> BuildItemProfile(Post post)...

    var passPosts = await _repository.GetRangeAsync<PassPost>(false, x => true, x => x.Post);

    var userProfile = BuildUserProfile(userId, passPosts);

    var userPosts = passPosts.Where(c => c.UserId == userId).Select(x => x.PostId).ToArray();

    var posts = await _repository.GetRangeAsync<Post>(false, x => x.UserId != userId && !x.Id.Contains(userPosts));

    var recommendations = new List<KeyValuePair<int, double>>();

    foreach (var post in posts)
    {
        var itemProfile = BuildItemProfile(post);
        var similarity = CalculateCosineSimilarity(
            userProfile.CategoryScores
                .Concat(userProfile.LocationScores)
                .Concat(userProfile.WordScores)
                .ToDictionary(k => k.Key, v => v.Value),
            itemProfile
        );
        recommendations.Add(new KeyValuePair<int, double>(post.Id, similarity));
    }

    var recommendationPostIds = recommendations.OrderByDescending(r => r.Value)
                            .Take(10)
                            .Select(r => r.Key)
                            .ToList();

    var allPosts = posts.Where(x => x.Id.Contains(recommendationPostIds));

    IEnumerable<PostItem> filteredPosts = _mapper.Map<IEnumerable<PostItem>>(allPosts);

    return filteredPosts;
}
```

Рисунок 5.5 – Код фільтрування на основі вмісту (рисунок створено самостійно)

```
ContentBasedUserProfile BuildUserProfile(int userId, IEnumerable<PassPost> passPosts)
{
    var userProfile = new ContentBasedUserProfile { UserId = userId };

    var userPasses = passPosts.Where(c => c.UserId == userId).ToList();

    foreach (var userPass in userPasses)
    {
        var post = userPass.Post;
        var categoryName = $"category_{post.CategoryId}";
        var locationName = $"location_{post.LocationId}";

        if (!userProfile.CategoryScores.ContainsKey(categoryName))
        {
            userProfile.CategoryScores[categoryName] = 0;
        }
        userProfile.CategoryScores[categoryName] += userPass.ReactionType == ReactionTypeEnum.Like ? 1 : -1;

        if (!userProfile.LocationScores.ContainsKey(locationName))
        {
            userProfile.LocationScores[locationName] = 0;
        }
        userProfile.LocationScores[locationName] += userPass.ReactionType == ReactionTypeEnum.Like ? 1 : -1;

        var words = post.Text.Split(' ');
        foreach (var word in words)
        {
            if (!userProfile.WordScores.ContainsKey(word))
            {
                userProfile.WordScores[word] = 0;
            }
            userProfile.WordScores[word] += userPass.ReactionType == ReactionTypeEnum.Like ? 1 : -1;
        }
    }

    return userProfile;
}
```

Рисунок 5.6 – Код BuildUserProfile (рисунок створено самостійно)

Коли є профіль користувача, достаються публікації, які користувач не проходив, створюється рекомендація. Для кожної публікації ствоюється профіль як у методі BuildUserProfile (див. рис. 5.6) але тільки для однієї публікації.

Після створення профілю публікації – розраховується косинусна схожість між профілем користувача та публікацією (див. рис. 5.4) за формулою 5.1.

На остаток, результат фільтрується з значенням, достаються рекомендовані публікації та відправляються на клієнт

5.3 Реалізація матричного розкладення

Метод матричного розкладання[8] також надає рекомендації за матричним розкладанням. Замість прямого порівняння між користувачами або публікаціями, як це робиться у колаборативному фільтруванні, метод матричного розкладання розглядає ці взаємодії через лінійну комбінацію складових. Для цього створюємо код (див. рис. 5.7) для матричного розкладання.

```
public async Task<IEnumerable<PostItem>> MatrixExpansion(int userId)
{
    var passPosts = await _repository.GetRangeAsync<PassPost>(false, x => true);
    int numUsers = passPosts.Select(x => x.UserId).Distinct().Count();
    int numPosts = passPosts.Select(x => x.PostId).Distinct().Count();

    int i = 0;
    Dictionary<int, int> userMapping = passPosts
        .Select(x => x.UserId).Distinct()
        .Select(x => new { UserId = x, Index = i++ }).ToDictionary(x => x.UserId, x => x.Index);
    i = 0;
    Dictionary<int, int> postMapping = passPosts
        .Select(x => x.PostId).Distinct()
        .Select(x => new { PostId = x, Index = i++ }).ToDictionary(x => x.PostId, x => x.Index);

    double[,] CreateRatingMatrix(IEnumerable<PassPost> passPosts, int numUsers, int numPosts)...

    (Matrix<double> U, Vector<double> Sigma, Matrix<double> Vt) PerformSVD(double[,] ratingMatrix)...

    double[,] PredictRatings(Matrix<double> U, Vector<double> Sigma, Matrix<double> Vt)...

    double[] GetRow(double[,] matrix, int rowIndex)...

    var ratingMatrix = CreateRatingMatrix(passPosts, numUsers, numPosts);
    var (U, Sigma, Vt) = PerformSVD(ratingMatrix);
    var predictedRatings = PredictRatings(U, Sigma, Vt);

    var userRatings = GetRow(predictedRatings, userMapping.GetValueOrDefault(userId));
    var recommendations = userRatings
        .Select((rating, index) => new { PostId = index + 1, Rating = rating })
        .OrderByDescending(r => r.Rating)
        .Take(10)
        .Select(r => r.PostId)
        .ToList();

    var allPosts = await _repository.GetRangeAsync<Post>(false, x => recommendations.Contains(x.Id));

    IEnumerable<PostItem> filteredPosts = _mapper.Map<IEnumerable<PostItem>>(allPosts);

    return filteredPosts;
}
```

Рисунок 5.7 – Код матричного розкладання (рисунок створено самостійно)

В цьому методі також використовуються взаємодії між публікаціями та користувачами.

Створюється матриця взаємодії між користувачами, де 1 – позитивна реакція, 0 – пуста, -1 – негативна.

Після цього матриця розкладається на три матриці:

- матриця U – містить ліві сингулярні вектори та представляє лінійні комбінації користувачів та їх взаємодій;
- вектор Sigma – містить сингулярні числа, що вказують на важливість кожного фактора у розкладі;
- матриця Vt – містить праві сингулярні вектори.

Це було виконано завдяки бібліотеці MathNet. Ці матриці використовуються для прогнозування рейтингів для конкретного користувача. Множенням матриць отримаємо прогнозоване значення матрицю користувачі, публікацій та значення схожості. Достаємо значення потрібного нам користувача, достаємо найрекомендованіші публікації та відправляємо їх на клієнт.

5.4 Реалізація глибинного навчання

Метод глибинного навчання[10] використовується для надання персоналізованих рекомендацій користувачам на основі аналізу складних взаємодій з публікаціями за допомогою нейронних мереж. Цей метод здатний автоматично витягувати високорівневі абстракції з даних, що дозволяє робити точні прогнози та надавати ефективні рекомендації. Для цього створюємо код (див. рис. 5.8) для матричного розкладання.

Метод глибинного навчання використовує нейронні мережі для прогнозування взаємодій користувачів з публікаціями. Спочатку завантажуються дані про взаємодії користувачів з публікаціями та перетворюються у формат, придатний для навчання моделі. RatingData в якій зберігається:

- поле UserId – ідентифікатор користувача;
- поле PostId – ідентифікатор публікації;
- поле Reaction – реакція користувача на дані.

```
public async Task<IEnumerable<PostItem>> DeepLearning(int userId)
{
    ITransformer TrainModel(MLContext mlContext, IEnumerable<PassPost> data)...

    var mlContext = new MLContext();

    var passPosts = await _repository.GetRangeAsync<PassPost>(false, x => true);

    var model = TrainModel(mlContext, passPosts);

    var predictionEngine = mlContext.Model.CreatePredictionEngine<RatingData, Prediction>(model);

    var posts = passPosts.Select(x => x.PostId).Distinct().ToList();

    var recommendations = posts.Select(postId => new
    {
        PostId = postId,
        Score = predictionEngine.Predict(new RatingData { UserId = userId, PostId = postId }).Score
    })
        .OrderByDescending(x => x.Score)
        .Take(10)
        .Select(x => x.PostId)
        .ToList();

    var allPosts = await _repository.GetRangeAsync<Post>(false, x => recommendations.Contains(x.Id));

    IEnumerable<PostItem> filteredPosts = _mapper.Map<IEnumerable<PostItem>>(allPosts);

    return filteredPosts;
}
```

Рисунок 5.8 – Код глибинного навчання (рисунок створено самостійно)

Навчаємо модель використовуючи конвеєр (pipeline), що включає перетворення значень у ключі та тренування моделі на основі матричної факторизації (див. рис. 5.9).

```
ITransformer TrainModel(MLContext mlContext, IEnumerable<PassPost> data)
{
    var dataView = mlContext.Data.LoadFromEnumerable(
        data.Select(x =>
            new RatingData()
            {
                UserId = x.UserId,
                PostId = x.PostId,
                Reaction = GetFloatFromReactionTypeEnum(x.ReactionType)
            }).ToArray());

    var pipeline = mlContext.Transforms.Conversion.MapValueToKey("UserId")
        .Append(mlContext.Transforms.Conversion.MapValueToKey("PostId"))
        .Append(mlContext.Recommendation().Trainers.MatrixFactorization(new MatrixFactorizationTrainer.Options
        {
            MatrixColumnIndexColumnName = "UserId",
            MatrixRowIndexColumnName = "PostId",
            LabelColumnName = "Reaction",
            NumberOfIterations = 20,
            ApproximationRank = 100
        }));


    var model = pipeline.Fit(dataView);
    return model;
}
```

Рисунок 5.9 – Код тренування моделі для глибинного навчання (рисунок створено самостійно)

Після навчання моделі, створюється інструмент прогнозування (prediction engine), який використовується для отримання прогнозів рейтингів для кожної публікації. На основі прогнозованих рейтингів вибираються потрібна кількість публікацій для рекомендацій, які потім отримуються з бази даних і перетворюються у формат, придатний для клієнтської частини.

5.5 Реалізація NLP

Метод обробки природної мови[10] (NLP) використовується для надання персоналізованих рекомендацій користувачам на основі аналізу текстового вмісту публікацій. Цей метод дозволяє враховувати текстові характеристики публікацій, щоб знайти схожі публікації та надати релевантні рекомендації.. Для цього створюємо код (див. рис. 5.10) для обробки природної мови.

```
public async Task<IEnumerable<PostItem>> Nlp(int userId)
{
    ITransformer TrainModel(MLContext mlContext, IEnumerable<PassPost> data)...

    var mlContext = new MLContext();

    var passPosts = await _repository.GetRangeAsync<PassPost>(false, x => true, x => x.Post);

    var model = TrainModel(mlContext, passPosts);

    var predictionEngine = mlContext.Model.CreatePredictionEngine<PostData, PostPrediction>(model);

    var userPosts = passPosts.Where(x => x.UserId == userId).ToList();
    var userCluster = predictionEngine.Predict(new PostData
    {
        UserId = userId,
        PostId = userPosts.First().PostId,
        Text = userPosts.First().Post.Text
    }).PredictedClusterId;

    var allPosts = passPosts.Select(x => x.PostId).Distinct().ToList();
    var posts = passPosts.Select(x => x.Post).ToArray();

    var recommendations = allPosts.Select(postId => new
    {
        PostId = postId,
        ClusterId = predictionEngine.Predict(new PostData
        {
            UserId = userId,
            PostId = postId,
            Text = posts.First(p => p.Id == postId).Text
        }).PredictedClusterId
    })
        .Where(x => x.ClusterId == userCluster)
        .Take(10)
        .Select(x => x.PostId)
        .ToList();

    var recommendedPosts = await _repository.GetRangeAsync<Post>(false, x => recommendations.Contains(x.Id));

    IEnumerable<PostItem> filteredPosts = _mapper.Map<IEnumerable<PostItem>>(recommendedPosts);

    return filteredPosts;
}
```

Рисунок 5.10 – Код обробки природної мови (NLP) (рисунок створено самостійно)

Метод NLP використовує текстовий аналіз для створення рекомендацій. Завантажуються дані про взаємодії користувачів з публікаціями та перетворюються у формат, придатний для навчання моделі. PostData в якій зберігається:

– поле UserId – ідентифікатор користувача;

– поле PostId – ідентифікатор публікації;

– поле Text – текс публікації.

Використовується конвеєр (pipeline), що включає фактуризацію тексту (FeaturizeText) для перетворення тексту у числові вектори, перетворення значень у ключі та тренування моделі на основі кластеризації KMeans. Тренування моделі представлено на рисунку 5.11.

```
ITransformer TrainModel(MLContext mlContext, IEnumerable<PassPost> data)
{
    var dataView = mlContext.Data.LoadFromEnumerable(
        data.Select(x =>
            new PostData()
            {
                UserId = x.UserId,
                PostId = x.PostId,
                Text = x.Post.Text
            }).ToArray());

    var pipeline = mlContext.Transforms.Text.FeaturizeText("TextFeatures", nameof(PostData.Text))
       .Append(mlContext.Transforms.Concatenate("Features", "TextFeatures"))
       .Append(mlContext.Transforms.Conversion.MapValueToKey("UserId"))
       .Append(mlContext.Transforms.Conversion.MapValueToKey("PostId"))
       .Append(mlContext.Clustering.Trainers.KMeans("Features", numberOfClusters: 10));

    var model = pipeline.Fit(dataView);
    return model;
}
```

Рисунок 5.11 – Код тренування моделі для обробки природної мови (рисунок створено самостійно)

Після навчання моделі, створюється інструмент прогнозування (prediction engine), який використовується для визначення кластера, до якого належать публікації користувача. На основі визначеного кластера, вибираються публікації, що належать до того ж кластера, що й публікації користувача. Публікації з цього кластера рекомендуються користувачеві та відправляються на клієнт.

6 АНАЛІЗ РЕЗУЛЬТАТІВ

6.1 Підготовка вхідних даних

Для аналізу результатів не вдалося зібрати відповідних великих даних з веб-застосунку, тому потрібно знайти підходящий великий, якісний датасет де можна проаналізувати роботу алгоритмів на ньому, для подальшого аналізу.

Сто відсоткова підходящого датасету не вдалось знайти але був знайдений датасет на історичну літературу [2] , він складається з (302,935 книжок, 2,066,193 оцінок користувачів).

Для заповнення даних постів використовувались дані з датасету книжок, він має вигляд json фалу (див. рис. 6.1)


```
{
    "isbn": "1599150603",
    "text_reviews_count": "7",
    "series": [],
    "country_code": "US",
    "language_code": "",
    "popular_shelves": [...],
    "asin": "",
    "is_ebook": "false",
    "average_rating": "4.13",
    "kindle_asin": "B00DU10PUG",
    "similar_books": [],
    "description": "Relates in vigorous prose the tale of Aeneas, the legendary ancestor of Romulus, who esc
        an epic poem by Virgil, to glorify the imperial city of Rome.",
    "format": "Paperback",
    "link": "https://www.goodreads.com/book/show/287141.The_Aeneid_for_Boys_and_Girls",
    "authors": [{"author_id": "3041852", "role": ""}],
    "publisher": "Yesterday's Classics",
    "num_pages": "162",
    "publication_day": "13",
    "isbn13": "9781599150604",
    "publication_month": "9",
    "edition_information": "",
    "publication_year": "2006",
    "url": "https://www.goodreads.com/book/show/287141.The_Aeneid_for_Boys_and_Girls",
    "image_url": "https://s.gr-assets.com/assets/nophoto/book/111x148-bcc042a9c91a29c1d680899eff700a03.png",
    "book_id": "287141",
    "ratings_count": "46",
    "work_id": "278578",
    "title": "The Aeneid for Boys and Girls",
    "title_without_series": "The Aeneid for Boys and Girls"
}
```


Рисунок 6.1 – Формат файлу книжок (рисунок створено самостійно)

З цього файлу знадобляться дані:

– book_id – для ідентифікації книжок задля створення зв’язків з користувачами;

– title – назва публікації;

– description – опис публікації;

– country_code – для локації публікації;

– publisher – для категорії публікації;

– similar_books – на майбутнє для розрахунків точності роботи алгоритму.

Щоб зв’язати користувачів з цими публікаціями використовуємо датасет з даними відгуків на ці публікації (див. рис. 6.2).


```
{
    "user_id": "8842281e1d1347389f2ab93d60773d4d",
    "book_id": "29893493",
    "review_id": "c23406fb584d6304d1dd4c75ce26ea3a",
    "rating": 5,
    "review_text": "I haven't read a non-fiction book this engaging
        companies, the story of Nike (or Blue Ribbon Shoes as it wa
        Japan at the age of 24, to walk into a Japanese shoe manufa
        start to successful companies (eg Zappos, Amazon, many more
        accounting background, and was well versed that the reason
        to the next, how twice they had to switch banks after being
        athletes to wear their shoes - often to athletes already we
        that changed over time. \n The story of Prefontaine was a t
        got a sense of the passion his has for the sport of running
        frothing, shrieking, the two men entered the final lap. It
        yard lead, then two, then five. We saw Young grimacing and
        company.\" \n The book is a fascinating telling of the four
        more. I could read 4 more volumes of this, as it feels like
    "date_added": "Thu Dec 15 10:51:26 -0800 2016",
    "date_updated": "Sun Mar 12 23:33:51 -0700 2017",
    "read_at": "Thu Mar 09 15:34:06 -0800 2017",
    "started_at": "Tue Feb 28 17:55:35 -0800 2017",
    "n_votes": 29,
    "n_comments": 8
}
```


Рисунок 6.2 – Формат файлу рецензій (рисунок створено самостійно)

З цього файлу знадобляться дані:

– book_id – для ідентифікації книжок;

– user_id – для ідентифікації користувача (кожна унікальна зустріч створює нового користувача);

– rating– оцінка користувача.

Взявши ці дані отримаємо для своєї системи датасет на якому можна проаналізувати дані, які заповнили даними всі необхідні таблиці для виконання алгоритмів (див. рис. 5.1).

6.2 Аналіз результатів експериментів

Використовуючи датасет з підрозділу 6.1 та алгоритми з розділу 5 – можна проаналізувати роботу цих алгоритмів та взяти відповідні характеристики. Датасет з підрозділу 6.1 містить інформацію про взаємодії користувачів з публікаціями, включаючи їхні оцінки, коментарі та реакції. Ці дані є основою для навчання та тестування алгоритмів.

Алгоритми, такі як колаборативне фільтрування, фільтрування на основі змісту, матричне розкладання, глибинне навчання та NLP, використовуються для побудови моделей, які можуть прогнозувати взаємодії користувачів з новими публікаціями. Для кожного з цих алгоритмів можна провести експерименти, щоб оцінити їхню ефективність та точність.

Замірювати будемо такі дані:

- точність – для вимірювання точності будемо порівнювати результат який надала рекомендація та фактичні результати рекомендації, які можна отримати через поле similar_books (див. рис. 6.1), де зберігаються ідентифікатори подібних публікацій, порівнявши кількість публікацій які повернув алгоритм можна порівняти с очікуваним та отримати розрахунки точності;
- повнота – вимірюється те скільки рекомендація з запланованих попали до результатів алгоритму, так що тут теж використовується поле similar_books (див. рис. 6.1);
- час – вимірюється час роботи алгоритму між викликом методу та поверненням рекомендації їм, для цього використовується статичний клас Stopwatch стандартної бібліотеки .NET. Вимірюється в мілісекундах;
- споживані ресурси – значення використаної пам’яті системи алгоритмом;
- масштабованість – оцінка роботи застосунку порівняно зі збільшенням обсягу даних, користувачів, кількості публікацій. Коефіцієнт, який показує наскільки час та споживані ресурси збільшуються від кількості даних які використовує алгоритм.

Знаходимо 3х користувачів, перший з великою кількістю рецензій, тобто в нього є велика кількість значень оцінок, на якій можна створювати рекомендацію, другий з середньою кількістю та третій – з найменшою, однією оцінкою.

Проводимо розрахунки для колаборативного фільтрування [4] та заносимо дані до таблиці 6.1. Для цього алгоритму аналізуємо лише дані до 1000, бо вже на матриці 1000 на 1000 вона показує довгу роботу застосунку.

Таблиця 6.1 – Результати колаборативного фільтрування (таблицю створено самостійно)

користувач	кількість даних	точність	повнота	час	ресурси	масштабованість
1	100	0.223	0.438	14566	415 Мб	206.26
1	500	0.287	0.472	264408	546 Мб	
1	1000	0.248	0.455	104288256	982 Мб	
2	100	0.272	0.483	14242	448 Мб	206.57
2	500	0.215	0.426	263815	504 Мб	
2	1000	0.290	0.490	104054364	906 Мб	
3	100	0.066	0.163	14243	410 Мб	207.63
3	500	0.039	0.142	296239	570 Мб	
3	1000	0.081	0.179	116843094	993 Мб	

На основі таблиці 6.1 створюємо графік збільшення швидкості роботи застосунку від збільшення кількості вибірки (див. рис. 6.3).

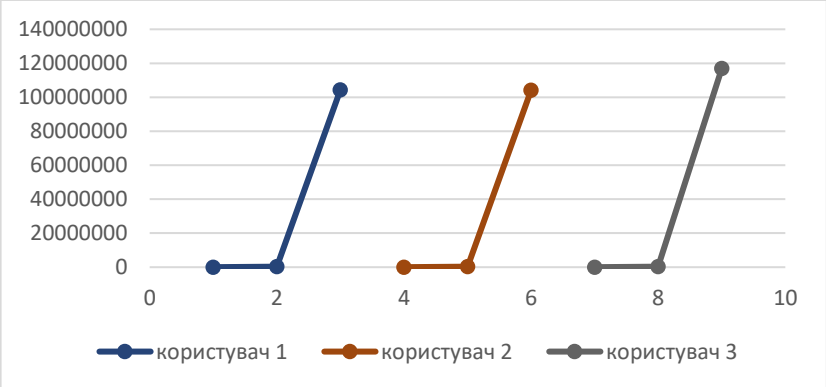

Рисунок 6.3 – Графік зростання часу від кількості даних (колаборативне фільтрування) (рисунок створено самостійно)

Точність та повнота цього алгоритму доволі непогана, але для користувача в якого не має відгуків – точність погана. Робота алгоритму доволі довга, а зі збільшенням кількості даних починає працювати набагато довше. Використання пам’яті непогана, збільшується з більшою кількістю даних, але не критична.

Проводимо розрахунки для фільтрування на основі вмісту [6] та заносимо дані до таблиці 6.2.

Таблиця 6.2 – Результати фільтрування на основі вмісту (таблицю створено самостійно)

користувач	кількість даних	точність	повнота	час	ресурси	масштабованість
1	100	0.299	0.483	17969	2600 Мб	1.12
1	500	0.299	0.483	25687	2700 Мб	
1	1000	0.299	0.483	27021	2700 Мб	
1	10000	0.299	0.483	24040	2700 Мб	
2	100	0.263	0.426	7808	1300 Мб	0.91
2	500	0.263	0.426	7533	1300 Мб	
2	1000	0.263	0.426	5222	1900 Мб	
2	10000	0.263	0.426	5566	1200 Мб	
3	100	0.233	0.321	4983	1200 Мб	1.02
3	500	0.233	0.321	6699	1000 Мб	
3	1000	0.233	0.321	6029	1100 Мб	
3	10000	0.233	0.321	4901	1000 Мб	

На основі таблиці 6.2 створюємо графік збільшення швидкості роботи застосунку від збільшення кількості вибірки (див. рис. 6.4).

Точність та повнота цього алгоритму напевно найкраща з усіх, але для користувача в якого не має відгуків – точність погана. Для кожного користувача точність та повнота однакові, так як кожен раз алгоритм для певного користувача повертає один й той самий результат тому результати однакові. Робота алгоритму одна з найкращій, масштабованість дуже низька та навіть на 10000 даних – працює досконало. Використання пам’яті непогана, вона не збільшується та тримається на одному й тому самому рівні не залежності від кількості даних.

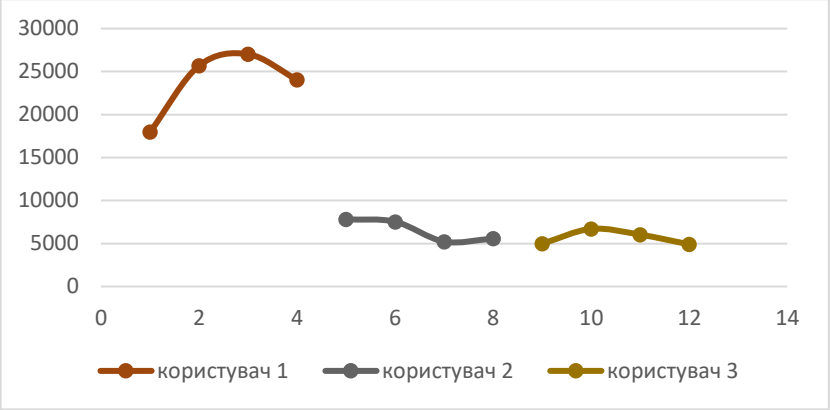

Рисунок 6.4 – Графік зростання часу від кількості даних (фільтрування на основі вмісту) (рисунок створено самостійно)

Проводимо розрахунки для матричного розкладання[8] та заносимо дані до таблиці 6.3.

Таблиця 6.3 – Результати матричного розкладання (таблицю створено самостійно)

користувач	кількість даних	точність	повнота	час	ресурси	масштабованість
1	100	0.083	0.089	1934	399 Мб	16.04
1	500	0.148	0.119	6542	573 Мб	
1	1000	0.186	0.114	46907	993 Мб	
1	10000	0.132	0.177	1762346	4600 Мб	
2	100	0.044	0.052	3156	412 Мб	14.57
2	500	0.067	0.057	7583	487 Мб	
2	1000	0.101	0.136	28398	962 Мб	
2	10000	0.159	0.198	1066943	4400 Мб	
3	100	0.073	0.097	2220	430 Мб	15.48
3	500	0.134	0.103	7625	575 Мб	
3	1000	0.148	0.179	41597	947 Мб	
3	10000	0.109	0.165	1562844	4300 Мб	

На основі таблиці 6.3 створюємо графік збільшення швидкості роботи застосунку від збільшення кількості вибірки (див. рис. 6.5).

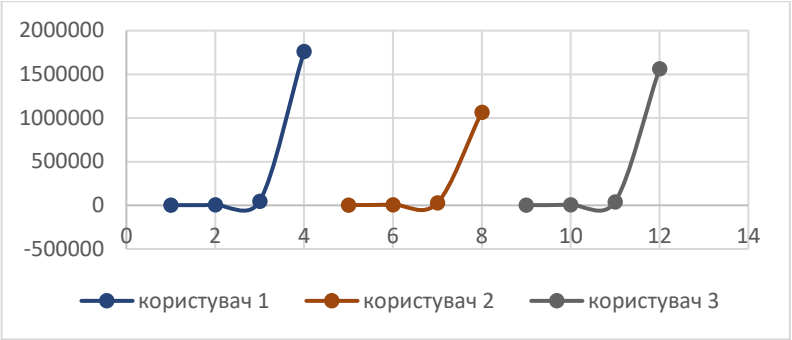

Рисунок 6.5 – Графік зростання часу від кількості даних (матричне розкладання) (рисунок створено самостійно)

Точність та повнота цього найслабкіші, але трішки зростає зі збільшенням кількості даних. Час роботи алгоритму краще ніж перший але все одно погане масштабування. Використання пам’яті непогана, але значно зростає з великою кількістю даних.

Проводимо розрахунки для глибинного навчання[10] та заносимо дані до таблиці 6.4.

Таблиця 6.4 – Результати глибинного навчання (таблицю створено самостійно)

користувач	кількість даних	точність	повнота	час	ресурси	масштабованість
1	100	0.238	0.438	3494	429 Мб	0.947
1	500	0.267	0.472	3551	407 Мб	
1	1000	0.285	0.455	2479	397 Мб	
1	10000	0.219	0.483	2794	383 Мб	
2	100	0.252	0.426	2289	444 Мб	1.075
2	500	0.293	0.490	2546	402 Мб	
2	1000	0.276	0.463	2973	361 Мб	
2	10000	0.241	0.442	2812	423 Мб	
3	100	0.262	0.479	2457	442 Мб	1.224
3	500	0.229	0.421	2029	766 Мб	
3	1000	0.278	0.495	3596	419 Мб	
3	10000	0.297	0.458	3857	431 Мб	

На основі таблиці 6.4 створюємо графік збільшення швидкості роботи застосунку від збільшення кількості вибірки (див. рис. 6.6).

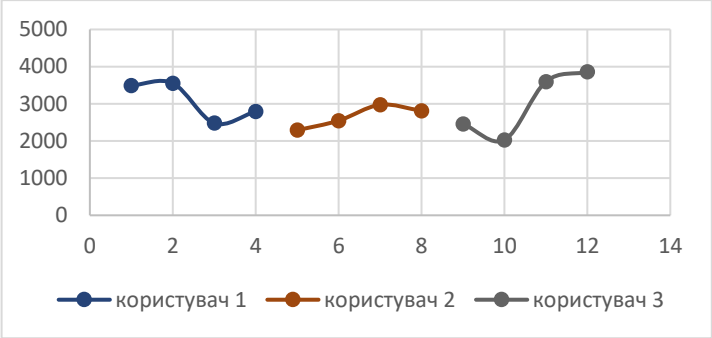

Рисунок 6.6 – Графік зростання часу від кількості даних (глибинне навчання) (рисунок створено самостійно)

Точність та повнота цього алгоритму не погана, на рівні з другим. Час роботи алгоритму один з найкращих. Використання пам’яті мінімальна та не залежить від кількості даних які обробляє.

Проводимо розрахунки для NLP[10] та заносимо дані до таблиці 6.5.

Таблиця 6.5 – Результати NLP (таблицю створено самостійно)

користувач	кількість даних	точність	повнота	час	ресурси	масштабованість
1	100	0.038	0.092	6696	1200 Мб	2.61
1	500	0.062	0.134	11925	1700 Мб	
1	1000	0.049	0.068	15787	2100 Мб	
1	10000	0.073	0.141	74749	7500 Мб	
2	100	0.015	0.099	6984	1100 Мб	2.49
2	500	0.058	0.058	14394	2100 Мб	
2	1000	0.024	0.073	18889	2600 Мб	
2	10000	0.081	0.122	77508	7800 Мб	
3	100	0.043	0.148	6335	1100 Мб	2.15
3	500	0.019	0.115	10514	1700 Мб	
3	1000	0.096	0.053	16650	2400 Мб	
3	10000	0.035	0.137	53669	7700 Мб	

На основі таблиці 6.5 створюємо графік збільшення швидкості роботи застосунку від збільшення кількості вибірки (див. рис. 6.7).

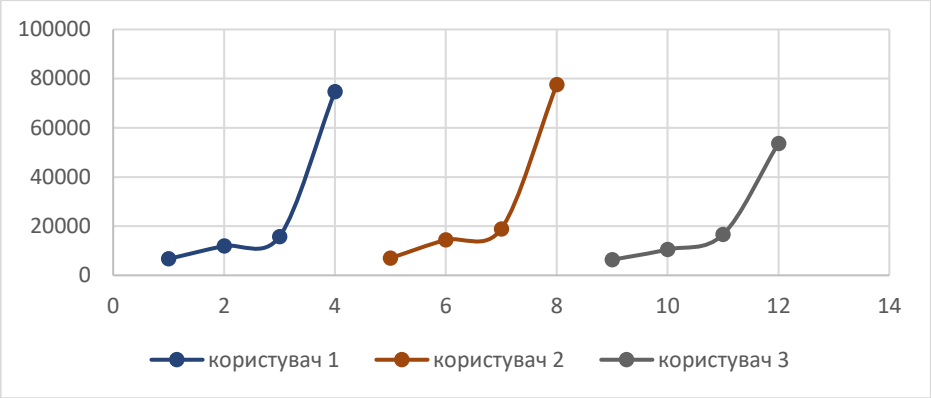

Рисунок 6.7 – Графік зростання часу від кількості даних (NLP) (рисунок створено самостійно)

Точність та повнота цього алгоритму напевно найгірша з усіх. Робота алгоритму потенційно швидка, зі збільшенням даних швидкість майже у 2 рази росте через таку масштабованість. Використання пам’яті велику кількість, найбільше використання пам’яті з усіх.

Проаналізуючи середні значення зібраних даних, для нашої системи найбільш ефективним являється – фільтрування на основі вмісту та глибинне навчання. Але це була оцінка датасету, та для більшого аналізу необхідно випустити застосунок у роботу, збирати відгуки користувачів та тестувати його на різних специфічних сценарію[12].

6.3 Можливий розвиток дослідження

По-перше, варто провести масштабне тестування алгоритмів на більш різноманітних реальних даних з системи, щоб забезпечити їхню узагальненість та надійність. По-друге, можна здійснити збір зворотного зв'язку від користувачів для оцінки точності та релевантної рекомендацій, що дозволить налаштовувати алгоритми відповідно до їхніх потреб та вподобань. Іншим важливим аспектом є дослідження впливу рекомендаційних систем на навчальний процес та результати користувачів.

ВИСНОВКИ

Результатом даного магістерського проекту є виявлення важливості використання рекомендаційних систем у сфері освіти та визначення їхнього потенціалу для покращення якості освіти та персоналізації навчання. Актуальність теми обумовлена стрімкими змінами в освітньому середовищі та необхідністю забезпечення ефективного навчання в умовах сучасного інформаційного суспільства.

Аналіз історії розвитку рекомендаційних систем загалом та насамперед в сфері освіти дає дуже міцний бекграунд.

Були розглянуті різні рекомендаційні алгоритми та порівняні завдяки коефіцієнтам, якими було приблизно підбиті важливі характеристики в сфері освіти, порівняно та обрано більш підходящий алгоритм.

Були розглянуті рекомендаційні алгоритми в сфері освіти, які розроблені в веб-системі з використанням таких технологій, як Angular, C# 12 [3] та SQL.

Обрані алгоритми були реалізовані та протестовані. Також провівся аналіз алгоритмів на якісному датасеті розміром 2,369,128 строк даних. Біли проаналізована робота за різними характеристиками: точність, повнота, швидкість, ресурси та масштабованість.

Біло обрано подальший шлях дослідження, для покращення результатів в цій сфері.

Загалом, результати даного дослідження відкривають нові можливості для розвитку рекомендаційних систем у сфері освіти та їх впровадження у реальні освітні середовища, сприяючи підвищенню якості та ефективності отримання знань.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. 28-й Міжнародний молодіжний форум «Радіоелектроніка та молодь у XXI столітті». Зб. матеріалів форуму. Т. 6., – Харків: ХНУРЕ. 2024. –958 с.

2. Goodreads Datasets. Mengting Wan. URL: https://mengtingwan.github.io/data/goodreads.html#datasets (дата звернення: 05.06.2024).

3. Бондарев В. М. Объектно-ориентированное программирование на C#. – Х.: Компания СМИТ, 2009. – 224 с.

4. Evaluating Collaborative Filtering Recommender Systems. URL: https://grouplens.org/site-content/uploads/evaluating-TOIS-20041.pdf (дата звернення 05.06.2024).

5. Pinela C. Recommender Systems – User-Based and Item-Based Collaborative Filtering. Medium. URL: https://medium.com/@cfpinela/recommender-systems-user-based-and-item-based-collaborative-filtering-5d5f375a127f (дата звернення: 05.06.2024).

6. Pazzani, M. J., & Billsus, D. "Content-Based Recommendation Systems." In "The Adaptive Web," edited by Peter Brusilovsky, Alfred Kobsa, and Wolfgang Nejdl, Springer. 2007. – 341 с.

7. Gupta B. Content Based Filtering in Machine Learning - Scaler Topics. Scaler Topics. URL: https://www.scaler.com/topics/machine-learning/content-based-filtering/ (дата звернення: 05.06.2024).

8. Koren, Y., Bell, R., & Volinsky, C. "Matrix Factorization Techniques for Recommender Systems." Computer, 42, 2009, – 37 с.

9. Gupta R. How Singular Value Decomposition (SVD) is used in Recommendation Systems, “Clearly Explained”. Medium. URL: https://medium.com/@ritik_gupta/how-singular-value-decomposition-svd-is-used-in-recommendation-systems-clearly-explained-201b24e175db (дата звернення: 05.06.2024).

10. What is ML.NET and how does it work? - ML.NET. Microsoft Learn: Build skills that open doors in your career. URL: https://learn.microsoft.com/en-us/dotnet/machine-learning/how-does-mldotnet-work (дата звернення: 05.06.2024).

11. Гребеннік І. В. Методи підтримки прийняття рішень : навч. посібник / І. В. Гребеннік, Т. Є. Романова, А. Д. Тевяшев, Г. М Яськов ; МОН України, Харк. нац. ун-т радіоелектроніки. – Харків : ХНУРЕ, 2010. – 127 с.

12. Chalyi S., Leshchynskyi V. METHOD OF CONSTRUCTING EXPLANATIONS FOR RECOMMENDER SYSTEMS BASED ON THE TEMPORAL DYNAMICS OF USER PREFERENCES. EUREKA: Physics and Engineering. 2020. T. 3. C. 43–50.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ ЗА НАУКОВИМИ НАПРЯМАМИ КЕРІВНИКА ТА НАУКОВЦІВ КАФЕДРИ ПРОГРАМНОЇ ІНЖЕНЕРІЇ

3. Бондарев В. М. Объектно-ориентированное программирование на C#. – Х.: Компания СМИТ, 2009. – 224 с.

11. Гребеннік І. В. Методи підтримки прийняття рішень : навч. посібник / І. В. Гребеннік, Т. Є. Романова, А. Д. Тевяшев, Г. М Яськов ; МОН України, Харк. нац. ун-т радіоелектроніки. – Харків : ХНУРЕ, 2010. – 127 с.

12. Chalyi S., Leshchynskyi V. METHOD OF CONSTRUCTING EXPLANATIONS FOR RECOMMENDER SYSTEMS BASED ON THE TEMPORAL DYNAMICS OF USER PREFERENCES. EUREKA: Physics and Engineering. 2020. Т. 3. С. 43–50.