Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:
Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання

Виконав: студент II курсу
групи ЗПІ−22м спеціальності
121 – Інженерія програмного забезпечення
Волинець Олександр Юрійович

Керівник: к.т.н., асист. каф. ПЗ Тужанський С. Є.
«11» грудня 2023 р.

Опонент: к.т.н., доцент каф. ОТ Кожем'яко А.В.
«11» грудня 2023 р.

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О. Н.
(прізвище та ініціали)
«11» грудня 2023 р.

ВНТУ – 2023

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О. Н.
«19» вересня 2023 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
СТУДЕНТУ

Волинцю Олександру Юрійовичу

1. Тема роботи – метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання.

Керівник роботи: Тужанський Станіслав Євгенович, к.т.н., асистент кафедри ПЗ, затверджені наказом вищого навчального закладу від « 18 » вересня 2023 р. № 247.

2. Строк подання студентом роботи – 5 грудня 2023 р.

3. Вихідні дані до роботи: середовище розробки Visual Studio та Visual Studio Code, мови програмування C#, TypeScript, операційна система Windows, використання алгоритмів ML.NET, веб платформи ASP.NET, фреймворку Angular та існуючих методів персоналізованих рекомендацій.

4. Зміст текстової частини: вступ, аналіз предметної області, аналіз існуючих методів персоналізованих рекомендацій, аналіз існуючих алгоритмів машинного навчання, аналіз необхідних даних для навчання моделі, висновки та постановка задачі, проектування додатків для створення засобу персоналізованих рекомендацій, проектування методу формування персоналізованих вподобань, проектування бібліотеки формування персоналізованих рекомендацій використовуючи машинне навчання, проектування утиліт для машинного навчання, проектування веб додатку з системою аутентифікації, проектування бази даних для додатків, вибір засобів реалізації, висновки, програмна реалізація та впровадження захисту у хмарну платформу, розробка сервісу аутентифікації користувачів,

розробка бібліотеки формування персоналізованих рекомендацій, розробка утиліт для машинного навчання, розробка веб сервісів з використанням покращеної системи персоналізованих рекомендацій, розробка інтерфейсу веб додатку і тестування бібліотек, перелік посилань, додатки.

5. Перелік ілюстративного матеріалу: блок-схеми алгоритмів, UML діаграми, діаграми залежностей, зображення вікон програмного засобу, тестування додатку.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Тужанський С.Є., асистент кафедри ПЗ	19.09.23	05.12.23
4	Причепа І.В., к.е.н., доцент каф. ЕПВМ	19.11.23	04.12.23

7. Дата видачі завдання 19 вересня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області	20.09.2023 – 10.10.2023	вик
2	Проектування методу і засобу персоналізованих рекомендацій	10.10.2023 – 30.10.2023	вик
3	Програмна реалізація та впровадження методу персоналізованих рекомендацій	01.11.2023 – 30.11.2023	вик
4.	Економічна частина	20.11.2023 – 01.12.2023	вик

Студент _____ **Волинець О.Ю.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Тужанський С.Є.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.912.032.26

Волинець О. Ю. Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання

Магістерська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2023. 163 с.

На укр. мові. Бібліогр.: 33 назва; рисунків: 34; таблиць 12.

У магістерській кваліфікаційній роботі подано результати дослідження алгоритмів створення рекомендацій. Обґрунтовано доцільність створення методу та програмного засобу персоналізованих рекомендацій на основі машинного навчання, який усуває вказані недоліки.

Магістерська робота містить аналіз відомих алгоритмів персоналізованих, а також методів машинного навчання для створення рекомендацій.

У роботі спроєктовано і створену бібліотеки класів і інтегровано в створений застосунок з функціональними можливостями і персоналізованими рекомендаціями. Для функціонування методу і застосунку створені спеціалізовані утиліти.

Наукова новизна досліджень полягає у створенні комплексного методу надання персоналізованих рекомендацій на основі навчання, який ефективніший за базові алгоритми надання рекомендацій.

Ключові слова: персоналізовані рекомендації, машинне навчання, алгоритми рекомендацій, колаборативний підхід, веб застосунки.

ABSTRACT

Volynets. O. Y. Method and software for personalized recommendations based on machine learning

Master's qualification work on specialty 121 - software engineering, educational program - software engineering. Vinnytsia: VNTU, 2023. 163 p.

In Ukrainian languages Bibliogr.: 33 name; drawings: 34; tables 12.

The master's thesis presents the results of the research of algorithms for creating recommendations. In the diploma, the text provides information about the expediency of creating a method and software research of personalized recommendations based on machine learning, which eliminates the specified shortcomings, is substantiated.

The master's thesis contains an analysis of well-known personalized algorithms, as well as machine learning methods for creating recommendations.

Class libraries were designed and created in the work and integrated into the created programs with functionality and personalized recommendations. Specialized utilities have been created for the operation and application of the method.

The scientific novelty of the research is carried out in the created comprehensive method of providing personalized recommendations based on machine learning, which is more effective than analyzed algorithms for providing recommendations.

Keywords: personalized recommendations, machine learning, recommendation algorithms, collaborative approach, web applications.

ЗМІСТ

ВСТУП

Актуальність теми. Популяризація інтернет послуг створює попит на ефективні системи рекомендацій. У сучасному світі щодня генерується велика кількість даних, аналіз яких у інформаційних системах є складним завданням. Постійне зростання обсягів інформації у мережах додатково ускладнює їх використання. Персоналізовані системи рекомендацій допомагають ефективно аналізувати та використовувати потрібні дані для покращення взаємодії з користувачем.

Крім того, конкурентна перевага на ринку для багатьох компаній визначається їх здатністю надавати персоналізовані рекомендації і послуги у найрізноманітніших сферах. Це допомагає підвищити лояльність клієнтів і конкурувати з іншими компаніями на ринку.

Розвиток технологій машинного навчання та зростаюча потужність обчислювальних систем дозволили підвищити точність та ефективність методів створення інформаційних систем рекомендацій.

Персоналізовані рекомендації допомагають покращити взаємодію з користувачем, допомагаючи знайти вміст або продукти, які найкраще відповідають потребам і вподобанням конкретного користувача. Вони покращують досвід використання веб додатків і допомагають користувачам досягти більш деталізованих результатів.

Таким чином, тема МКР «Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання» є актуальною та має потенціал для застосування у різних типах додатків включно із веб застосунками. Розробка програмного засобу з використанням покращеного методу персоналізованих рекомендацій є перспективною для застосування в різних галузях, зокрема у бізнесі, медіа, соціальних мережах та інших.

Мета і задачі дослідження. Метою роботи є підвищення точності генерації персоналізованих рекомендацій у рекомендаційних системах за

рахунок використання методів машинного навчання з урахуванням індивідуальних потреб користувачів.

Основними задачами дослідження є:

– Аналіз сучасних методів і підходів до створення персоналізованих рекомендацій та рекомендаційних систем;

– Розробка та вдосконалення навчання моделей з рекомендаційними алгоритмами, які враховують контентні та колаборативні підходи, аналіз та порівняння різних моделей за їхньою ефективністю;

– Удосконалення алгоритму формування рекомендацій з урахуванням результатів моделі машинного навчання;

– Розробка програмного засобу, який базується на удосконалених моделях та забезпечує створення персоналізованих рекомендацій для кінцевих користувачів;

– Проведення тестування розробленого програмного засобу та аналіз ефективності рекомендацій з використанням інтегрованих бібліотек.

Об’єкт дослідження – процес надання персоналізованих рекомендацій.

Предмет дослідження – методи і засоби формування персоналізованих рекомендацій.

Методи дослідження. У роботі використані такі методи дослідження, як: теорія чисел та чисельних методів, лінійна алгебра, теорія машинного навчання, теорія тестування програмних засобів. комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна роботи. Удосконалено надання персоналізованих рекомендацій на основі навчання, який є на відміну від інших є поєднанням класичних методів рекомендацій із машинним навчанням та побудовою матриць уподобань за принципом бінарного пошуку, що дозволило сформувати більш ефективні і точні персоналізовані рекомендації на основі попередніх даних шляхом побудови залежностей вподобань з урахуванням

результатів моделі машинного навчання.

Практичне значення одержаних результатів. На основні запропонованого методу розроблено програмний засіб у вигляді сервісів і веб додатків, який використовує удосконалений метод надання персоналізованих рекомендацій на основі машинного навчання з відповідною навченою моделлю на основі реальних даних.

Особистий внесок здобувача. Усі наукові результати, викладені у магістерській кваліфікаційній роботі, отримані автором особисто.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на міжнародній науково-практичній інтернет-конференції "Електронні інформаційні ресурси: створення, використання, доступ 2023".

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Сучасний інтернет, супроводжуваний безмежним обсягом інформації та контенту, створює велике суспільне прагнення до персоналізації взаємодії з веб-середовищем. Користувачі очікують індивідуального підходу до своїх потреб та інтересів, незалежно від того, чи мова йде про онлайн-магазини, стрімінгові платформи, медіа-сервіси або соціальні мережі. У зв'язку з цим, системи рекомендацій стали важливим інструментом для забезпечення користувачів змістом та продуктами, які відповідають їхнім унікальним вимогам.

Рекомендаційні системи використовуються для передбачення та підтримки індивідуальних потреб користувачів, сприяючи покращенню їхнього досвіду в онлайн-середовищі. Вони можуть рекомендувати товари, статті, фільми, музику, ігри або навіть потенційних друзів, залежно від контексту застосування.

Основна ідея підходів до рекомендацій полягає в тому, щоб використовувати інформацію про минулу поведінку або думки існуючої спільноти користувачів, щоб передбачити, які предмети поточний користувач системи найімовірніше отримає, подобається або цікавить. Такі типи систем широко використовуються в промисловості, зокрема, як інструмент на сайтах продажу товарів чи послуг в Інтернеті, щоб задовольнити потреби конкретного клієнта і таким чином просувати додаткові товари та збільшувати продажі [1].

Підходи до побудови персоналізованих рекомендаційних систем стали дуже актуальними завдяки росту обсягів даних та розвитку методів машинного навчання.

Мета персоналізованих рекомендацій - надавати користувачам контент, який найкраще відповідає їхнім індивідуальним потребам і інтересам. При цьому, системи рекомендацій можуть використовувати

різноманітні підходи, включаючи колаборативний та контент-базований підхід, а також машинне чи глибоке навчання.

1.1 Аналіз існуючих методів персоналізованих рекомендацій

Аналіз існуючих методів персоналізованих рекомендацій є ключовим етапом в розробці програмного засобу для створення рекомендаційної системи на основі машинного навчання. Дослідження існуючих методів допомагає зрозуміти сучасний стан цієї предметної області і визначити найбільш ефективні та підходящі підходи для досягнення поставленої мети.

Розглянемо найбільш популярні методи персоналізованих рекомендацій.

Один із найбільш популярних підходів до побудови персоналізованих рекомендацій - це колаборативний фільтринг. Він базується на ідеї аналізу взаємодії користувачів та об'єктів (наприклад, товарів, фільмів, тощо). Цей метод поділяється на два підтипи:

– Колаборативний фільтринг на основі споживачів (User-Based Collaborative Filtering): Цей підхід рекомендує об'єкти на основі інших користувачів, які мають схожі уподобання або історію взаємодії.

– Колаборативний фільтринг на основі об'єктів (Item-Based Collaborative Filtering): В цьому випадку рекомендації створюються на основі схожості об'єктів або товарів, які користувач раніше взаємодіяв.

Типові питання, які виникають у контексті колаборативного фільтрингу включають наступне:

– Як знайти користувачів із схожими смаками на користувача, для якого нам потрібен рекомендація?

– Як ми вимірюємо подібність?

– Що робити з новими користувачами, у яких ще немає історії покупок доступний [1]?

Кожен метод має свої переваги та недоліки, зокрема колаборативний

фільтринг є легким для реалізації та зрозуміння, що робить його популярним в початкових розробках рекомендаційних систем, а також відсутність потреби в контенті: Не потребується додаткової інформації про об'єкти, тільки історія взаємодії користувачів. Однак, основною проблемою є холодний старт. Він не ефективний для нових користувачів або об'єктів, оскільки немає історії взаємодії [2].

Ще одним популярним методом є контент-базований фільтринг. Цей підхід використовує аналіз властивостей об'єктів та інтересів користувачів. Наприклад, для рекомендацій фільмів це може включати аналіз жанру, акторів, режисера та інших характеристик фільмів.

– У контексті рекомендацій на основі змісту виникають наступні запитання:

– Як системи можуть автоматично отримувати та постійно вдосконалювати користувача профілі?

– Як ми визначаємо, які предмети відповідають або принаймні схожі чи сумісні з інтересами користувача?

– Які методи можна використати для автоматичного вилучення або вивчення предмета описів, щоб зменшити ручне втручання?

Останнім часом глибоке навчання, зокрема з використанням нейронних мереж, набуло великої популярності у розробці рекомендаційних систем. Відмінність полягає в тому, що глибокі моделі можуть автоматично виявляти складні зв'язки в даних та забезпечувати високу точність рекомендацій.

1.2 Аналіз існуючих алгоритмів машинного навчання

Останнім часом глибоке і машинне навчання, зокрема з використанням нейронних мереж, набуло великої популярності у розробці рекомендаційних систем. Відмінність полягає в тому, що глибокі моделі можуть автоматично виявляти складні зв'язки в даних та забезпечувати

високу точність рекомендацій.

Розпочнемо з матричної факторизації (Matrix Factorization). Цей підхід розглядає взаємодію користувачів і об'єктів як матрицю та намагається розкласти її на більш прості компоненти. Це дозволяє підкреслити важливі зв'язки і створити рекомендації на основі цих компонент [4].

Матрична факторизація — це клас алгоритмів спільної фільтрації, які використовуються в системах рекомендацій. Алгоритми матричної факторизації працюють шляхом розкладання матриці взаємодії користувач-елемент на добуток двох прямокутних матриць меншої розмірності. Цей набір методів став більш популярним завдяки нагороді Netflix, яка була вручена за їхню високу ефективність. Ініціатором цього був Саймон Фанк, який поділився своїми висновками у своєму блозі ще у 2006 році з науковою спільнотою [5]. Згодом виявилося, що результати прогнозу можна покращити, використовуючи різні ваги регуляризації прихованим факторам, які базуються на популярності об'єктів та активності користувачів.

Ідея матричної факторизації полягає в тому, щоб представити користувачів і об'єкти (елементи) у меншому латентному просторі низької розмірності. Після першої роботи Функа у 2006 році щодо рекомендаційних систем було запропоновано численні методи матричної факторизації. На деякі з найпоширеніших і простих з них ми далі звернемо увагу.

Funk MF - оригінальний алгоритм, запропонований Саймоном Функом у його дописі в блозі, розкладає матрицю оцінки елементів користувача на множники як добуток двох матриць меншої розмірності, перша має рядок для кожного користувача, а друга має стовпець для кожного елемента. Рядок або стовпець, пов'язані з конкретним користувачем або елементом, називаються прихованими факторами. Зауважте, що в Funk MF не застосовано декомпозицію сингулярного

значення, це SVD (Singular Value Decomposition) - подібна модель машинного навчання [4].

Зокрема, прогнозована оцінка, яку користувач дасть елементу і, обчислюється як:

$$r_{ui} = \sum_{f=0}^{nfactors} H_{u,f} W_{f,i} \tag{1.1}$$

де r - це матриця оцінки елементів користувача, H - містить приховані фактори користувача, W - приховані фактори елемента.

Асиметричний SVD має на меті поєднати переваги SVD++, будучи алгоритмом на основі моделі, отже, дає змогу розглядати нових користувачів із кількома оцінками без необхідності перенавчання всієї моделі. На відміну від SVD на основі моделі, тут матриця латентного фактора користувача H замінена на Q, яка вивчає вподобання користувача як функцію його оцінок [3].

Передбачувана оцінка, яку користувач дасть елементу, обчислюється як:

$$r_{ui} = \mu + b_i + b_u + \sum_{f=0}^{nfactors} \sum_{j=0}^{nitems} r_{uj} Q_{j,f} W_{f,i} \tag{1.2}$$

Сингулярний розклад, або SVD, - це математичний алгоритм, який використовується в багатьох областях, включаючи аналіз даних і рекомендації. В контексті систем рекомендацій SVD допомагає розкривати складні залежності між користувачами і елементами (товари, послуги тощо). SVD++ є розширенням базового SVD для покращення якості рекомендацій. Його ключовою ідеєю є врахування не тільки історії оцінок

користувача, але також активності користувача, наприклад, як часто він дивився певний фільм або читав категорію книг. Це допомагає створити більш точні та персоналізовані рекомендації.

Групозалежний SVD є розширенням SVD, яке додає поняття груп або категорій користувачів. У реальних системах рекомендацій користувачі можуть бути частиною різних груп, і їхні інтереси можуть відрізнятися залежно від групи, до якої вони відносяться.

Group-specific SVD дозволяє враховувати це різноманіття шляхом розробки рекомендацій для окремих груп користувачів. Наприклад, він може рекомендувати книги фанам фантастики та книги жахів окремо, навіть якщо користувачі належать до обох груп. Це допомагає створити більш точні та спеціалізовані рекомендації для кожної групи користувачів.

Останніми роками було розроблено багато інших моделей матричної факторизації для використання постійно зростаючої кількості та різноманітності доступних даних про взаємодію та випадків використання. Алгоритми гібридної матриці факторизації здатні об’єднувати явні та неявні взаємодії або як вміст, так і спільні дані [6].

Спільні фактори, також відомі як "Factorization Machines" (Факторизаційні машини), - це алгоритм машинного навчання, який використовується для аналізу взаємодії між різними факторами, що можуть впливати на рекомендації. Цей метод дозволяє враховувати різні види даних і вагомість окремих факторів, не обмежуючись тільки схожістю.

Спільні фактори використовують ідею факторизації, де ознаки (фактори) подаються у вигляді добутку параметрів для кожної пари факторів. Це дозволяє моделі краще захоплювати взаємодію між факторами, і вона може бути дуже ефективною для завдань, де важливі взаємодії між різними ознаками [7].

Цей алгоритм враховує взаємодію між різними факторами, які можуть впливати на рекомендації. Він може обробляти різноманітні види даних та

дозволяє враховувати не тільки схожість, а й важливість факторів.

Нейронні мережі (Neural Networks): Глибоке навчання і нейронні мережі набули популярності в останні роки завдяки своїй здатності виявляти складні зв'язки в даних. Вони використовуються для створення складних рекомендаційних систем, здатних адаптуватися до змін в поведінці користувачів. З використанням нейронних мереж, глибоке навчання дозволяє автоматично виявляти складні зв'язки та патерни в даних. Цей підхід може застосовуватися для роботи з великими обсягами даних та досягнення високої точності в рекомендаціях.

Зважаючи на різнорідність алгоритмів машинного навчання, кожен з них має свої переваги і недоліки.

Переваги колаборативного фільтрингу:

– простота реалізації: Колаборативний фільтринг є легким для реалізації та розуміння, що робить його популярним в початкових розробках рекомендаційних систем.

– відсутність потреб в контенті: Ви не потребуєте додаткової інформації про об'єкти (наприклад, зміст фільмів), тільки історія взаємодії користувачів.

Недоліки колаборативного фільтрингу:

– проблеми з розрідженістю: Колаборативний фільтринг може мало працювати, коли у вас є розріджені дані (багато користувачів, які оцінили лише декілька об'єктів).

– холодний старт: Він не ефективний для нових користувачів або об'єктів, оскільки немає історії взаємодії.

Переваги контент-базованого фільтрингу:

– інтерпретованість: Ви знаєте, чому рекомендація була зроблена, оскільки ви враховуєте властивості об'єктів та інтереси користувачів.

– вирішує проблему холодного старту: Цей метод можна використовувати для рекомендацій новим користувачам або об'єктам,

оскільки він заснований на контенті.

Недоліки контент-базованого фільтрингу:

– вимоги до даних: Вам потрібна відповідна інформація про об'єкти, їхні властивості та користувачів.

– обмежена рекомендація: Цей метод обмежується властивостями, які ви враховуєте, і може недооцінювати складні взаємозв'язки між об'єктами.

Переваги матричного розкладу та факторизаційних машин:

– універсальність: Вони можуть моделювати складні залежності між користувачами і об'єктами.

– можливість враховувати додаткові дані: Ви можете легко включити додаткові ознаки, щоб покращити якість рекомендацій.

Недоліки матричного розкладу та факторизаційних машини:

– складність підбору параметрів: Налаштування моделей може бути складним завданням.

– залежність від обсягу даних: Вони вимагають великого обсягу даних для точних рекомендацій.

Переваги методів заснованих на нейронних мережах:

– здатність до роботи з різнорідними даними: Нейронні мережі можуть ефективно обробляти інформацію різних типів, включаючи текстові, графічні, аудіо та числові дані. Це робить їх корисними для створення комплексних рекомендаційних систем, які враховують різноманітні аспекти користувачів та контенту.

– здатність до автоматичного вивчення рекомендацій: Нейронні мережі можуть навчатися з історії користувачів і вибору контенту, щоб піддавати аналізу складні взаємодії та попереджати майбутні рекомендації.

Недоліки методів заснованих на нейронних мережах:

– вимоги до обчислювальних ресурсів: Нейронні мережі можуть бути великими та складними, що потребує великої кількості обчислювальних ресурсів. Це може бути проблемою для систем з обмеженими

обчислювальними можливостями.

– необхідність великого обсягу даних: Для ефективного навчання нейронних мереж необхідний значний обсяг даних. У випадку, коли дані обмежені, може бути складно створити точні та надійні рекомендації.

– чорна скринька: Нейронні мережі часто вважаються "чорними скриньками", оскільки їх рішення можуть бути важко інтерпретовані. Це може ускладнити пояснення рекомендацій користувачам та супервізорам.

Таблиця 1.1 - Переваги і недоліки методів персоналізованих рекомендацій

Назва методу	Простота реалізації	Зрозумілість вимог до контенту	Ефективність для нових користувачів	Урахування зв'язків
Колаборативний фільтринг	+	++	-	++
Контент-базованоий фільтринг	+-	--	+	-
Матричний розкладу та Факторизаційні машини	+-	+-	+-	-
Методи засновані на Нейронних мережах	--	--	+	+

Кожен з цих алгоритмів має свої переваги та недоліки, і вибір конкретного методу залежить від специфіки завдань та доступних даних.

1.3 Аналіз необхідних даних для навчання моделі

Аналіз необхідних даних для навчання моделі є кроком важливого

підготовчого процесу у розробці системи персоналізованих рекомендацій на основі машинного навчання.

Для ефективної роботи системи персоналізованих рекомендацій необхідно чітко визначити джерела даних. Це можуть бути дані про користувачів, їхню активність на платформі, відгуки, інформація про контент (такий як статті, фільми, музика, ігри) та інші дані, які відображають взаємодію користувачів з системою.

Збір даних включає в себе розробку процедур для отримання інформації з вищезазначених джерел. Важливо враховувати правила конфіденційності та безпеки під час обробки даних користувачів. Після збору дані піддаються обробці, включаючи очищення від помилок та форматування для подальшого використання у моделях.

Зазвичай дані перебувають у невідповідному форматі для навчання моделей і використання цих даних, тому після знаходження необхідної колекції даних (чи створення під час експлуатації додатку, використовують методологію ETL (Extract, Transform, Load).

ETL – це методологія та набір процесів, які використовуються для збору, обробки та завантаження даних з одного джерела в інше [8]. Цей підхід відіграє ключову роль у сфері обробки даних і використовується для підготовки та інтеграції даних з різних джерел для подальшого аналізу та використання в бізнес-системах.

Вилучення (Extract): Перший крок ETL - вилучення даних з різних джерел. Це можуть бути бази даних, файлові системи, зовнішні API, та інші джерела. Під час цього процесу дані збираються та екстрагуються з джерел у структурований формат, придатний для обробки.

Трансформація (Transform): Наступний етап – трансформація даних. Під час цього кроку дані очищаються, перетворюються, об'єднуються та агрегуються з метою підготовки до завантаження. Цей крок може включати в себе фільтрацію непотрібних даних, перетворення форматів, обчислення

нових показників та багато інших операцій.

Завантаження (Load): Останній етап ETL - завантаження оброблених даних в цільову базу даних або сховище даних, яке використовується для подальшого аналізу та звітування. Завантаження може бути виконано у пакетному режимі або в режимі реального часу, залежно від потреб бізнесу.

Основна мета ETL - забезпечити якість даних та їхню готовність до подальшого аналізу. Цей процес допомагає уникнути помилок та незгод у даних, підвищити ефективність бізнес-процесів та сприяє прийняттю обґрунтованих рішень на основі даних. ETL відіграє важливу роль у великих системах обробки даних, таких як дата-склади, бізнес-інтелект системи та системи аналітики [9].

Важливим моментом є обсяг і якість даних. Необхідна накопичувати дані для більш точно результату і слідкувати за якістю, оскільки важко передбачити вплив хибних даних на роботи алгоритмів.

Машинне навчання зазвичай вимагає великої кількості даних для навчання та тестування моделей. Ці дані можуть бути отримані з різних джерел та сервісів. Дані для навчання можуть бути отримані з таких джерел.

Колекції даних для досліджень (Datasets for Research):

– Багато університетів та дослідницьких установ надають доступ до публічних наборів даних, які можуть бути використані для наукових досліджень. Наприклад, "UCI Machine Learning Repository" та "Kaggle Datasets" - це джерела, де можна знайти різні набори даних для машинного навчання.

– Соціальні мережі, такі як Steam, Twitter, Facebook, та LinkedIn, надають API для доступу до даних, які можна використовувати для аналізу настроїв, класифікації користувачів та багатьох інших завдань машинного навчання.

– Веб-скрейпінг дозволяє отримувати дані з веб-сайтів. Це може бути корисним для створення власних наборів даних, особливо для текстових

даних, новин та інших інформаційних ресурсів, але можуть бути проблеми з правами на використання таких даних.

– Відкриті дані. Багато урядів надають публічний доступ до даних, таких як населення, економіка, екологія, транспорт тощо. Ці дані можуть бути використані для різних аналітичних завдань.

– Компанії та організації, які збирають великі обсяги даних, можуть надавати доступ до цих даних через API. Наприклад, сервіси Google Cloud та Amazon Web Services надають доступ до різних даних через свої платформи.

– Колекції даних для розробників. Деякі компанії, такі як Twitter, надають API для розробників, які дозволяють отримувати доступ до їхніх даних для розробки додатків та аналізу.

– Колекції даних для досліджень. Багато університетів та дослідницьких установ надають доступ до публічних наборів даних, які можуть бути використані для наукових досліджень. Наприклад, "UCI Machine Learning Repository" та "Kaggle Datasets" - це джерела, де можна знайти різні набори даних для машинного навчання.

Ці джерела та сервіси надають доступ до різноманітних даних, які можна використовувати для створення та навчання моделей машинного навчання. Вибір конкретного джерела залежить від вашої конкретної задачі та потреб.

1.4 Висновки та постановка задачі

Для створення покращеного методу персоналізованих рекомендацій на основі машинного навчання і розробки застосунку, який використовує даний метод слід спроектувати та програмно реалізувати бібліотеки, утиліти і веб сервіси з використанням удосконаленого алгоритму.

Зважаючи на переваги і недоліки, новий алгоритм має поєднати матричну факторизацію та колаборативний фільтринг, які будуть частково

компенсувати свої недоліки і примножувати переваги. Комбінація даних методів залишить проблема використання алгоритму для нових користувачів, тому слід удосконалити його шляхом додаткового алгоритму, який буде брати до уваги нових користувачів і створювати матрицю вподобань при холодному старті.

При проєктувані системи варто звернути увагу на дані для навчання моделі, якість даних і обсяг. Оскільки запропонований метод не потребує складних вимог до контенту, вибір колекції даних не є проблемою.

В розділі також здійснено порівняльний аналіз існуючих методів персоналізованих рекомендацій та алгоритмів машинного навчання. Детально розглянуто їх особливості при використанні, переваги і недоліки кожного методу, поставлені вимоги до даних для навчання і типові задачі при побудові моделей.

2 ПРОЕКТУВАННЯ МЕТОДУ І ЗАСОБУ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ

2.1 Проектування методу формування персоналізованих вподобань

Розробка ефективної системи рекомендацій вимагає уваги до кількох ключових аспектів, таких як обробка та аналіз даних, вибір підходів до моделювання, а також впровадження та оцінку роботи системи.

До поєднання двох обраних методів відповідно до аналізу, треба усунути недоліки кожного з методів, а саме проблеми холодного старту і досвід рекомендацій для нових користувачів.

Принципово новий алгоритм для побудови матриці вподобань конкретного користувача має бути зрозумілим для реалізації, та мати допустимий час виконання (рис. 2.1).

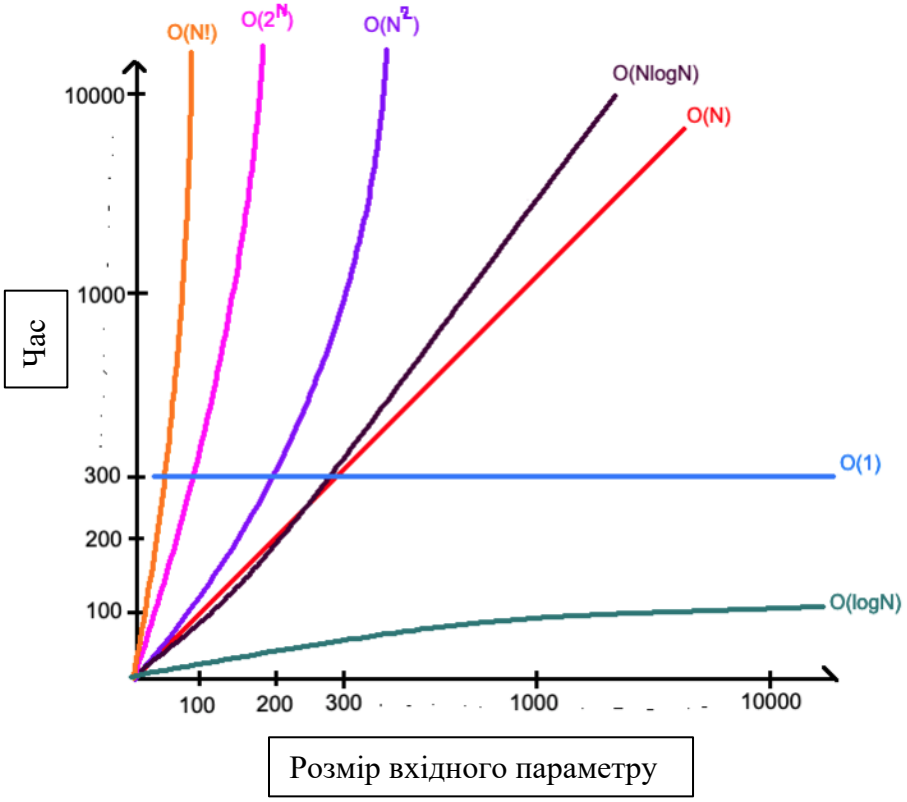


Рисунок 2.1 – Графік алгоритмічної складності

Алгоритмічна складність даного алгоритму визначається кількістю

порівнянь, які потрібно зробити для знаходження шуканого елемента. Основною характеристикою складності є логарифмічна часова складність.

Оскільки цільова продуктивність має бути логарифмічною, це дає спорідненість нового алгоритму до бінарного пошуку, в яку на кожному кроці розмір пошукового діапазону зменшується приблизно вдвічі, оскільки він поділяється на дві частини. Тобто, якщо у вас є список довжиною "n", то алгоритм вимагає логарифмічну кількість порівнянь, яка виражається як O(log n). Це означає, що алгоритм залишається дуже швидким, навіть при зростанні розміру списку.

Бінарний пошук - це швидкий алгоритм пошуку в відсортованому списку. Він працює наступним чином: алгоритм порівнює цільовий елемент з елементом посередині списку. Якщо вони співпадають, пошук завершується. Якщо цільовий елемент менший за центральний елемент, пошук продовжується тільки в лівій половині списку, інакше - тільки в правій. Алгоритм це робить до тих пір, поки не знайде цільовий елемент або діапазон пошуку стане порожнім. Бінарний пошук дуже швидкий і ефективний для великих відсортованих списків.

У загальному випадку логарифмічна складність - це дуже добра характеристика для алгоритмів пошуку, і бінарний пошук відзначається своєю швидкістю при роботі з великими даними.

Блок-схема надає чітке уявлення про послідовність операцій та призначення кожного компонента системи. Дана схема являє собою графічне представлення послідовності кроків, яке використовує блоки різних форм і з'єднує їх лініями, щоб показати порядок виконання дій. Кожен блок у блок-схемі представляє певний етап або операцію, яка виконується у процесі виконання алгоритму

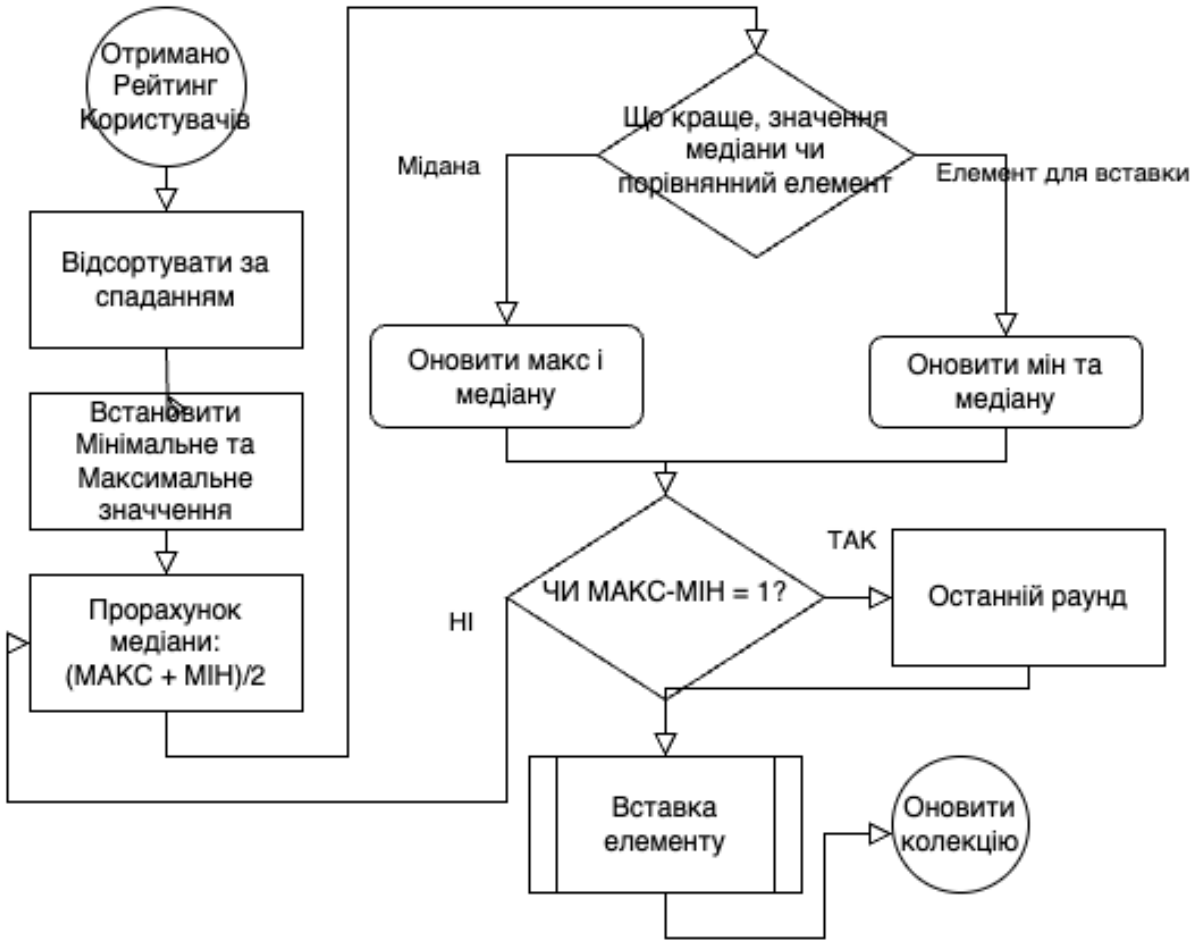


Рисунок 2.2 – Блок схема алгоритму вставки рекомендацій з логарифмічною складністю

Надана схема зображає послідовність дій, які виконуються для вставки певного елемента у впорядкованому масиві даних. Основні етапи включають:

– Старт - початок алгоритму на основі існуючого рейтингу користувача, якщо це не перший запуск алгоритму.

– Впорядкування даних - створення вхідного відсортованого масиву та елемента, який потрібно вставити.

– Встановлення індексів та медіани - встановлення початкового і кінцевого індексів.

– Умова пошуку - перевірка умови, доки ліва границя менша або дорівнює правій границі.

– Обчислення середнього елемента - визначення середнього індексу для поточного діапазону.

– Перевірка на необхідність продовження.

– Вставка залежно від результату перевірки.

– Зупинка - завершення алгоритму.

Блок-схема допомагає візуалізувати і розуміти послідовність дій алгоритму, допомагаючи програмістам та інженерам легше реалізувати цей алгоритм у своєму програмному коді.

Даний опис і блок схема є достатньою для реалізації алгоритму.

2.2 Проектування бібліотеки формування персоналізованих рекомендацій з використанням машинного навчання

В проєктуванні бібліотеки, яка використовуватиме машинне навчання та проводитиме тренування моделі для аналізу результатів, ключовим завданням є поєднання двох методів, а саме матричної факторизації та колаборативного фільтрингу, враховуючи описаний метод на основі бінарного пошуку. Дане поєднання і запропонований алгоритм буде ураховувати потреби нових користувачів та формувати матрицю вподобань для холодного старту.

Під час проєктування системи, важливо звернути увагу на дані для навчання моделі, їх якість і обсяг. Запропонований метод не має високих вимог до складності контенту, тому вибір колекції даних не є проблемою.

Дана бібліотека покликана вирішувати проблеми традиційних методів рекомендаційних систем, а також забезпечити відмінну ефективність для широкого кола використання, незалежно від обсягу та складності даних.

Оскільки, поєднання декількох методів і алгоритмів можуть викликати складнощі при підтримці даного рішення, варто проєктувати бібліотеки зважаючи на доцільне розділення коду задня спрощення подальшої розробки.

Розподілення коду на проєкти та керування залежностями - це критичний аспект будь-якого програмного проекту, який значно полегшує подальший супровід і розвиток програмного забезпечення. Ось чому це важливо:

– Розподілення дозволяє розбити складний код на менші, більш зрозумілі частини. Кожен проєкт або компонент виконує конкретну функцію, і це полегшує розуміння та відлагодження коду для розробників.

– Коли кілька розробників працюють над великим одиничним кодом, конфлікти при злитті можуть бути складними для вирішення. Розподілення на проєкти зменшує ймовірність конфліктів, оскільки кожен проєкт може розроблятися окремо.

– Розподілення сприяє модульності. Стає можливим легко додавати, видаляти або змінювати окремі частини коду, не впливаючи на інші проєкти. Це робить програмний код більш гнучким та швидко змінюваним.

– Інша важлива перевага полягає в керуванні залежностями. В проєкті може бути сказано, які бібліотеки або проєкти залежать від інших, і зручно керувати оновленнями та сумісністю між ними.

– Код може бути відокремленим, що стосується інфраструктури та конфігурації, від бізнес-логіки. Це полегшує підтримку та розгортання на різних середовищах.

– Легкість тестування - ще одна перевага розподілу. Ви можете проводити тестування окремих проєктів, не впливаючи на інші частини системи.

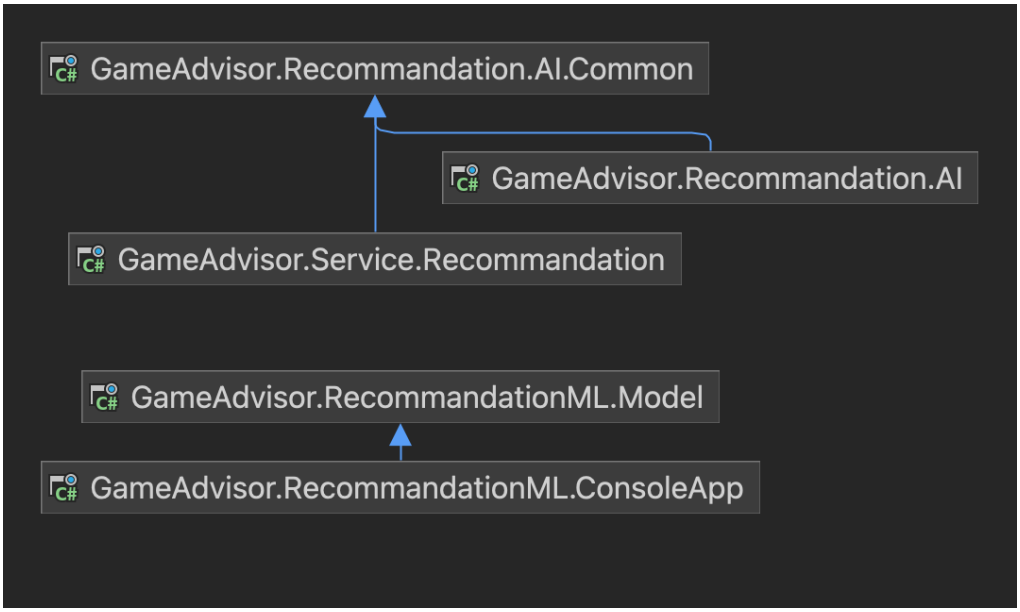


Рисунок 2.3 – Діаграма залежностей створених бібліотек по машинному навчанню

Загалом, розподілення коду на проєкти та правильне управління залежностями спрощує життя розробників та допомагає підтримувати великі програмні проекти.

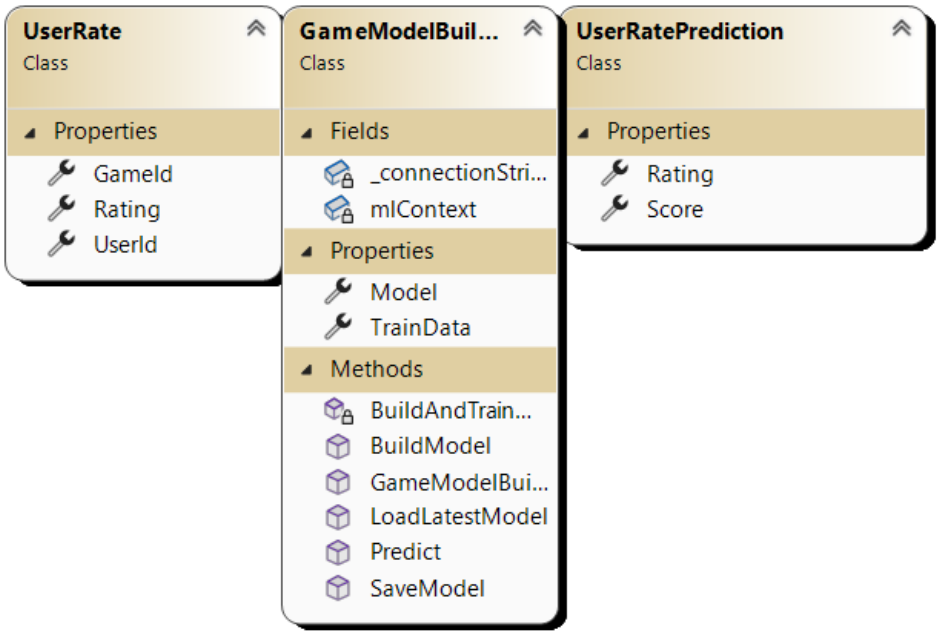


Рисунок 2.4 UML діаграма сутностей уподобань і машинного навчання

Клас UserRate (рис. 2.4) містить властивості, що відображають дані про користувачів та їх оцінки ігор. Властивості UserId, GameId та Rating відповідають ідентифікаторам користувачів, ігор та оцінок.

Клас GameModelBuilder є компонентом для побудови моделі рекомендацій системи для ігор. Він використовує ML.NET (MLContext) для навчання моделі та прогнозування рейтингів для нових користувачів. Клас має методи для побудови, збереження та завантаження моделі, а також для прогнозування рейтингів користувачів.

Клас UserRatePrediction використовується для представлення прогнозованої оцінки Rating та її вірогідності Score для користувача, який вказав свою оцінку гри.

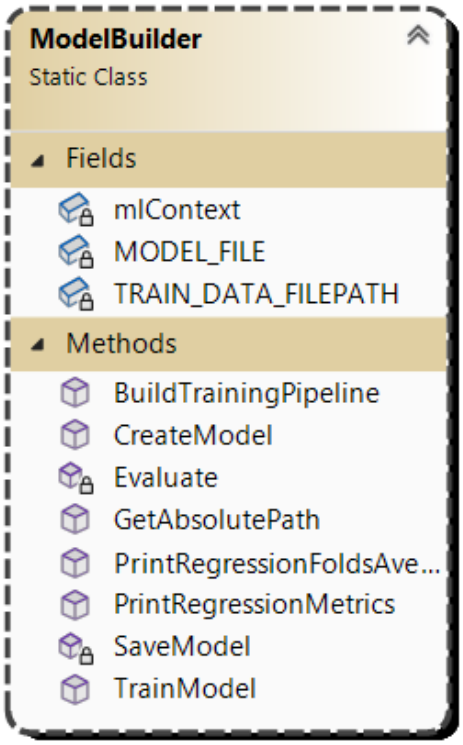


Рисунок 2.5 UML діаграма класу навчання і виконання моделі

Клас ModelBuilder є компонентом для створення та навчання моделі машинного навчання. Він містить методи, які допомагають у побудові моделі регресії або інших моделей, оцінці та збереженні моделі. Основні

методи цього класу включають:

– CreateModel(): Ініціалізує потрібні компоненти для створення моделі.

– BuildTrainingPipeline(): Створює порядок і елементи навчання, яка включає попередню обробку даних, вибір алгоритму навчання та налаштування моделі.

– TrainModel(): Навчає модель за допомогою переданого порядку і елементів навчання, та даних для тренування.

– Evaluate(): Оцінює якість моделі на основі навчальних даних.

– SaveModel(): Зберігає навчену модель для подальшого використання.

– GetAbsolutePath(): Повертає абсолютний шлях до файлу.

– PrintRegressionMetrics(): Виводить метрики регресії, такі як середньоквадратичне відхилення та коефіцієнт детермінації.

– PrintRegressionFoldsAverageMetrics(): Виводить середні метрики регресії для перехресної валідаційних наборів даних.

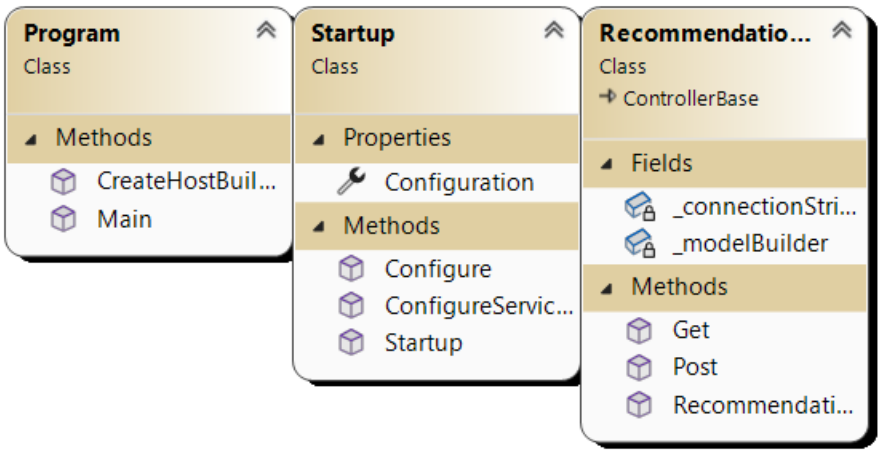


Рисунок 2.6 UML діаграми класів додатку роботи з моделлю

На рисунку 2.6 зображено класи для обробки HTTP-запитів, пов'язаних з рекомендаціями ігор, Налаштування сервісів та обробки HTTP-запитів та клас, який запускає веб-сервер. Методи в класах серверу мають таку функцію:

– ConfigureServices: Додає необхідні сервіси до контейнера

залежностей додатка.

– Configure: Налаштовує проміжні компоненти для обробки HTTP-запитів, включаючи обробку виключень, маршрутизацію, перенаправлення HTTPS.

– CreateHostBuilder: Налаштовує веб-сервер та ініціалізує додаток за допомогою класу Startup.

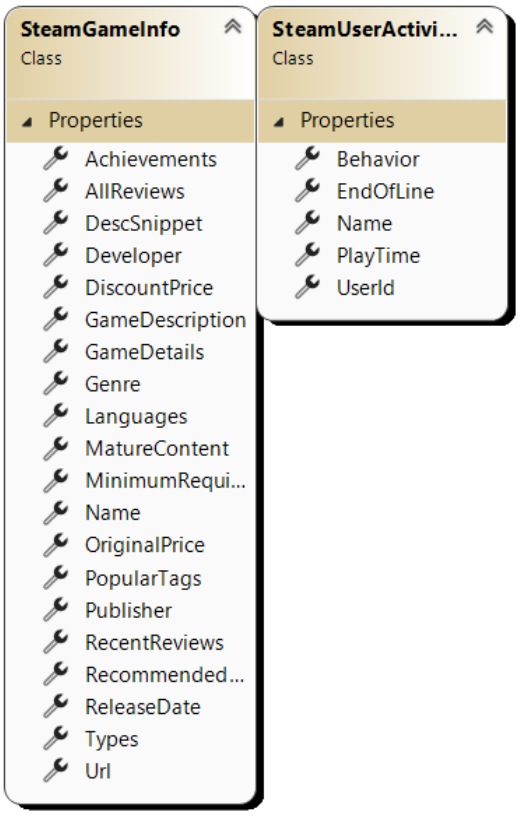


Рисунок 2.6 UML діаграма сутностей колекції даних для навчання моделі

Клас SteamUserActivity – це модель, що представляє активність користувачів у грі на платформі Steam і має такі поля:

– UserId: Ідентифікатор користувача на платформі Steam.

– Name: Ім'я користувача.

– Behavior: Опис поведінки користувача.

– PlayTime: Час гри користувача.

– EndOfLine: Показник закінчення.

Клас SteamGameInfo – це також модель, що містить інформацію про гру на платформі Steam:

– Url: URL-адреса гри.
– Types: Типи гри.
– Name: Назва гри.
– DescSnippet: Уривок опису.
– RecentReviews: Останні відгуки.
– AllReviews: Усі відгуки.
– ReleaseDate: Дата випуску.
– Developer: Розробник.
– Publisher: Видавець.
– PopularTags: Популярні теги.
– GameDetails: Деталі гри.
– Languages: Мови.
– Achievements: Досягнення.
– Genre: Жанр.
– GameDescription: Опис гри.
– MatureContent: Вміст для дорослих.
– MinimumRequirements: Мінімальні вимоги.
– RecommendedRequirements: Рекомендовані вимоги.
– OriginalPrice: Початкова ціна.
– DiscountPrice: Знижена ціна.

Ці класи включають в себе дані про активність користувачів та інформацію про гру на платформі Steam.

Описані класи і представлені у вигляді UML діаграм, представляють спроєктовану систему для підготовки моделей використовуючи машинне навчання.

2.3 Проектування утиліт для машинного навчання

Діаграма залежностей у проєкті є потужним інструментом для візуалізації та аналізу взаємозв'язків між різними компонентами та модулями проєкту. Вона надає глибокий уявлення про те, як окремі частини системи взаємодіють одна з одною, і є важливою частиною супроводу та розвитку програмного забезпечення.

Дана діаграма допомагає виявити, як зміни в одній частині можуть вплинути на інші, а також спрощує процес розробки, супроводу та тестування, оскільки вона дозволяє виділити ключові залежності і сконцентрувати увагу на них. Це важливо не лише для програмістів, але і для всієї команди, що працює над проєктом, включаючи тестувальників, аналітиків і менеджерів.

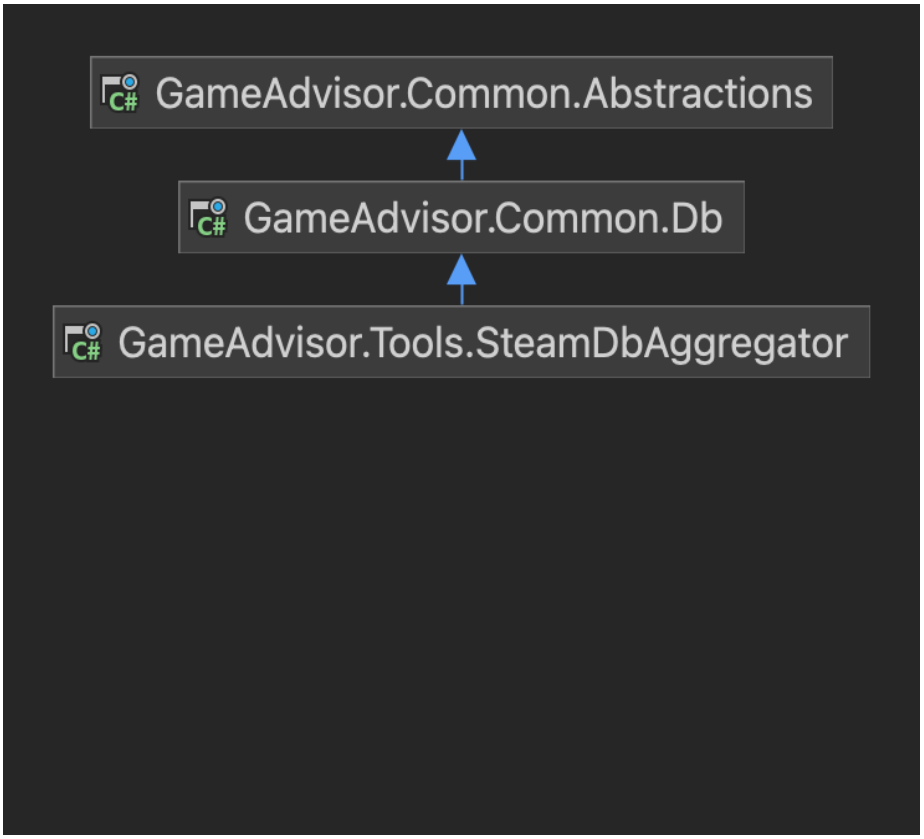


Рисунок 2.7 – Діаграма залежностей утиліти по створенню моделі і роботи з базою даних

Для утиліти по створенню моделі і роботи з базою даних, необхідно створити 3 бібліотеки класів, включаючи абстракції, які дозволять слідувати принципу SOLID.

SOLID - це п'ять основних принципів об'єктно-орієнтованого дизайну, які сприяють створенню масштабованих, підтримуваних та розширюваних програмних систем. Кожна літера у слові SOLID представляє один із цих принципів [10]:

– Принцип єдиної відповідальності (Single Responsibility Principle - SRP): Кожен клас повинен мати лише одну причину для зміни. Це означає, що клас має виконувати лише один вид функціональності.

– Принцип відкритості/закритості (Open/Closed Principle - OCP): Програмні сутності повинні бути відкритими для розширення, але закритими для модифікації. Це можна досягнути через використання абстракцій та інтерфейсів.

– Принцип підстановки Барбари Лісков (Liskov Substitution Principle - LSP): Об'єкти підкласів повинні бути замінюваними за об'єкти базового класу без зміни бажаної коректності програми.

– Принцип інверсії залежностей (Dependency Inversion Principle - DIP): Високорівневі модулі не повинні залежати від низькорівневих модулів. Обидва мають залежати від абстракцій. Цей принцип також стверджує, що деталі повинні залежати від абстракцій, а не навпаки.

– Принцип інтерфейсів (Interface Segregation Principle - ISP): Клієнти не повинні залежати від інтерфейсів, які вони не використовують. Це означає, що класи повинні реалізувати тільки ті методи, які їм потрібні.

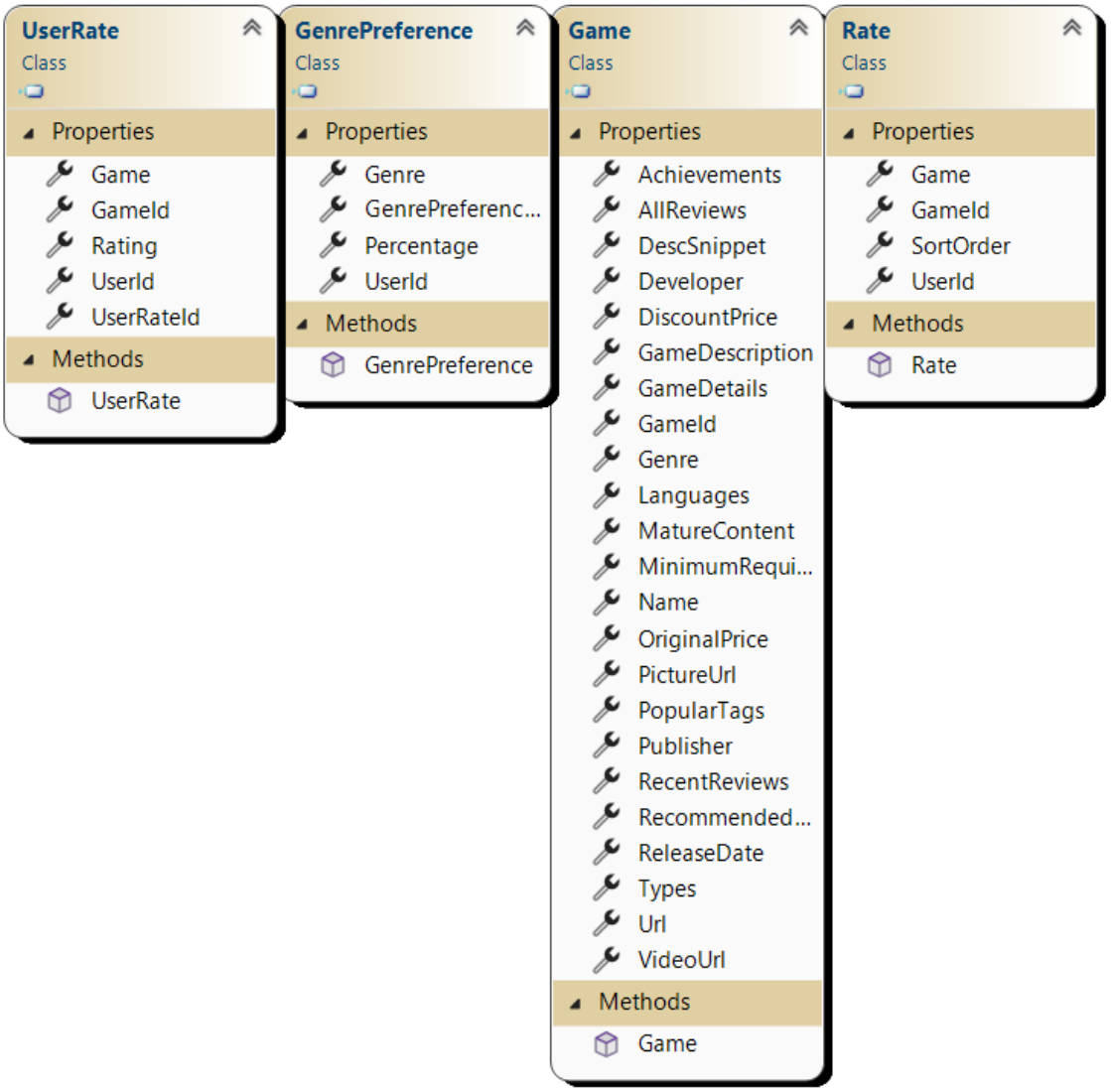


Рисунок 2.8 UML діаграма сутностей при обробки інформації

Моделі класів для утиліт частково перетинаються з полями класів зібраних для збору даних та машинного навчання, оскільки мають подібну структуру, але відповідно до побудови залежностей і архітектури проєктів, не мають бути взаємозалежними, тому уособлюють різні класи. Виключенням є клас GenrePreference:

– GenrePreferenceId: Ідентифікатор вподобання жанру.

– UserId: Ідентифікатор користувача.

– Genre: Назва жанру.

– Percentage: Відсоток, що відображає ступінь вподобання цього жанру користувачем.

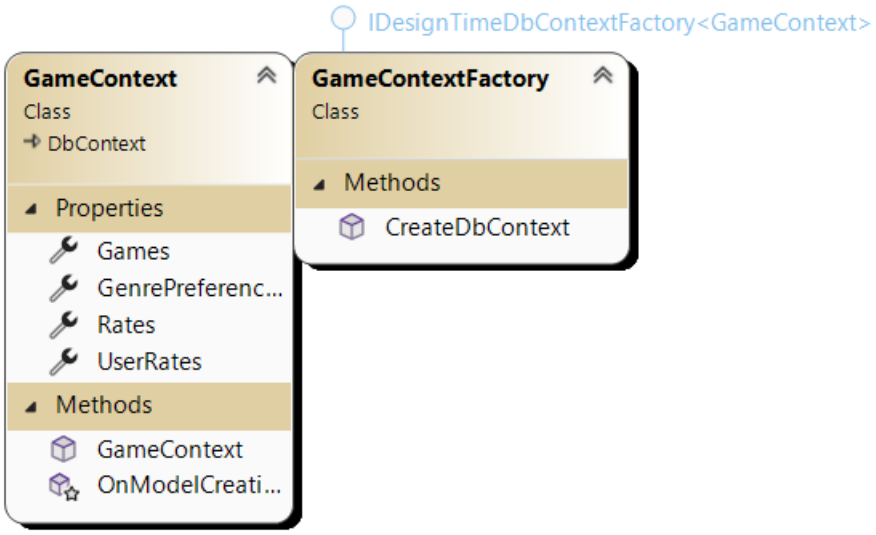


Рисунок 2.9 UML діаграма класів для операцій з базою даних

Клас GameContext є контекстом бази даних для взаємодії з різними таблицями і містить властивості DbSet, які відповідають за роботу з таблицями бази даних. Він також успадковує клас DbContext для використання можливостей Entity Framework. У конструкторі GameContext передаються параметри підключення до бази даних.

Клас GameContextFactory відповідає за створення екземпляра GameContext у випадках, коли потрібно створити контекст бази даних в дизайн-часі, наприклад, при міграціях. У методі CreateDbContext повинна бути реалізована логіка для створення нового контексту бази даних.

2.4 Проектування веб додатку з системою аутентифікації

Проєктуючи безпосередньо веб додатки, слід використати попередні бібліотеки і розмістити залежності відповідно до минулих діаграм. Критично важливо слідувати даним залежностям, оскільки проєкт може стрімко рости, а проблеми циклічних залежностей і неясності відношень стають критичними.

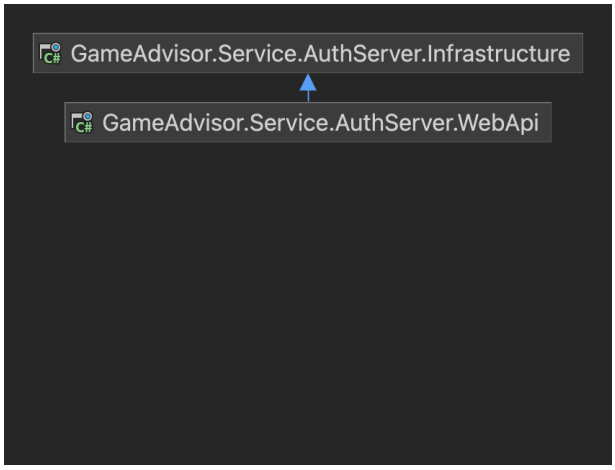


Рисунок 2.10 – Діаграма залежностей АПІ для аутентифікації

Побудувавши діаграми додатків і визначивши їх залежності надає суттєву перевагу у можливості дотримання чистої архітектури.

Чиста архітектура - це підхід до проєктування програмних систем, який прагне розділити програму на окремі логічні шари та компоненти. Його основна ідея полягає в тому, щоб система була легко змінюваною та тестовою, зі збереженням високої якості коду [11].

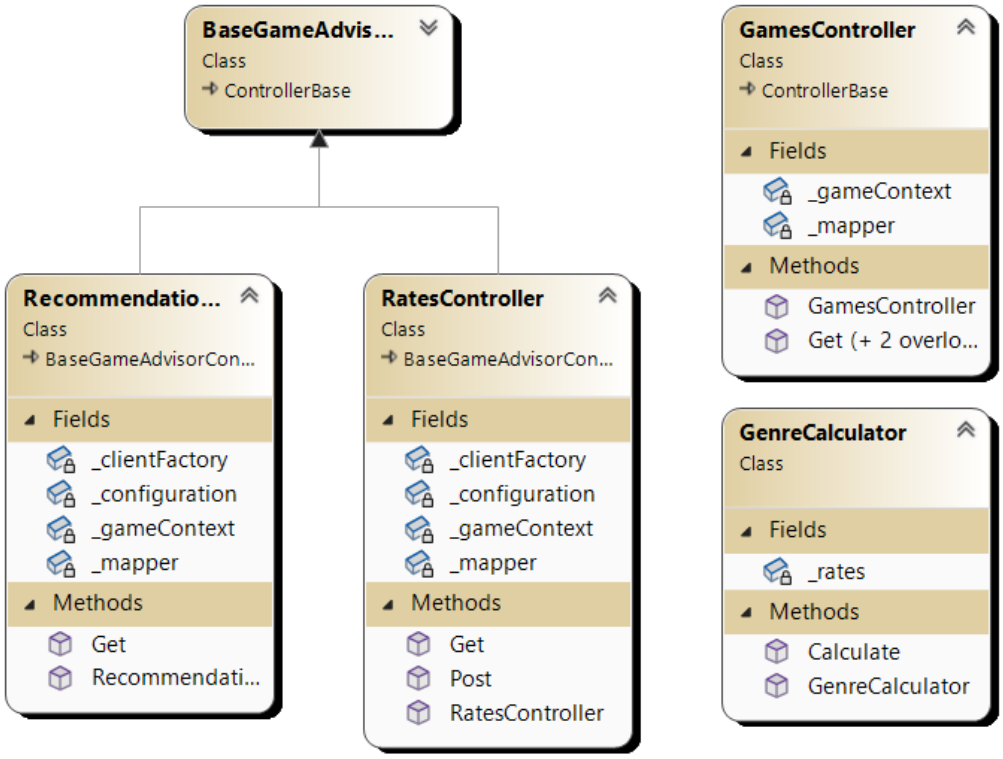


Рисунок 2.11 UML діаграма основних класів веб застосунку

Діаграми класів зображають методи й залежності від абстракцій веб застосунку. Кожен контролер декомпозований відповідно до доменної області задля покращення розуміння і підтримки кодової бази. Для зменшення обсягів кодової бази, трансформації моделей має бути автоматизовано за допомогою спеціалізованих «об'єктів-мепперів», які автоматично мають змогу ініціалізуватти властивості.

Меппер — це шаблон проектування, який використовується для перетворення даних з одного типу об'єкта в інший, як правило, між різними моделями даних або структурами. У розробці програмного забезпечення меппери допомагають перетворювати інформацію з одного представлення в інше, дотримуючись певних правил, угод або конфігурацій [12].

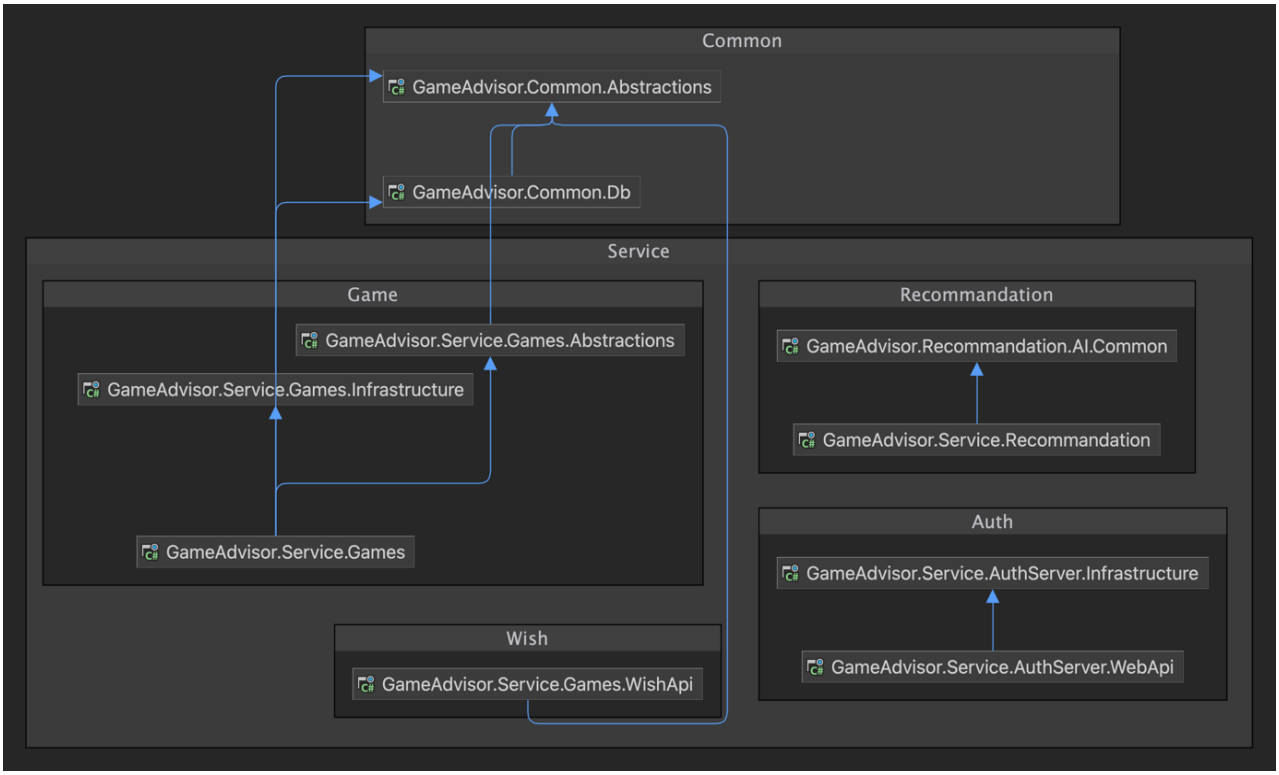


Рисунок 2.12 – Діаграма залежностей веб сервісів

Дана архітектура допомагає забезпечити високу залежність системи від зовнішніх факторів, полегшує тестування та розвиток, а також робить систему більш гнучкою та сумісною з майбутніми змінами.

2.5 Проектування бази даних для додатків

База даних виступає як фундамент, на якому будується весь функціонал програми. Правильне проектування бази даних гарантує, що програма буде ефективною, швидкою та легко розширюваною.

Реляційна база даних (RDBMS) - це тип бази даних, яка зберігає дані в структурованому форматі, використовуючи таблиці з рядками і стовпцями для зберігання інформації. У реляційній базі даних дані організовані у вигляді таблиць, де кожен рядок представляє конкретний запис, а кожен стовпець визначає певний атрибут або характеристику [13].

Реляційні бази даних є широко використовуваними через такі переваги:

– Структурованість даних. Дані зберігаються в таблицях, що спрощує доступ, пошук, та маніпулювання ними.

– Гнучкість. Реляційні бази даних підтримують різні типи запитів, що дозволяє отримувати та оновлювати дані в різних способах.

– Цілісність даних. Вони дозволяють встановлювати обмеження і правила цілісності, щоб гарантувати правильність та цілісність даних.

– Нормалізація. Реляційні бази даних дозволяють нормалізувати дані для зменшення дублювання і забезпечення узгодженості [14].

Використання реляційних баз даних виправдане, коли потрібно зберігати структуровану інформацію, яка має взаємозв'язки між різними частинами даних. Це дозволяє ефективно зберігати, оновлювати, та отримувати доступ до даних, а також забезпечує даним цілісність та гнучкість у використанні.

Детальне проектування бази даних допомагає запобігти проблемам у подальшій розробці, спрощує процес зберігання, оновлення та отримання даних.

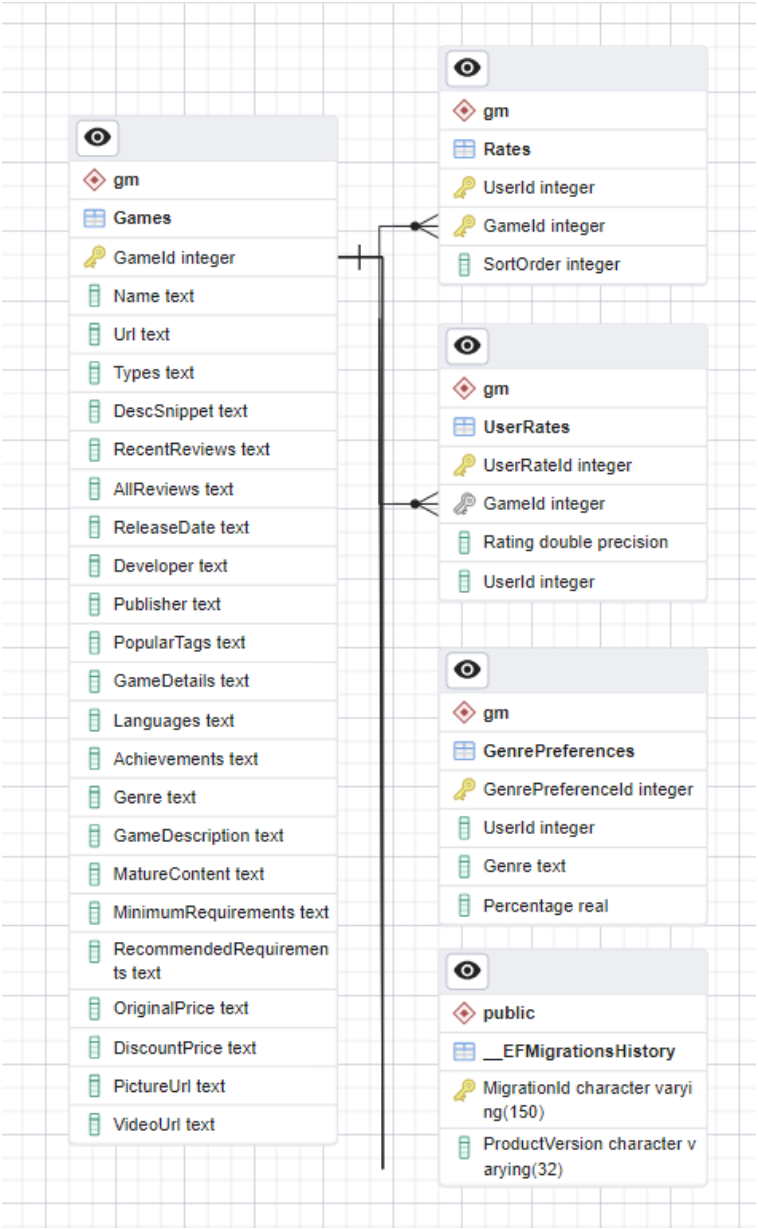


Рисунок 2.13 – Діаграма структури бази даних

На діаграмі (рис 2.13) візуалізовано основні таблиці і зв’язки у базі даних, а саме «Ігри», «Рейтинги», «Рейтинги Користувачів системи», «Уподобання» та інфраструктурна таблиця для міграцій.

Таблиця «Ігри» (Games) містить інформацію про ігри. Основні поля мають включати унікальний ідентифікатор "GameId", а також дані про назву гри ("Name"), URL ("Url"), тип гри ("Types"), короткий опис ("DescSnippet"), відгуки ("RecentReviews", "AllReviews"), дату виходу ("ReleaseDate"), розробника ("Developer"), видавця ("Publisher"), популярні теги

("PopularTags"), деталі гри ("GameDetails"), мови ("Languages"), досягнення ("Achievements"), жанр ("Genre"), опис гри ("GameDescription"), вміст для дорослих ("MatureContent"), мінімальні та рекомендовані вимоги ("MinimumRequirements", "RecommendedRequirements"), початкову та знижену ціну ("OriginalPrice", "DiscountPrice"), URL картинки ("PictureUrl") і URL відео ("VideoUrl"). Ця база даних має бути наповнена за допомогою спроектованої утиліти для імпорту даних. Дана таблиця матиме унікальний ключ "PK_Games", який є первинним ключем ідентифікації записів, визначений за допомогою поля "GameId". Така структура бази даних дозволяє зберігати та організовувати різноманітну інформацію про ігри для подальшого використання у веб застосунку та при формуванні персоналізованих рекомендацій.

«Рейтинги» ("GenrePreferences") - це таблиця, яка міститиме інформацію про вподобання жанрів користувачів. У таблиці є поля: "GenrePreferenceId", яке є унікальним ідентифікатором вподобань жанрів, "UserId", що є ідентифікатором користувача, "Genre", яке містить назву жанру, та "Percentage", який представляє відсоток вподобання для цього жанру. Дана таблиця призначена для зберігання відомостей про вибрані користувачами жанри та їх відсоткове співвідношення.

Таблиця з назвою «Рейтинги Користувачів системи» ("Rates") використовуватиметься для зберігання оцінок користувачів для ігор. У таблиці є поля "UserId", що є ідентифікатором користувача, "GameId", яке є ідентифікатором гри, та "SortOrder", що визначає порядок сортування.

Первинним ключем цієї таблиці є "PK_Rates", який складається з комбінації полів "UserId" та "GameId". Це забезпечує унікальність комбінацій користувача та гри для кожної оцінки. Також, є зовнішній ключ "FK_Rates_Games_GameId", який посилається на таблицю "Games" та забезпечує зв'язок між полем "GameId" у таблиці "Rates" та "GameId" у таблиці "Games". Також, має бути створений індекс "IX_Rates_GameId" для

поля "GameId", що підвищить швидкодію запитів, які використовують це поле для пошуку та сортування.

Остання з основних таблиць «Вподобання» ("UserRates") призначена для зберігання рейтингів, які користувачі виставляють іграм. У таблиці мають бути наступні поля: "UserRateId", що є унікальним ідентифікатором рейтингу користувача, "GameId", яке є ідентифікатором гри, "Rating", що представляє оцінку користувача, та "UserId", що є ідентифікатором користувача.

Спроєктовані таблиці мають мати такі відношення:

– Games та UserRates - одна гра може мати багато оцінок від користувачів (один до багатьох). Користувачі надають рейтинг кожній грі, що відображається в таблиці UserRates. Отже, у таблиці UserRates поля GameId та UserId вказують на GameId та UserId в таблиці Games.

– Games та Rates – має певну схожість на попереднє відношення, однак Rates може включати у себе лише перелік уподобаних/сортованих ігор користувача, без конкретних оцінок. Однак, зв'язок залишається один до багатьох, кожен запис Rates вказує на конкретну гру в таблиці Games.

– Games та GenrePreferences - дане відношення дозволяє кожному користувачеві визначити свої вподобання щодо різних жанрів в іграх. Кожен запис GenrePreferences вказує на конкретну гру в таблиці Games, забезпечуючи зв'язок між жанром і конкретною грою.

Створена діаграми з таблицями та проектування бази даних допоможе уникнути проблем з подальшим розвитком додатку і сприяє його швидкому впровадженню [15]. Крім того, воно сприяє оптимізації роботи застосунку, забезпечуючи швидкий доступ до даних та зручний їх пошук.

2.6 Вибір засобів реалізації

Для реалізації бібліотек машинного навчання, утилік та веб застосунку було обрано мову програмування C# в поєднанні з .NET

Framework. C# представляє собою об'єктно-орієнтовану мову програмування з безпечною системою типізації, розроблену для платформи .NET. Синтаксис C# подібний до мов програмування, таких як C++ та Java. Вона володіє строгою статичною типізацією та підтримує такі концепції, як поліморфізм, перевантаження операторів, властивості, атрибути та винятки.

Ця мова виникла, спираючись на досвід різних мов програмування, таких як C++, Delphi, Модула і Smalltalk, і виключила деякі моделі, що виявились проблематичними при розробці програмних систем, наприклад, множинне спадкування класів.

Однією з основних переваг обраної мови та платформи є їхні можливості. Є декілька реалізацій платформ, які працюють з C#, такі як .NET, .NET Core, .Mono і DotGNU. Проте, найбільш привабливою для використання є платформа .NET, оскільки її розробником виступає компанія Microsoft, і вона є найстарішою та повною.

.NET Framework, також відома як .NET, є центральною платформою, яка дозволяє створювати різноманітні програми та веб-додатки на різних мовах програмування, включаючи C#. Ця платформа була створена як альтернатива Java, і вона знайшла своє застосування як у традиційному розробці програмного забезпечення, так і в веб-програмуванні [16].

Найважливішою особливістю .NET є те, що вона підтримує різні мови програмування, а C# є однією з найпопулярніших серед них. Вона відкриває перед розробником безліч можливостей, адже сама платформа .NET розроблена з використанням C# [17].

Розробка веб додатків відбувається на платформі ASP.NET, що входить до платформи .NET. Її вибір також зумовлений даним по безпеці від компанії Positive Technologies (табл. 2.1) [18].

Таблиця 2.1 – Загрози по технологіям

PHP	Доля сайтів (%)	Java	Доля сайтів (%)	ASP.NET	Доля сайтів (%)
Cross-Site Scripting	90	Cross-Site Scripting	80	Cross-Site Scripting	73
Credential/Session Prediction	86	Fingerprinting	60	Brute Force	73
Brute Force	81	Brute Force	45	Fingerprinting	55
Information Leakage	67	Credential/Session Prediction	45	Cross-Site Request Forgery	55
SQL Injection	62	Server Misconfiguration	35	Credential/Session Prediction	45
Fingerprinting	43	Information Leakage	30	Information Leakage	45
Cross-Site Request Forgery	43	XML External Entities	30	Server Misconfiguration	36
Server Misconfiguration	29	SQL Injection	25	Application Misconfiguration	36
Insufficient Authorization	29	Cross-Site Request Forgery	25	XML External Entities	36
Application Misconfiguration	19	Insufficient Authorization	15	SQL Injection	27

З аналізу даних стає очевидним, що серед популярних технологій найбільш безпечною платформою для розробки хмарної платформи є ASP.NET. В сучасному світі існує безліч різних систем управління базами даних (СУБД), включаючи MS SQL, MySQL, MongoDB, NoSQL,

PostgreSQL та інші. У зв'язку з різноманітними відносинами у даному додатку, вибір був зроблений на користь реляційної бази даних.

Як базу даних було обрано Postgres. Це обрано через її зручну інтеграцію з C#, вбудований рівень захисту, швидкодію, велику спільноту та інші переваги.

PostgreSQL (або Postgres) - це потужна об'єктно-реляційна система управління базами даних, яка відкрита, надійна та розширювана. Вона має великий набір функцій, що дозволяють здійснювати операції з даними, такі як взаємодія зі складними структурами, виконання операцій атомарності та транзакційності, підтримка ACID (atomicity, consistency, isolation, durability) властивостей, робота з геопросторовими даними, індексами тощо [19].

Запропонована база даних є одною з найбільш розповсюджених інструментів для управління базами даних у світі відкритого програмного забезпечення. Її популярність полягає в тому, що вона є відкритою і має велику спільноту користувачів та розробників, яка постійно внесення вдосконалення.

Postgres SQL також забезпечує максимальний рівень безпеки за рахунок інтеграції мережевої безпеки з сервером безпеки. Забезпечується обмежений доступ користувачів до записів відповідно до їхніх привілеїв. Дані зберігаються на окремому сервері з обмеженням несанкціонованого доступу.

Реляційна база даних оптимізує передачу даних через мережу, передаючи лише результати запитів, що покращує час відгуку та ефективність. Обслуговування Postgres серверу досить просте, включаючи можливість змін в структурі даних та резервне копіювання без зупинки роботи сервера.

Для реалізації веб застосунку для надання рекомендацій було обрано ASP.NET MVC, фреймворк, що використовує шаблон Model-View-Controller (MVC) [20]. MVC дозволяє розділити додаток на компоненти, такі

як модель, представлення та контролер, спрощуючи розробку та тестування. Такий підхід дозволяє ефективно керувати логікою введення, бізнес-логікою та інтерфейсною логікою, що полегшує розробку та спільну роботу великих розробничих команд.

ASP.NET MVC забезпечує більшу гнучкість та гнучкість у порівнянні з альтернативами, оскільки він побудований на архітектурі Model-View-Controller, де кожен компонент має свою визначену роль:

– Моделі (Model): Вони представляють логіку додатку та управляють даними. Моделі можуть взаємодіяти з базою даних та виконувати бізнес-логіку. Це дозволяє ефективно керувати даними та їх обробкою.

– Представлення (View): Відповідають за відображення даних користувачу. Представлення може бути створене на основі даних моделі. Вони відображають інтерфейс, з яким взаємодіє користувач.

– Контролери (Controller): Вони обробляють взаємодію користувача, передачу даних моделі та вибір представлення для відображення. Контролери допомагають виконувати логіку введення та взаємодії з моделлю.

Такий розподіл логіки дозволяє зосередитися на кожному аспекті додатку окремо, полегшуючи розробку та забезпечуючи чітке розділення обов'язків між командами розробників.

ASP.NET MVC також надає можливість розширювати інфраструктуру маршрутизації, що важливо для великих команд та веб-розробників, які потребують високого рівня контролю над поведінкою програми [21].

Зважаючи на вибір мови програмування C# та платформи ASP.NET є доречним використовувати бібліотеки того ж самого стеку, як проєкт компанії Microsoft під назвою ML.NET.

ML.NET - це відкрита платформа для машинного навчання, розроблена Microsoft, яка дозволяє розробникам .NET створювати моделі

машинного навчання та вбудовувати їх у свої додатки на мові програмування C#. Вона надає доступність і простоту використання для розробників, що працюють з екосистемою .NET, і може бути використана в різних областях від рекомендаційних систем до обробки природної мови та аналізу даних. Дана бібліотека забезпечує широкий спектр інструментів, таких як бібліотеки для побудови моделей (які забезпечують підтримку для навчання з учителем і без учителя), оптимізовані алгоритми, можливість використання вбудованих моделей, що вже навчені та можливість інтеграції з іншими .NET фреймворками [22].

Для розробки користувацького інтерфейсу варто використовувати фреймворк, задля уникнення рутинних задач та перевикористання компонентів. Один з найпопулярніших і найбільш розвинутих це Angular.

Angular - це популярний фреймворк розробки веб-інтерфейсів, розроблений Google, який використовується для створення односторінкових застосунків (Single Page Applications, SPA). Він надає зручний інструментарій для розробки високопродуктивних, масштабованих і динамічних веб-застосунків.

Переваги використання Angular для розробки користувацького інтерфейсу:

– Angular використовує архітектуру MVVM (Model-View-ViewModel), що спрощує структуру програми та дозволяє легко розділити логіку додатку та його представлення.

– Даний фреймворк використовує HTML як мову для опису веб-інтерфейсу, що робить код більш читабельним та легким для розуміння.

– Функція двосторонньої прив’язки - функція дозволяє автоматично оновлювати користувацький інтерфейс при зміні даних в моделі та навпаки, що робить роботу з даними більш простою.

– Компонентна Структура фреймворку, який побудований на основі структури, що дозволяє розбити додаток на невеликі, перевикористовувані

компоненти, що полегшує розробку та підтримку коду.

– Angular надає широкий спектр інструментів для тестування, дебагу, підтримки та пакування додатків.

– Спільнота: фреймворку і бібліотек, яка постійно актуалізується, розробляє додатки, плагіни та розширення,.

– Підтримка мов програмування JavaScript і TypeScript. Angular підтримує JavaScript, а також TypeScript - мову програмування, яка надає статичну типізацію та багато інших функцій, що полегшують розробку великих проектів [23].

Для написання коду на C# є три найбільш популярні рішення, а саме Visual Studio, Rider – як повноцінні інтегровані середовища розробки (IDE), чи Visual Studio Code – як редактор коду, з широким вибором плагінів. Кожне рішення має ряд переваг, і відповідно до потреб кожного розробника, може бути обраний будь-який з цих засобів.

Rider від JetBrains - це кросплатформна IDE, спеціально розроблена для розробників .NET. Він підтримує різні мови, включаючи C#, VB.NET, F# і ASP.NET, серед інших. Вона пропонує багатий набір функцій, таких як аналіз коду, інструменти рефакторингу, вбудований налагоджувач і підтримка розробки Unity. Rider добре інтегрується з ReSharper, іншим продуктом JetBrains, надаючи розширений аналіз коду, навігацію по коду та пропозиції (рис. 2.14).

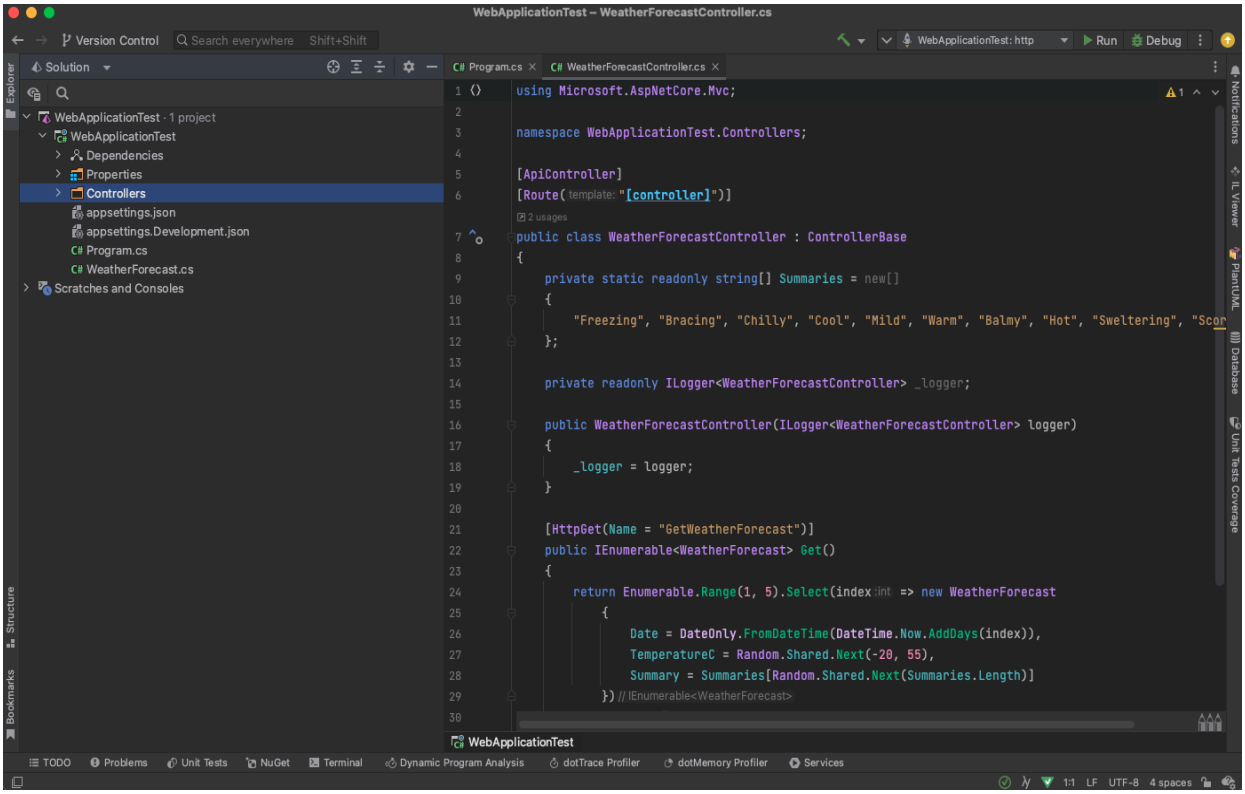


Рисунок 2.14– Головне вікно JetBrains Rider IDE

Будучи кросплатформним, Rider працює на Windows, macOS і Linux, що робить його привабливим вибором для розробників, які використовують різні операційні системи [24].

Visual Studio від Microsoft — це комплексна та надійна IDE, яка підтримує різні мови програмування та фреймворки, включаючи .NET. Вона має широкий набір функцій, включаючи потужний налагоджувач, IntelliSense для завершення коду, інструменти профілювання та багатий набір розширень. Visual Studio забезпечує цілісне середовище розробки для програм .NET, веб-розробки, хмарних служб, мобільних програм та ігор, але доступна виключно для операційної системи Windows [25].

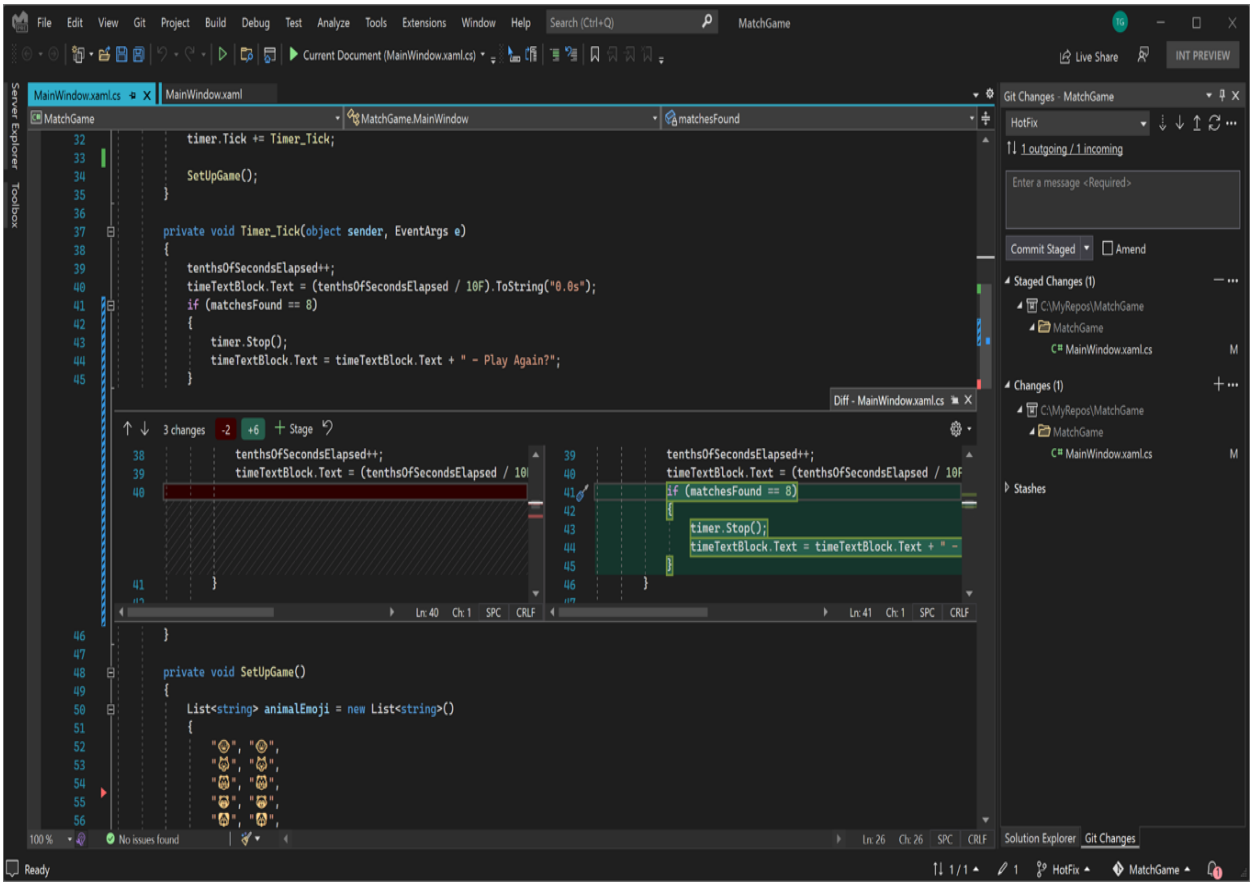


Рисунок 2.15– Головне вікно Microsoft Visual Studio

VS Code (Visual Studio Code) — це легкий редактор коду з відкритим кодом, розроблений Microsoft. Хоча це не повноцінна IDE, як Visual Studio або Rider, вона забезпечує надійну підтримку розробки .NET через розширення. Його можна налаштовувати завдяки величезному магазину розширень, що пропонує різні плагіни для розробки .NET. VS Code доступний у Windows, macOS і Linux і відомий своєю швидкістю, простотою використання та універсальністю. Розробники цінують його гнучкість і можливість адаптувати його до своїх конкретних потреб розробки [26].

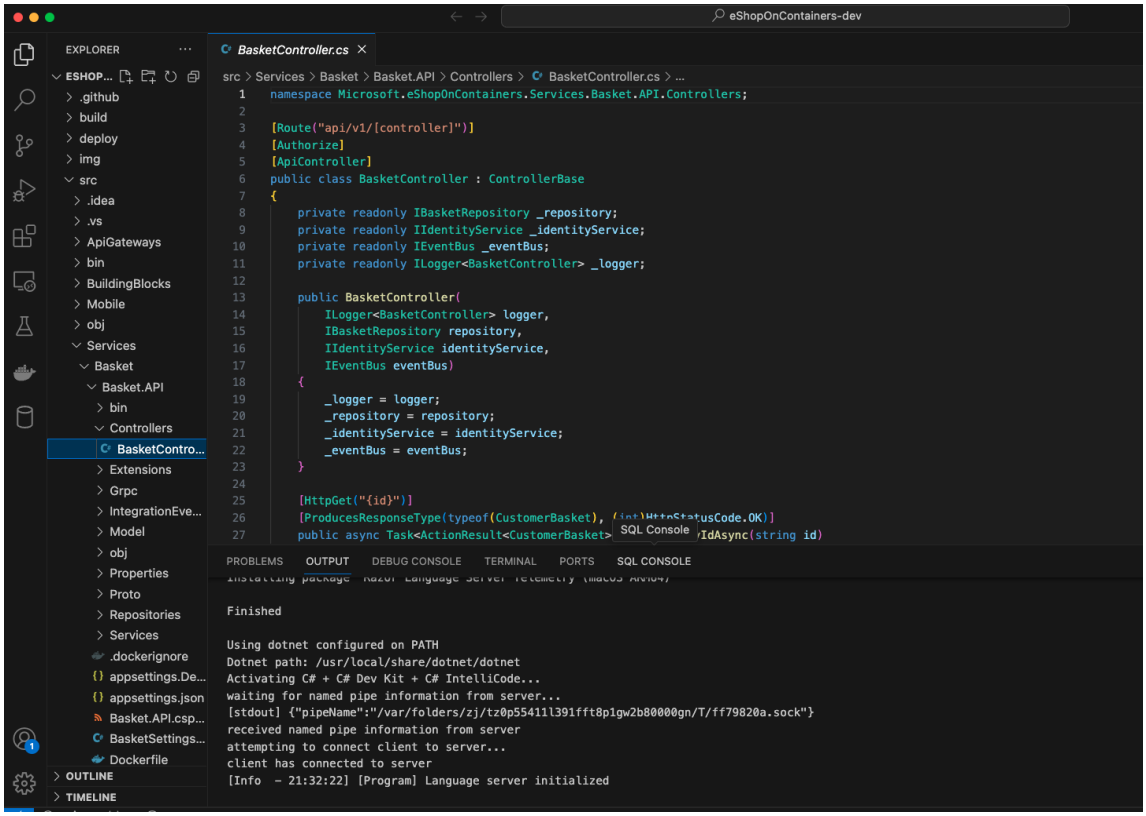


Рисунок 2.16 – Головне вікно Visual Studio Code

Для полегшення підтримки утиліт, бібліотек і застосунку, варто зважати на доступність написання коду використовуючи різні операційні системи, так і зважати на вподобання розробників, які впилвають на продуктивність. Згідно цього, обрано формування проєктів використовуючи загально підтримувані конфігурації доступні для цих трьох засобів, такі як “editor.config”, “csproj” та “sln” файли.

Цей вибір технологій, такий як C#, ASP.NET, ML.NET, Agnular та Postgres, допомагає створити ефективні і розширювані застосунки, як утілити, так і веб додатки.

2.7 Висновки

В проектування різних аспектів створення засобу персоналізованих рекомендацій, було розглянуто і вивчено важливі аспекти і складності цього завдання. Було розглянути методи і алгоритми, які можна використовувати

для створення персоналізованих рекомендацій. Даний етап є фундаментом проекту та визначає ключові аспекти аналізу даних і рекомендацій. За допомогою цього нового методу, усунуто ключові недоліки проаналізованих методів і алгоритмів рекомендацій.

Також було спроєктовано новий метод, який є поєднанням машинного навчання, методу рекомендацій і новоствореного методу побудови матриці вподобань по принципу бінарного пошуку.

Було створено діаграми корисних утиліт для роботи з машинним навчанням є важливим аспектом проекту і розглянуто способи підготовки та обробки даних, а також важливість забезпечення якості даних для навчання моделей.

База даних є основою для зберігання даних і результатів обчислень. Тож було досліджено і здійснений вибір бази даних, та проектування її структури. Під час проєктування і реалізації системи персоналізованих рекомендацій був проведений аналіз різних фреймворків та бібліотек, необхідних для розробки спроєктованих класів і застосунків. Цей аналіз включав у себе вивчення особливостей кожного з інструментів, їхні переваги та сумісність стеку технологій та можливості інтеграції з іншими компонентами проекту.

На основі цього аналізу було вибрано набір фреймворків і бібліотек, які найбільше відповідали вимогам проєкту. Ці інструменти будуть використані для створення класів, модулів і застосунків, які утворюють основну частину системи персоналізованих рекомендацій.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ВПРОВАДЖЕННЯ МЕТОДУ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ

3.1 Розробка сервісу аутентифікації користувачів

Рішення складається з проєктів і файлів у сервісі аутентифікації користувачі, компіляцією якої можна керувати використовуючи Visual Studio чи інший редактор коду. Проєкти та файли в цьому сервісі відображаються у наступному вигляді:

– Проєкти є верхнім рівнем організації. Вони представляють собою логічні групи або області, в межах яких користувачі можуть створювати та керувати файлами, а також посилатись на ці проєкти з інших проєктів ефективно перевикористовуючи код. Кожен проєкт може мати свої унікальні налаштування та дозволи на доступ.

– Файли представляють собою конкретні ресурси або об'єкти, які належать до певного проєкту. Як правило, використовуючи мову програмування C#, кожен файл має мати один і тільки один клас.

Ця структура допомагає користувачам системи легко організовувати та керувати своїми ресурсами для аутентифікації, забезпечуючи впорядковану та логічну організацію проєктів та файлів.

Для спрощення побудови системи аутентифікації, варто використовувати загальні принципи перевірки доступів, такий як модель керування доступом на основі ролей (RBAC).

Даний підхід - метод регулювання доступу до комп'ютерних або мережевих ресурсів на основі ролей окремих користувачів у системи, або сімейства систем. У цьому контексті доступ є здатністю окремих користувачів виконувати певне завдання, наприклад, переглядати, створювати або змінювати дані. Ролі визначаються відповідно до компетенції, повноважень та відповідальності в межах системи. Після належного впровадження, RBAC дозволяє користувачам здійснювати

широке коло авторизованих завдань шляхом динамічного регулювання їхніх дій відповідно до гнучких функцій, взаємозв’язків та обмежень [27].

Одна з найвідоміших бібліотек для керування користувачами, яка легко інтегрується з ASP.NET – це Identity Platform.

ASP.NET Identity - це платформа, яка надає механізми для аутентифікації і авторизації користувачів в програмному забезпеченні. Вона забезпечує інструменти для керування доступом до різних ресурсів системи та контролю за обліковими записами користувачів [28].

Переваги використання Identity Platform включають:

– Забезпечення безпеки через різні механізми аутентифікації, такі як багаторівнева автентифікація, одноразові паролі, двофакторна аутентифікація та інші.

– Платформа може мати готові інтеграції з іншими інструментами і сервісами, що спрощує розробку та інтеграцію аутентифікації в додатки.

– Здатність керувати правами доступу користувачів до різних ресурсів системи, обмеження доступу до функціональності або даних в залежності від ролей або прав користувача.

– Спрощення роботи з обліковими записами користувачів: Дозволяє легко керувати обліковими записами користувачів, змінювати паролі, відновлювати доступ та інше.

– Можливість розширення функціоналу та налаштувань для відповідності конкретним потребам проєкту.

Для інтеграції сервісу авторизацію з платформою, необхідно реалізувати класи IdentityDbContext, для комунікації з базою даних, передавши такі сутності в системі як AppUser, IdentityRole.

Сам клас Startup має включати код налаштування залежностей. Дану конфігурацію можна виконати за допомогою методів розширення AddIdentity, з підключенням до бази даних передавши конфігурацію у методи AddDbContext та AddEntityFrameworkStores.

Після цього, необхідно налаштувати аудентифікацію за допомогою токена використовуючи метод AddAuthentication для схеми JwtBearerDefaults.AuthenticationScheme.

Загальний вигляд сервісу має відповідно до проектування 2 шари, це шар комунікації з базою даних і веб інтерфейс для комунікації (рис 3.1).

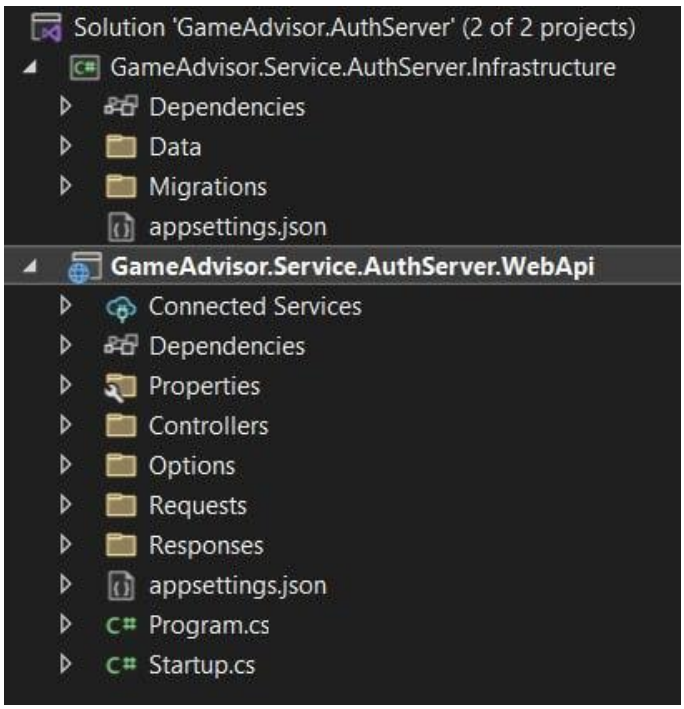


Рисунок 3.1– Вікно рішення з файлами проєктів для сервісу аутентифікації користувачів

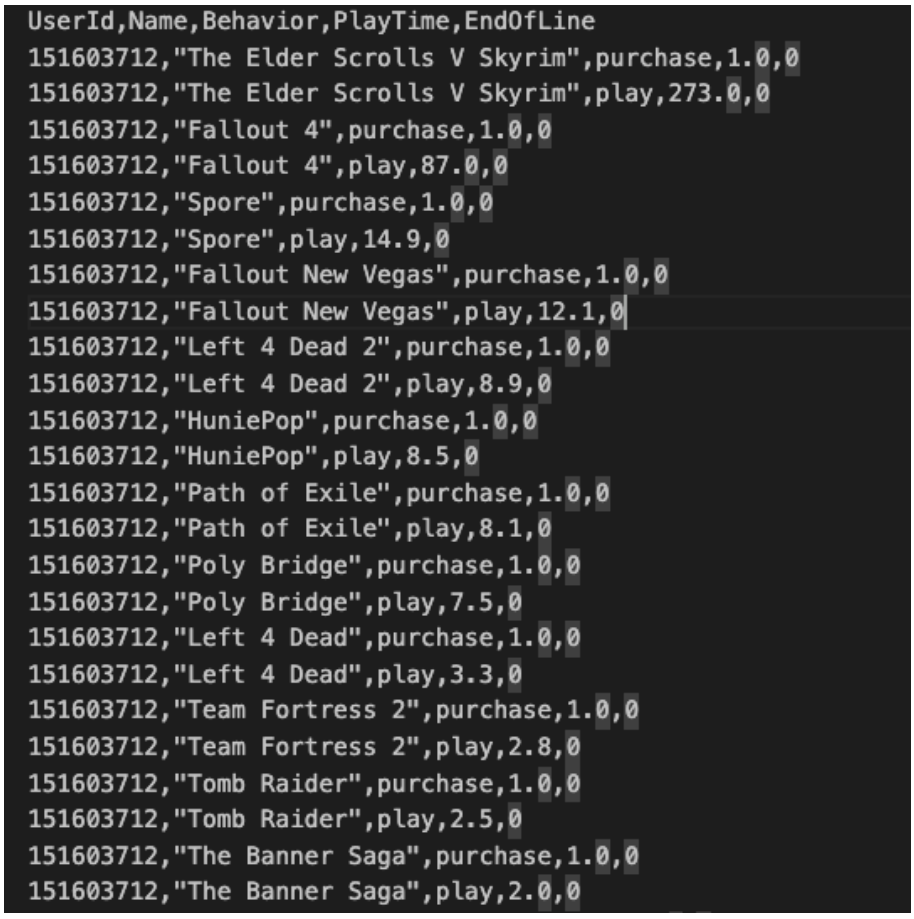
Сервіс аутентифікації використовує Idenity Platform, яка входить до проєктів .NET Foundation, завдяки чому має чудову документацію. Згідно документації заключним етапом є додавання перевірок і чергу фільтрів веб застосунку за допомогою UseAuthentication та UseAuthorization.

3.2 Розробка бібліотеки формування персоналізованих рекомендацій

Проєкт бібліотеки алгоритмів персоналізованих рекомендацій є ключовим етапом в розробці імовірностної системи рекомендацій.

Під час реалізації варто розпочати з підготовка даних для навчання моделі, а саме збір та підготовка даних, таких як інформація про користувачів, предмети, рейтинги, взаємодії.

Відповідно до ідеї кінцевого застосунку чудово підходять данні Steam, які містять в собі дані про ігри та вподобання користувачів з унікальним ідентифікатором. Steam надає CSV (comma-separated values) файли з діями користувачів, який можна використовувати для навчання і перевірки моделі (рис. 3.2).



```
UserId,Name,Behavior,PlayTime,EndOfLine
151603712,"The Elder Scrolls V Skyrim",purchase,1.0,0
151603712,"The Elder Scrolls V Skyrim",play,273.0,0
151603712,"Fallout 4",purchase,1.0,0
151603712,"Fallout 4",play,87.0,0
151603712,"Spore",purchase,1.0,0
151603712,"Spore",play,14.9,0
151603712,"Fallout New Vegas",purchase,1.0,0
151603712,"Fallout New Vegas",play,12.1,0
151603712,"Left 4 Dead 2",purchase,1.0,0
151603712,"Left 4 Dead 2",play,8.9,0
151603712,"HuniePop",purchase,1.0,0
151603712,"HuniePop",play,8.5,0
151603712,"Path of Exile",purchase,1.0,0
151603712,"Path of Exile",play,8.1,0
151603712,"Poly Bridge",purchase,1.0,0
151603712,"Poly Bridge",play,7.5,0
151603712,"Left 4 Dead",purchase,1.0,0
151603712,"Left 4 Dead",play,3.3,0
151603712,"Team Fortress 2",purchase,1.0,0
151603712,"Team Fortress 2",play,2.8,0
151603712,"Tomb Raider",purchase,1.0,0
151603712,"Tomb Raider",play,2.5,0
151603712,"The Banner Saga",purchase,1.0,0
151603712,"The Banner Saga",play,2.0,0
```

Рисунок 3.2 – CSV файл з даними взаємодії користувачів з іграми

Наведений CSV файл для навчання моделі містить наступні стовпці:

– UserId: Ідентифікатор користувача, який використовує гру.
– Name: Назва гри, з якою пов'язана відповідна дія користувача.
– Behavior: Тип дії, яку користувач виконав щодо гри (наприклад,

покупка, гра тощо).

– PlayTime: Кількість годин, які користувач витратив на гру.

– EndOfLine: кінець запису.

Для визначення для рівня вподобання певної гри варто врахувати час витрачений у грі відносно часу її повного проходження. Тобто, гравець проводить більше часу у грі, яка відповідає його вподобанням і витрачає не менше час, ніж займає середньостатистичне базове проходження.

Над даними з CSV файлу варто виконати очищення та попередню обробку даних, такі як:

– заповнення пропущених значень;

– виключення зайвих маніпуляцій з грою, не важливих для формування уподобань;

– масштабування;

– видалення дублікатів.

Як структуру даних і основне джерело визначено, переходимо до створення класів моделей. Розроблення класів для представлення даних, включаючи класи, які відображають користувачів, предмети та їх взаємодії. Класи мають відповідати діаграмам з розділу проектування, та містити визначені характеристики.

Маючи дані і моделі, переходимо безпосередньо до реалізації класу відповідального за навчання, передбачення і зберігання моделі. Реалізація має відповідати діаграмі створеній під час проектування даного класу.

Створюємо клас GameModelBuilder, який відповідає за створення, збереження, завантаження, навчання та прогнозування моделі, використовуючи фреймворк ML.NET. В ньому містяться різні методи для роботи з моделлю машинного навчання:

– BuildModel() - метод відповідає за створення моделі. У цьому методі здійснюються методи та функції для побудови самої моделі з використанням алгоритмів машинного навчання. Особливо увагу приділено

методу MatrixFactorizationTrainer для побудови моделі. Даний алгоритм базується на факторизації матриць для прогнозування рейтингів, що створює векторні представлення користувачів та ігор для прогнозування рейтингів, які користувачі можуть дати іграм.

– SaveModel - метод використовується для збереження побудованої моделі в певному місці зберігання. Модель зберігається у файлі архіву, і в подальшому може бути збережена в базі даних для використання в прогнозуванні на нових даних.

– LoadLatestModel - використовується для завантаження найновішої (останньої) версії збереженої моделі з місця збереження. Це дозволяє використовувати оновлену модель для прогнозування або обробки нових даних.

– Predict - метод використовує навчену модель для прогнозування результату на основі вхідних даних. У випадку прогнозування рейтингу, метод отримує об'єкт UserRate, в якому, містяться дані про гру та користувача, і повертає прогнозований рейтинг чи інший очікуваний результат.

– BuildAndTrainModel - метод використовується для побудови та навчання моделі з використанням контексту та даних, які передаються у форматі IDataView для навчання моделі.

Створені методи дозволяють ефективно працювати з машинним навчанням і здійснювати різні операції з моделлю, додаючи бібліотеку зі створеним класом у необхідні проєкти (веб застосунок, утиліти).

Задля тестування алгоритму і моделі, створимо API (application programming interface), з посиланням на дану бібліотеку задля перевикористання попередньо створеного коду. Методи створеного API будуть відповідати за оновлення моделі та передбачення рейтингу гри для конкретного користувача.

Створенне рішення представляє собою набір алгоритмів та

інструментів, призначених для створення персоналізованих рекомендацій для користувачів в різних сферах, від електронної комерції до забезпечення контенту, яку можна використати у будь-якому проєкті створеному використовуючи .NET (рис. 3.3).

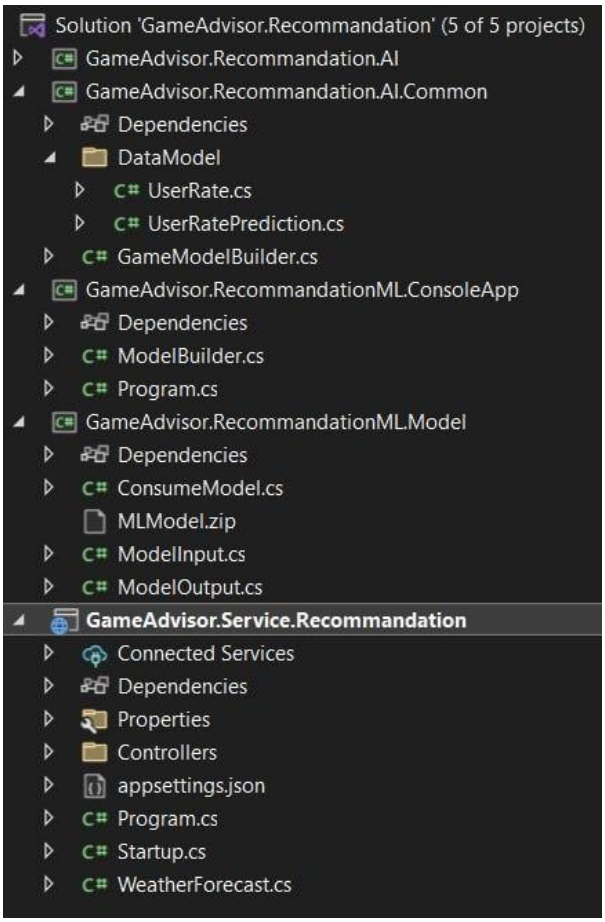


Рисунок 3.3 – Вікно рішення з файлами бібліотек алгоритмів

Маючи можливість тренувати модель, а також робити передбачення рейтингу запропонованої гри, лишається коректно сформувати вподобання користувача використовуючи спроектований метод.

Даний метод можна використовувати, як на сервернів частині, так і на клієнті задля економії ресурсів сервісу, таким чином доречна реалізація цього методу має бути створена за допомогою TypeScript.

Створений код виконує дії, пов'язані з операціями вставки елементів у список вподобань користувача відповідно до позицій у списку і має такі

етапи:

– спочатку відбувається вивід інформації, отриманої з події;

– визначається змінна gameToInsertSelected, яка підпорядковується умові порівняння event.data.gameId з this.gameToInsert.game.gameId;

– виконується перевірка умови this.isLastStep, що визначає, чи є поточний крок останнім. У випадку, якщо це останній крок, відбувається додавання елементів у ratedGames відповідно до певних умов та позицій у списку.

– якщо це не останній крок, виконується пошук в межах позицій startIndex та endIndex за допомогою методу search;

– метод search встановлює позиції startIndex та endIndex на вхідних значеннях, обчислених шляхом визначення середнього індексу middleGameIndex між ними та перевірки умов на визначення останнього кроку;

– при завершення пошуку, відбувається операція вставки елементів у список ratedGames;

– викликаються додаткові методи, такі як sendRates, який здійснює операцію збереження оцінок у coreService.

Дане рішення містить у собі бібліотеки класів, консольний застосунок, TypeScript код для формування вподобань, який може буте використаний у веб додатку, та API для маніпуляції з алгоритмом.

3.3 Розробка утиліт для машинного навчання

Консольна утиліта передбачає створення бази даних і імпорт ігрових даних з CSV файлу у визначеному форматі. Для цього необхідно створити схему бази даних, адаптовану для збереження інформації про ігри.

Для створення бази даних варто використовувати класи, які належать до пакету Entity Framework в рамках платформи .NET. Основним класом для взаємодії з базою даних є GameContext, який є підкласом DbContext з

методом OnModelCreating. Цей метод дозволяє визначити конфігурацію моделі даних, включаючи структуру таблиць, стовпці, обмеження, зв'язки тощо.

Клас GameContextFactory є реалізацією інтерфейсу IDesignTimeDbContextFactory і має метод CreateDbContext, призначений для створення екземпляра GameContext. Цей підхід використовується в процесі розробки та міграцій бази даних (рис. 3.4).

Рисунок 3.4 – Вікно створених таблиць у PgAdmin

Для виконання міграцій, Entity Framework надає функціональність Code-First Migrations, яка дозволяє створювати, застосовувати та скасовувати міграції бази даних на основі змін у коді моделі даних. Цей підхід дозволяє зберігати структуру бази даних у вигляді коду, що спрощує розгортання та розвиток бази, дозволяючи контролювати версії та еволюції схем баз даних.

Для імпорту даних використовуємо два згадані файли, а саме файл з

вподобаннями і загальна інформація про ігри. Вказаний CSV файл містить інформацію про посилання, тип, назву, опис, огляди, дату випуску, розробника, видавця, теги, деталі гри, мови, досягнення, жанр, опис гри, вміст для дорослих, мінімальні та рекомендовані вимоги, оригінальну ціну та знижену ціну (рис. 3.5).

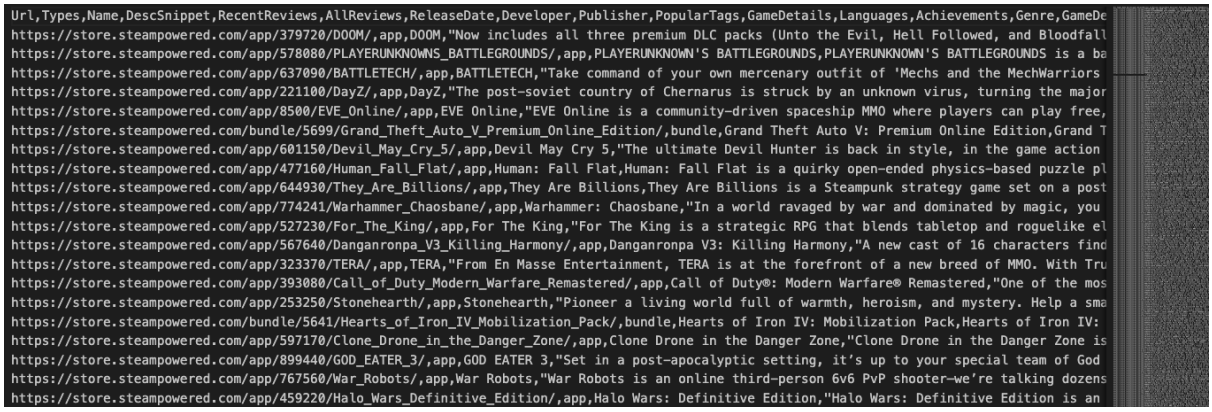


Рисунок 3.5 – CSV файл з даними по іграм

Дана утиліта також буде виконувати навчання і тестування моделі використовуючи створену бібліотеку персоналізованих рекомендацій. Під час навчання виконуються такі операції:

– поділ даних на тренувальний та валідаційний набори;

– тренування моделі на тренувальних даних для прогнозування рейтингів;

– оцінка та налаштування моделі на валідаційних даних для отримання оптимальних результатів.

– Для зручності використання утиліти та потенційної автоматизації введемо такі консольні параметри:

– migrateDb - параметр вказує, чи потрібно виконати міграцію бази даних перед використанням утиліти. Якщо значення true, то перед роботою утиліти буде виконано міграцію бази даних, згідно зі змінами, які знаходяться в контексті бази даних;

– connectionString - параметр представляє рядок підключення до бази

даних. Він містить інформацію про сервер бази даних, ім'я бази даних, аутентифікаційні дані та інші необхідні параметри для підключення;

– activityFile - параметр містить шлях до файлу, що містить активність користувачів на платформі Steam. Утиліта використовує цей файл для отримання інформації про поведінку користувачів у відношенні до ігор;

– dataFile - параметр містить шлях до файлу, що містить дані про ігри на платформі Steam. Утиліта використовує цей файл для отримання детальної інформації про кожну гру.

Фінальне рішення утиліти містить в собі файли з даними, розділені на бібліотеки шари та консольну утиліту (рис. 3.6).

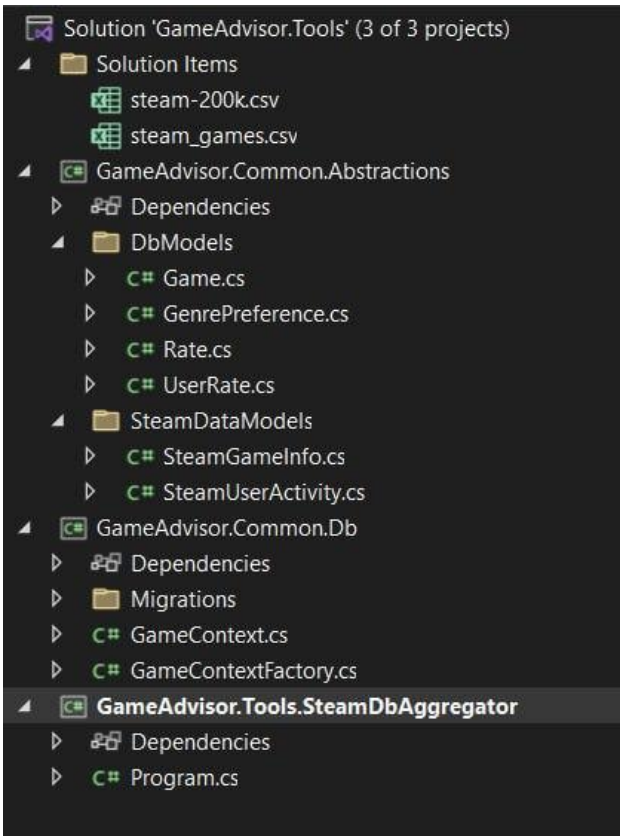


Рисунок 3.6– Вікно рішення з файлами рішення утиліт

Під час роботи утиліти, вона звітує про прогрес міграції та імпорту даних у реальному часі, для відслідковування етапів, оскільки завантаження даних і тренування моделі займає суттєвий час через обсяг (рис. 3.7).

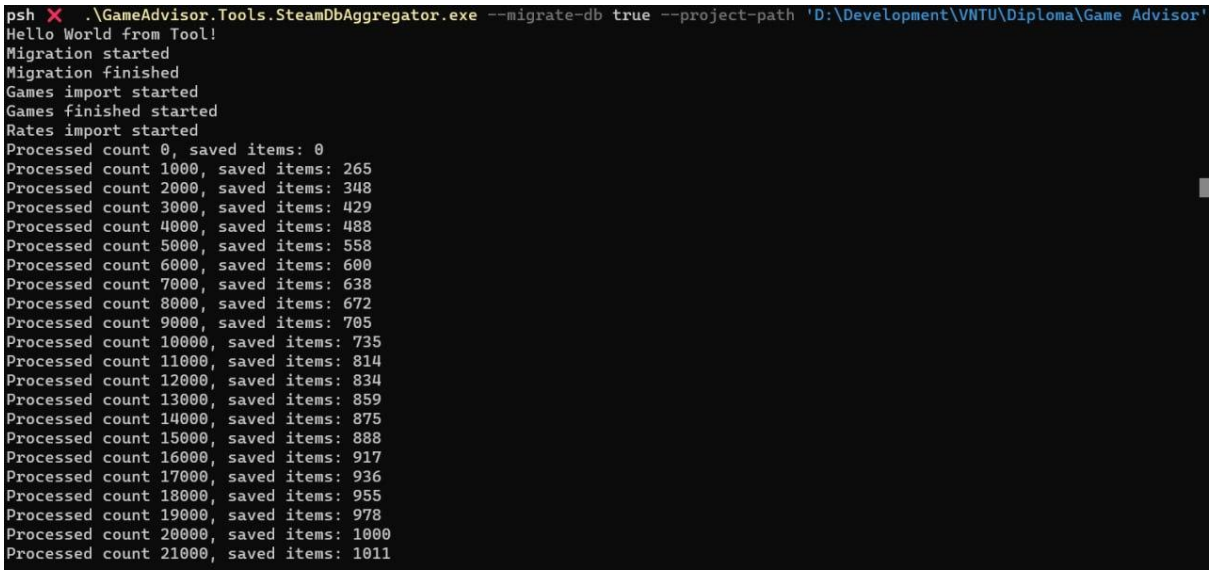


Рисунок 3.7 – Інтерфейс утиліти імпорту даних

Розроблена утиліта дозволяє підтримувати базу даних і модель машинного навчання в актуальному стані, а також розгортати середовища розробника автоматизованим чином.

3.4 Розробка веб сервісів з використанням покращеної системи персоналізованих рекомендацій

Загальне рішення містить у собі посилання попередньо створених бібліотек класів, а також нові веб сервіси, які являють собою єдиний застосунок.

Кожен веб проєкт в середині рішення являє собою типову трьохшарову архітектури [20]:

– Представлення (Presentation Layer) - шар відповідає за відображення даних і взаємодію з користувачем. В ASP.NET MVC це виглядає як набір "Views" (відображень) чи ActionResult (JSON чи XML відповідей), які відображають дані користувачам. Вони обробляють користувацькі запити;

– Логіка бізнесу (Business Logic Layer) - шар містить бізнес-логіку додатку. Це можуть бути класи, які обробляють дані, проводять розрахунки,

виконують правила бізнесу і так далі;

– Доступ до даних (Data Access Layer) - шар відповідає за доступ до даних, зазвичай бази даних. Він включає класи, які взаємодіють з базою даних, виконують запити та забезпечують зв'язок між бізнес-логікою і даними. Це можуть бути моделі, які представляють сутності додатку та відображають дані.

Рішення включає в себе декілька API, які реалізують трьошарову архітектуру та є мікросервісами для різних функціональних областей:

– Мікросервіс Ігор - відповідає за управління іграми. Він надає можливість отримувати інформацію про ігри, додавати нові та оновлювати існуючі записи про гру;

– мікросервіс рекомендацій - відповідає за створення персоналізованих рекомендацій для користувачів на основі їхніх уподобань та історії взаємодії з іграми;

– мікросервіс списку бажань - дозволяє користувачам додавати ігри до свого списку бажань, переглядати та керувати цим списком;

– сервіс авторизації – кожен мікросервіс посилається на цей сервіс для авторизації користувачів, надаючи доступ до функціональності та захищаючи ресурси API.

Розроблені сервіси реалізують REST (Representational State Transfer) підхід - це стиль архітектури для розробки мережевих програмних систем, який базується на принципах простоти, однорідності, прозорості та масштабованості. Розроблені сервіси реалізують REST, дотримуючись кількох ключових принципів:

– клієнт-серверна архітектура - система поділена на клієнтів та сервери. Це дозволяє їм розвиватися незалежно один від одного, забезпечуючи більшу масштабованість і простоту взаємодії;

– безстановість (Statelessness) - кожен запит від клієнта до сервера повинен містити всю необхідну інформацію для обробки запиту. Сервер не

зберігає жодної інформації про стан клієнта між запитами. Це сприяє масштабованості та простоті кешування;

– кешування - Сервер може вказувати клієнтам, чи можна кешувати відповіді. Це покращує продуктивність та знижує навантаження на сервер;

– інтерфейс уніфікованих ресурсів (Uniform Interface) - це принцип REST, який дозволяє кожному ресурсу бути уніфікованим шляхом звернення. Клієнти отримують доступ до ресурсів через URL-адреси [29].

Безстановість досягнена шляхом використання JWT (JSON Web Token) під час аутентифікації та авторизації. JWT - це компактний та самостійний спосіб передачі інформації між сторонами у вигляді JSON-об'єкта. Використання відбувається шляхом передачі токена в заголовку запиту під час авторизованих запитів до сервера, що дозволяє перевіряти доступ користувача до ресурсів.

Самі кінцеві точки запитів також реалізують принципи REST, а саме HTTP методи використовують для маршрутизації, наприклад:

– GET - використовується для отримання ресурсів. У випадку наших сервісів гарним прикладом є отримання інформації про гру по її ID – GET “api/games/id”;

– POST - Використовується для створення нових ресурсів, або виконання операцій. Наприклад, створення рейтинга - POST “api/rates”;

– PUT: Використовується для оновлення існуючих ресурсів. Наприклад, оновлення інформації про гру: PUT “api/games/id”;

– DELETE: Використовується для видалення ресурсів. Наприклад, видалення даних про гру: DELETE “api/games/id”.

REST і використання HTTP методів спрощують розробку API, роблять його більш підтримуваним та прозорим, сприяючи взаємодії клієнтів і серверів через інтерфейс уніфікованих ресурсів.

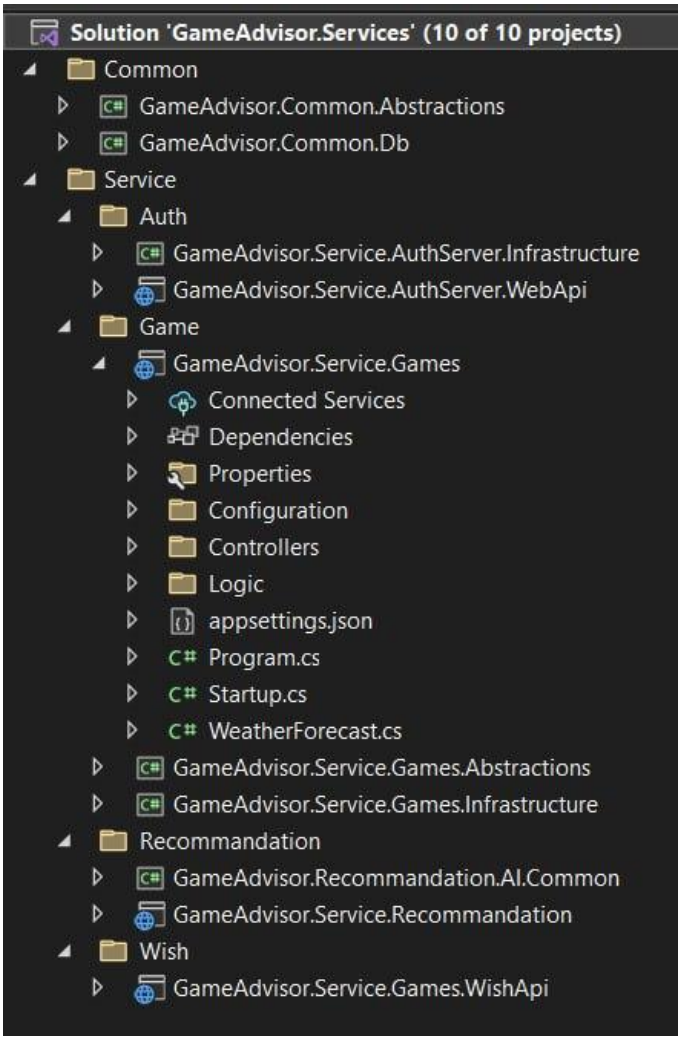


Рисунок 3.8 - Вікно рішення веб сервісів персоналізованих рекомендацій ігор

Розроблені мікросервіси виконують різні завдання і надають певні функції, специфічні для своєї області взаємодії з користувачем. Трьошарова архітектура дозволяє їм працювати як окремі компоненти з чіткою відповідальністю, спрощуючи розширення, тестування та підтримку коду. Оскільки кожен шар має свою власну відповідальність і може бути розроблений та тестований окремо – це полегшує розширення і супровід додатку. Будь-який з розроблених мікросервісів може бути розгорнутий окремо, що робить систему більш масштабованою та гнучкою (рис. 3.8).

3.5 Розробка інтерфейсу веб додатку і тестування бібліотек

Розробка інтерфейсу веб-додатку для тестування бібліотек і самого алгоритму є невід'ємною частиною цілісного застосунку. Під час розробки веб-додатків на Angular, важливим етапом є створення структури папок, що відображає організацію проекту, та визначення зручних npm команд для швидкого розгортання, розвитку та тестування.

Процес розпочинається з визначення базової структури каталогів, що відображає найкращі практики та основні принципи організації коду в Angular додатках (рис. 3.9).

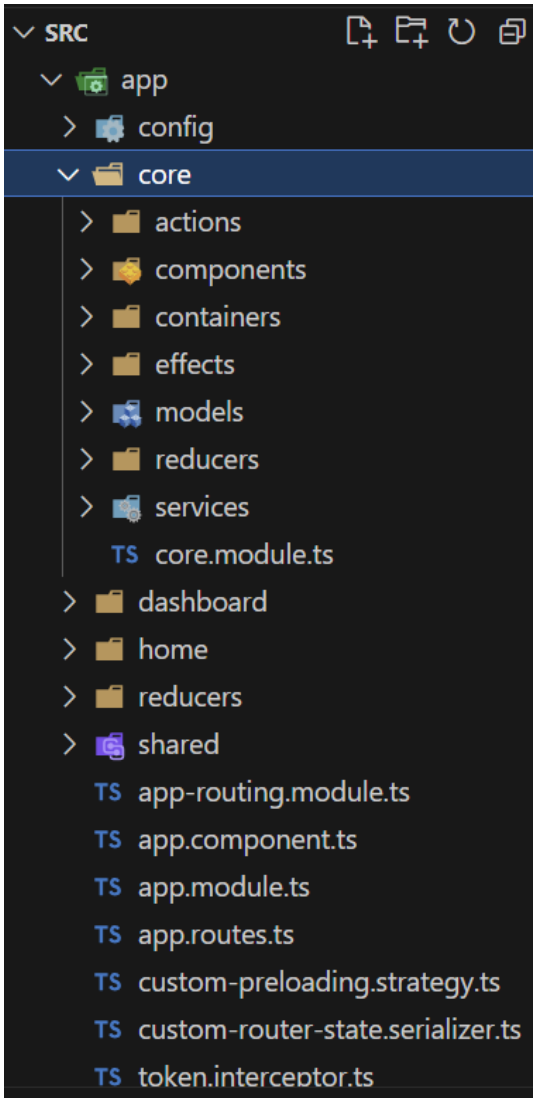


Рисунок 3.9 - Структура проєкту інтерфейсу застосунку

Далі, встановлення правильного набору npm скриптів та команд дозволяє створити ефективний розробничий процес, що спрощує роботу розробників та дозволяє легко виконувати тести, розгортання та інші повсякденні завдання (рис. 3.10).

```
"scripts": {
  "ng": "ng",
  "start": "ng serve --proxy-config proxy.conf.json --port 4300",
  "build": "ng build",
  "test": "ng test",
  "lint": "ng lint",
  "e2e": "ng e2e"
},
"private": true,
"dependencies": {
  "@angular/animations": "~9.1.4",
  "@angular/cdk": "^11.0.0",
```

Рисунок 3.10 - Структура проєкту інтерфейсу застосунку

Розпочнемо створення інтефейсу з головної сторінки, яка коротко розповідає про застосунок і його ціль, а також має приваблювати користувачів (рис. 3.11).

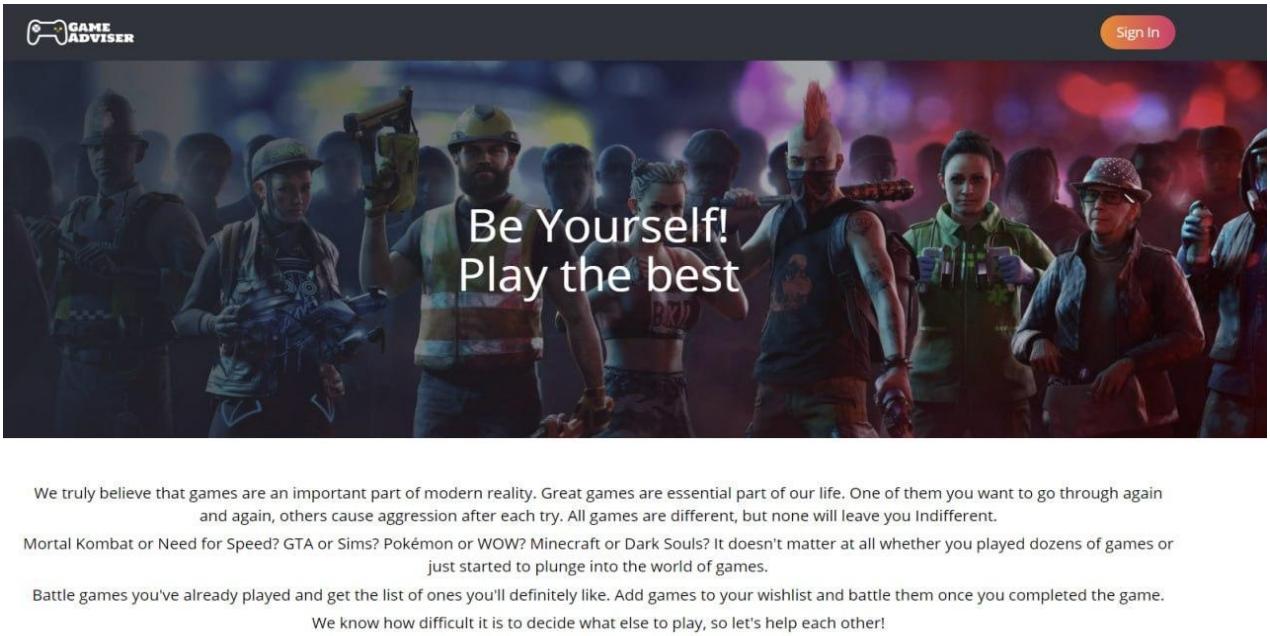


Рисунок 3.11 - Головне вікно веб додатку

Відобразимо інформацію про ігри одразу ж на головній сторінці за допомогою віджетів та мікросервісу ігор. Використавши ресурс «api/games?itemCount=10&page=0&filterPublisher=Activision» з фільтрацією і пагінацією елементу будуть оптимізовано отримати з бази і відображені по 10 ігор відповідно до категорії (рис. 3.12).

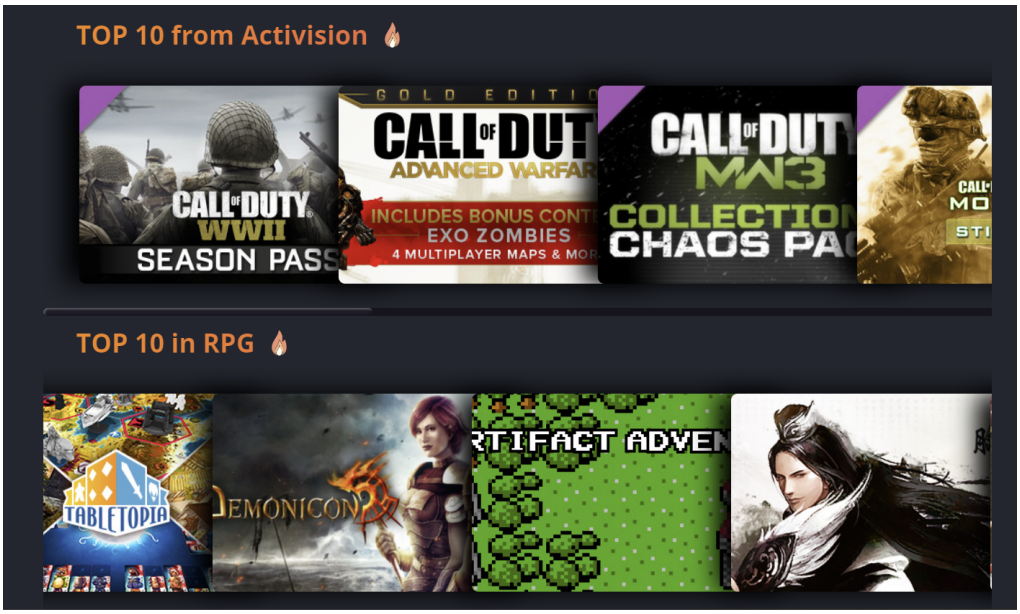


Рисунок 3.12 - Вигляд сторінки переліку запропонованих позицій

Для формування персоналізованих рекомендацій першочергово варто ідентифікувати користувача, тому створимо вікно реєстрації і логіну, яке використовуватиме розроблений сервіс аутентифікації (рис. 3.13).

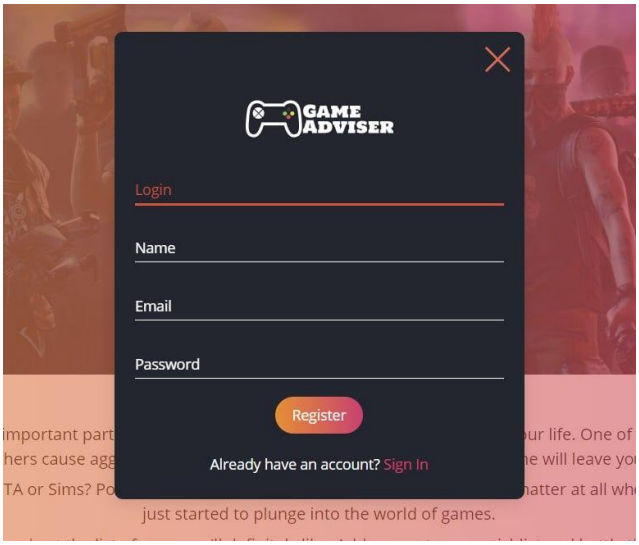


Рисунок 3.13 - Вигляд вікна реєстрації нового користувача

Під час реєстрації користувача, для вирішення описаної проблеми холодного старту, одразу ж використовуємо сервіс вподобань і за допомогою метода формуємо пріорітет вподобань користувача (рис. 3.14). Також, даний вибір і майбутні ігри користувачів будуть використовуватись для покращення роботи моделі машинного навчання, таким чином чим більше використовується додаток, тим точніші будуть персоналізовані рекомендації.

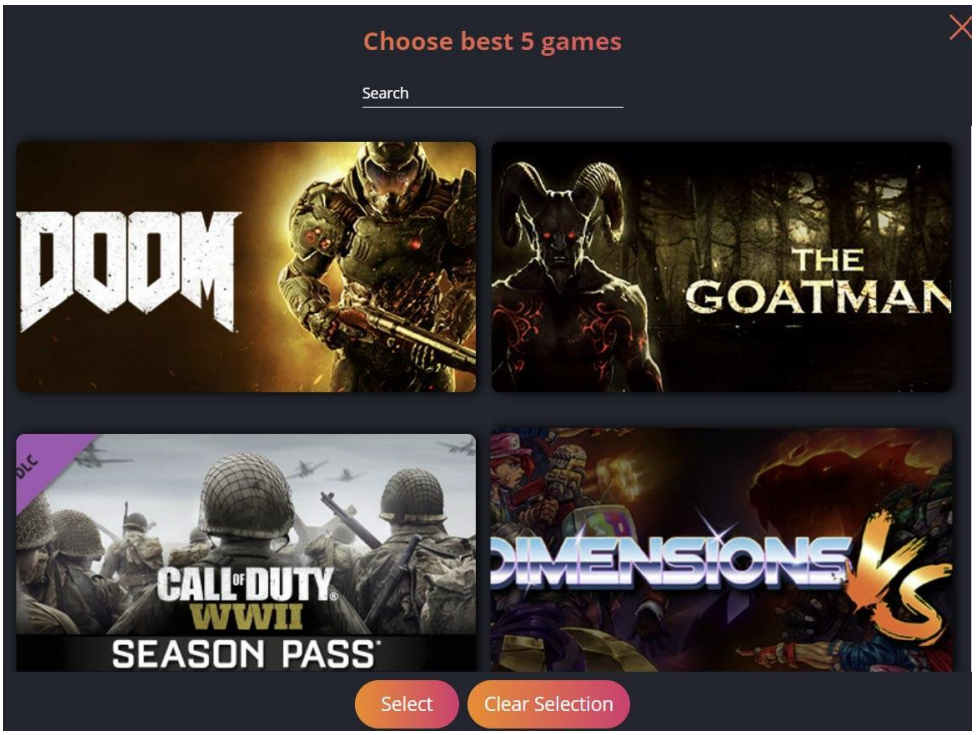


Рисунок 3.14– Вигляд сторінки раундів перевірки вподобань нового користувача

Після проходження вибірки і виконання раундів методи, дані відправляються до мікросервісу рейтингу з HTTP методом POST для створення початкових вподобань користувача відповідно до пріоритетів. З кожним запитом також відправляється JWT Bearer токен для ідентифікації користувача і відображення виключно персоналізованого змісту та переліку ігор, які можуть зацікавити відповідно до сформованого простору зацікавлень і перевірки варіантів з залученням моделі (рис. 3.15).

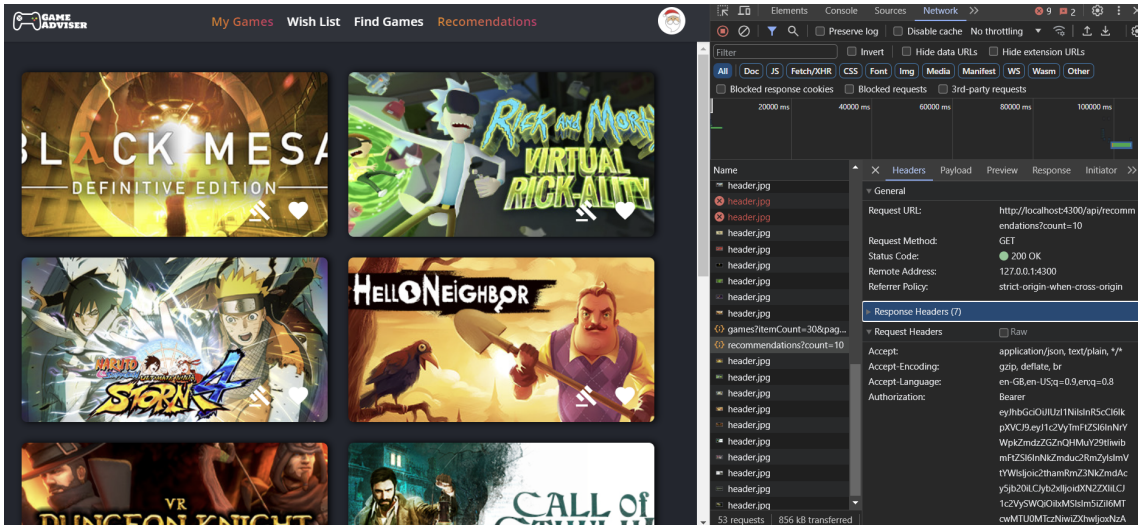


Рисунок 3.15 – Вигляд сторінки рекомендацій і вікно запитів застосунку з токеном

Після узгодження вподобань під час холодного старту, користувач має можливість додавати нові ігри і орієнтувати їх відповідно попередніх вподобань задля більш тонкого налаштування пріорітетів. При обранні іконки «Молот» починається новий раунд на основі методу формування персоналізованих рекомендацій (рис. 3.16).

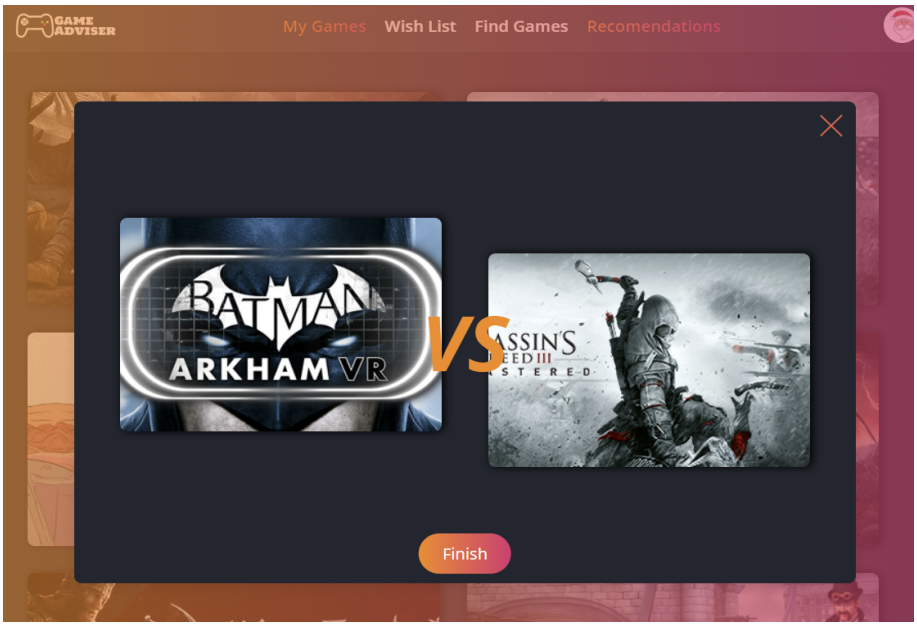


Рисунок 3.16 – Вигляд вікна переформування вподобань відповідно до нової гри

Кількість раундів відповідно реалізованого методу і описаного в діаграмі, залежить від вибору користувача і можливості ідентифікації конкретного місця цієї гри при порівняннях. Після закінчення переформування вподобань, користувач має змогу подивитись свої ігри відсортовані на основі вподобань та раундів методу (рис. 3.17).

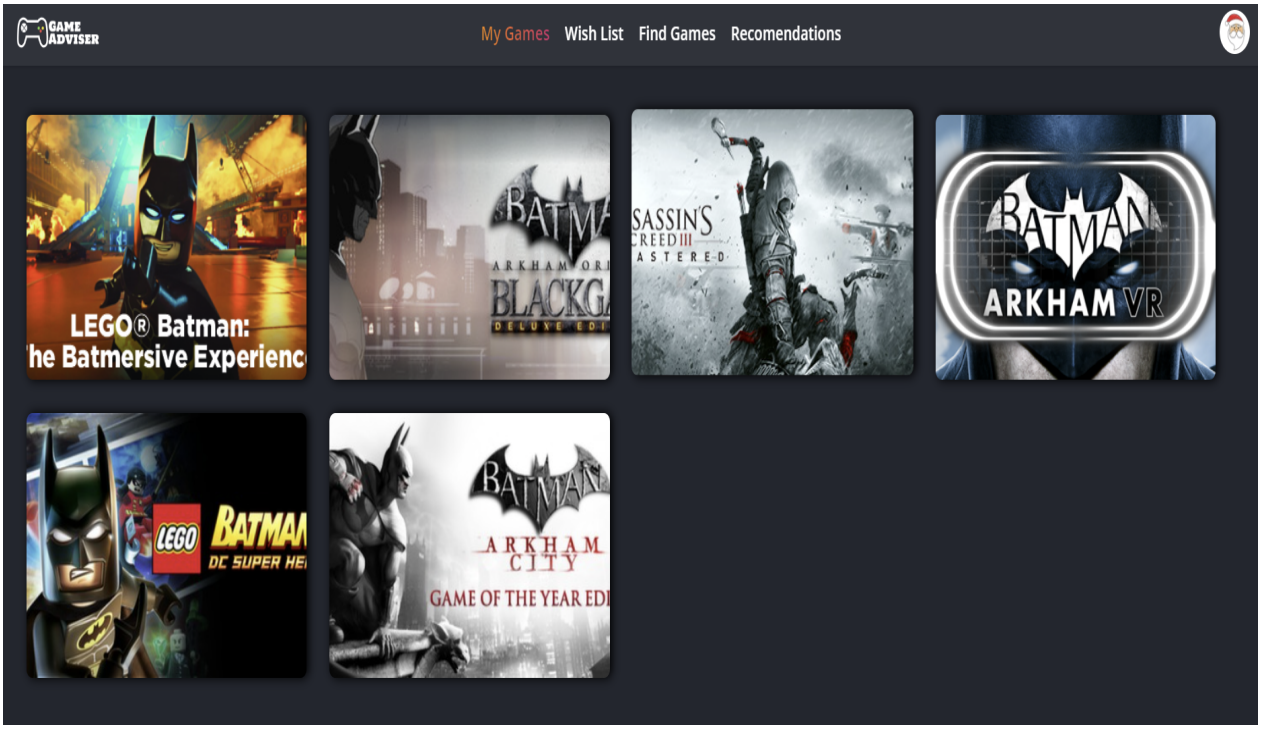


Рисунок 3.17 – Вигляд сторінки відформатованих вподобань користувача

Для детального розуміння гри і можливості прийняти рішення до проходження, з сторінки рекомендацій користувач має можливість перейти і переглянути деталі до гри описані під час проєктування базу даних відповідно до імпортованого джерела (рис. 3.18). Дану сторінку перевикористаємо в інших місцях відображення піктограм ігор.

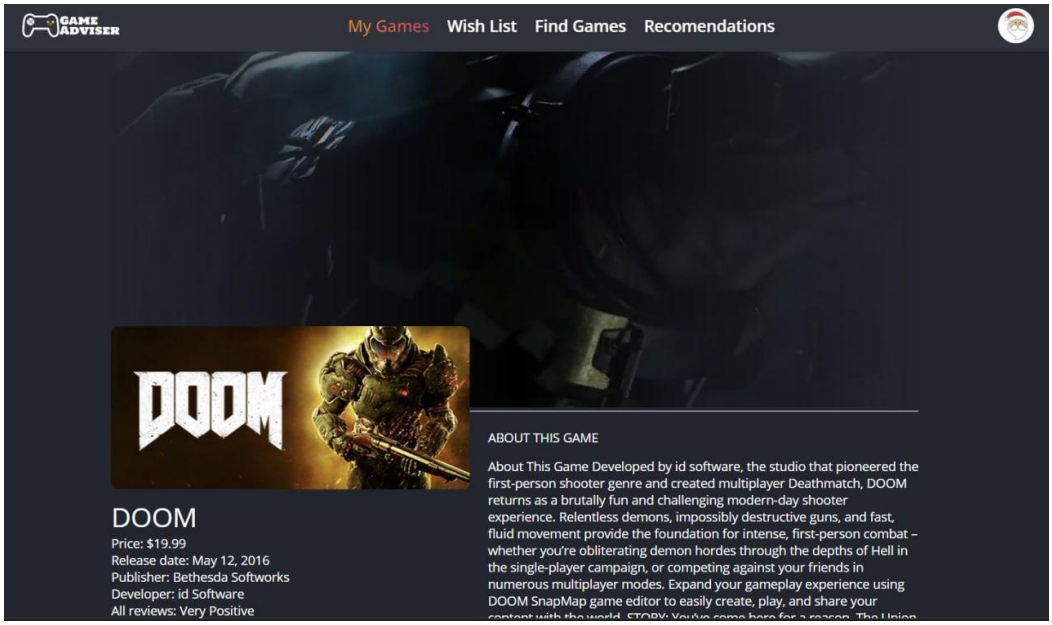


Рисунок 3.18 - Вигляд сторінки перегляду інформації про гру

Тестування програмного забезпечення є невід'ємною частиною процесу розробки. Воно допомагає забезпечити працездатність системи та її відповідність вимогам користувачів. Один з найбільш ефективних методів тестування є метод "чорного ящика".

Тестування "чорного ящика" - це метод тестування програмного забезпечення, при якому тестувальник не знає про внутрішню структуру та реалізацію системи. Тестувальник взаємодіє з системою лише через її інтерфейс, надаючи вхідні дані та отримуючи вихідні результати. Це дозволяє тестувати систему з точки зору кінцевого користувача, щоб переконатися, що вона відповідає його потребам [30].

Відповідно створеного інтерфейсу і використання API сервісів, можна скласти такий порядок дій для тестування основного функціоналу застосунку. Даний шлях буде стосуватися кожного сервісу. Методу формування персоналізованих уподобань і персоналізованих рекомендацій на основі сформованих уподобань та моделі навченої за допомогою машинного навчання:

– перехід на головну сторінку з відображенням найбільш популярних

ігор відповідно категорії;

– створення профілю користувача чи вхід за допомогою існуючого;

– холодний старт та обрання попередньо вподобаних ігор;

– перегляд персоналізованих рекомендацій відповідно до дії методу;

– перехід на обрану гру зі списку рекомендацій для перегляду деталей;

– проходження раундів переформування вподобань;

– перегляд новосформованого списку власних ігор і попередніх вподобань.

Сформований сценарій допоможе підтримувати застосунок більше ефективно маючи змоги контролювати зміни основних компонентів системи. Запропонований сценарій може частково повторно використовуватись під час застосування бібліотек і утиліт для визначення персоналізованих рекомендацій в інших сферах діяльності, як кінофільми, книги, або музика.

3.6 Висновки

В результаті проектування було розроблено розробки сервіси й бібліотеки, що містять метод персоналізованих рекомендацій, утиліти для машинного навчання та веб-застосунок, а також були розглянуті і описані ключові складові системи.

Відповідно до вибору засобів, даний функціонал було реалізовано мовою C# та TypeScript на платформі .NET, використовуючи технології для побудови API та фреймворку користувацького інтерфейсу Angular.

Після реалізації усіх компонентів системи, було сформовано сценарій і протестовано систему використовуючи метод чорного ящика.

4 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

Магістерська кваліфікаційна робота за темою «Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання» відноситься до науково-технічних робіт, які орієнтовані на виведення на ринок (або рішення про виведення науково-технічної розробки на ринок може бути прийнято у процесі проведення самої роботи), тобто коли відбувається так звана комерціалізація науково-технічної розробки. Цей напрямок є пріоритетним, оскільки результатами розробки можуть користуватися інші споживачі, отримуючи при цьому певний економічний ефект. Але для цього потрібно знайти потенційного інвестора, який би взявся за реалізацію цього проекту і переконати його в економічній доцільності такого кроку.

Для наведеного випадку нами мають бути виконані такі етапи робіт:

1) проведено комерційний аудит науково-технічної розробки, тобто встановлення її науково-технічного рівня та комерційного потенціалу;

2) розраховано витрати на здійснення науково-технічної розробки;

3) розрахована економічна ефективність науково-технічної розробки у випадку її впровадження і комерціалізації потенційним інвестором і проведено обґрунтування економічної доцільності комерціалізації потенційним інвестором.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями, наведеними в табл. 4.1 [33].

Таблиця 4.1 – Рекомендовані критерії оцінювання науково-технічного рівня і комерційного потенціалу розробки та бальна оцінка

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження таблиці 4.1

Бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено працездатність продукту в реальних умовах
Ринкові переваги (недоліки)					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою

Продовження таблиці 4.1

7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві

Продовження таблиці 4.1

11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Результати оцінювання науково-технічного рівня та комерційного потенціалу науково-технічної розробки потрібно звести до таблиці.

Таблиця 4.2 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	5	5	5
2. Ринкові переваги (наявність аналогів)	4	5	4
3. Ринкові переваги (ціна продукту)	4	3	3
4. Ринкові переваги (технічні властивості)	4	4	3
5. Ринкові переваги (експлуатаційні витрати)	3	3	3

Продовження таблиці 4.2

6. Ринкові перспективи (розмір ринку)	3	4	3
7. Ринкові перспективи (конкуренція)	3	3	4
8. Практична здійсненність (наявність фахівців)	3	3	3
9. Практична здійсненність (наявність фінансів)	3	3	4
10. Практична здійсненність (необхідність нових матеріалів)	3	3	3
11. Практична здійсненність (термін реалізації)	3	4	4
12. Практична здійсненність (розробка документів)	4	3	3
Сума балів	42	43	42
Середньоарифметична сума балів $СБ_с$	42,3		

За результатами розрахунків, наведених в таблиці 4.2, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в табл. 5.3 [33]

Таблиця 4.3 – Науково-технічні рівні та комерційні потенціали розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Науково-технічний рівень та комерційний потенціал розробки
41…48	Високий
31…40	Вище середнього
21…30	Середній
11…20	Нижче середнього
0…10	Низький

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання» становить 42,3 бала, що,

відповідно до таблиці 4.3, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

4.2 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

4.2.1 Витрати на оплату праці

До статті «Витрати на оплату праці» належать витрати на виплату основної та додаткової заробітної плати керівникам відділів, лабораторій, секторів і груп, науковим, інженерно-технічним працівникам та ін.

Основна заробітна плата дослідників

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [33, 34]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p} \qquad (4.1)$$

де k – кількість посад дослідників залучених до процесу досліджень;

M_{ni} – місячний посадовий оклад конкретного дослідника, грн;

t_i – число днів роботи конкретного дослідника, дні;

T_p – середнє число робочих днів в місяці, T_p=21 день.

$З_о$ = 35000,00 ·60 / 21 = 100000 грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 4.4 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Керівник проекту	35000	1666,67	60	100000,00
Інженер-розробник програмного забезпечення	23000	1095,24	60	65714,29
Консультант	15000	714,29	20	14285,71
Всього				180000

Основна заробітна плата робітників

Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i \quad , \tag{4.2}$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} \quad , \tag{4.3}$$

де $М_М$ – розмір прожиткового мінімуму працездатної особи, або мінімальної місячної заробітної плати (в залежності від діючого законодавства), приймемо $Мм$=6700,00 грн;

K_i – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду (табл. Б.2, додаток Б) [33]

K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об'єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

T_p – середнє число робочих днів в місяці, приблизно T_p = 21 день;

$t_{зм}$ – тривалість зміни, год.

$C_1 = 6700{,}00 \cdot 1{,}10 \cdot 1{,}65 / (21 \cdot 8) = 72{,}38$ грн.
$З_{р1} = 72{,}38 \cdot 4{,}00 = 289{,}54$ грн.

Таблиця 4.5 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Установка електронно-обчислювального обладнання	4	2	1,1	72,38	289,54
Підготовка робочого місця дослідника	3	2	1,1	72,38	217,15
Інсталяція програмного забезпечення	3	5	1,7	111,87	335,60
Тестування системи	96	2	1,1	72,38	6948,86
Всього					7791,14

Додаткова заробітна плата дослідників та робітників

Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (4.4)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 11%.

$З_{дод}$ = (180000+ 7791,14) · 11 / 100% = 20657,03 грн.

4.2.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (4.5)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%.

Зн = (180000+ 7791,14+ 20657,03) · 22 / 100% = 45858,6 грн.

4.3.3 Сировина та матеріали

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби і предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень.

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} Н_{вj} \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц, \quad (4.6)$$

де H_j – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

$Ц_j$ – вартість матеріалу j-го найменування, грн/кг;

$К_j$ – коефіцієнт транспортних витрат, ($К_j$ = *1,1 … 1,15*);

$В_j$ – маса відходів j-го найменування, кг;

$Ц_{вj}$ – вартість відходів j-го найменування, грн/кг.

$М_1 = 3 \cdot 200{,}00 \cdot 1{,}1 - 0{,}000 \cdot 0{,}00 = 660{,}0$ грн.

Проведені розрахунки зведемо до таблиці.

Таблиця 4.6 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за од	Норма витрат, од	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Офісний папір Calipso Plus A4-500-80	200	3	0	0	660
Канцелярське приладдя (набір офісного працівника)	175	2	0	0	385
Flesh-пам'ять Kingston 16 GB	130	1	0	0	143
Всього					1188

4.2.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання» відсутні.

4.3.5 Спецустаткування для наукових (експериментальних) робіт

До статті «Спецустаткування для наукових (експериментальних) робіт» належать витрати на виготовлення та придбання спецустаткування

необхідного для проведення досліджень, також витрати на їх проектування, виготовлення, транспортування, монтаж та встановлення.

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \quad , \qquad (4.7)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн;

$C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань устаткування.

$B_{спец} = 550000 \cdot 1 \cdot 1,1 = 60500$ грн.

Отримані результати зведемо до таблиці:

Таблиця 4.7 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
ASUS S500MC-5114000460 Intel i5-11400/16/512F/NVD3060-12/No	1	52000	57200
HP Elite 600-G9 TWR, Intel i7-12700, 16GB, F512GB	1	55000	60500
Всього			117700

4.2.6 Програмне забезпечення для наукових (експериментальних) робіт

До статті «Програмне забезпечення для наукових (експериментальних) робіт» належать витрати на розробку та придбання спеціальних програмних засобів і програмного забезпечення, (програм, алгоритмів, баз даних) необхідних для проведення досліджень, також витрати на їх проектування, формування та встановлення.

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \quad , \tag{4.8}$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

$B_{прг} = 5500,00 \cdot 2 \cdot 1,1 = 12320$ *грн.*

Отримані результати зведемо до таблиці:

Таблиця 4.8 – Витрати на придбання програмних засобів по виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
ОС Windows 11	2	11600	25984
Прикладний пакет Microsoft Office 2019	2	5500	12320
Всього			38304

4.2.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (4.9)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл} = (57200{,}00 \cdot 2) / (3 \cdot 12) = 3177{,}78$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 4.9– Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
ASUS S500MC-5114000460 Intel i5-11400/16/512F/NVD3060-12/No	57200	3	2	3177,78

Продовження таблиці 4.9

HP Elite 600-G9 TWR, Intel i7-12700, 16GB, F512GB	60500	3	2	3361,11
Робоче місце дослідника	10000	5	2	333,33
Оргтехніка	6000	4	2	250,00
ОС Windows 11	11600	2	2	966,67
Прикладний пакет Microsoft Office 2019	5500	2	2	458,33
Всього				8547,22

4.2.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (4.10)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_e$ – вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), приймемо $Ц_e$ = 7,50 грн;

$K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$;

η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$.

$B_e = 0,6 \cdot 480,0 \cdot 7,50 \cdot 0,95 / 0,97 = 2115,46$ *грн.*

Проведені розрахунки зведемо до таблиці.

Таблиця 4.10– Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Ноутбук HP	0,6	480	2115,46
Ноутбук ASUS	0,6	480	2115,46
Робоче місце дослідника	0,15	480	528,87
Оргтехніка	0,45	20	66,11
Всього			4825,90

4.2.9 Службові відрядження

До статті «Службові відрядження» дослідної роботи належать витрати на відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, аспірантів, зайнятих розробленням досліджень, відрядження, пов’язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з’їзди, конференції, наради, пов’язані з виконанням конкретних досліджень.

Витрати за статтею «Службові відрядження» розраховуємо як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{св} = (З_о + З_р) \cdot \frac{H_{св}}{100\%}, \qquad (4.11)$$

де $H_{св}$ – норма нарахування за статтею «Службові відрядження», приймемо $Н_{св}$ = 20%.

$B_{св}$ = (180000+ 7791,14) · 20 / 100% = 37558,23грн.

4.2.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (4.12)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 35%.

$B_{сп}$ = (130952,38+ 987,05) · 35 / 100% = 65726,90 грн.

4.2.11 Інші витрати

До статті «Інші витрати» належать витрати, які не знайшли відображення у зазначених статтях витрат і можуть бути віднесені безпосередньо на собівартість досліджень за прямими ознаками.

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (4.13)$$

де $Н_{ів}$ – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%.

$І_в$ = (180000+ 7791,14) · 50 / 100% = 93895,57грн.

4.2.12 Накладні (загальновиробничі) витрати

До статті «Накладні (загальновиробничі) витрати» належать: витрати, пов'язані з управлінням організацією; витрати на винахідництво та раціоналізацію; витрати на підготовку (перепідготовку) та навчання кадрів; витрати, пов'язані з набором робочої сили; витрати на оплату послуг банків; витрати, пов'язані з освоєнням виробництва продукції; витрати на науково-технічну інформацію та рекламу та ін.

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (4.14)$$

де $Н_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 120%.

$В_{нзв}$ = (180000+ 7791,14) · 120 / 100% = 225 349,37 грн.

Витрати на проведення науково-дослідної роботи на тему «Вдосконалення методу генерації цифрових ключів за допомогою зліпка обличчя» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$B_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (4.15)$$

$В_{заг}$=180000+ 7791,14+20657,03+45858,59+1188+0,00 +117700+ 38304+ 8547,22+ 4825,90+37558,23+65726,90+93895,57+225 349,37 = 528477,93грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{B_{заг}}{\eta}, \quad (4.16)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9.

ЗВ = 528477,93/ 0,9 = 941557,73грн.

4.3 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

В ринкових умовах узагальнюючим позитивним результатом, що його може отримати потенційний інвестор від можливого впровадження результатів тієї чи іншої науково-технічної розробки, є збільшення у потенційного інвестора величини чистого прибутку.

Результати дослідження проведені за темою ««Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання» передбачають комерціалізацію протягом 3-х років реалізації на ринку.

В цьому випадку майбутній економічний ефект буде формуватися на основі таких даних:

ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик;

1-й рік – 20000 користувачів за рік;

2-й рік – 35000 користувачів за рік;

3-й рік – 25000 користувачів за рік.

N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо 200000 користувачів;

$Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 1100,00 грн (підписка на рік);

$\pm\Delta Ц_o$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 200,00 грн.

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 3-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [33]:

$$\Delta П_i = (\pm\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (4.17)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2023 році ставка податку на додану вартість складає 20%, а коефіцієнт $\lambda = 0,8333$;

ρ – коефіцієнт, який враховує рентабельність інноваційного продукту. Приймемо ρ =30%;

ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2023 році $\vartheta = 18\%$;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ = (200000,00·200,00+1300,00·20000)·0,83·0,3·(1-0,18/100%)= 13524588 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ = (200000,00·200,00+1300,00·(20000+35000))·0,83·0,3·(1-0,18/100%)= 22848357 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3 =$ (20000,00·200,00+1300,00·(20000+35000+25000))·0,83·0,3·(1-0,18/100%)= 29508192 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (4.18)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,2;

t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

ПП = 13524588/(1+0,2)[1]+ 22848357/(1+0,2)[2]+ 29508192/(1+0,2)[3]= 44213904,58 грн.

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$ВВ = k_{інв} \cdot \quad , \tag{4.19}$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$=3;

$ЗВ$ – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 941557,73грн.

$ВВ = k_{інв} \cdot$ = 3· 941557,73= 2824673,205грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \tag{4.20}$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 44213904,58 грн;

PV – теперішня вартість початкових інвестицій, 2824673,205грн.

$E_{абс} = ПП - PV$ = 44213904,58 - 2824673,205= 41389231,38грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Тж]{1 + \frac{E_{абс}}{PV}} - 1 \quad , \tag{4.21}$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 41389231,38грн;

PV – теперішня вартість початкових інвестицій, 2824673,205грн;

$T_{ж}$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 3 роки.

$$E_{в} = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1 = (1+ 41389231,38/ 2824673,205)^{1/3}= 1,5.$$

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \tag{4.22}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2023 році в Україні d =0,12;

f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,2.

$\tau_{мін}$ = 0,12+0,2 = 0,32 < 1,5 свідчить про те, що внутрішня економічна дохідність інвестицій $E_{в}$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \tag{4.23}$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій.

$T_{ок} = 1 / 1,5 = 0,67$ року.

$T_{ок} <$ 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

4.4 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання» становить 42,3 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки високий).

Також термін окупності становить 0,67 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання».

ВИСНОВКИ

Під час виконання магістерської кваліфікаційної роботи було розроблено метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання. Використовуючи створений метод і засіб, було розроблено веб застосунок для рекомендації ігор відповідно до персональних вподобань. Для розробки було використано Visual Studio 2022, як основне середовище програмування та Visual Studio Code для розробки користувацького інтерфейсу. Робота оформлена згідно методичних вказівок [31].

На основі проведеного аналізу методів персоналізованих рекомендацій та алгоритмів машинного навчання була визначена мета проєкту: створення покращеного методу персоналізованих рекомендацій, який комбінує Матричну факторизацію і Колаборативний фільтринг, що компенсує недоліки та посилює переваги кожного методу. Такий алгоритм допоможе вирішити проблему використання нового методу для нових користувачів шляхом додаткового алгоритму для створення матриці вподобань під час холодного старту.

Процес створення засобів персоналізованих рекомендацій містив у собі реалізацію бібліотек, утиліт і веб-сервісів, які використовують розроблений алгоритм. Після реалізації засобу, було виконано його тестування. Найважливішими аспектами цього процесу є вибір даних для навчання моделей, якість і обсяг даних.

Результатом проведеного аналізу та розробки є створення методу персоналізованих рекомендацій, який усуває недоліки інших методів, і його подальша інтеграція в програмний засіб для подальшого використання. Даний метод є актуальним і має високий практичний рівень, тому потребує подальшої розробки для досягнення поставленої мети.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Recommender Systems An Introduction - Дітмар Яннах, Маркус Занкер, Олександр Фельферніг, Герхард Фрідріх, 2010.

2. Волинець О. Ю., Тужанський С. Є. Персоналізовані Рекомендації У Цифрових Бібліотеках – 2023, матеріали Міжнародної науково-практичної Інтернет-конференціїї "Електронні інформаційні ресурси: створення, використання, доступ 2023"

3. A Gentle Introduction to Recommender Systems - Sun Jackson

4. Matrix factorization (recommender systems). [Електронний ресурс]. URL: https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems)

5. "Netflix Update: Try This at Home" - Funk, Simon.

6. "Kernelized Probabilistic Matrix Factorization: Exploiting Graphs and Side Information" - Zhou, Tinghui; Shan, Hanhuai, 26 April 2012.

7. Factorization machines. In Proceedings of the 10th IEEE International Conference on Data Mining (ICDM) - Rendle, S., 2010.

8. ETL Testing: A Complete Guide - Пейман Персі

9. ETL Processes and Data Engineering - Карл Фассбендер

10. Хмарні технології. [Електронний ресурс]. URL: http://j.parus.ua/ua/358

11. Clean Code: A Handbook of Agile Software Craftsmanship - Robert C. Martin

12. Instant Automapper Paperback - Taswar Bhatti (2013).

13. Конноллі Т. Бази даних. Проектування, реалізація та супровід. Теорія та практика. 3-тє видання / Т. Конноллі, К. Бегг.

14. Гарсіа-Моліна Г. Системи баз даних. Повний курс / Г. Гарсіа-Моліна, Дж. Ульман, Дж. Уідом, 2003.

15. Грабер М. Введення в SQL - Лорі, 2010.

16. Введення в мову C# та платформу .NET Framework [Електронний ресурс]. URL: https://msdn.microsoft.com/ua/library/z1zx9t92.aspx

17. C# 6.0 and the .NET 4.6 Framework 7th ed - ANDREW TROELSEN. 2015 Edition - ISBN-13: 978-1484213339

18. PositiveTechnology. [Електронний ресурс]. URL: https://www.ptsecurity.com/

19. PostgreSQL: Up and Running 2nd Edition - Regina Obe, Leo Hsu

20. Jose Rolando Guay Paz Beginning ASP.NET MVC 4 2013th Edition

21. Jess Chadwick - Programming ASP.NET MVC 4: Developing Real-World Web Applications with ASP.NET MVC - ISBN-13: 978-1449320317

22. Hands-On Machine Learning with ML.NET: Getting started with Microsoft ML.NET to implement popular machine learning algorithms in C# - Jarred Capellman

23. Angular Development with TypeScript 2nd Edition - Yakov Fain, Anton Moiseev

24. Документація JetBrains Rider Cross platform IDE. [Електронний ресурс]. URL: https://www.jetbrains.com/rider/documentation.

25. Документація Microsoft Visual Studio 2022. [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/visualstudio/?view=vs-2022

26. Документація Visual Studio Code. [Електронний ресурс]. URL: https://code.visualstudio.com/docs.

27. Удосконалення моделі керування доступом на основі ролей у приватних хмарних середовищах - О. Ю. Волинець, Д. В. Куліш, А. В. Приймак, Я. Ю. Яремчук.

28. Документація ASP.NET Identity Platform. [Електронний ресурс]. URL: https://learn.microsoft.com/en-us/aspnet/identity/

29. RESTful API Design - Matthias Biehl

30. Myers G. The Art of Software Testing – Chicago, 2012

31. Положення про кваліфікаційну роботу на другому (вищому) рівні вищої освіти. Розробники: А.О. Семенов, Л.П. Громова, Т.В. Макарова, О.В. Сердюк – Вінниця, ВНТУ, 2021

32. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад.: В. О. Козловський, О. Й. Лесько, В. В. Кавецький. Вінниця : ВНТУ, 2021. 42 с.

33. Кавецький В. В**.** Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа. Вінниця : ВНТУ, 2016. 113 с.

ДОДАТКИ

Додаток А

Технічне завдання

Міністерство освіти і науки України

Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ

д.т.н., проф. О. Н. Романюк

"19" вересня 2023 р.

Технічне завдання

на магістерську кваліфікаційну роботу « «Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання» за спеціальністю

121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:

к.т.н., асистент. каф. ПЗ Тужанський С.Є.

" 19 " вересня 2023 р.

Виконав:

студент гр.3ПІ-22М О.Ю. Волинець

" 19 " вересня 2023 р.

Вінниця – 2023 року

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання».

Галузь застосування – застосунки з розповсюдження контенту та товарів.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № 247 від 18 вересня 2023 року ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є підвищення точності генерації персоналізованих рекомендацій за рахунок використання методів машинного навчання з урахуванням індивідуальних потреб користувачів.

Призначення роботи – розробка методів і застосунку для персоналізованих вподобань з можливістю подальшої інтеграції бібліотек у інші застосунки.

3 Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Recommender Systems An Introduction 2010 - Дітмар Яннах, Маркус Занкер, Олександр Фельферніг, Герхард Фрідріх..
2. A Gentle Introduction to Recommender Systems - Sun Jackso.
3. Factorization machines. In Proceedings of the 10th IEEE International Conference on Data Mining (ICDM) - Rendle, S. (2010).
4. C# 6.0 and the .NET 4.6 Framework 7th ed. 2015 Edition - ANDREW TROELSEN.

4. Технічні вимоги

Вхідні дані – набір даних вподобань ігор популярної; вихідні дані – список персоналізованих рекомендацій ігор для конкретного користувача відносно минулих виборів.

5. Конструктивні вимоги.

Конструкція застосунку і бібліотек повинна відповідати найкращим практикам та використовувати загально відомі шаблони. Код має компілюватись без помилок і проблем.

6. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

7. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

8. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз предметної області	20.09.2023 – 10.10.2023
2	Проектування методу і засобу персоналізованих рекомендацій	10.10.2023– 30.10.2023
3	Програмна реалізація та впровадження методу персоналізованих рекомендацій	01.11.2023– 30.11.2023
4.	Економічна частина	20.11.2023 – 01.12.2023

9. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи. Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком.

Додаток Б

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: «Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання»

Тип роботи: магістерська кваліфікаційна робота

Підрозділ: кафедра програмного забезпечення, ФІТКІ, ЗПІ-22м

Науковий керівник: к.т.н., асистент каф. ПЗ Тужанський С. Є.

Unicheck	
Оригінальність	**96.6%**
Схожість	3.4%

Аналіз звіту подібності

1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.

3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Черноволик Г.О.

Опис прийнятого рішення: **допустити до захисту**

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck

Автор роботи _____ Волинець О.Ю.

Керівник робота _____ Тужанський С. Є.

Додаток В

Лістинг коду бібліотеки

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;
using Microsoft.ML;
using GameAdvisor_RecommandationML.Model;

namespace GameAdvisor_RecommandationML.Model
{
  public class ConsumeModel
  {
    private static Lazy<PredictionEngine<ModelInput, ModelOutput>> PredictionEngine = new
Lazy<PredictionEngine<ModelInput, ModelOutput>>(CreatePredictionEngine);

    public static string MLNetModelPath = Path.GetFullPath("MLModel.zip");
    public static ModelOutput Predict(ModelInput input)
    {
      ModelOutput result = PredictionEngine.Value.Predict(input);
      return result;
    }

    public static PredictionEngine<ModelInput, ModelOutput> CreatePredictionEngine()
    {
      // Create new MLContext
      MLContext mlContext = new MLContext();

      // Load model & create prediction engine
      ITransformer mlModel = mlContext.Model.Load(MLNetModelPath, out var modelInputSchema);
      var predEngine = mlContext.Model.CreatePredictionEngine<ModelInput, ModelOutput>(mlModel);

      return predEngine;
    }
  }
}

namespace GameAdvisor_RecommandationML.Model
```

```
{
   public class ModelInput
   {
      [ColumnName("UserId"), LoadColumn(0)]
      public float UserId { get; set; }


      [ColumnName("Name"), LoadColumn(1)]
      public string Name { get; set; }


      [ColumnName("Behavior"), LoadColumn(2)]
      public string Behavior { get; set; }


      [ColumnName("PlayTime"), LoadColumn(3)]
      public float PlayTime { get; set; }


      [ColumnName("EndOfLine"), LoadColumn(4)]
      public float EndOfLine { get; set; }


   }

using Microsoft.ML.Data;
using Npgsql;
using System;
using System.Data.Common;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.ML;
using GameAdvisor.Recommandation.AI.Common.DataModel;
using Microsoft.ML.Trainers;
using static Microsoft.ML.DataOperationsCatalog;
using System.IO;

namespace GameAdvisor.Recommandation.AI.Common
{
   public class GameModelBuilder
```

```
{
    private readonly string _connectionString;
    private readonly MLContext mlContext;

    public TrainTestData TrainData { get; set; }
    public ITransformer Model { get; private  set; }

    public GameModelBuilder(string connectionString, MLContext context = null)
    {
        _connectionString = connectionString ?? throw new ArgumentNullException(nameof(connectionString));
        mlContext = context ?? new MLContext();
    }

    public void BuildModel()
    {
        DatabaseLoader loader = mlContext.Data.CreateDatabaseLoader<UserRate>();

        string sqlCommand = "SELECT CAST(\"UserId\" AS REAL), CAST(\"GameId\" AS REAL), CAST(\"Rating\" AS REAL)
FROM gm.\"UserRates\"";

        NpgsqlConnection connection = new NpgsqlConnection(_connectionString);
        var factory = DbProviderFactories.GetFactory(connection);
        DatabaseSource dbSource = new DatabaseSource(factory, _connectionString, sqlCommand);

        IDataView data = loader.Load(dbSource);

        TrainData = mlContext.Data.TrainTestSplit(data, 0.2);

        Model = BuildAndTrainModel(mlContext, TrainData.TrainSet);
    }

    public void SaveModel()
    {
        if (Model == null)
            throw new ArgumentNullException(nameof(Model));

        var modelPath = Path.Combine(Environment.CurrentDirectory,
            $"GameAIModel{DateTime.Now:MM-dd-yy-H-mm-ss}.zip");

        mlContext.Model.Save(Model, TrainData.TrainSet.Schema, modelPath);
    }
```

```
    public ITransformer LoadLatestModel()
    {
        DirectoryInfo currentDirectory = new DirectoryInfo(Environment.CurrentDirectory);
        FileInfo[] aiFiles = currentDirectory.GetFiles("GameAIModel*.zip");

        var filePath = aiFiles.OrderByDescending(a => a.Name).FirstOrDefault().FullName;

        DataViewSchema modelSchema;
        Model = mlContext.Model.Load(filePath, out modelSchema);

        return Model;
    }

    public float Predict(UserRate userRate)
    {
        var predictionEngine = mlContext.Model.CreatePredictionEngine<UserRate, UserRatePrediction>(Model);

        var prediction = predictionEngine.Predict(userRate);

        return float.IsNaN(prediction.Score) ? 0 : prediction.Score;
    }

    private static ITransformer BuildAndTrainModel(MLContext mlContext, IDataView trainingDataView)
    {
        IEstimator<ITransformer> estimator =
                mlContext.Transforms.Conversion.MapValueToKey(outputColumnName: "UserId", inputColumnName:
"UserId")
            .Append(mlContext.Transforms.Conversion.MapValueToKey(outputColumnName: "GameId", inputColumnName:
"GameId"));

        var options = new MatrixFactorizationTrainer.Options
        {
            MatrixColumnIndexColumnName = "UserId",
            MatrixRowIndexColumnName = "GameId",
            LabelColumnName = "Rating",
            NumberOfIterations = 100,
            ApproximationRank = 100,
            LearningRate = 0.05f,

        };

        var trainerEstimator = estimator.Append(mlContext.Recommendation().Trainers.MatrixFactorization(options));
```

```
            ITransformer model = trainerEstimator.Fit(trainingDataView);

            return model;
        }
    }
}


namespace GameAdvisor.Service.Recommandation.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class RecommendationRoundController : ControllerBase
    {
        private readonly string _connectionString;
        private readonly GameModelBuilder _modelBuilder;

        public RecommendationRoundController(IConfiguration configuration)
        {
            _connectionString = configuration.GetConnectionString("DefaultConnection");
            _modelBuilder = new GameModelBuilder(_connectionString);

        }

        [HttpGet()]
        public float Get([FromQuery] int userId, [FromQuery] int gameId)
        {
            // predict with latest model
            _modelBuilder.LoadLatestModel();

            return _modelBuilder.Predict(new UserRate() { GameId = gameId, UserId = userId });
        }

        [HttpPost()]
        public ActionResult Post()
        {
            _modelBuilder.BuildModel();
            _modelBuilder.SaveModel();

            return Ok();
        }
```

```
    }
}
namespace GameAdvisor.Recommandation.AI.Common.DataModel
{
    public class UserRate
    {
        [LoadColumn(0)]
        public float UserId { get; set; }
        [LoadColumn(1)]
        public float GameId { get; set; }
        [LoadColumn(2)]
        public float Rating { get; set; }
    }
}
```

```
            namespace GameAdvisor.Recommandation.AI.Common.DataModel
{
    public class UserRatePrediction
    {
        public float Rating { get; set; }
        public float Score { get; set; }
    }
}
```

Додаток Г

Лістинг коду утиліти

```
namespace GameAdvisor.Common.Abstractions.SteamDataModels
{
   public class SteamGameInfo
   {
      public string Url { get; set; }
      public string Types { get; set; }
      public string Name { get; set; }
      public string DescSnippet { get; set; }
      public string RecentReviews { get; set; }
      public string AllReviews { get; set; }
      public string ReleaseDate { get; set; }
      public string Developer { get; set; }
      public string Publisher { get; set; }
      public string PopularTags { get; set; }
      public string GameDetails { get; set; }
      public string Languages { get; set; }
      public string Achievements { get; set; }
      public string Genre { get; set; }
      public string GameDescription { get; set; }
      public string MatureContent { get; set; }
      public string MinimumRequirements { get; set; }
      public string RecommendedRequirements { get; set; }
      public string OriginalPrice { get; set; }
      public string DiscountPrice { get; set; }
   }
}

namespace GameAdvisor.Common.Abstractions.SteamDataModels
{
   public class SteamUserActivity
   {
      public int UserId { get; set; }
      public string Name { get; set; }
      public string Behavior { get; set; }
      public double PlayTime { get; set; }
      public int EndOfLine { get; set; }
   }
}
```

```
using GameAdvisor.Common.Abstractions.DbModels;
using Microsoft.EntityFrameworkCore;

namespace GameAdvisor.Common.Db
{
   public class GameContext : DbContext
   {
      public DbSet<Game> Games { get; set; }
      public DbSet<UserRate> UserRates { get; set; }
      public DbSet<Rate> Rates { get; set; }
      public DbSet<GenrePreference> GenrePreferences { get; set; }
      public GameContext(DbContextOptions options) : base(options) { }

      protected override void OnModelCreating(ModelBuilder modelBuilder)
      {
         modelBuilder.HasDefaultSchema("gm");

         modelBuilder.Entity<Game>().Property(g => g.GameId).ValueGeneratedOnAdd();
         modelBuilder.Entity<UserRate>().Property(g => g.UserRateId).ValueGeneratedOnAdd();
         modelBuilder.Entity<GenrePreference>().Property(g => g.GenrePreferenceId).ValueGeneratedOnAdd();



         modelBuilder.Entity<Rate>().HasKey(c => new { c.UserId, c.GameId });

         base.OnModelCreating(modelBuilder);
      }
   }
}

using AutoMapper;
using CsvHelper;
using GameAdvisor.Common.Abstractions.DbModels;
using GameAdvisor.Common.Abstractions.SteamDataModels;
using GameAdvisor.Common.Db;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Globalization;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
```

```
namespace GameAdvisor.SteamDbAggregator
{
   class Program
   {
      /// <summary>
      /// Entry point of the tool
      /// </summary>
      /// <param name="migrateDb">true to migrate DB before usage</param>
      /// <param name="connectionString">DB connection string</param>
      /// <param name="activityFile">path to Steam Activity file</param>
      /// <param name="dataFile">path to Steam Games file</param>
      /// <param name="projectPath">path for git root folder, will be used for all files</param>
      static void Main(bool migrateDb, string connectionString, string activityFile, string dataFile, string projectPath)
      {
         Console.WriteLine("Hello World from Tool!");

         // Init
         var mapper = ReturnMapper();

         IEnumerable<SteamUserActivity> activityRecords;
         IEnumerable<SteamGameInfo> dataRecords;

         // Path for files
         if (string.IsNullOrEmpty(projectPath))
         {
            if (string.IsNullOrEmpty(activityFile))
            {
               activityFile = Path.GetFullPath(@"Game Advisor\data\steam-200k.csv");
            }

            if (string.IsNullOrEmpty(dataFile))
            {
               dataFile = Path.GetFullPath(@"Game Advisor\data\steam_games.csv");
            }
         }
         else
         {
            activityFile = Path.GetFullPath($@"{projectPath}\data\steam-200k.csv");
            dataFile = Path.GetFullPath($@"{projectPath}\data\steam_games.csv");
         }
```

```
        // Getting of the Steam hour rates

        using (var reader = new StreamReader(activityFile))
        using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture))
        {
            activityRecords = csv.GetRecords<SteamUserActivity>().Where(ar => ar.Behavior == "play").ToList();
        }

        // Getting of the Steam game data

        using (var reader = new StreamReader(dataFile))
        using (var csv = new CsvReader(reader, CultureInfo.InvariantCulture))
        {
            dataRecords = csv.GetRecords<SteamGameInfo>().ToList();
        }


        string dbconnectionString = connectionString;
        if (string.IsNullOrEmpty(dbconnectionString))
        {
            dbconnectionString =
"Host=127.0.0.1;Port=5432;Database=gameadvisor;Username=postgres;Password=Welcome1**;Keepalive=150;Applicatio
n Name=GameAdvisorTool";
        }

        var optionsBuilder = new DbContextOptionsBuilder<GameContext>();
        optionsBuilder.UseNpgsql(dbconnectionString);

        var gameContext = new GameContext(optionsBuilder.Options);

        if (migrateDb)
        {
            Console.WriteLine("Migration started");
            gameContext.Database.Migrate();
            Console.WriteLine("Migration finished");
        }

        Console.WriteLine("Games import started");
        foreach (var gameItem in dataRecords)
        {
            var game = mapper.Map<Game>(gameItem);
            gameContext.Games.Add(game);
```

```
}

gameContext.SaveChanges();
Console.WriteLine("Games finished started");

Console.WriteLine("Rates import started");
Dictionary<string, int> existingGames = new Dictionary<string, int>();

int counter = 0;
foreach (var rateItem in activityRecords)
{
   int existingGameId = 0;

   if (existingGames.ContainsKey(rateItem.Name.ToLower()))
   {
      existingGameId = existingGames[rateItem.Name.ToLower()];
   }
   else
   {
      var existingGame = gameContext.Games.FirstOrDefault(g => g.Name.ToLower() == rateItem.Name.ToLower());
      if (existingGame != null)
      {
         existingGames.Add(existingGame.Name.ToLower(), existingGame.GameId);
         existingGameId = existingGame.GameId;
      }
   }

   if (existingGameId != 0)
   {
      gameContext.UserRates.Add(new UserRate()
      {
         GameId = existingGameId,
         UserId = rateItem.UserId,
         Rating = GetRateByTime(rateItem.PlayTime)
      });
   }

   if (counter % 1000 == 0)
   {
      gameContext.SaveChanges();
      Console.WriteLine($"Processed count {counter}, saved items: {existingGames.Count()}");
   }
```

```
            counter++;
        }
        Console.WriteLine("Rates import finished");

        gameContext.SaveChanges();

        Console.ReadLine();
    }

    private static IMapper ReturnMapper()
    {
        var config = new MapperConfiguration(cfg =>
        {
            cfg.CreateMap<SteamGameInfo, Game>();
            cfg.CreateMap<SteamUserActivity, UserRate>();
        }
        );

        return config.CreateMapper();
    }

    private static double GetRateByTime(double playTime)
    {
        return playTime switch
        {
            double value when value >= 40=> 1, // perfect game
            double value when value >= 10 && value < 40 => 0.9, // very good
            double value when value >= 8 && value < 10 => 0.8, // good enough
            double value when value >= 4 && value < 8 => 0.6, // medium
            double value when value >= 2 && value < 4 => 0.3, // pretty bad
            _ => 0.1 // every work worse smth
        };
    }
  }
}
```

Додаток Д

Лістинг коду веб застосунку

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;

namespace GameAdvisor.Service.AuthServer.WebApi
{
  public class Program
  {
    public static void Main(string[] args)
    {
      CreateHostBuilder(args).Build().Run();
    }

    public static IHostBuilder CreateHostBuilder(string[] args) =>
      Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
        {
          webBuilder.UseStartup<Startup>();
        });
  }
}

using Microsoft.IdentityModel.Tokens;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Http;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using GameAdvisor.Service.AuthServer.WebApi.Options;
using Microsoft.Extensions.Configuration;
using Microsoft.EntityFrameworkCore;
using GameAdvisor.Service.AuthServer.Infrastructure.Data.Context;
using GameAdvisor.Service.AuthServer.Infrastructure.Data.Model;
```

```
using Microsoft.AspNetCore.Identity;

namespace GameAdvisor.Service.AuthServer.WebApi
{
   public class Startup
   {
      public IConfiguration Configuration { get; }

      public Startup(IConfiguration configuration)
      {
         Configuration = configuration;
      }

      public void ConfigureServices(IServiceCollection services)
      {
         string connectionString = Configuration.GetConnectionString("DefaultConnection");

         services.AddDbContext<AppIdentityDbContext>(options => options.UseNpgsql(connectionString));

         services.AddIdentity<AppUser, IdentityRole<int>>()
            .AddEntityFrameworkStores<AppIdentityDbContext>()
            .AddDefaultTokenProviders();

         services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
            .AddJwtBearer(options =>
            {
               options.RequireHttpsMetadata = false;
               options.TokenValidationParameters = new TokenValidationParameters
               {
                  ValidateIssuer = true,
                  ValidIssuer = AuthOptions.ISSUER,

                  ValidateAudience = true,
                  ValidAudience = AuthOptions.AUDIENCE,

                  ValidateLifetime = true,
                  IssuerSigningKey = AuthOptions.GetSymmetricSecurityKey(),
                  ValidateIssuerSigningKey = true
               };
            });

         //services.AddCors(options =>
```

```
        //{
        //    options.AddDefaultPolicy(builder =>
        //    {
        //        builder.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader();
        //    });
        //});

        services.AddControllers();

    }

    // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }

        app.UseRouting();
        app.UseDefaultFiles();
        app.UseStaticFiles();
        //app.UseCors();

        app.UseAuthentication();
        app.UseAuthorization();
        app.UseEndpoints(endpoints =>
        {
            endpoints.MapControllers();
        });
    }
  }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using AutoMapper;
using GameAdvisor.Common.Abstractions.DbModels;
using GameAdvisor.Common.Db;
using GameAdvisor.Service.Games.Abstractions.Models;
using GameAdvisor.Service.Games.Abstractions.Requests;
```

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;

namespace GameAdvisor.Service.Games.Controllers
{
   [ApiController]
   [Route("api/[controller]")]
   public class GamesController : ControllerBase
   {
      private readonly GameContext _gameContext;
      private readonly IMapper _mapper;

      public GamesController(GameContext gameContext, IMapper mapper)
      {
         _gameContext = gameContext;
         _mapper = mapper;
      }
      // GET: api/<GamesController>
      [HttpGet]
      public IEnumerable<GameModel> Get([FromQuery] GameListRequest request)
      {
         if (request == null)
         {
            throw new ArgumentException(nameof(request));
         }

         var query = _gameContext.Games.Skip(request.Page * request.ItemCount);
         if (!string.IsNullOrEmpty(request.FilterName))
         {
            query = query.Where(x => x.Name.ToLower().Contains(request.FilterName.ToLower()));
         }
         if (!string.IsNullOrEmpty(request.FilterDeveloper))
         {
            query = query.Where(x => x.Developer.ToLower().Contains(request.FilterDeveloper.ToLower()));
         }
         if (!string.IsNullOrEmpty(request.FilterGenre))
         {
            query = query.Where(x => x.Genre.ToLower().Contains(request.FilterGenre.ToLower()));
         }
         if (!string.IsNullOrEmpty(request.FilterPublisher))
         {
```

```
            query = query.Where(x => x.Publisher.ToLower().Contains(request.FilterPublisher.ToLower()));
        }
        if (!string.IsNullOrEmpty(request.FilterTag))
        {
            query = query.Where(x => x.PopularTags.ToLower().Contains(request.FilterTag.ToLower()));
        }

        // Nicer default games
        query = query.OrderBy(x => x.GameId);

        return _mapper.Map<IEnumerable<Game>, IEnumerable<GameModel>>(query.Take(request.ItemCount).ToList());
    }

    // GET api/<GamesController>/5
    [HttpGet("{id}")]
    public GameModel Get(int id)
    {
        return _mapper.Map<Game, GameModel>(_gameContext.Games.FirstOrDefault(x => x.GameId == id));
    }

    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    [HttpGet("test")]
    public bool Get()
    {
        return true;
    }
  }
}
```

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Security.Claims;
using System.Threading.Tasks;
using AutoMapper;
using GameAdvisor.Common.Abstractions.DbModels;
using GameAdvisor.Common.Db;
using GameAdvisor.Service.Games.Abstractions.Models;
using GameAdvisor.Service.Games.Abstractions.Requests;
using GameAdvisor.Service.Games.Configuration;
using GameAdvisor.Service.Games.Logic;
```

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;

// For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860

namespace GameAdvisor.Service.Games.Controllers
{
   [Route("api/[controller]")]
   [ApiController]
   public class RatesController : BaseGameAdvisorController
   {
      private readonly GameContext _gameContext;
      private readonly IMapper _mapper;
      private readonly IConfiguration _configuration;
      private readonly IHttpClientFactory _clientFactory;

      public RatesController(GameContext gameContext, IMapper mapper,
            IConfiguration configuration, IHttpClientFactory clientFactory)
      {
         _gameContext = gameContext;
         _mapper = mapper;
         _configuration = configuration;
         _clientFactory = clientFactory;
      }

      // GET: api/<RatesController>
      [HttpGet]
      [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
      public IEnumerable<RateModel> Get()
      {
         int userId = GetUserId();

         var rates = _gameContext.Rates.Include(r => r.Game).Where(r => r.UserId == userId).OrderByDescending(r =>
r.SortOrder).ToList();

         return _mapper.Map<IEnumerable<RateModel>>(rates);
      }

      // POST api/<RatesController>
```

```
[HttpPost]
[Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
public async Task Post([FromBody] CreateRateRequest request)
{
   int userId = GetUserId();

   var rates = new List<Rate>();

   // Clear old rates
   var currentRates = _gameContext.Rates.Where(r => r.UserId == userId).ToList();
   _gameContext.Rates.RemoveRange(currentRates);

   // Save new rates
   foreach (var item in request)
   {
      _gameContext.Rates.Add(new Rate()
      {
         GameId = item.GameId,
         SortOrder = item.SortOrder,
         UserId = userId
      });
   }
   _gameContext.SaveChanges();

   // Calculate genres
   var newRates = _gameContext.Rates.Include(r => r.Game).Where(r => r.UserId == userId).ToList();
   var calculator = new GenreCalculator(newRates);
   var genresDict = calculator.Calculate();

   // Clear old preferences
   var oldPreferences = _gameContext.GenrePreferences.Where(r => r.UserId == userId).ToList();
   _gameContext.GenrePreferences.RemoveRange(oldPreferences);
   _gameContext.SaveChanges();

   // Save new preferences
   foreach (var genre in genresDict)
   {
      _gameContext.GenrePreferences.Add(new GenrePreference() {
         Genre = genre.Key,
         Percentage = genre.Value,
         UserId = userId
      });
```

```
        }

        // Clear old UserRates for game
        var oldUserRates = _gameContext.UserRates.Where(r => r.UserId == userId).ToList();
        _gameContext.UserRates.RemoveRange(oldUserRates);
        _gameContext.SaveChanges();


        // Add new userRates
        var rateStep = 1f /request.Count;
        var rateAcum = 1f;
        //var rateCalcPart = 100f / totalRatePart;
        foreach (var rateItem in request.OrderByDescending(r => r.SortOrder).ToList())
        {
           _gameContext.UserRates.Add(new UserRate()
           {
              UserId = userId,
              GameId = rateItem.GameId,
              Rating = rateAcum != 0 ? rateAcum : 0.1
           });
           rateAcum -= rateStep;
        }

        _gameContext.SaveChanges();

        // Regenerate AI file
        var recommendationRoundRequest = new HttpRequestMessage(HttpMethod.Post,
           $"{_configuration.GetValue(typeof(string), "recommendationRoundApi")}RecommendationRound");
        using (var client = _clientFactory.CreateClient("recommendationRound"))
        {
           var response = await client.SendAsync(recommendationRoundRequest);
        }
     }
  }
}
            using System;
using System.Collections.Generic;
using System.Globalization;
using System.Linq;
using System.Net.Http;
using System.Security.Claims;
using System.Threading.Tasks;
```

```
using AutoMapper;
using GameAdvisor.Common.Abstractions.DbModels;
using GameAdvisor.Common.Db;
using GameAdvisor.Service.Games.Configuration;
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;

// For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860

namespace GameAdvisor.Service.Games.Controllers
{
  [Route("api/[controller]")]
  [ApiController]
  public class RecommendationsController : BaseGameAdvisorController
  {
    private readonly GameContext _gameContext;
    private readonly IMapper _mapper;
    private readonly IConfiguration _configuration;
    private readonly IHttpClientFactory _clientFactory;

    public RecommendationsController(GameContext gameContext, IMapper mapper,
        IConfiguration configuration, IHttpClientFactory clientFactory)
    {
      _gameContext = gameContext;
      _mapper = mapper;
      _configuration = configuration;
      _clientFactory = clientFactory;
    }

    // GET: api/<RecommendationsController>
    [HttpGet]
    [Authorize(AuthenticationSchemes = JwtBearerDefaults.AuthenticationScheme)]
    public async Task<IEnumerable<Game>> Get([FromQuery] int count)
    {
      int userId = GetUserId();

      // Get game preferences
      var genrePreferences = _gameContext.GenrePreferences.Where(gp => gp.UserId == userId).ToList();
      var gamesByGenre = new Dictionary<string, List<Game>>();
```

```
        var orientedGameCount = count * genrePreferences.Count();

        var gameCount = _gameContext.Games.Count();
        var rnd = new Random();
        var rndOffset = rnd.Next(0, gameCount);

        var finalGames = new List<Game>();

        foreach (var genre in genrePreferences.OrderByDescending(g => g.Percentage).ToList())
        {
          // Get random games by genre
          var genredGames = _gameContext.Games.Skip(rndOffset)
            .Where(g => g.Genre.Contains(genre.Genre))
            .Take(orientedGameCount).ToList();

          if (!gamesByGenre.ContainsKey(genre.Genre))
          {
            gamesByGenre[genre.Genre] = new List<Game>();
          }

          // Exclude current games
          var ratedGames = _gameContext.Rates.Include(r => r.Game)
            .Where(x => x.UserId == userId).Select(r => r.Game).ToList();
          genredGames = genredGames.Except(ratedGames).ToList();

          foreach (var gameCandidate in genredGames.Take(count).ToList())
          {
            // Do AI here

            try
            {
              var recommendationRoundRequest = new HttpRequestMessage(HttpMethod.Get,
                $"{_configuration.GetValue(typeof(string),
"recommendationRoundApi")}RecommendationRound?userId={userId}&gameId={gameCandidate.GameId}");
              using (var client = _clientFactory.CreateClient("recommendationRound"))
              {
                var response = client.SendAsync(recommendationRoundRequest);

                var predictedResult = await response.Result.Content.ReadAsStringAsync();

                var predictedValue = float.Parse(predictedResult, CultureInfo.InvariantCulture.NumberFormat);
```

```
                if (predictedValue >= GameConsts.AcceptablePrediction)
                {
                    gamesByGenre[genre.Genre].Add(gameCandidate);
                }
            }
        }
        catch (Exception ex)
        {
            // Just for demo, but haven't seen any issues
            if (gameCandidate.GameId % 2 == 1)
            {
                gamesByGenre[genre.Genre].Add(gameCandidate);
            }
            throw;
        }
    }

    if (gamesByGenre[genre.Genre].Count < count)
    {
        gamesByGenre[genre.Genre].AddRange(genredGames.Take(count).ToList());
        gamesByGenre[genre.Genre] = gamesByGenre[genre.Genre].Distinct().ToList();
    }

    // Add game in percentage
    var countToTakeFloat = count * genre.Percentage;
    var countToTake = (int)countToTakeFloat;
    if (countToTakeFloat != countToTake)
    {
        countToTake++;
    }

    finalGames.AddRange(gamesByGenre[genre.Genre].Take(countToTake).ToList());

    rndOffset = rnd.Next(0, gameCount);
  }

  return finalGames.Take(count).ToList();
 }
 }
}
```

```
            using GameAdvisor.Common.Abstractions.DbModels;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace GameAdvisor.Service.Games.Logic
{
   public class GenreCalculator
   {
      private List<Rate> _rates;

      public GenreCalculator(List<Rate> rates) => _rates = rates ?? throw new ArgumentNullException(nameof(rates));

      public Dictionary<string, float> Calculate()
      {
         var genresDict = new Dictionary<string, int>();
         var accumulator = 0;

         foreach (var rate in _rates.OrderByDescending(r => r.SortOrder).ToList())
         {
            var genres = rate.Game.Genre.Split(',');
            foreach (var genre in genres)
            {
               if (!genresDict.ContainsKey(genre))
               {
                  genresDict[genre] = 0;
               }
               genresDict[genre] += rate.SortOrder;
               accumulator += rate.SortOrder;
            }
         }

         var calcPart = 1f / accumulator;

         var resultWithPercent = new Dictionary<string, float>();

         foreach (var genreItem in genresDict)
         {
            resultWithPercent.Add(genreItem.Key, genreItem.Value * calcPart);
         }
```

```
            return resultWithPercent;
        }
    }
}
                using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using Microsoft.EntityFrameworkCore;
using GameAdvisor.Common.Db;
using AutoMapper;
using GameAdvisor.Service.Games.Configuration;
using Microsoft.IdentityModel.Tokens;
using GameAdvisor.Service.AuthServer.WebApi.Options;

namespace GameServiceAdvisor.Service.GameService
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddHttpClient();

            var mapperConfig = new MapperConfiguration(mc =>
            {
```

```
        mc.AddProfile(new AutomapperProfile());
    });

    IMapper mapper = mapperConfig.CreateMapper();
    services.AddSingleton(mapper);

    string connection = Configuration.GetConnectionString("DefaultConnection");
    services.AddDbContext<GameContext>(options =>
        options.UseNpgsql(connection));
    services.AddAuthentication("Bearer")
        .AddJwtBearer("Bearer", opt =>
        {
            opt.RequireHttpsMetadata = false;
            opt.TokenValidationParameters = new TokenValidationParameters
            {
                ValidateIssuer = true,
                ValidIssuer = AuthOptions.ISSUER,

                ValidateAudience = true,
                ValidAudience = AuthOptions.AUDIENCE,

                ValidateLifetime = true,
                IssuerSigningKey = AuthOptions.GetSymmetricSecurityKey(),
                ValidateIssuerSigningKey = true
            };
        });
    services.AddControllers();
}

// This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    UpdateDatabase(app);

    app.UseHttpsRedirection();

    app.UseRouting();
```

```
            app.UseAuthentication();
            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
                endpoints.MapControllers();
            });
        }

        private static void UpdateDatabase(IApplicationBuilder app)
        {
            using (var serviceScope = app.ApplicationServices
                .GetRequiredService<IServiceScopeFactory>()
                .CreateScope())
            {
                using (var context = serviceScope.ServiceProvider.GetService<GameContext>())
                {
                    context.Database.Migrate();
                }
            }
        }
    }
}
```

Додаток Е

Лістинг коду інтерфейсу

```
import { Component, OnInit, Inject } from '@angular/core';
import { MatDialogRef, MAT_DIALOG_DATA } from '@angular/material/dialog';
import { CoreService } from '../../services';
import { mergeMap, catchError } from 'rxjs/operators';
import { EMPTY, of } from 'rxjs';
import { AppState } from 'src/app/reducers';
import { Store } from '@ngrx/store';
import * as gameSelectors from '../../../dashboard/game/selectors/game.selectors';
import { ToastrService } from 'ngx-toastr';

@Component({
  selector: 'app-battle-screen-dialog',
  templateUrl: './battle-screen-dialog.component.html',
  styleUrls: ['./battle-screen-dialog.component.scss']
})
export class BattleScreenDialogComponent implements OnInit {
  gameToInsert: any = { }; // getting from wish-list
  suggestedGame: any;  // middle game

  selectedGame: any; // game that user select

  ratedGames: any;

  startIndex: number;
  endIndex: number;
  middleGameIndex: number;
  isLastStep: boolean;

  constructor(
    public dialogRef: MatDialogRef<BattleScreenDialogComponent>,
    private coreService: CoreService,
    private store$: Store<AppState>,
    private toastr: ToastrService,
    @Inject(MAT_DIALOG_DATA) public data: any,
  ) {

    this.gameToInsert.game = data;

  }
```

```
ngOnInit(): void {
  this.coreService.getRate().subscribe((games) => {
    this.ratedGames = games.map(game => {
      return {
        ...game,
        game: {
          ...game.game,
          imageUrl: this.getImageUrl(game.game.url)
        }
      };
    });

    this.onStart(this.ratedGames);
  });
}

onStart(games: Array<any>){
  this.startIndex = 0;
  this.endIndex = games.length - 1;
  this.middleGameIndex = Math.floor((this.startIndex + this.endIndex) / 2);

  this.suggestedGame = games[this.middleGameIndex];
}

onClose() {
  this.dialogRef.close();
}

onSelectEvent(event) {
  switch (event.action) {
    case 'selectedGame':
      console.log(event.data);

      const gameToInsertSelected = event.data.gameId === this.gameToInsert.game.gameId;

      if (this.isLastStep){
        const isInsertAtEnd = this.endIndex === this.ratedGames.length - 1;

        if (gameToInsertSelected){
          isInsertAtEnd ?
            this.ratedGames.splice(this.endIndex + 1, 0, this.gameToInsert.game) :
```

```
          this.ratedGames.splice(this.endIndex, 0, this.gameToInsert.game);
        }else{
          isInsertAtEnd ?
          this.ratedGames.splice(this.endIndex, 0, this.gameToInsert.game) :
          this.ratedGames.splice(this.startIndex, 0, this.gameToInsert.game);
        }

        this.sendRates(this.ratedGames);
        this.onClose();
        this.toastr.success('Successfully Battled ^-^', 'Wow!');
        return;
      }

      if (gameToInsertSelected){
        this.search(this.middleGameIndex, this.endIndex, gameToInsertSelected);
      }else{
        this.search(this.startIndex, this.middleGameIndex, gameToInsertSelected);
      }

      break;
    default:
      break;
  }
}

search(start: any, end: any, gameToInsertSelected: boolean){
  this.startIndex = start;
  this.endIndex = end;

  if ((end - start) === 1) {
    this.isLastStep = true;
    if (end === this.ratedGames.length - 1) {
      this.suggestedGame = this.ratedGames[this.ratedGames.length - 1];
    }else if (start === 0){
      this.suggestedGame = this.ratedGames[0];
    }
    else{
      // FINISH
      const insertIndex = gameToInsertSelected ? this.middleGameIndex + 1 : this.middleGameIndex;
      this.ratedGames.splice(insertIndex, 0, this.gameToInsert.game);

      this.sendRates(this.ratedGames);
```

```
            this.onClose();
            this.toastr.success('Successfully Battled ^-^', 'Wow!');
          }
          return;
        }
        this.middleGameIndex = Math.floor((start + end) / 2);

        this.suggestedGame = this.ratedGames[this.middleGameIndex];
      }

      sendRates(games: any){
        const preparedRates = games.map((game, i) => {
          return {
            gameId: game.gameId,
            sortOrder: games.length - i
          };
        });

        this.coreService.saveRate(preparedRates).subscribe();
      }

      public binarySearch = (data: Array<any>, target: any, start: any, end: any) => {
        if (end < 1) { return data[0]; }
        const middle = Math.floor((start + (end - start) / 2));
        if (target === data[middle]) { return data[middle]; }
        if ((end - 1) === start) { return Math.abs(data[start] - target) > Math.abs(data[end] - target) ? data[end] :
data[start]; }
        if (target > data[middle]) { return this.binarySearch(data, target, middle, end); }
        if (target < data[middle]) { return this.binarySearch(data, target, start, middle); }
      }

      getImageUrl(url){
        const key = url.split('/')[4];
        return `https://cdn.cloudflare.steamstatic.com/steam/apps/${key}/header.jpg`;
      }

    }

    import { Action } from '@ngrx/store';
    import { Dialog } from '../models/dialog';

    export enum LayoutActionTypes {
```

```
  OPEN_DIALOG = '[Layout] Open dialog',
  OPEN_DIALOG_SUCCESS = '[Layout] Open dialog success',
  CLOSE_DIALOG = '[Layout] Close dialog',
  SET_MESSENGER_STATUS = '[Layout] Set Messenger Status',
  ADD_WARNING_NOTIFICATION = '[Layout] Add Warning Notification',
  DELETE_WARNING_NOTIFICATION = '[Layout] Delete Warning Notification',
  CLEAR_WARNING_NOTIFICATIONS = '[Layout] Clear Warning Notifications',
}

/** Dialog actions */
export class OpenDialogAction implements Action {
  readonly type = LayoutActionTypes.OPEN_DIALOG;

  constructor(public payload: Dialog) {}
}

export class OpenDialogSuccessAction implements Action {
  readonly type = LayoutActionTypes.OPEN_DIALOG_SUCCESS;

  constructor(public payload: string) {}
}

export class CloseDialogAction implements Action {
  readonly type = LayoutActionTypes.CLOSE_DIALOG;

  constructor(public payload: string) {}
}

/** Set Messenger Status (opened / closed) */
export class SetMessengerStatus implements Action {
  readonly type = LayoutActionTypes.SET_MESSENGER_STATUS;
  constructor(public payload: boolean) {}
}

/** Add Warning Notification */
export class AddWarningNotification implements Action {
  readonly type = LayoutActionTypes.ADD_WARNING_NOTIFICATION;
  constructor(public payload: {type: string, content: any}) {}
}

/** Delete Warning Notification */
export class DeleteWarningNotification implements Action {
```

```
  readonly type = LayoutActionTypes.DELETE_WARNING_NOTIFICATION;
  constructor(public payload: string) {}
}

/** Clear Warning Notifications */
export class ClearWarningNotifications implements Action {
  readonly type = LayoutActionTypes.CLEAR_WARNING_NOTIFICATIONS;
}

export type LayoutActions =
  CloseDialogAction
  | OpenDialogAction
  | OpenDialogSuccessAction
  | SetMessengerStatus
  | AddWarningNotification
  | DeleteWarningNotification
  | ClearWarningNotifications
;
```

```
<span class="close-button" (click)="onClose()"></span>
<div class="battle-screen-container">
  <app-single-card [game]="suggestedGame?.game"
    [selectable]="true"
    [canSelectGames]="true"
    [showFavoriteButton]="false"
    (gameEvent)="onSelectEvent($event)"
    [battleScreen]="true"></app-single-card>

  <span class="vs">VS</span>

  <app-single-card [game]="gameToInsert?.game"
    [selectable]="true"
    [canSelectGames]="true"
    [showFavoriteButton]="false"
    (gameEvent)="onSelectEvent($event)"
    [battleScreen]="true"></app-single-card>
</div>
<button mat-button class="default-button big" [mat-dialog-close]="true" cdkFocusInitial>Finish</button>

@import 'variables';
```

```
.battle-screen-container {
 display: flex;
 align-items: center;
 justify-content: center;
 height: 100%;

 app-single-card {
  margin: 0 25px;
  &:first-of-type {
   margin-bottom: 40px;
  }
  &:last-of-type {
   margin-top: 40px;
  }
 }

 .vs {
  font-size: 85px;
  font-family: $font-italic;
  color: #ff8a00;
  position: absolute;
  z-index: 5;
  top: 50%;
  left: 50%;
  transform: translate(-50%,-50%);
 }

 @media screen and (max-width: 967px) {
  flex-direction: column;

  app-single-card {
   margin: 0;

   &:first-of-type,
   &:last-of-type {
    margin: 0;
   }
  }

  .vs {
   position: static;
   transform: initial;
```

```
    }
   }
  }
```

```
  .default-button {
   position: absolute;
   bottom: 10px;
   left: 50%;
   transform: translateX(-50%);
  }
  <span class="close-button" (click)="onClose()"></span>
  <h2 class="best-five-title">Choose best 5 games</h2>
  <mat-form-field class="search-field">
   <mat-label>Search</mat-label>
   <input matInput type="text" (keyup.enter)="findGames()" [(ngModel)]="searchField">
   <button mat-button *ngIf="searchField" matSuffix mat-icon-button aria-label="Clear" (click)="searchField=''"
style="color: red;">
    &#9932;
   </button>
  </mat-form-field>
  <mat-dialog-content>
   <div class="choose-five-container" *ngIf="!hide">
    <app-single-card
     *ngFor="let game of allGames$ | async"
     [game]="game"
     [selectable]="true"
     [showFavoriteButton]="false"
     [canSelectGames]="selectedGames.length < 5"

     (gameEvent)="onSelectEvent($event)"
    ></app-single-card>
   </div>
  </mat-dialog-content>
  <mat-dialog-actions align="center">
   <button mat-button class="default-button big" (click)="onSubmit()">Select</button>
   <button mat-button class="default-button big" (click)="clearSelectedGames()">Clear Selection</button>
  </mat-dialog-actions>
  @import 'mixins';
```

```
h2.best-five-title {
  @include text-linear-gradient-background();
  margin-bottom: 10px;
}

.choose-five-container {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(350px, 1fr));
  gap: 40px 15px;
}

.search-field {
  margin: 0 auto;
  display: block;
  max-width: 250px;
  color: white;
}
import { Component, OnInit, ChangeDetectorRef } from '@angular/core';
import { MatDialogRef } from '@angular/material/dialog';
import { select, Store } from '@ngrx/store';
import { Observable } from 'rxjs/internal/Observable';
import { AppState } from 'src/app/reducers';
import * as gameSelectors from '../../../dashboard/game/selectors/game.selectors';
import { CoreService } from '../../services';
import { GetGames } from 'src/app/dashboard/game/actions/game.actions';
import { Router } from '@angular/router';

@Component({
  selector: 'app-choose-five-dialog',
  templateUrl: './choose-five-dialog.component.html',
  styleUrls: ['./choose-five-dialog.component.scss']
})
export class ChooseFiveDialogComponent implements OnInit {
  searchField = '';
  allGames$: Observable<Array<any>>;
  selectedGames = [];
  currentUserId = JSON.parse(localStorage.getItem('userId'));
  hide = false;

  constructor(
    public dialogRef: MatDialogRef<ChooseFiveDialogComponent>,
    private store$: Store<AppState>,
```

```
  private coreService: CoreService,
  private changeDetector: ChangeDetectorRef,
  private router: Router
) {
 this.allGames$ = this.store$.pipe(select(gameSelectors.getAllGames));
}

ngOnInit(): void {}

onClose() {
  this.dialogRef.close();
}

onSubmit() {
  if (this.selectedGames && this.selectedGames.length === 5) {

    localStorage.setItem('isActive', 'true');
    this.coreService.topFiveSelected(this.currentUserId).subscribe();
    this.coreService.saveRate(this.selectedGames).subscribe();
    this.router.navigate(['/', 'games']);
    this.onClose();
  }
}

onSelectEvent(event) {
  switch (event.action) {
    case 'selectedGame':
      console.log(event.data);
      this.addGameToSelected(event.data);

      break;
    default:
      break;
  }
}

addGameToSelected(game: any) {
  this.selectedGames.push({
      gameId: game.gameId,
      sortOrder: this.selectedGames.length + 1
    });
```

```
    /*const index = this.selectedGames.findIndex(v => v.gameId === game.gameId);
    index === -1 ? this.selectedGames.push({gameId: game.gameId}) : this.selectedGames.splice(index, 1);*/
  }

  clearSelectedGames(){
    this.selectedGames = [];
    this.hide = true;
    this.changeDetector.detectChanges();
    this.hide = false;
  }

  findGames(){
    this.store$.dispatch(new GetGames(30, 0, this.searchField));
  }
}
```

```
<mat-toolbar>
  <div routerLink="/" class="logo-box">
    <img src="../../../../assets/img/logo-1.png" alt="logo">
  </div>
  <app-nav-item [routerLinkActive]="['active-link']">
  </app-nav-item>
  <app-user-settings
    (userEvent)="onUserEvents($event)"
  >
  </app-user-settings>
</mat-toolbar>

<div id="router-wrap" class="router-wrap">
  <router-outlet></router-outlet>
</div>
@import 'variables';
@import 'mixins';

.logo-box {
  cursor: pointer;
  height: 100%;

  img {
    height: 60px;
    width: auto;
```

```
  }
 }

 .nav-list-wrapper {
  padding-bottom: 60px;
 }

 .mat-toolbar {
  display: flex;
  align-items: center;
  justify-content: space-between;
  z-index: 3;
  position: relative;
  height: 60px;
  padding: 0 36px;
  background-color: #22262ff0;
  width: 100%;
  @include box-shadow(0, 1px, 3px, 0, rgba(0, 0, 0, 0.17));
 }

 .router-wrap {
  background-color: #22262f;
  top: 60px;
  bottom: 0;
  padding: 15px 32px 30px 25px;
  position: absolute;
  overflow-x: hidden;
  width: 100%;
 }
 import { Component, OnDestroy, OnInit } from '@angular/core';
 import { Observable } from 'rxjs';
 import { select, Store } from '@ngrx/store';
 import { AppState } from '../../../reducers';
 import * as authActions from '../../../home/actions/auth.actions';
 import * as selectFromAuth from '../../../home/selectors/auth.selectors';
 import { CoreService } from '../../services';
 import { Config } from 'src/app/config';
 import { BattleScreenDialogComponent } from '../../components/battle-screen-dialog/battle-screen-
dialog.component';
 import { ChooseFiveDialogComponent } from '../../components/choose-five-dialog/choose-five-
dialog.component';
 import { Router } from '@angular/router';
```

```
@Component({
 selector: 'app-main',
 templateUrl: './main.component.html',
 styleUrls: ['./main.component.scss']
})
export class MainComponent implements OnInit, OnDestroy {
 userProfile$: Observable<any>;
 allGames$: Observable<any>;
 allGames: any;
 isTopFivePassed = JSON.parse(localStorage.getItem('isActive'));

 constructor(
  private store$: Store<AppState>,
  private coreService: CoreService,
  private config: Config,
  private router: Router,
 ) {
  this.userProfile$ = this.store$.pipe(select(selectFromAuth.getCurrentUser));
 }

 ngOnInit(): void {
  if (!this.isTopFivePassed) {
   this.openChooseFiveDialog();
  }
 }

 ngOnDestroy(): void {
 }

 openBattleScreenDialog() {
  this.coreService.openDialog(
   BattleScreenDialogComponent,
   this.config.defaultDialogConfig
  );
 }

 openChooseFiveDialog() {
  this.coreService.openDialog(
   ChooseFiveDialogComponent,
   this.config.fullscreenDialogConfig
  );
```

```
  }

  onUserEvents(event) {
    switch (event.action) {
      case 'logout':
        this.store$.dispatch(new authActions.LogOut());
        this.router.navigate(['/', 'home']);
        break;
      default:
        break;
    }
  }
}
```

Додаток Є

ІЛЮСТРАТИВНА ЧАСТИНА

«Метод і програмний засіб персоналізованих рекомендацій на основі машинного навчання»

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення

Презентація магістерської кваліфікаційної роботи на тему:

МЕТОД І ПРОГРАМНИЙ ЗАСІБ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ НА ОСНОВІ МАШИННОГО НАВЧАННЯ

Виконав студент групи ЗПІ-22м Волинець О.Ю.

Науковий керівник к.т.н., асист. каф. ПЗ Тужанський С. Є.

Мета, предмет та об'єкт дослідження

- Мета і задачі дослідження. Метою роботи є підвищення точності генерації персоналізованих рекомендацій у рекомендаційних системах за рахунок використання методів машинного навчання з урахуванням індивідуальних потреб користувачів.
- Об'єкт дослідження – процес надання персоналізованих рекомендацій.
- Предмет дослідження – методи і засоби формування персоналізованих рекомендацій.

Новизна роботи та Практичне значення одержаних результатів

- Удосконалено надання персоналізованих рекомендацій на основі навчання, який є на відміну від інших є поєднанням класичних методів рекомендацій із машинним навчанням та побудовою матриць уподобань за принципом бінарного пошуку, що дозволило сформувати більш ефективні і точні персоналізовані рекомендації на основі попередніх даних шляхом побудови залежностей вподобань з урахуванням результатів моделі машинного навчання.
- На основні запропонованого методу розроблено програмний засіб у вигляді сервісів і веб додатків, який використовує удосконалений метод надання персоналізованих рекомендацій на основі машинного навчання з відповідною навченою моделлю на основі реальних даних.

Актуальність роботи

Популяризація інтернет послуг створює попит на ефективні системи рекомендацій. У сучасному світі щодня генерується велика кількість даних, аналіз яких у інформаційних системах є складним завданням. Постійне зростання обсягів інформації у мережах додатково ускладнює їх використання. Персоналізовані системи рекомендацій допомагають ефективно аналізувати та використовувати потрібні дані для покращення взаємодії з користувачем.

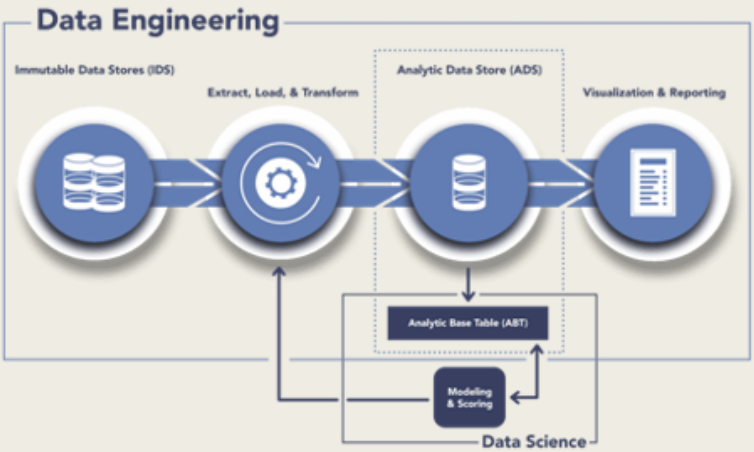
Постановка задачі

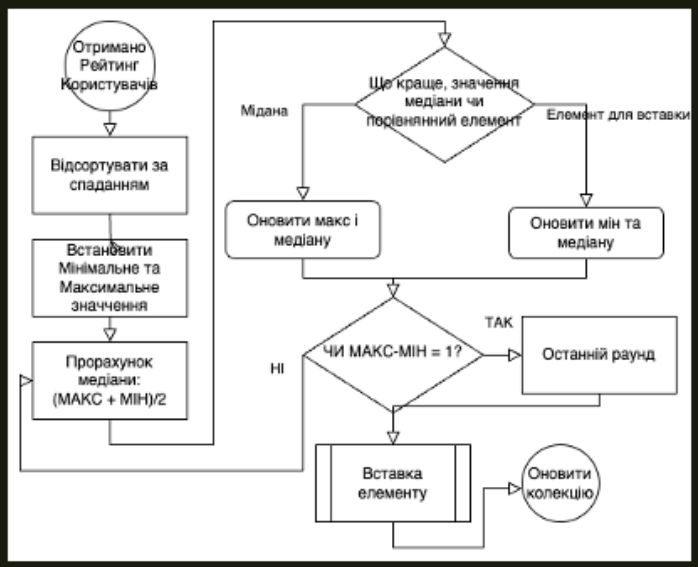
- Аналіз сучасних методів і підходів до створення персоналізованих рекомендацій та рекомендаційних систем;
- Розробка та вдосконалення навчання моделей з рекомендаційними алгоритмами, які враховують контентні та колаборативні підходи, аналіз та порівняння різних моделей за їхньою ефективністю;
- Удосконалення алгоритму формування рекомендацій з урахуванням результатів моделі машинного навчання;
- Розробка програмного засобу, який базується на удосконалених моделях та забезпечує створення персоналізованих рекомендацій для кінцевих користувачів;
- Проведення тестування розробленого програмного засобу та аналіз ефективності рекомендацій з використанням інтегрованих бібліотек

АНАЛІЗ АНАЛОГІВ

Назва методу	Простота реалізації	Зрозумілість вимог до контенту	Ефективність для нових користувачів	Урахування зв'язків
Колаборативний фільтринг	+	++	-	++
Контент-базованоий фільтринг	+-	–	+	-
Матричний розкладу та Факторизаційні машини	+-	+-	+-	-
Методи засновані на Нейронних мережах	–	–	+	+

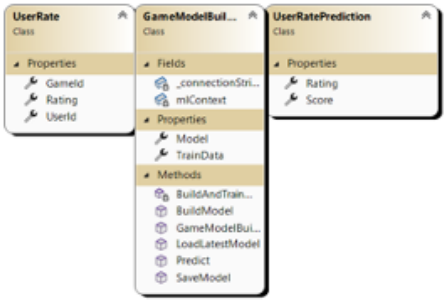
Аналіз вимог і операцій над даними

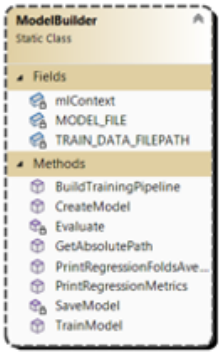




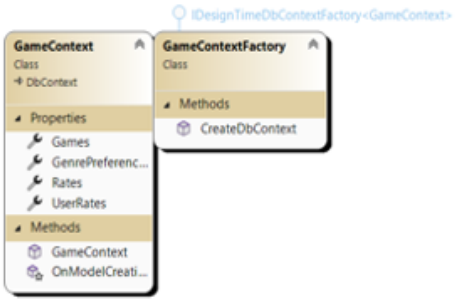
ПРОЕКТУВАННЯ МЕТОДУ ФОРМУВАННЯ ПЕРСОНАЛІЗОВАНИХ ВПОДОБАНЬ

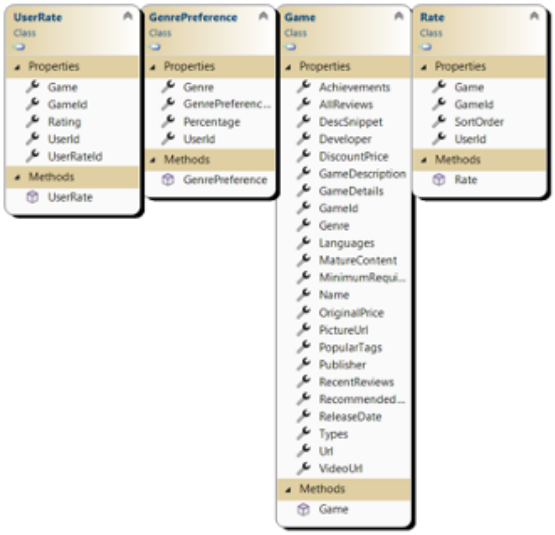
Проектування бібліотеки формування персоналізованих рекомендацій з використанням машинного навчання



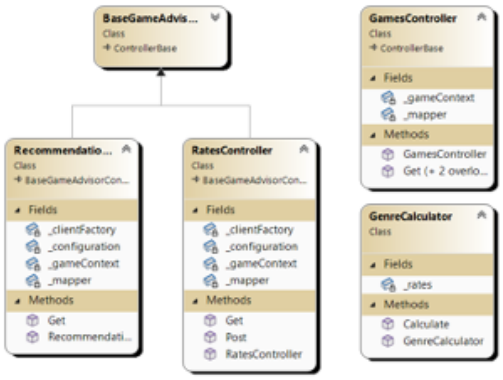


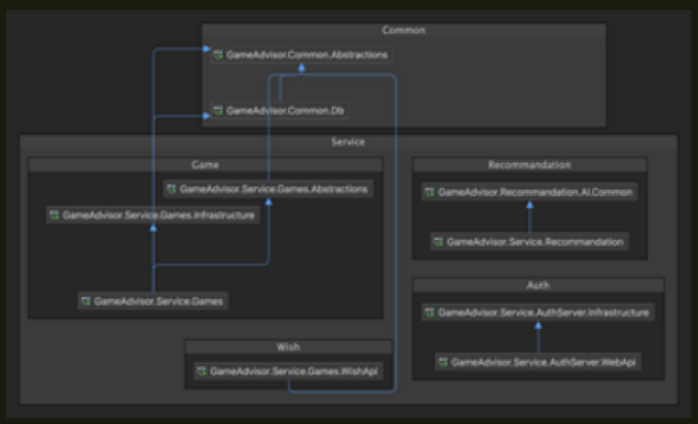
Проектування утиліт для машинного навчання





Проектування веб додатку з сервісом аутентифікації і база даних





ВІЗУАЛІЗАЦІЯ ЗАЛЕЖНОСТЕЙ

Вибір засобів

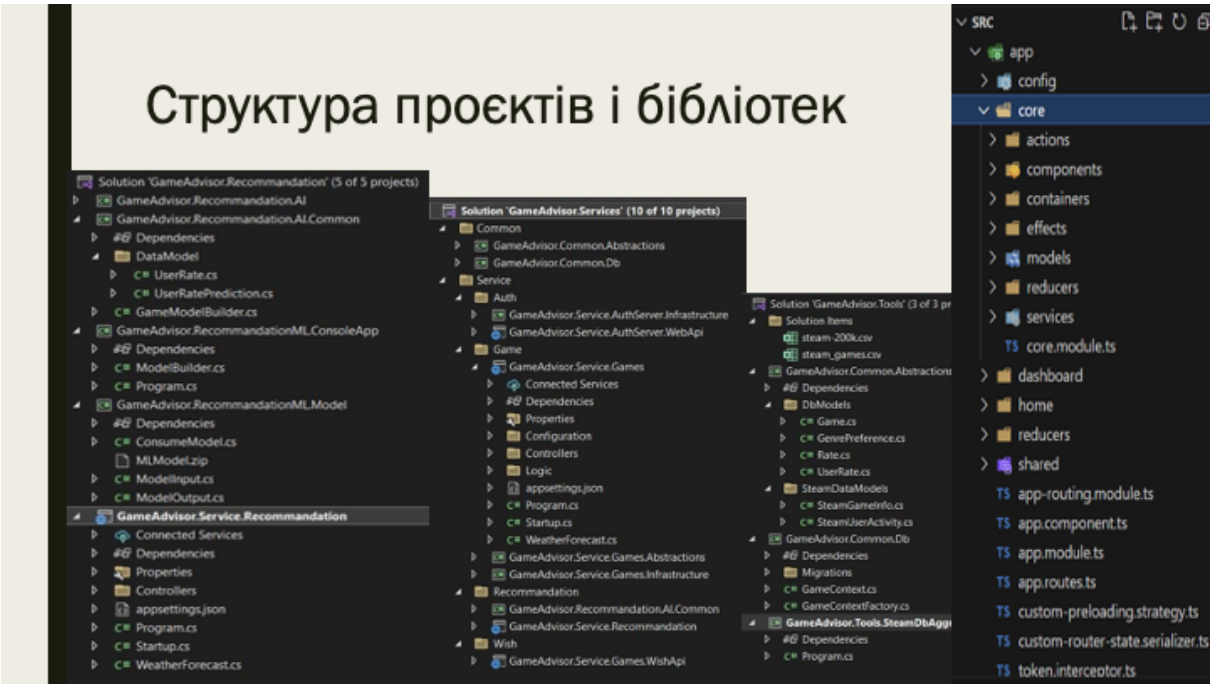
- Мови програмування C#, TypeScript
- База даних Postgres
- Платформи і фреймворки .NET, ASP.NET, ML.NET
- Засоби розробки користувацького інтерфейсу Angular, CSS, JS, Html, TypeScript
- IDE і редактори коду Visual Studio, Visual Studio Code

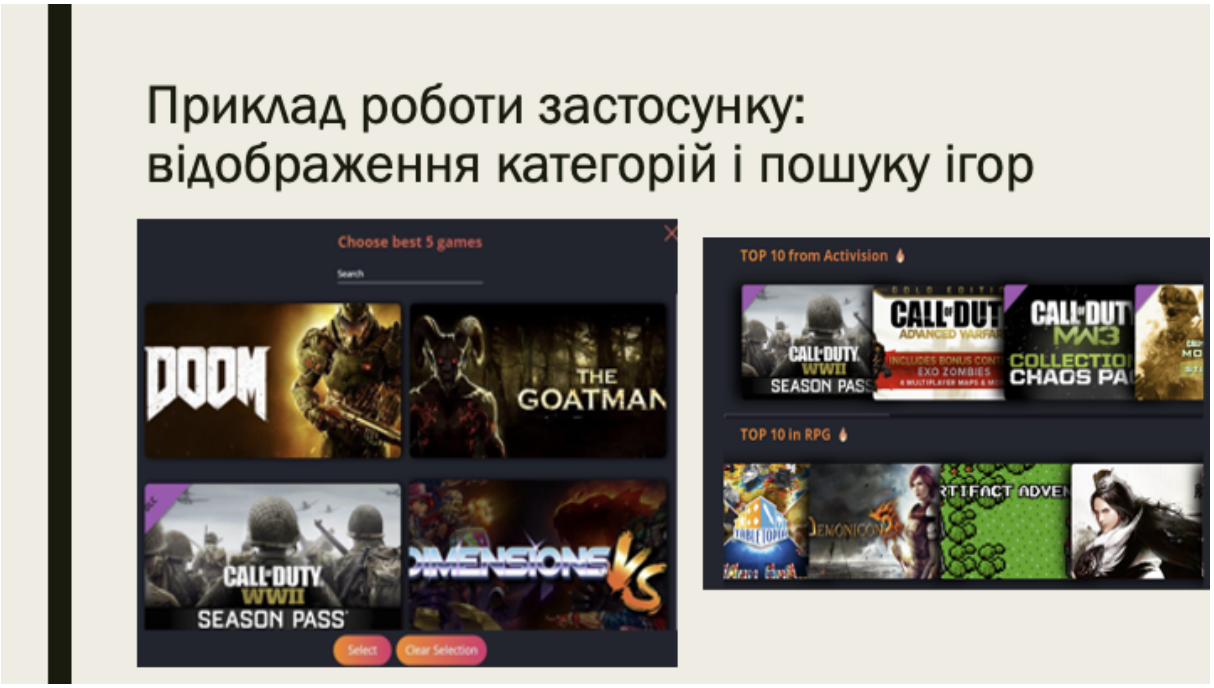


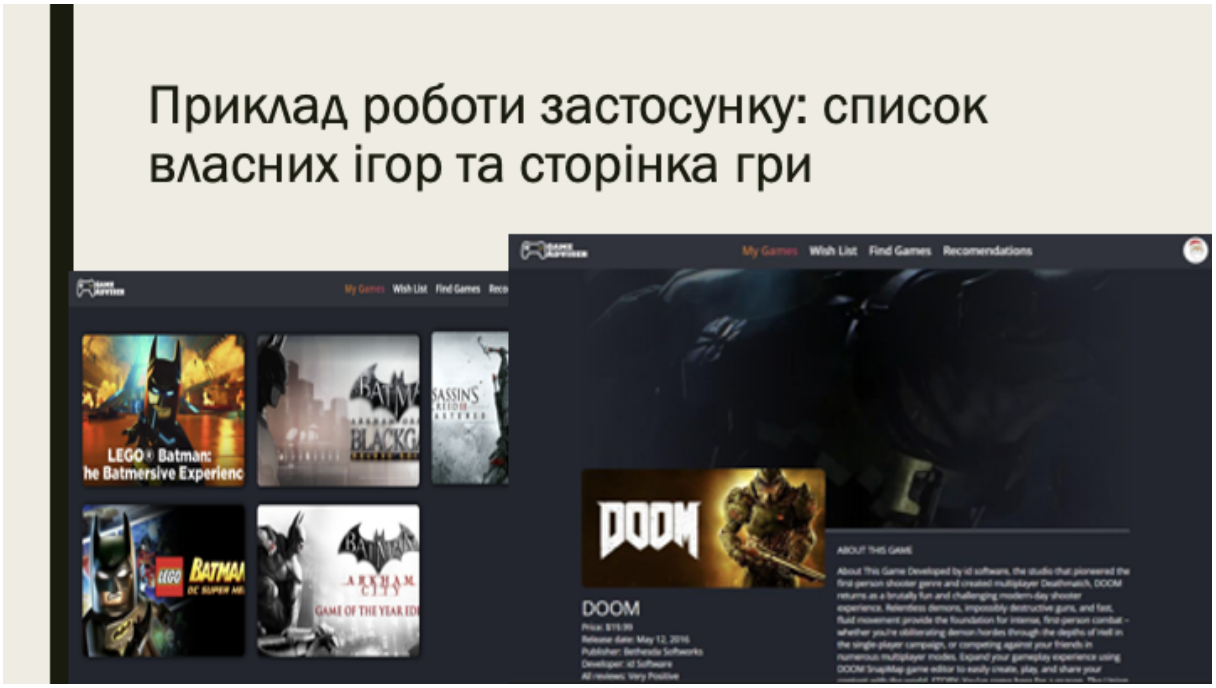

Дані для навчання

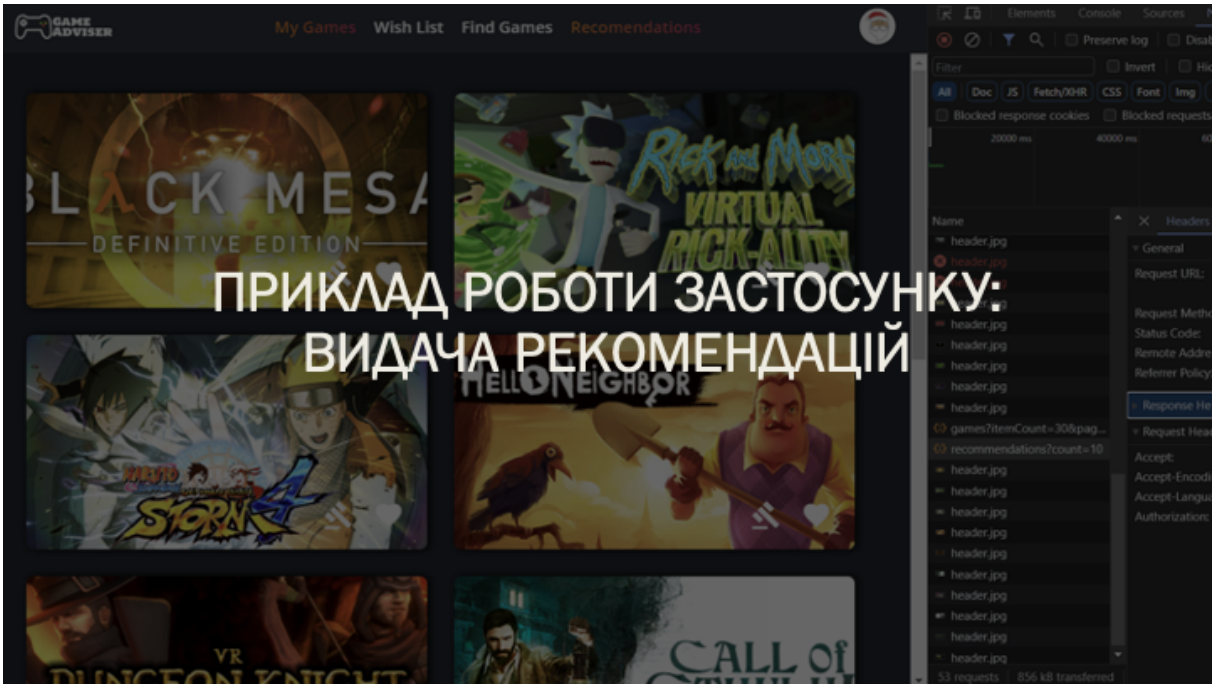
```
151603712,"The Elder Scrolls V Skyrim",purchase,1.0,0
151603712,"The Elder Scrolls V Skyrim",play,273.0,0
151603712,"Fallout 4",purchase,1.0,0
151603712,"Fallout 4",play,87.0,0
151603712,"Spore",purchase,1.0,0
151603712,"Spore",play,14.9,0
151603712,"Fallout New Vegas",purchase,1.0,0
151603712,"Fallout New Vegas",play,12.1,0
151603712,"Left 4 Dead 2",purchase,1.0,0
151603712,"Left 4 Dead 2",play,8.9,0
151603712,"HuniePop",purchase,1.0,0
151603712,"HuniePop",play,8.5,0
151603712,"Path of Exile",purchase,1.0,0
151603712,"Path of Exile",play,8.1,0
151603712,"Poly Bridge",purchase,1.0,0
151603712,"Poly Bridge",play,7.5,0
151603712,"Left 4 Dead",purchase,1.0,0
151603712,"Left 4 Dead",play,3.3,0
151603712,"Team Fortress 2",purchase,1.0,0
151603712,"Team Fortress 2",play,2.8,0
151603712,"Tomb Raider",purchase,1.0,0
151603712,"Tomb Raider",play,2.5,0
151603712,"The Banner Saga",purchase,1.0,0
151603712,"The Banner Saga",play,2.0,0
```

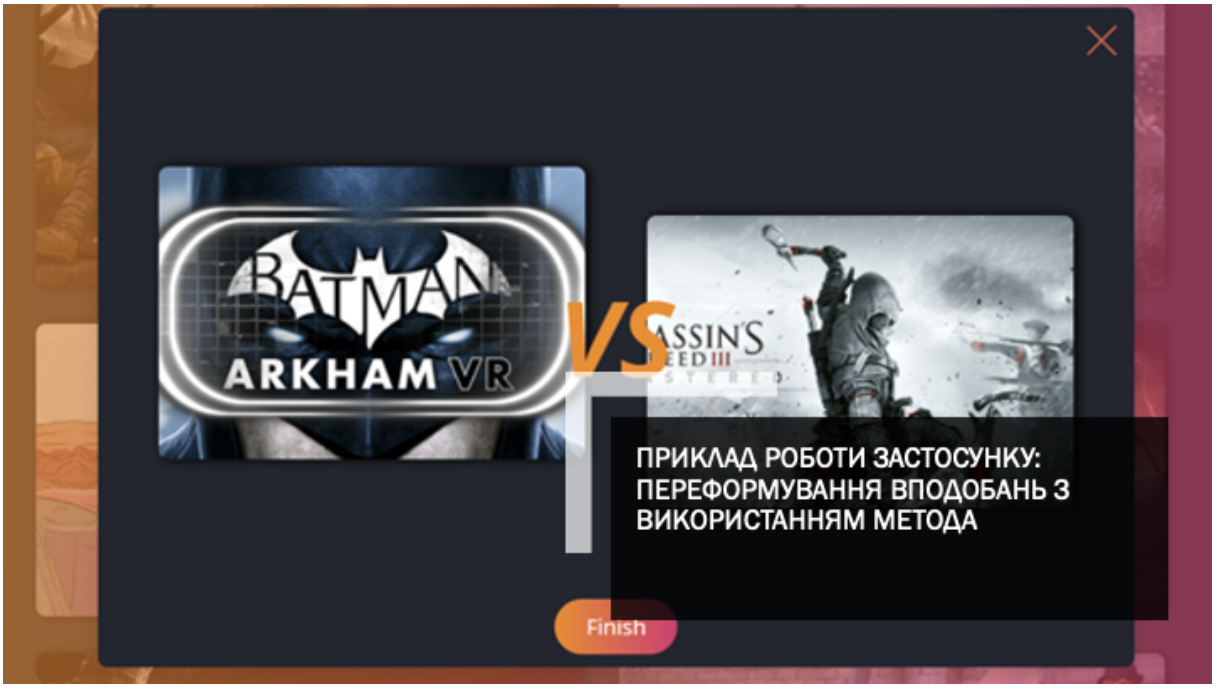
```
Url,Types,Name,DescSnippet,RecentReviews,AllReviews,ReleaseDate,Developer,Publisher,
https://store.steampowered.com/app/379720/DOOM/,app,DOOM,"Now includes all three pre
https://store.steampowered.com/app/578080/PLAYERUNKNOWNS_BATTLEGROUNDS/,app,PLAYERUN
https://store.steampowered.com/app/637090/BATTLETECH/,app,BATTLETECH,"Take command o
https://store.steampowered.com/app/221100/DayZ/,app,DayZ,"The post-soviet country of
https://store.steampowered.com/app/8500/EVE_Online/,app,EVE Online,"EVE Online is a
https://store.steampowered.com/bundle/5699/Grand_Theft_Auto_V_Premium_Online_Edition
https://store.steampowered.com/app/601150/Devil_May_Cry_5/,app,Devil May Cry 5,"The
https://store.steampowered.com/app/477160/Human_Fall_Flat/,app,Human: Fall Flat,Huma
https://store.steampowered.com/app/644930/They_Are_Billions/,app,They Are Billions,T
https://store.steampowered.com/app/774241/Warhammer_Chaosbane/,app,Warhammer: Chaosb
https://store.steampowered.com/app/527230/For_The_King/,app,For The King,"For The Ki
https://store.steampowered.com/app/567640/Danganronpa_V3_Killing_Harmony/,app,Dangan
https://store.steampowered.com/app/323370/TERA/,app,TERA,"From En Masse Entertainment, TERA is at the forefront of a new breed of MMO. With Tru
https://store.steampowered.com/app/393080/Call_of_Duty_Modern_Warfare_Remastered/,app,Call of Duty®: Modern Warfare® Remastered,"One of the mos
https://store.steampowered.com/app/253250/Stonehearth/,app,Stonehearth,"Pioneer a living world full of warmth, heroism, and mystery. Help a sma
https://store.steampowered.com/bundle/5641/Hearts_of_Iron_IV_Mobilization_Pack/,bundle,Hearts of Iron IV: Mobilization Pack,Hearts of Iron IV:
https://store.steampowered.com/app/597170/Clone_Drone_in_the_Danger_Zone/,app,Clone Drone in the Danger Zone,"Clone Drone in the Danger Zone is
https://store.steampowered.com/app/899440/GOD_EATER_3/,app,GOD EATER 3,"Set in a post-apocalyptic setting, it's up to your special team of God
https://store.steampowered.com/app/767560/War_Robots/,app,War Robots,"War Robots is an online third-person 6v6 PvP shooter-we're talking dozens
https://store.steampowered.com/app/459220/Halo_Wars_Definitive_Edition/,app,Halo Wars: Definitive Edition,"Halo Wars: Definitive Edition is an
```


Структура проєктів і бібліотек
Solution 'GameAdvisor.Recommandation' (5 of 5 projects)
GameAdvisor.Recommandation.AI
GameAdvisor.Recommandation.AI.Common
Dependencies
DataModel
UserRate.cs
UserRatePrediction.cs
GameModelBuilder.cs
GameAdvisor.RecommandationML.ConsoleApp
Dependencies
ModelBuilder.cs
Program.cs
GameAdvisor.RecommandationML.Model
Dependencies
ConsumeModel.cs
MLModel.zip
ModelInput.cs
ModelOutput.cs
GameAdvisor.Service.Recommandation
Connected Services
Dependencies
Properties
Controllers
appsettings.json
Program.cs
Startup.cs
WeatherForecast.cs
Solution 'GameAdvisor.Services' (10 of 10 projects)
Common
GameAdvisor.Common.Abstractions
GameAdvisor.Common.Db
Service
Auth
GameAdvisor.Service.AuthServer.Infrastructure
GameAdvisor.Service.AuthServer.WebApi
Game
GameAdvisor.Service.Games
Connected Services
Dependencies
Properties
Configuration
Controllers
Logic
appsettings.json
Program.cs
Startup.cs
WeatherForecast.cs
GameAdvisor.Service.Games.Abstractions
GameAdvisor.Service.Games.Infrastructure
Recommandation
GameAdvisor.Recommandation.AI.Common
GameAdvisor.Service.Recommandation
Wish
GameAdvisor.Service.Games.WishApi
Solution 'GameAdvisor.Tools' (3 of 3 pr
Solution Items
steam-200k.csv
steam_games.csv
GameAdvisor.Common.Abstraction
Dependencies
DbModels
Game.cs
GenrePreference.cs
Rate.cs
UserRate.cs
SteamDataModels
SteamGameInfo.cs
SteamUserActivity.cs
GameAdvisor.Common.Db
Dependencies
Migrations
GameContext.cs
GameContextFactory.cs
GameAdvisor.Tools.SteamDbAgg
Dependencies
Program.cs
SRC
app
config
core
actions
components
containers
effects
models
reducers
services
core.module.ts
dashboard
home
reducers
shared
app-routing.module.ts
app.component.ts
app.module.ts
app.routes.ts
custom-preloading.strategy.ts
custom-router-state.serializer.ts
token.interceotor.ts


Приклад роботи застосунку:
відображення категорій і пошуку ігор
Choose best 5 games
Search
DOOM
THE GOATMAN
CALL DUTY WWII
SEASON PASS
Select
Clear Selection
TOP 10 from Activision
CALL DUTY WWII
SEASON PASS
CALL DUT
ADVANCED WARFARE
CALL DUT MW3
COLLECTION
CHAOS PA
TOP 10 in RPG


Приклад роботи застосунку: список власних ігор та сторінка гри
My Games
Wish List
Find Games
Recomendations
DOOM


ПРИКЛАД РОБОТИ ЗАСТОСУНКУ:
ВИДАЧА РЕКОМЕНДАЦІЙ
My Games
Wish List
Find Games
Recomendations


BATMAN
ARKHAM VR
VS
ПРИКЛАД РОБОТИ ЗАСТОСУНКУ:
ПЕРЕФОРМУВАННЯ ВПОДОБАНЬ З
ВИКОРИСТАННЯМ МЕТОДА
Finish


Апробації та публікації
Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на міжнародній науково-практичній інтернет-конференції "Електронні інформаційні ресурси: створення, використання, доступ 2023" - Волинець О. Ю., Тужанський С. Є.
Персоналізовані рекомендації у цифрових бібліотеках

Висновки

- ✓ Розроблено метод персоналізованих рекомендацій на основі машинного навчання.
- ✓ Створено програмний засіб для рекомендації ігор з урахуванням персональних вподобань.
- ✓ Використано середовища розробки: Visual Studio 2022 для основного програмування та Visual Studio Code для користувацького інтерфейсу.
- ✓ Виконано роботу відповідно до методичних вказівок.

Важливі етапи та висновки:

- ✓ Визначена мета проєкту: розробка покращеного методу персоналізованих рекомендацій, що комбінує Матричну факторизацію і Колаборативний фільтринг.
- ✓ Реалізацію бібліотек, утиліт і веб-сервісів.
- ✓ Використання розробленого алгоритму для цих інструментів.
- ✓ Тестування засобу після його реалізації.

Важливі аспекти процесу:

- ❖ Вибір даних для навчання моделей.
- ❖ Звернення до якості та обсягу даних.

ДЯКУЮ ЗА УВАГУ