НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ

І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ПОГОДЖЕНО	**ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ**
Декан факультету	**Завідувач кафедри інформаційних систем і технологій**
_____ Ігор БОЛБОТ	**_____ Михайло ШВИДЕНКО**
(підпис)	(підпис)
«___» _____ 2026 р.	«___» _____ 2026 р.

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Інтелектуальна веб-система персоналізованих рекомендацій розважального контенту»

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Гарант освітньої програми д.е.н., професор	_____ (підпис)	**Роман РУДЕНСЬКИЙ**
Керівник бакалаврської кваліфікаційної роботи асистент	_____ (підпис)	**Ярослав ПОНЗЕЛЬ**
Консультант к.п.н., доцент	_____ (підпис)	**Тетяна ВОЛОШИНА**
Виконав	_____ (підпис)	**Дмитро САФОНЧИК**

Київ-2026

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

Факультет інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри інформаційних систем і технологій

к.е.н., доцент _____ Михайло ШВИДЕНКО
(підпис)

«___» _____ 2026 р.

ЗАВДАННЯ
ДО ВИКОНАННЯ БАКАЛАВРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
ЗДОБУВАЧУ
Сафончику Дмитру Олександровичу

Спеціальність «122 Комп'ютерні науки»

Освітньо-професійна програма «Комп'ютерні науки»

Тема бакалаврської кваліфікаційної роботи

«Інтелектуальна веб-система персоналізованих рекомендацій розважального контенту»
затверджена наказом від «06» січня 2026 р. № 46 «С»
Термін подання завершеної роботи на кафедру «22» травня 2026 р.
Вихідні дані до бакалаврської кваліфікаційної роботи (проєкту):
Інформація про розважальний контент; дані про користувацькі вподобання та історію взаємодії з контентом; використання алгоритмів рекомендацій; інтеграція із зовнішніми джерелами контенту через API
Перелік питань, що підлягають дослідженню:
Аналіз підходів до побудови рекомендаційних систем. Розробка алгоритмічного та програмного забезпечення для формування персоналізованих рекомендацій на основі аналізу вподобань і поведінкових даних користувачів.

Дата видачі завдання «06» січня 2026 р.

Керівник бакалаврської кваліфікаційної роботи _____ (підпис) **Ярослав ПОНЗЕЛЬ**

Консультант _____ (підпис) **Тетяна ВОЛОШИНА**

Завдання взяв до виконання _____ (підпис) **Дмитро САФОНЧИК**

Зміст

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – База даних

СУБД – Система управління базами даних

API (Application Programming Interface) – інтерфейс прикладного програмування

ASP.NET (Active Server Pages .NET) – платформа розробки вебзастосунків від Microsoft

CF (Collaborative Filtering) – колаборативна фільтрація

CLI (Command Line Interface) – інтерфейс командного рядка

CRUD (Create, Read, Update, Delete) – чотири базові функції управління даними (створення, читання, оновлення, видалення)

DOM (Document Object Model) – об'єктна модель документа

DTO (Data Transfer Object) – об'єкт передавання даних

ER (Entity-Relationship) – модель «сутність – зв'язок»

HTML (HyperText Markup Language) – мова гіпертекстової розмітки

HTTP (HyperText Transfer Protocol) – протокол передавання гіпертексту

IP (Internet Protocol) – міжмережевий протокол

ISBN (International Standard Book Number) – міжнародний стандартний номер книги

JSON (JavaScript Object Notation) – текстовий формат обміну даними

JWT (JSON Web Token) – відкритий стандарт для безпечної передачі інформації у вигляді JSON-об'єкта

LTS (Long Term Support) – версія програмного забезпечення з довгостроковою підтримкою

ORM (Object-Relational Mapping) – технологія об'єктно-реляційного відображення

RAWG – база даних та API відеоігор

REST (Representational State Transfer) – архітектурний стиль для розробки мережевих застосунків

RPG (Role-Playing Game) – комп'ютерна рольова гра

SDK (Software Development Kit) – набір засобів розробки програмного забезпечення

SPA (Single Page Application) – односторінковий вебзастосунок

SSD (Solid-State Drive) – твердотілий накопичувач

TF-IDF (Term Frequency – Inverse Document Frequency) – статистична міра для оцінки важливості слова в документі

TMDB (The Movie Database) – база даних та API фільмів і телесеріалів

UI (User Interface) – користувацький інтерфейс

UML (Unified Modeling Language) – уніфікована мова моделювання

URL (Uniform Resource Locator) – уніфікований покажчик ресурсу

UX (User Experience) – користувацький досвід

ВСТУП

Розвиток глобальної мережі Інтернет та вдосконалення технологій зберігання і передачі даних призвели до безпрецедентного зростання обсягів цифрового контенту у світі. Сучасний користувач, маючи підключення до мережі, отримує миттєвий доступ до мільйонів кінофільмів, телевізійних серіалів, різноманітних відеоігор та неосяжної кількості електронних книг. З одного боку, це є надзвичайним досягненням людства, що надає безмежні можливості для дозвілля, навчання та саморозвитку. З іншого боку, такий стан речей породив серйозну психологічну та технологічну проблему, відому як «інформаційне перевантаження» або «параліч вибору».

Параліч вибору виникає тоді, коли кількість доступних когнітивних варіантів настільки велика, що процес прийняття рішення стає стресовим, виснажливим і забирає невиправдано багато часу. У контексті індустрії розваг це призводить до того, що користувачі витрачають години на нескінченний пошук відповіді на запитання «що подивитися сьогодні ввечері» або «у що пограти на вихідних», часто так і не приймаючи остаточного рішення. Як наслідок, людина відчуває втому від прийняття рішень, що різко знижує рівень задоволеності від самого процесу споживання контенту.

Ефективним і технологічно виправданим вирішенням цієї проблеми стали рекомендаційні системи. Сьогодні рекомендаційні технології є критично важливим компонентом успіху найбільших світових технологічних корпорацій, таких як Netflix, Amazon, Spotify, YouTube та Steam. Проте, незважаючи на надзвичайно високий рівень розвитку таких інтелектуальних систем, більшість із них є вузькоспеціалізованими і функціонують у межах суворо закритих екосистем. Наприклад, алгоритми Steam блискуче аналізують ігровий досвід, алгоритми Netflix досконало розуміють кінематографічні уподобання, а Goodreads відстежує читацькі інтереси.

У реальному житті інтереси людини є багатогранними і часто перетинаються між різними медіа-доменами. Вузькоспеціалізовані платформи фізично не здатні вловити ці глибинні крос-доменні зв'язки, оскільки не володіють даними з інших сфер життя користувача, що суттєво обмежує їхню здатність пропонувати дійсно релевантний, цілісний та різноманітний контент. Саме тому створення універсальної крос-медійної рекомендаційної системи, яка б інтегрувала інформацію про фільми, серіали, ігри та книги на єдиній платформі, є надзвичайно актуальним, своєчасним та науково-практичним завданням.

Метою кваліфікаційної роботи є комплексне проєктування та повноцінна програмна реалізація масштабованої мікросервісної інформаційної системи, що виконує функції потужного агрегатора розважального контенту та забезпечує генерацію високоточних персоналізованих рекомендацій на основі передових гібридних алгоритмів фільтрації.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1. Провести глибинний аналіз предметної області, дослідити існуючі теоретичні підходи до побудови рекомендаційних систем та виявити ключові недоліки сучасних розважальних платформ.
2. Спроєктувати гнучку архітектуру програмного комплексу на базі мікросервісного підходу, визначити ефективні стратегії комунікації між сервісами.
3. Розробити оптимізовану структуру реляційної бази даних для зберігання різнорідних типів контенту з використанням архітектурного патерну Table-Per-Type.
4. Дослідити та математично обґрунтувати алгоритми контентної та колаборативної фільтрації, синтезувати на їх основі зважену гібридну модель.

5. Реалізувати відмовостійку серверну частину системи з інтеграцією зовнішніх джерел даних для наповнення каталогу.
6. Розробити оптимізований односторінковий клієнтський застосунок на фреймворку Angular з підтримкою механізмів розумної пагінації для роботи з великими масивами даних без втрати продуктивності браузера.
7. Реалізувати високопродуктивне алгоритмічне ядро рекомендацій мовою Python та безшовно інтегрувати його в загальну екосистему.
8. Контейнеризувати усі компоненти системи за допомогою технології Docker та провести комплексне тестування розробленого програмного забезпечення.

Об’єкт дослідження – процес формування персоналізованих крос-доменних рекомендацій розважального контенту на основі аналізу користувацьких уподобань та історії взаємодії.

Предмет дослідження – методи, алгоритми, математичні моделі та програмні засоби побудови крос-медійної рекомендаційної інформаційної системи на базі мікросервісної архітектури.

Для розв’язання поставлених інженерних та наукових завдань у роботі було використано комплекс методів:

- системний аналіз предметної області, існуючих платформ та аналогів для формування вимог до ПЗ;
- методи математичного моделювання та машинного навчання для побудови високопродуктивного рекомендаційного ядра;
- об’єктно-орієнтований аналіз та уніфікована мова моделювання для проєктування логічної структури й динамічної поведінки системи;
- архітектурні патерни Table-Per-Type для поліморфного проєктування реляційної бази даних та Backend-For-Frontend з API Gateway для побудови мікросервісів;

- методи контейнеризації та оркестрації для забезпечення відмовостійкого розгортання.

Основні теоретичні положення, архітектурні підходи та практичні результати розробки інформаційної системи були висвітлені у тезах доповіді на тему «Проєктування та розробка рекомендаційної системи розважального контенту на базі мікросервісної архітектури» на Науково-практичній конференції студентів і аспірантів «Теоретичні та прикладні аспекти розробки комп’ютерних систем».

Кваліфікаційна робота складається з переліку умовних позначень, вступу, чотирьох розділів, висновків та списку використаних джерел. Повний обсяг пояснювальної записки становить 66 сторінок. Текст роботи містить 17 рисунків та 2 таблиці. Список використаних джерел налічує 19 найменувань.

Короткий зміст розділів роботи:

- У першому розділі обґрунтовано актуальність теми крос-доменних рекомендацій, проведено глибокий аналіз методів побудови рекомендаційних систем, здійснено порівняльний аналіз існуючих світових аналогів та сформульовано чіткі функціональні й нефункціональні вимоги до системи.
- У другому розділі представлено інженерне обґрунтування мікросервісної архітектури системи з використанням патерну API Gateway, спроєктовано схему бази даних PostgreSQL за патерном Table-Per-Type, виконано UML-моделювання поведінки системи, а також математично описано алгоритмічне ядро рекомендацій.
- У третьому розділі описано безпосередню програмну реалізацію серверної частини на платформі ASP.NET Core, архітектуру фонової агрегації та нормалізації даних через зовнішні API, розробку Single Page Application клієнта на Angular з імплементацією «розумної

пагінації», створення рекомендаційного модуля на Python та контейнеризацію всієї екосистеми через Docker.

- У четвертому розділі сформульовано комплексну стратегію тестування ПЗ, наведено специфікацію та результати функціонального й навантажувального тестування клієнтської і серверної частин, проведено оцінку релевантності гібридного алгоритму на базі штучно змодельованих профілів, а також визначено перспективи горизонтального масштабування та модернізації інфраструктури платформи.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Актуальність теми та проблематика вибору розважального контенту

Індустрія розваг зазнала колосальних трансформацій за останні два десятиліття. Перехід від фізичних носіїв інформації до цифрової дистрибуції повністю змінив парадигму споживання контенту. Стрімкий розвиток інфраструктури Інтернету та зменшення вартості зберігання даних призвели до того, що користувачі отримали доступ до практично необмежених обсягів інформації у будь-якій точці світу. Згідно зі звітами міжнародних аналітичних агенцій, щодня у світі випускаються десятки великобюджетних та інді-відеоігор, сотні годин професійного та аматорського відеоконтенту, а також тисячі нових літературних творів. Замість того, щоб відчувати абсолютну свободу вибору та задоволення, сучасні користувачі дедалі частіше стикаються з потужним психологічним феноменом, відомим у наукових колах як «information overload» (інформаційне перевантаження).

Інформаційне перевантаження критично ускладнює процес прийняття рішень людським мозком. Коли перед людиною стоїть вибір із трьох-п'яти фільмів, вона може легко проаналізувати їхні жанри, акторський склад, тривалість та зробити свідомий вибір. Коли ж варіантів стає тисячі, процес аналізу стає надто ресурсомістким для когнітивної системи людини. Цей феномен призводить до втоми від прийняття рішень. Користувач відчуває тривогу через побоювання зробити «неправильний» вибір та змарнувати свій вільний час на неякісний продукт. Як наслідок, замість активного відпочинку, людина часто відмовляється від перегляду фільму чи проходження гри на

користь пасивного і менш ресурсомісткого гортання стрічки новин у соціальних мережах.

У цьому складному контексті рекомендаційні системи виступають у ролі життєво необхідних інтелектуальних фільтрів. Вони виконують важку попередню алгоритмічну обробку всього доступного масиву даних, відсіюють нерелевантну або низькоякісну інформацію та пропонують користувачеві стислий, чітко структурований список об'єктів, які з найвищою ймовірністю задовольнять його унікальні потреби в конкретний момент часу. Таким чином, рекомендаційні алгоритми еволюціонували від статусу простої «зручної фічі» до критично необхідної технологічної бази для функціонування будь-якої сучасної контент-орієнтованої платформи.

Окремою та дуже важливою проблемою сучасної індустрії розваг є жорстка доменна ізоляція компаній-гігантів. Кожен користувач щодня залишає свій унікальний «цифровий слід» на багатьох різних сервісах: він зберігає та оцінює книги на Goodreads, публікує розгорнуті відгуки на відеоігри у Steam, переглядає серіали на Netflix та слухає саундтреки на Spotify. Проблема полягає в тому, що ці розрізнені платформи жодним чином не обмінюються даними між собою. Сервіс Netflix алгоритмічно не знає, що користувач щойно провів сотню годин у грі «The Witcher 3: Wild Hunt» і, ймовірно, був би вкрай зацікавлений у першочерговому перегляді однойменного фентезійного серіалу або прочитанні циклу книг Анджея Сапковського. Створення агрегованої бази даних, яка об'єднує різні типи медіа в одному місці, дозволяє технологічно подолати цей бар'єр і будувати рекомендації на основі цілісного профілю інтересів особистості, а не лише її фрагментарної активності в одному вузькому домені.

1.2 Аналіз методів побудови рекомендаційних систем

Світові наукові дослідження у сфері Data Science та машинного навчання виділяють три фундаментальні підходи до побудови алгоритмів для

рекомендаційних систем: контентна фільтрація, колаборативна фільтрація та різноманітні гібридні методи. Розуміння переваг та недоліків кожного з них є ключовим для правильного проєктування системи.

Контентна фільтрація (Content-based Filtering). Цей класичний метод повністю базується на глибокому аналізі властивостей самих об'єктів контенту та зіставленні їх з наявним профілем користувача. Кожен об'єкт (наприклад, конкретна гра або повнометражний фільм) математично представляється у вигляді набору характеристик. Для розважального контенту такими ознаками є жанри (RPG, Action, Drama), користувацькі теги (Singleplayer, Dark Fantasy, Cyberpunk), рік випуску, режисер, студія-розробник тощо. Профіль користувача формується алгоритмом динамічно шляхом агрегації характеристик тих об'єктів, які користувачеві сподобалися раніше (наприклад, отримали оцінку 4 або 5 зірок). Процес самої рекомендації зводиться до математичного пошуку об'єктів у базі даних, які є максимально схожими на профіль користувача. Головною перевагою цього підходу є абсолютна відсутність проблеми «холодного старту» для нових об'єктів: як тільки новий маловідомий фільм додається до бази даних і отримує набір тегів, система одразу, тієї ж секунди, може рекомендувати його відповідним користувачам. Серйозним недоліком є тенденція алгоритму до надмірної спеціалізації – система замикається у так званій «інформаційній бульбашці» і рідко пропонує користувачеві щось принципово нове, що він раніше не споживав.

Колаборативна фільтрація (Collaborative Filtering). Цей потужний метод кардинально відрізняється від контентного, оскільки він повністю ігнорує внутрішні властивості самого контенту (системі абсолютно байдуже, що це за фільм, який у нього жанр чи хто режисер). Алгоритм спирається виключно на глобальну матрицю взаємодій «Користувач-Контент» (користувацькі оцінки, лайки, історію переглядів). Базова математична гіпотеза звучить так: якщо два різні користувачі мали схожі смаки у минулому, вони з великою ймовірністю

матимуть схожі смаки й у майбутньому. Колаборативна фільтрація традиційно поділяється на два основні підтипи:

1) User-based CF (на основі користувачів): алгоритм знаходить групу користувачів зі схожою історією оцінок (так званих «найближчих сусідів») і рекомендує активному користувачеві те, що найбільше сподобалося цій групі.

2) Item-based CF (на основі об'єктів): алгоритм аналізує схожість між самими об'єктами виключно на основі того, як ці об'єкти спільно оцінювалися одними й тими самими людьми у базі даних (наприклад, якщо користувачі, які грали у гру А, майже завжди грають у гру Б з високою оцінкою, то алгоритм вважає ігри А і Б дуже схожими). Безперечною перевагою методів колабаративної фільтрації є їхня здатність робити дуже несподівані, інтуїтивні, але напрочуд точні крос-жанрові рекомендації, викликаючи у користувача ефект приємної несподіванки. Головні недоліки методу – яскраво виражена проблема «холодного старту» (новий об'єкт не буде нікому рекомендований, поки не набере критичну масу оцінок від перших користувачів) та проблема розрідженості даних, яка виникає, коли матриця оцінок містить занадто багато порожніх значень, оскільки кожен користувач оцінює мізерну долю від мільйонного каталогу бази даних.

Гібридні системи (Hybrid Recommender Systems). Оскільки жоден із перелічених базових методів машинного навчання не є ідеальним і містить критичні архітектурні недоліки, на практиці у промислових системах найчастіше застосовуються різноманітні гібридні підходи. Вони покликані скомбінувати сильні сторони контентної та колаборативної фільтрації, одночасно нівелюючи їхні слабкості. Існує багато способів гібридизації алгоритмів: зважена гібридизація, де результати кількох алгоритмів складаються у відповідних пропорціях; змішана гібридизація, де результати видаються одночасно в різних блоках UI; каскадна гібридизація, де один алгоритм фільтрує базу, а другий сортує залишки. У межах даної кваліфікаційної роботи найбільш раціональним та оптимальним є використання зваженої гібридної моделі, яка дозволяє системі

динамічно налаштовувати математичний вплив кожного алгоритму на фінальний результат залежно від обсягу наявних даних про конкретного користувача.

1.3 Огляд та порівняльний аналіз існуючих платформ

Для формування чітких функціональних вимог до власної системи було проведено аналіз інтерфейсів та рекомендаційних алгоритмів провідних світових платформ, що спеціалізуються на розважальному контенті. До аналізу було залучено платформи Netflix (фільми та серіали), Steam (відеоігри) та Goodreads (література).

Netflix є одним із світових лідерів у застосуванні алгоритмів машинного навчання для утримання уваги користувачів. Рекомендаційна система Netflix базується на складних гібридних алгоритмах. Інтерфейс системи побудований у вигляді горизонтальних списків з категоріями, такими як «Тому що ви дивилися», «Топ-10 у вашій країні», «Схоже на...». Важливою особливістю є розрахунок відсотка збігу, який показує ймовірність того, що контент сподобається конкретному користувачеві. Однак платформа обмежена виключно відеоконтентом.

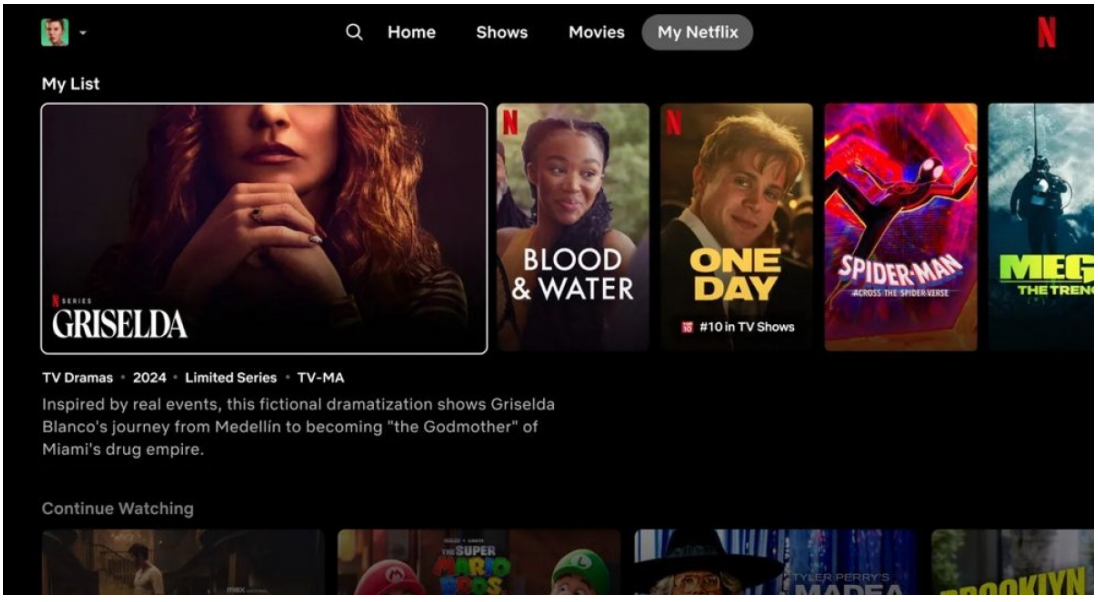

Рис. 1.1. Інтерфейс блоку персоналізованих рекомендацій Netflix

Платформа цифрової дистрибуції ігор Steam від Valve використовує потужну систему тегів, що генеруються самими користувачами. Контентна фільтрація тут спирається саме на ці теги. У магазині Steam реалізовано механізм «Черга рекомендацій» та блок «Схожі товари», які враховують не лише жанри, але й ігри, які є в бібліотеці користувача, час, проведений у них, та ігри, які користувач додав до списку бажаного.

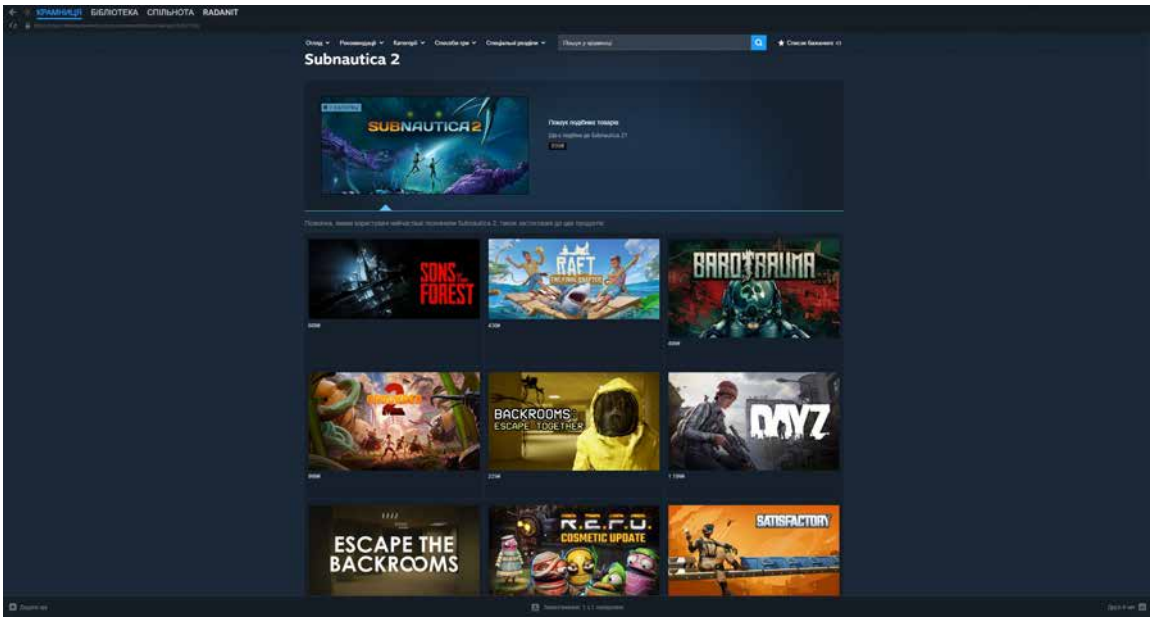

Рис. 1.2. Відображення схожого контенту на базі системи тегів у Steam

Goodreads є найбільшою платформою для шанувальників книг. Її рекомендаційна система сильно спирається на соціальну взаємодію та колаборативну фільтрацію. Користувачі створюють «полиці» та виставляють оцінки за 5-бальною шкалою. Система аналізує ці полки і пропонує розділ «Рекомендації на основі ваших оцінок». Проте інтерфейс платформи виглядає застарілим і часто перевантажений текстовою інформацією.

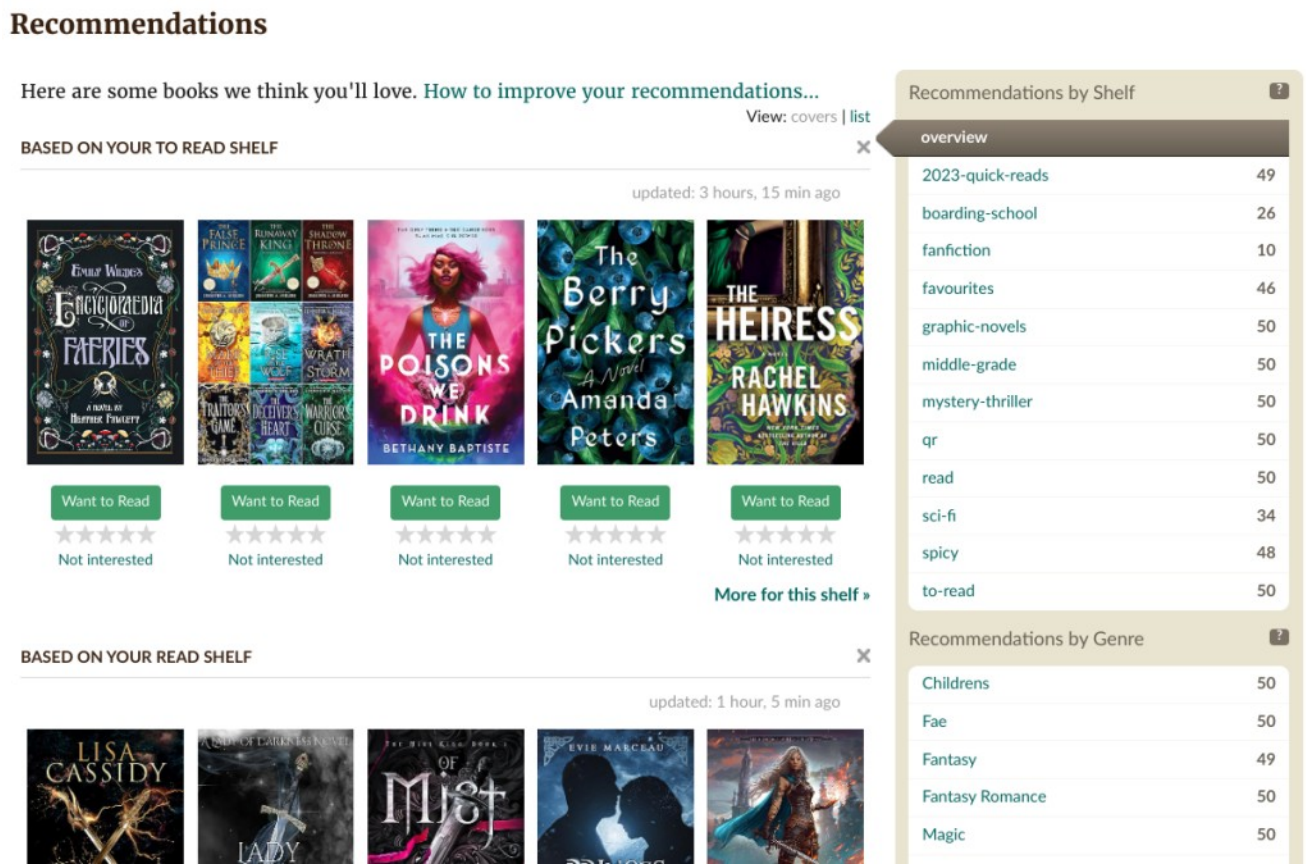

Рис. 1.3. Блок рекомендацій книг на платформі Goodreads

Кожна з проаналізованих платформ має потужне рекомендаційне ядро, проте всі вони є вузькоспеціалізованими: Netflix орієнтований виключно на відеоконтент, Steam — на комп'ютерні ігри, а Goodreads — на літературу. Для користувача, який споживає різні види розважального контенту, виникає потреба в одночасному використанні кількох окремих застосунків. На основі проведеного аналізу аналогів переваги розроблюваної системи наочно продемонстровано у табл. 1.1.

Таблиця 1.1

Функціональні можливості розроблюваної системи та аналогів

Функціональна можливість	**Netflix**	**Steam**	**Goodreads**	**Розроблювана система**
Крос-доменність	Ні	Ні	Ні	Так
Колаборативна фільтрація	Ні	Так	Так	Так
Контентна фільтрація (за описом/тегами)	Так	Так	Так	Так

1.4 Формулювання мети, завдань та функціональних вимог до системи

На основі ґрунтовного аналізу недоліків конкурентів сформульовано чіткі інженерні вимоги до інформаційної системи, що розробляється. Система повинна бути сучасним веб-орієнтованим застосунком з розподіленою клієнт-серверною архітектурою.

Ключові функціональні вимоги:

1. Підсистема імпорту: наявність модулів для автоматизованого пакетного парсингу великих обсягів даних із зовнішніх публічних API (зокрема RAWG для ігор, TMDB для фільмів та серіалів, Google Books для літератури) з наступним збереженням нормалізованих даних у єдиній реляційній БД.

2. Підсистема адміністрування: наявність зручної, чуйної адміністративної панелі для безпечного управління каталогом контенту. Панель повинна підтримувати всі базові CRUD-операції (створення, читання, оновлення, видалення) і працювати без візуальних затримок при рендерингу тисяч записів.

3. Підсистема користувача: забезпечення повноцінного механізму реєстрації, авторизації, створення власних профілів, де користувач може зберігати історію взаємодій та виставляти рейтинги контенту за 5-бальною шкалою.

4. Рекомендаційна підсистема: автоматична генерація персоналізованих рекомендаційних списків за допомогою математично обґрунтованих методів контентної та колаборативної фільтрації, з можливістю їх гібридного об'єднання.

Ключові нефункціональні вимоги:

1. Масштабованість: архітектура програмного коду повинна бути модульною і підтримувати безболісне додавання нових типів медіа (наприклад,

музики чи подкастів) у майбутньому без кардинального переписування базового ядра системи.

2. Продуктивність: складні математичні обчислення матриць не повинні блокувати роботу веб-сервера. Час генерації гібридних рекомендацій не повинен перевищувати прийнятних для UX меж (максимум 2-4 секунди). Відмальовка великих таблиць даних в адміністративній панелі фронтенду має відбуватися миттєво.

3. Відмовостійкість: використання мікросервісів має гарантувати, що вихід з ладу одного другорядного компонента (наприклад, сервісу фонового імпорту чи відключення зовнішнього API) жодним чином не призведе до фатальної непрацездатності всієї розважальної платформи.

2 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1 Обґрунтування вибору мікросервісної архітектури

Проєктування базової архітектури є критично важливим, фундаментальним етапом розробки програмного забезпечення, оскільки саме воно визначає майбутню здатність системи до масштабування під навантаженням, легкість технічної підтримки та можливість впровадження нових фіч. Традиційним, історично сформованим підходом до створення веб-додатків є монолітна архітектура. У моноліті весь програмний код (інтерфейс користувача, бізнес-логіка домену, шар доступу до бази даних, фонові задачі) жорстко скомпільований у єдиний виконуваний файл або розгорнутий на одному сервері. Монолітний підхід дозволяє дуже швидко розпочати розробку та легко дебажити код на початкових етапах. Проте, з часом та ростом бази коду моноліт неминуче перетворюється на «велику грудку бруду» – антипатерн, де будь-яка незначна зміна в одному модулі може непередбачувано зламати зовсім інший модуль, а масштабування вимагає дублювання всієї величезної програми, навіть якщо перевантаженим є лише один дрібний компонент.

Враховуючи вимоги до відмовостійкості та гнучкості, для даного проєкту було стратегічно обрано мікросервісну архітектуру. Цей сучасний архітектурний стиль передбачає декомпозицію великої системи на набір невеликих, слабо пов'язаних сервісів. Кожен такий сервіс відповідає виключно за певну вузьку бізнес-функцію, працює у власному ізольованому процесі операційної системи та спілкується з іншими мікросервісами за допомогою стандартизованих, легковагових протоколів.

У рамках розробленої системи виділено наступні ключові мікросервіси:

ContentService (розроблений мовою C# на базі ASP.NET Core) – головне ядро управління контентом платформи. Він відповідає за збереження, структурування та видачу метаданих про всі ігри, фільми, серіали та книги.

Окрім забезпечення повного циклу CRUD-операцій для фронтенду, цей сервіс також бере на себе інтеграційну функцію – комунікацію із зовнішніми постачальниками даних (RAWG API, TMDB API, Google Books API) для автоматизованого імпорту та оновлення медіа-каталогу. Сервіс безпосередньо взаємодіє з власною базою даних PostgreSQL.

UserService (розроблений мовою C# на базі ASP.NET Core) – ізольований мікросервіс, який відповідає за управління обліковими записами та персональними даними. Він зберігає інформацію профілів користувачів, їхні демографічні дані та базові вподобання (задані при реєстрації пріоритетні жанри, теги чи мови). Сервіс оперує базою даних UserDb та формує статичну складову профілю інтересів особистості.

InteractionService (розроблений мовою C# на базі ASP.NET Core) – спеціалізований сервіс, що фокусується виключно на зборі та обробці поведінкових даних. Він фіксує та логує всю історію взаємодій користувачів із платформою: виставлені оцінки контенту, збереження об'єктів до списку «Вибране» та історію отриманих рекомендацій. Цей сервіс накопичує дані у базі InteractionDb, формуючи критично важливий масив інформації, який є фундаментом для роботи алгоритмів колаборативної фільтрації.

RecommendationService (розроблений мовою Python) – спеціалізоване математичне ядро системи. Оскільки екосистема мови Python має об'єктивно найкращий у світі та найшвидший інструментарій (зокрема, бібліотеки Pandas та Scikit-learn) для роботи з векторами даних та побудови алгоритмів машинного навчання, цей критичний модуль виділено в окремий сервіс. Він отримує агреговані дані про контент, користувачів та їхні взаємодії з інших баз, проводить складні матричні обчислення в оперативній пам'яті (гібридна фільтрація) і повертає через API Gateway готові відсортовані JSON-відповіді з персоналізованими рекомендаціями.

Використання мікросервісів, попри всі свої переваги, породжує відому проблему комунікації між клієнтським додатком та десятком розрізнених

серверів. Якщо клієнтський додаток буде напряму звертатися до кожного мікросервісу, це неминуче призведе до надзвичайно жорсткої зв'язності архітектури. Крім того, це спричинить явище «over-fetching» (коли клієнт змушений робити безліч дрібних запитів, щоб зібрати одну сторінку) та постійні проблеми з політикою браузерів.

Для елегантного вирішення цієї фундаментальної проблеми було спроєктовано та імплементовано архітектурний патерн Backend-For-Frontend, реалізований за допомогою спеціального вузла – API Gateway.

API Gateway виступає як єдина публічна точка входу для всіх без винятку клієнтських запитів. Фронтенд-додаток звертається виключно до одного шлюзу за однією адресою. Шлюз, у свою чергу, містить таблицю маршрутизації і сам знає, на який саме внутрішній прихований мікросервіс у мережі Docker слід перенаправити запит. Більше того, API Gateway виконує не лише маршрутизацію, але й складну агрегацію даних: він може прийняти один запит від клієнта, зробити три паралельні внутрішні запити (наприклад, отримати інформацію про гру з ContentService, її рейтинг від користувача з InteractionService та позицію в топі з RecommendationService), склеїти їх і повернути клієнту один підсумковий, ідеально відформатований JSON. Важливо зазначити, що шлюз також забезпечує стандартизацію відповідей, вирішуючи поширену проблему розбіжності регістрів (наприклад, коли мікросервіс на C# серіалізує ключі у форматі PascalCase «ContentId», «Title», а фронтенд на Angular суворо очікує формат camelCase «contentId», «title»).

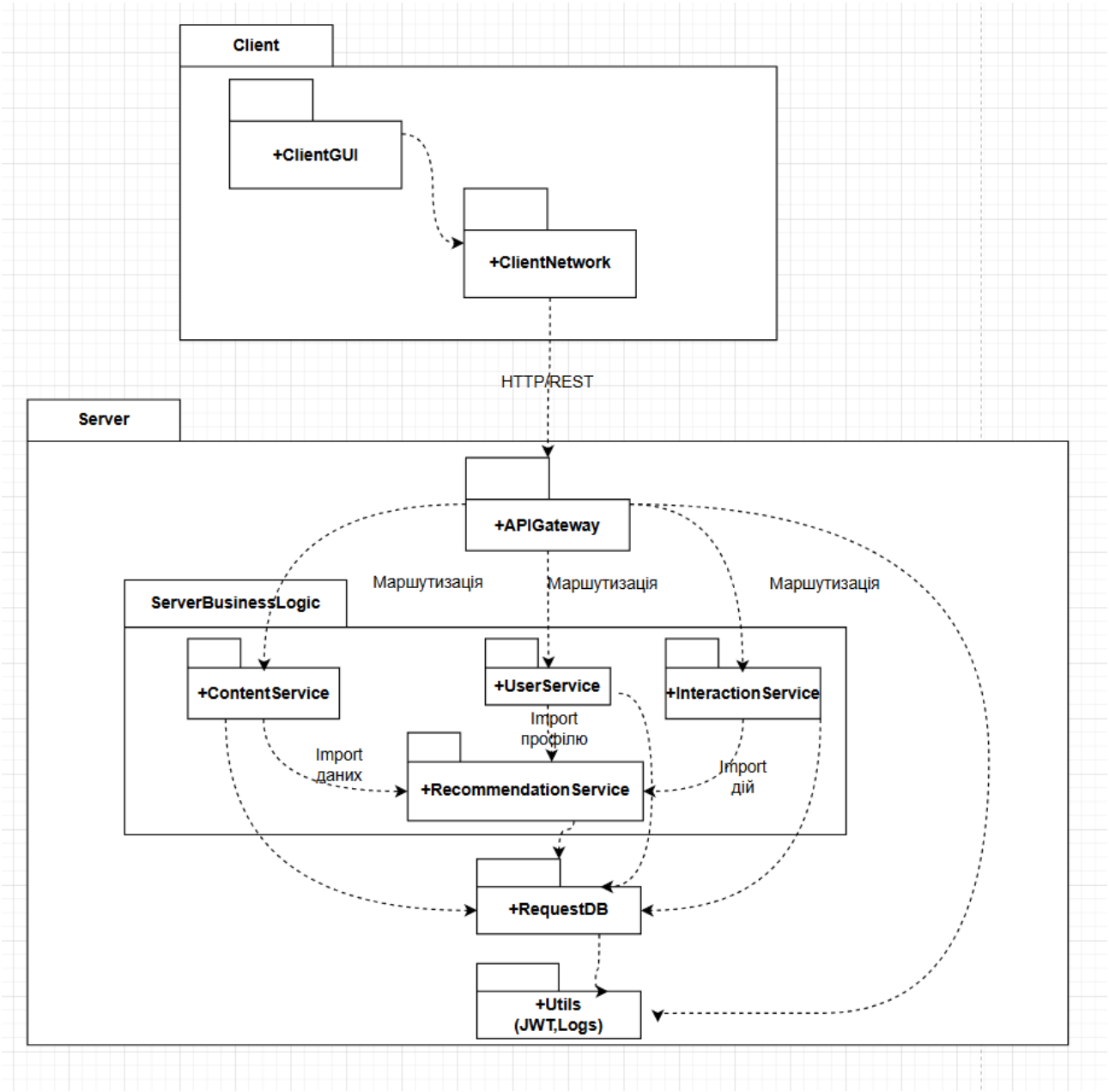

Рис. 2.1 Архітектура мікросервісної системи з використанням патерну API Gateway

2.2 Проєктування бази даних

Специфіка розважального контенту вимагає надзвичайно гнучкого та продуманого підходу до проєктування схеми бази даних. Ігри, фільми, серіали та книги мають величезну кількість спільних базових характеристик: у всіх них є назва, текстовий опис, середня оцінка, дата релізу, та URL-посилання на постер обкладинки. Але водночас кожен тип медіа має абсолютно унікальні атрибути,

притаманні лише йому. Наприклад, відеогра обов'язково має «Розробника». Фільм характеризується наявністю «Режисера», та касових зборів. Книга ж має «Автора», унікальний міжнародний ідентифікатор «ISBN» та кількість сторінок.

У реляційних базах даних для успішної реалізації поліморфізму та наслідування об'єктно-орієнтованих сутностей використовують три основні патерни: Table-Per-Hierarchy (TPH – одна величезна таблиця для всього), Table-Per-Type (TPT – таблиця для базового класу і таблиці для нащадків) та Table-Per-Concrete-Class (TPC – окрема незалежна таблиця для кожного фінального класу).

Для вирішення завдань даного проєкту було науково обґрунтовано та обрано патерн Table-Per-Type, реалізований за допомогою передового ORM-фреймворку Entity Framework Core від Microsoft. Згідно з логікою цього патерну, у базі даних створюється базова таблиця «Contents», яка фізично містить колонки для всіх спільних атрибутів та головний первинний ключ «ContentID». Далі створюються окремі дочірні таблиці: «Games», «Films», «Series», «Books». Кожна з цих дочірніх таблиць має свій первинний ключ «ContentID», який водночас є суворим зовнішнім ключем, що посилається на базову таблицю «Contents». Крім того, дочірні таблиці містять колонки виключно для своїх специфічних атрибутів. Цей архітектурний підхід забезпечує ідеальну нормалізацію даних до третьої нормальної форми, повну відсутність великої кількості порожніх колонок та значно полегшує безболісне додавання нових типів медіа (наприклад, музичних альбомів) у майбутньому без зміни структури існуючих таблиць.

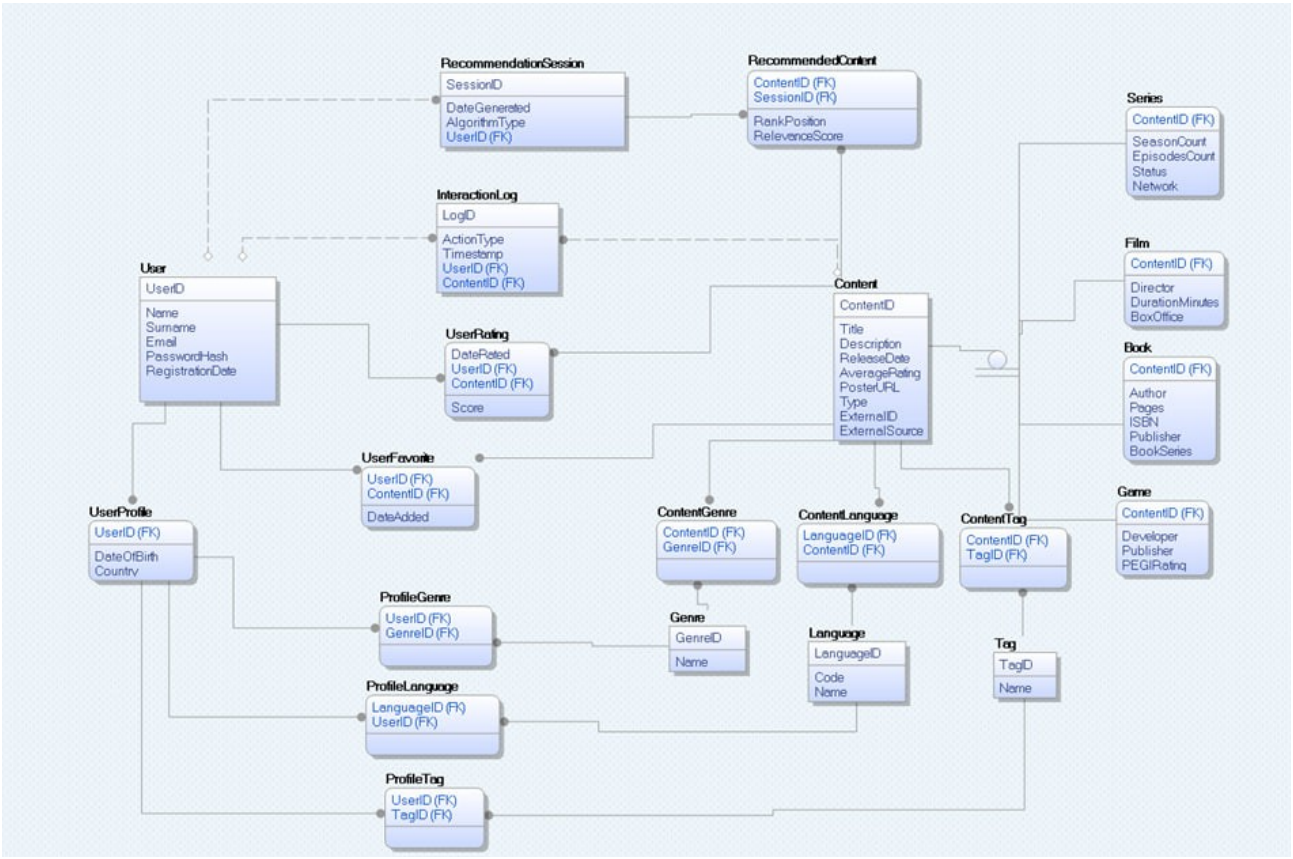

Рис. 2.2 ER-діаграма реляційної бази даних системи

У розробленій системі як основне і надійне джерело зберігання даних використовується потужна об'єктно-реляційна СУБД PostgreSQL. Вона ефективно справляється з обробкою інформації різної природи в межах одного проєкту. Для швидкого доступу до даних, які надзвичайно часто запитуються клієнтами, але майже ніколи не змінюються (наприклад, статичні словники жанрів для UI-фільтрів, списки доступних тегів), спроєктовано оптимізовану структуру таблиць із застосуванням відповідних баз даних індексів. Завдяки грамотній нормалізації, ефективному налаштуванню реляційних зв'язків та внутрішнім механізмам кешування і оптимізації самої PostgreSQL, час виконання як простих вибірок, так і складних багатоступеневих JOIN-запитів зведено до мінімуму. Це суттєво підвищує загальну продуктивність та гарантує високу пропускну здатність платформи навіть при пікових навантаженнях.

2.3 Моделювання поведінки системи за допомогою UML-діаграм

Для забезпечення чіткого розуміння архітектури, поведінки та логіки роботи розроблюваної системи на етапі проєктування було використано об’єктно-орієнтований підхід та уніфіковану мову моделювання (UML). Побудова візуальних моделей дозволила детально описати функціональні вимоги, алгоритми поведінки користувачів, обмін повідомленнями між компонентами та фізичне розгортання системи. Нище наведено детальний опис ключових діаграм проєкту.

2.3.1 Діаграма прецедентів

Діаграма прецедентів описує систему на концептуальному рівні з точки зору кінцевих користувачів, визначаючи межі системи та доступний функціонал. У розроблюваній вебсистемі персоналізованих рекомендацій розважального контенту виділено двох основних акторів, між якими встановлено відношення узагальнення:

- Користувач – базовий суб'єкт системи;
- Адміністратор – суб'єкт із розширеними правами, який успадковує всі можливості звичайного Користувача та має додаткові прецеденти для управління платформою.

До основних прецедентів, з якими взаємодіє актор Користувач, належать:

- Ідентифікація особи – базовий процес входу, який за необхідності розширюється прецедентом «Налаштування особистих інтересів»;
- Пошук контенту – процес пошуку, що обов'язку включає прецедент «Вибір за характеристиками» для деталізації запиту;

- Ознайомлення з контентом – перегляд інформації про об'єкт. Цей прецедент може розширюватися додатковими діями за бажанням користувача: «Висловлення вражень» та «Збереження в улюблене»;
- Отримання персональної добірки – ключова функція системи для кінцевого споживача.

Актор Адміністратор додатково має доступ до таких прецедентів:

- Управління контентом – комплексний процес, що включає підпроцеси «Додавання контенту», «Оновлення контенту» та «Вилучення контенту»;
- Перегляд статистики – аналіз показників роботи платформи.

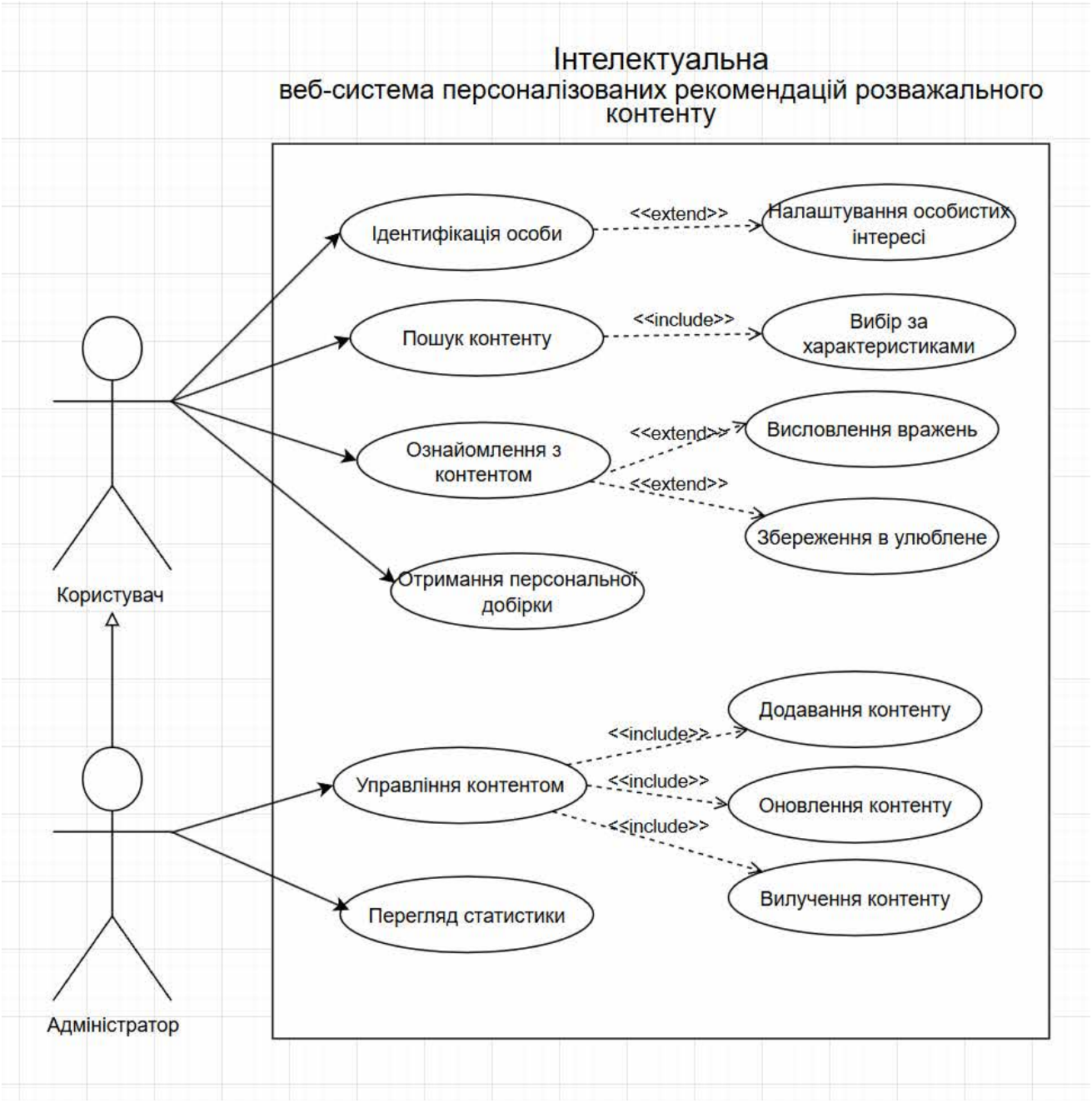

Рис. 2.3. Діаграма прецедентів інтелектуальної рекомендаційної системи

2.3.2 Діаграма діяльності

Діаграма діяльності моделює динаміку бізнес-процесу взаємодії користувача із системою, починаючи від входу і закінчуючи оновленням його профілю. Алгоритм роботи розгортається за такими етапами:

1. Старт та ідентифікація. Процес розпочинається з дії «Визначення особи».

2. Розгалуження. Система перевіряє статус клієнта. Якщо це новий користувач, ініціюється «Базове опитування щодо улюблених жанрів». Якщо це постійний користувач, виконується «Аналіз попереднього досвіду перегляду».
3. Формування пропозицій. Обидві гілки сходяться на етапі «Формування стартових пропозицій».
4. Паралельний вибір шляху. На цьому етапі користувачеві надається три альтернативні шляхи пошуку контенту: ознайомлення з персональними рекомендаціями (на основі роботи алгоритмів), самостійний вибір за критеріями (ручний пошук) або отримання випадкового контенту.
5. Взаємодія. Незалежно від обраного шляху, потоки об'єднуються на кроці «Ознайомлення з обраним контентом».
6. Зворотний зв'язок. Користувач виконує дію «Формування вражень / Залишення відгуку».
7. Оновлення системи. На основі отриманих даних відбувається «Оновлення профілю вподобань», після чого бізнес-процес завершується.

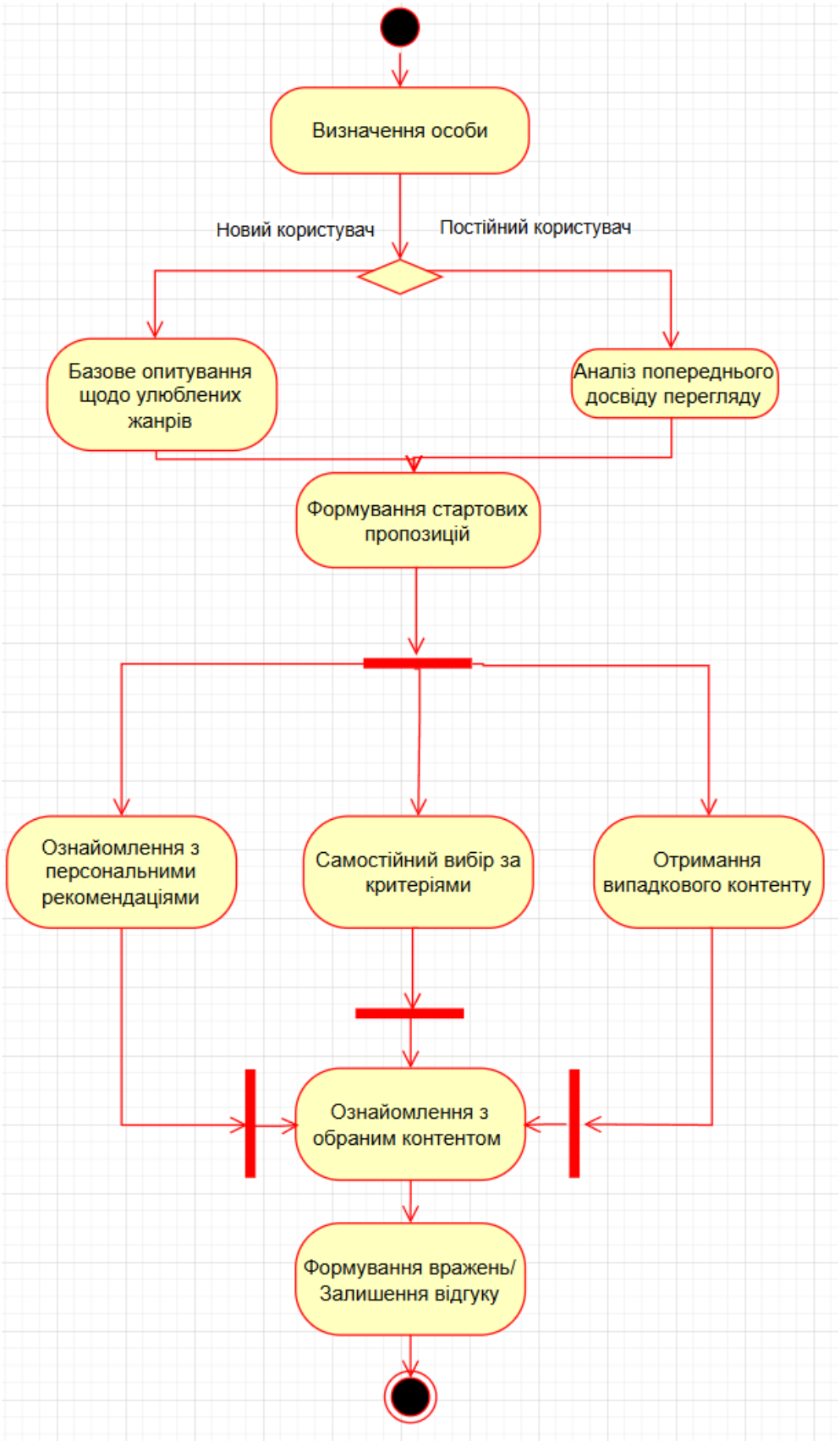

Рис. 2.4. Діаграма діяльності процесу генерації гібридних рекомендацій

2.3.3 Діаграма послідовностей

Діаграма послідовностей відображає часову динаміку взаємодії та обмін повідомленнями між об'єктами і сутностями під час роботи рекомендаційної системи. У процесі беруть участь такі лінії життя (Lifelines): Користувач, Профіль вподобань, Модель рекомендацій та Каталог контенту. Взаємодія розділена на два логічні фрейми:

Фрейм «Взаємодія та збір даних»: Користувач ініціює дію «Ознайомлення з твором та висловлення вражень», звертаючись до компонента «Профіль вподобань». Після цього «Профіль вподобань» виконує внутрішню дію – «Оновлення вподобань», фіксуючи нові дані в системі.

Фрейм «Інтелектуальний підбір контенту»: Користувач відправляє запит до «Моделі рекомендацій»: «Запит на нові пропозиції (Що подивитися?». «Модель рекомендацій» звертається до «Профілю вподобань» із «Запитом актуальних інтересів», який повертає необхідні дані через «Передачу списку інтересів та історії».

Отримавши профіль, «Модель рекомендацій» формує «Запит матеріалів за визначеними критеріями» до «Каталогу контенту». «Каталог контенту» повертає результат через «Надання списку доступного контенту». На основі цього «Модель рекомендацій» здійснює внутрішню алгоритмічну обробку – «Зіставлення профілю з контентом». Після завершення обчислень «Модель рекомендацій» повертає Користувачеві фінальний результат: «Надання готової персональної добірки».

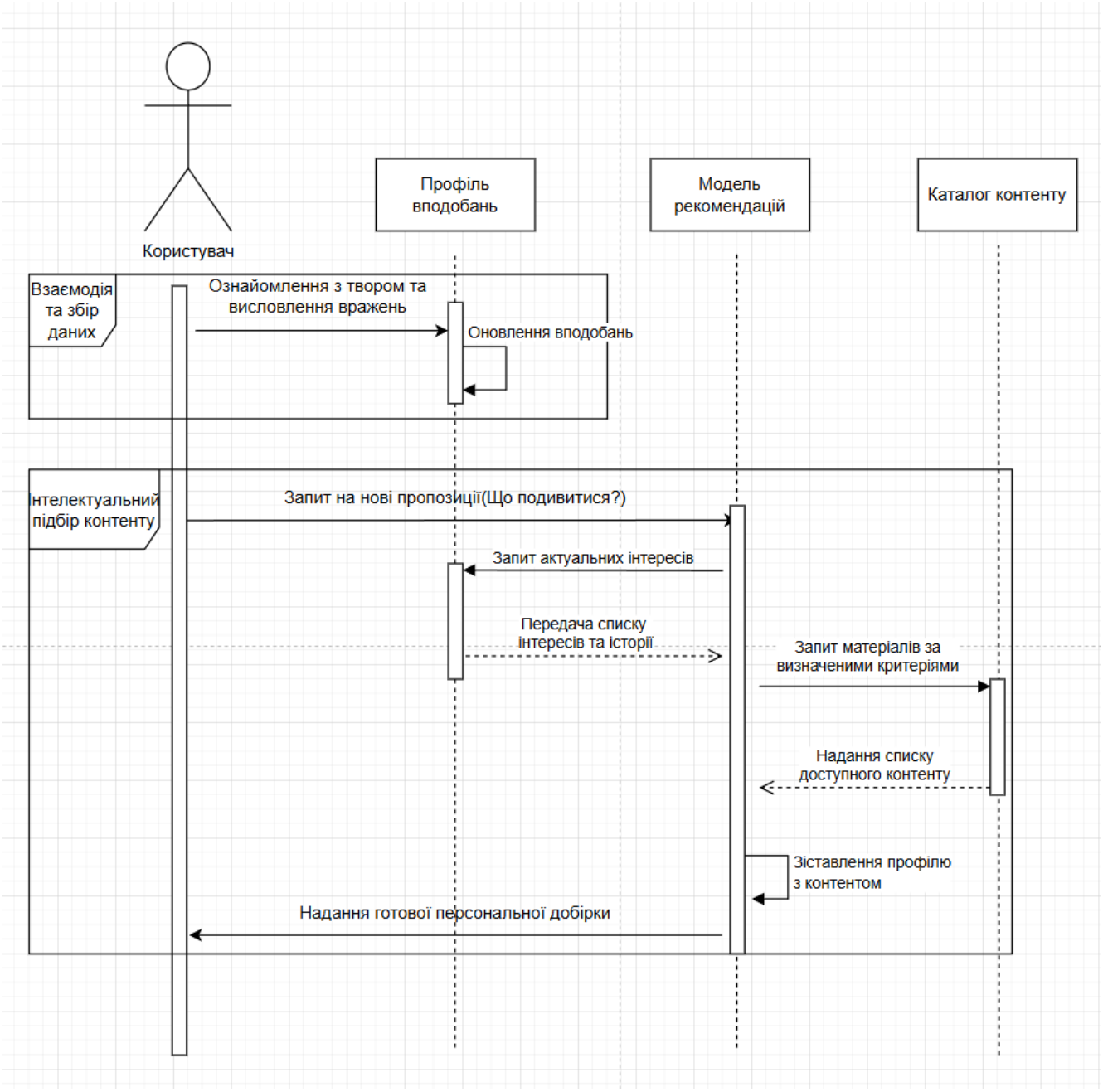

Рис. 2.5. Діаграма послідовностей взаємодії мікросервісів при оцінюванні контенту

2.3.4 Діаграма розгортання

Діаграма розгортання ілюструє фізичну топологію апаратних засобів та розподіл програмних компонентів і модулів системи між ними. Інфраструктура платформи складається з трьох основних вузлів:

1. Вузол «PC». Містить програмний компонент «Веб-браузер». Зв'язок із сервером здійснюється за протоколом HTTP/HTTPS.
2. Вузол «Сервер модулів». Є центральною обчислювальною ланкою системи. Точкою входу є компонент API Gateway, який приймає зовнішні REST-запити від веббраузера. API Gateway здійснює «Маршрутизацію» запитів до чотирьох незалежних внутрішніх мікросервісів:
    - *User Service* (управління користувачами);
    - *Content Service* (управління інформацією про медіа);
    - *Interaction Service* (управління історією та оцінками);
    - *Recommendation Service* (ядро генерації добірок).
3. Вузол «Сервер БД». Містить систему управління базами даних – компонент PostgreSQL, який забезпечує роботу з артефактом «Розподілена БД».

Усі чотири мікросервіси із «Сервера модулів» підключаються безпосередньо до бази даних через захищене з'єднання за протоколом TCP/IP для виконання транзакцій та забезпечення збереження даних.

Діаграма розгортання ілюструє фізичну топологію апаратних засобів та розподіл програмних компонентів і модулів системи між ними. Інфраструктура платформи складається з трьох основних вузлів (Nodes):

1. Вузол «PC» (Клієнтський пристрій). Містить програмний компонент «Веб-браузер». Зв'язок із сервером здійснюється за протоколом HTTP/HTTPS (REST API).
2. Вузол «Сервер модулів» (Application Server). Є центральною обчислювальною ланкою системи. Точкою входу є компонент API Gateway, який приймає зовнішні REST-запити від веббраузера. API Gateway здійснює «Маршрутизацію» запитів до чотирьох незалежних внутрішніх мікросервісів:

- User Service (управління користувачами);
- Content Service (управління інформацією про медіа);
- Interaction Service (управління історією та оцінками);
- Recommendation Service (ядро генерації добірок).

3. Вузол «Сервер БД» (Database Server). Містить систему управління базами даних – компонент PostgreSQL, який забезпечує роботу з артефактом «Розподілена БД».

Усі чотири мікросервіси із «Сервера модулів» підключаються безпосередньо до бази даних через захищене з'єднання за протоколом TCP/IP (SQL) для виконання транзакцій та забезпечення збереження даних.

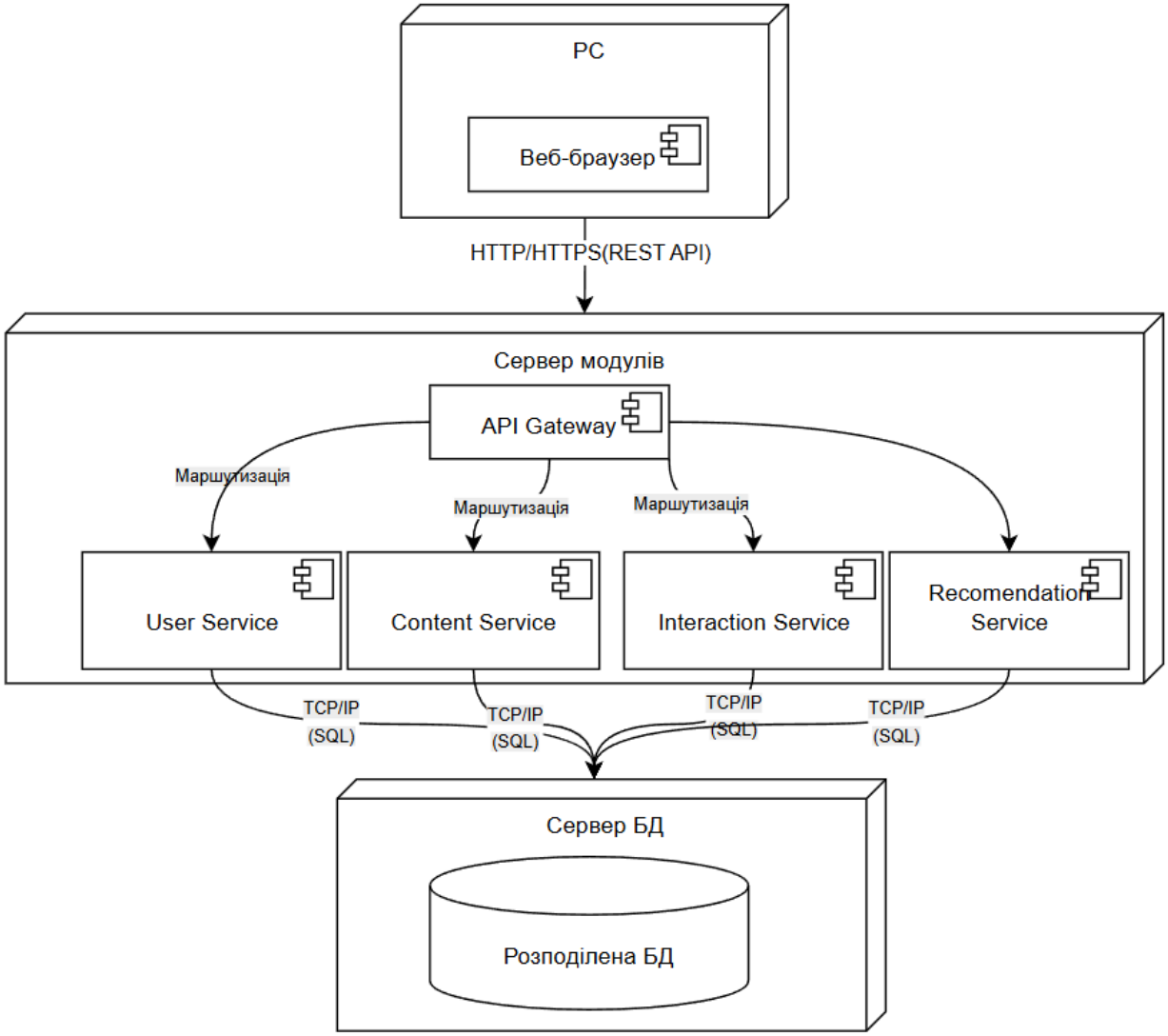

Рис. 2.6. Діаграма розгортання компонентів системи

2.4 Математична модель та алгоритмічне ядро рекомендацій

Серцем розробленої розважальної платформи є її алгоритмічне ядро. Побудова якісного, точного та релевантного алгоритму рекомендацій вимагає застосування суворого математичного апарату. Для вирішення проблеми «паралічу вибору» було спроєктовано та імплементовано гібридний підхід. Розглянемо детальніше математичну основу кожного з його складових компонентів.

1. Контентна фільтрація (Content-Based Filtering). Цей компонент алгоритму здійснює аналіз текстової інформації за допомогою потужного методу векторизації текстів TF-IDF (Term Frequency – Inverse Document Frequency). Основна ідея методу полягає в тому, що кожен об'єкт контенту в базі даних представляється як цілісний текстовий «документ». Цей документ програмно генерується шляхом злиття в один довгий рядок усіх назв жанрів об'єкта, його тегів та імен авторів чи режисерів. Далі алгоритм підраховує вагу кожного слова: слова, які зустрічаються рідко в усій базі, але часто в конкретному описі гри (наприклад, специфічний тег «Cyberpunk»), отримують найвищу математичну вагу, тоді як загальні слова (наприклад, жанр «Action») пеналізуються.

Для обчислення ступеня подібності між «вектором профілю користувача» (який синтезується з тегів усього контенту, який цей користувач раніше оцінив дуже високо) та багатовимірними векторами всього доступного невідомого контенту використовується класична метрика косинусної подібності. Вона обчислюється за формулою:

$$sim(u, i) = cos(\theta) = (u \cdot i) / (||u|| \times ||i||) \quad (2.1)$$

де u та i – це відповідні багатовимірні TF-IDF вектори профілю користувача та конкретного об'єкта контенту. Фізичний зміст цієї метрики полягає у визначенні косинуса кута між двома векторами у n-вимірному просторі ознак: якщо кут між ними близький до нуля (тобто вектори вказують в одному напрямку, а отже їхній косинус дорівнює 1), об'єкти вважаються максимально схожими за своїми внутрішніми властивостями.

2. Колаборативна фільтрація (Collaborative Filtering). Даний компонент алгоритму базується на перевіреному індустрією Item-Based підході. Він також використовує косинусну міру або статистичну кореляцію Пірсона, але застосовує їх зовсім в іншій площині – не до текстових ознак, а виключно до розрідженої матриці користувацьких рейтингів. Програмно будується велика Data-матриця, де рядки представляють унікальних користувачів платформи, стовпці – конкретні об'єкти контенту (ігри чи фільми), а значеннями в комірках

є оцінки (від 1 до 5), поставлені цими користувачами цим об'єктам. Подібність у такому алгоритмі обчислюється математично між самими об'єктами. Якщо статистично багато людей систематично ставлять однаково високі або однаково низькі оцінки одночасно грі А та фільму Б, ці об'єкти отримують високий коефіцієнт подібності sim(A, B). Це відбувається повністю незалежно від їхнього жанру, опису чи розробника, дозволяючи системі знаходити глибокі приховані закономірності у поведінці людей.

$$sim(a, b) = \sum(r_{a,p} - \bar{r}_a)(r_{b,p} - \bar{r}_b) / (\sqrt{(\sum(r_{a,p} - \bar{r}_a)^2)} \cdot \sqrt{(\sum(r_{b,p} - \bar{r}_b)^2)}) \quad (2.2)$$

3. Гібридний розрахунок (Weighted Hybrid Algorithm). Оскільки контентна фільтрація страждає від проблеми передбачуваності, а колаборативна – від проблеми «холодного старту» (нові фільми без оцінок ніколи не потраплять у видачу), система використовує їхній синтез. Для кожного об'єкта в базі обчислюється фінальна інтегральна оцінка релевантності за формулою зваженої суми:

$$P_{final} = w_{cb} \cdot P_{cb} + w_{cf} \cdot P_{cf} \quad (2.3)$$

де Pcb – це нормалізована (приведена до діапазону від 0 до 1) оцінка релевантності від контентного алгоритму, Pcf – аналогічна оцінка від колаборативного алгоритму, а wcb та wcf -спеціальні експертні вагові коефіцієнти, сума яких строго дорівнює одиниці (наприклад, 0.6 та 0.4). Ця математична формула, програмно реалізована у Python-коді, відіграє вирішальну роль: вона гарантує стабільну, збалансовану видачу рекомендацій за будь-яких умов. Якщо про гру ще ніхто нічого не знає (Pcf = 0), контентна частина формули (Pcb) все одно витягне її в топ завдяки правильним жанровим тегам.

2.5 Компонентна діаграма архітектури системи

Для наочного представлення фізичної організації компонентів системи та їх взаємодії розроблено компонентну діаграму UML. Діаграма демонструє модульність системи та шляхи обміну даними між інфраструктурними вузлами.

Центральним елементом діаграми є компонент API Gateway, розгорнутий у середовищі ASP.NET Core, який приймає всі зовнішні запити та виконує їх маршрутизацію. Від шлюзу розходяться зв'язки до мікросервісів: User Service (профілі), Content Service (дані медіа) та Recommendation Service (машинне навчання на Python). Взаємодія між цими компонентами відбувається за допомогою внутрішніх HTTP/REST викликів у закритій мережі Docker. Нижній рівень компонентної моделі займає «Сервер баз даних», де розгорнуто компонент СУБД PostgreSQL. Кожен з .NET мікросервісів має свій власний незалежний доступ (через протокол TCP/IP з виконанням SQL-запитів) до виділеної йому частини розподіленої бази даних, що чітко відображено на схемі у вигляді стрілок залежності. Така декомпозиція гарантує високу ступінь інкапсуляції та відмовостійкості архітектури.

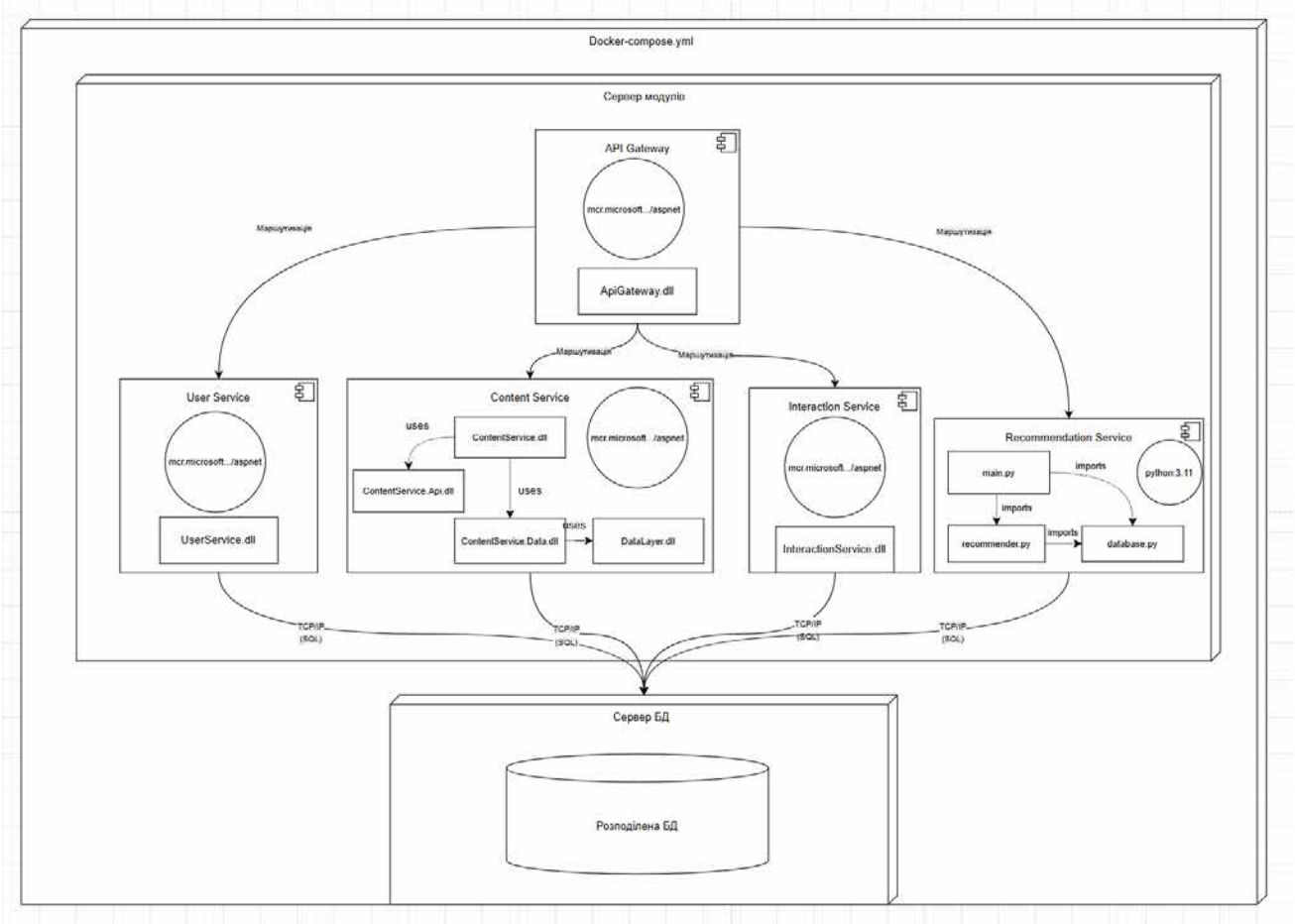

Рис. 2.7 UML Діаграма компонентів та розміщення системи

3 РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ

3.1 Вимоги до апаратного та програмного забезпечення

Ефективна розробка та стабільне функціонування мікросервісної інтелектуальної системи рекомендацій потребує обґрунтованого підбору технічних засобів та системного оточення. Оскільки архітектура проєкту передбачає використання контейнеризації для ізоляції окремих модулів, вимоги до середовища розробки та виконання є досить високими.

3.1.1 Програмне забезпечення

Для реалізації серверної частини, клієнтського застосунку та алгоритмічного ядра було обрано сучасний технологічний стек. В якості основної операційної системи для розробки використовувалася Windows 11, однак для серверного розгортання в Docker-контейнерах рекомендовано застосовувати Linux Ubuntu 22.04 LTS. Керування оркестрацією мікросервісів здійснюється через середовище віртуалізації Docker Desktop версії 4.20 або новішої з підтримкою Docker Compose V2.

Основним інструментарієм написання коду для .NET-сервісів обрано інтегроване середовище Microsoft Visual Studio 2022. Для розробки фронтенд-частини на Angular та написання скриптів машинного навчання на мові Python застосовувався редактор Visual Studio Code з відповідним набором розширень. Керування базами даних PostgreSQL 16 здійснюється за допомогою графічного клієнта pgAdmin 4. Серед важливих залежностей системи слід виділити платформу .NET 8 SDK, середовище виконання Node.js для Angular CLI, а також інтерпретатор Python 3.11 з бібліотеками Pandas та Scikit-learn для обробки даних.

3.1.2 Апаратне забезпечення

Враховуючи специфіку роботи системи, що включає одночасний запуск декількох баз даних, кешування та ресурсомістких процесів векторизації тексту в Python-модулі, визначено необхідні потужності обладнання. Для мінімального розгортання системи достатньо процесора рівня Intel Core i3 або AMD Ryzen 3 з чотирма логічними ядрами та 8 ГБ оперативної пам'яті, з яких не менше половини має бути виділено для ресурсів Docker.

Для стабільної роботи системи під навантаженням та швидкого навчання рекомендаційних моделей рекомендовано використовувати процесори Intel Core i5 або i7 11-го покоління і вище. Обсяг оперативної пам'яті в такому випадку має становити 16 ГБ для уникнення затримок при векторизації великих масивів даних. Також обов'язковою умовою є наявність NVMe SSD накопичувача з вільним простором від 10 ГБ для зберігання образів контейнерів та кешування великих обсягів інформації. Стабільне з'єднання з мережею Інтернет на швидкості від 50 Мбіт/с необхідне для коректної інтеграції з зовнішніми API-сервісами RAWG та TMDB.

3.2 Розробка серверної частини на платформі ASP.NET Core

Головна серверна частина інформаційної системи написана сучасною об'єктно-орієнтованою мовою C# на базі потужного кросплатформного фреймворку ASP.NET Core. Вибір саме цього технологічного стеку обґрунтовано низкою вагомих архітектурних переваг: винятково високою продуктивністю виконання скомпільованого коду, вбудованою «з коробки» системою впровадження залежностей, чудовою підтримкою створення RESTful API та наявністю одного з найкращих у світі ORM-фреймворків – Entity Framework Core.

Архітектура C#-додатка побудована за принципами чистої архітектури (з чітким розділенням відповідальності на шари). Це гарантує, що логіка роботи з

базою даних не змішується з логікою обробки HTTP-запитів. Для забезпечення безпеки всі кінцеві точки (endpoints), які вимагають авторизації, захищені атрибутами [Authorize], що змушує фреймворк валідувати вхідні JWT-токени за допомогою спеціального Middleware.

3.3 Інтеграція із зовнішніми API та архітектура фонової агрегації даних

Для успішного функціонування крос-доменної рекомендаційної платформи критично необхідна велика кількість якісних, актуальних та чітко структурованих персистентних даних. Ручне внесення десятків тисяч найменувань відеоігор, фільмів, серіалів та книг адміністраторами платформи є фізично неможливим, неефективним та раціонально незахищеним інженерним завданням.

Для автоматизації цього процесу в архітектурі серверної частини було спроєктовано та розроблено надійний програмний комплекс фонової агрегації даних. Взаємодія із зовнішніми інформаційними середовищами забезпечується за допомогою стійких HTTP-клієнтів, реалізованих через системний інтерфейс IHttpClientFactory. Це дозволяє централізовано керувати життєвим циклом з'єднань, оптимізувати використання системних ресурсів хост-машини та автоматизувати інтеграцію з трьома провідними публічними джерелами інформації: RAWG API (глобальна база даних відеоігор), The Movie Database (TMDB) API (інформаційне середовище кінематографу) та Google Books API (джерело метаданих для літературних творів).

Процес імпорту первинного масиву інформації організовано керованими пакетами. Адміністратор системи через спеціалізований графічний інтерфейс панелі управління ініціює запит, вказуючи лише номер необхідної сторінки каталогу. Після цього серверний сервіс формує асинхронний запит, підписаний унікальним токеном розробника, та надсилає його до відповідного зовнішнього

ендпоінту. У відповідь система отримує пакет, який стандартно містить масив із сирих об'єктів у текстовому форматі JSON.

Процес подальшої обробки отриманого пакета вимагає побудови надійного конвеєра трансформації та нормалізації даних через суттєву асиметрію та структурні розбіжності у вхідних потоках даних від різних провайдерів. Зовнішні сервери повертають відповіді у вигляді складних, глибоко вкладених деревоподібних структур, формати яких кардинально відрізняються залежно від домену. Наприклад, у відповідях кінематографічного API TMDB жанри та категорії передаються у вигляді плоских масивів числових ідентифікаторів, тоді як ігровий API RAWG транслює їх як повноцінні вкладені об'єкти з текстовими назвами та мітками, а літературний сервіс Google Books передає категорії у вигляді суцільних рядків текстових описів.

Програмний комплекс агрегації послідовно виконує такі етапи обробки отриманих JSON-пакетів:

Десеріалізація та проміжна фільтрація: сирий текстовий потік JSON, отриманий через мережевий клієнт, перетворюється в об'єкти передачі даних на платформі .NET Core. На цьому етапі відсікаються надлишкові дані, які не використовуються рекомендаційною системою.

Нормалізація та мапінг: оскільки для коректної роботи математичного рекомендаційного ядра потрібен уніфікований текстовий опис об'єкта, сервіс-адаптер трансформує розрізнені зовнішні атрибути у єдину сутність внутрішньої моделі даних. Масиви текстових тегів, ключових слів та назв жанрів піддаються деструктуризації. Сервіс зчитує лише їхні рядкові значення, очищає від зайвих символів пунктуації, переводить у нижній.

Контроль унікальності: перед фіксацією змін у сховищі система виконує превентивну перевірку на наявність дублікатів. За допомогою унікального зовнішнього ідентифікатора об'єкта та прапорця його типу система виконує швидкий пошук у локальній таблиці. Якщо об'єкт уже присутній у базі даних,

транзакція відхиляється, що гарантовано дозволяє уникнути дублювання інформації при повторних запусках імпорту або накладанні пакетів.

Персистентне збереження: після успішного проходження всіх етапів перевірки та трансформації, об'єкти передаються до контексту даних Entity Framework Core. Запис у базу даних PostgreSQL виконується асинхронно із суворим дотриманням складної ієрархії таблиць типу TPT, що забезпечує чітке розділення загальних характеристик медіаконтенту та специфічних атрибутів, притаманних лише окремим доменам (іграм, фільмам чи книгам).

Парсинг великих обсягів даних у реальному часі неминуче пов'язаний із жорсткими інфраструктурними обмеженнями кількості запитів, що накладаються власниками зовнішніх сервісів для захисту від перевантаження. Для запобігання блокуванню IP-адреси нашого сервера, в інтеграційному модулі реалізовано продумані механізми алгоритмічної затримки між послідовними викликами ендпоінтів.

У разі виникнення помилок мережевого з'єднання або отриманні від зовнішнього сервера специфічного HTTP-статусу 429 Too Many Requests, система активує надійну політику повторних спроб. Цей механізм автоматично призупиняє виконання конвеєра обробки даних, розраховує експоненційну затримку часу та повторює запит доти, доки зовнішній сервер не підтвердить готовність віддати пакет даних.

3.4 Розробка клієнтського застосунку на Angular та оптимізація пагінації

Візуальний фронтенд системи побудовано за концепцією Single Page Application за допомогою потужного компонентного фреймворку Angular від Google. Розробка велася мовою TypeScript, що забезпечило строгу типізацію даних на клієнті, зменшило кількість помилок під час виконання та покращило якість коду. Взаємодія з бекендом реалізована через парадигму реактивного

програмування за допомогою потужної бібліотеки RxJS та об'єктів типу Observable.

Особливої уваги під час реалізації заслуговує розробка центральної адміністративної панелі. Графічний інтерфейс користувача спроєктовано за логічним архітектурним патерном Hub-and-Spoke, де існує зручний центральний вузол з навігаційними віджетами макрокатегорій (Ігри, Фільми, Серіали, Книги) та детальні внутрішні сторінки для кожної з категорій з великими інтерактивними таблицями CRUD-операцій.

У процесі тестування розробленого інтерфейсу виникла критична технологічна проблема: при завантаженні з бекенду великого масиву, що містив понад 900+ об'єктів контенту, і спробі вивести їх у стандартну HTML-таблицю за допомогою структурної директиви *ngFor, браузер користувача повністю «заморожувався» через надмірне, неприпустиме навантаження на DOM-дерево. Для вирішення цієї поширеної проблеми було проаналізовано та свідомо відхилено варіант класичної серверної пагінації (яка б вимагала складних додаткових запитів до БД при кожному кліку). Натомість було спроєктовано та успішно реалізовано інтелектуальний алгоритм «розумної клієнтської пагінації» безпосередньо в оперативній пам'яті браузера.

Алгоритм працює наступним чином: масивні дані завантажуються з бекенду лише один раз і зберігаються у глобальній компонентній змінній allTableData. Для безпосереднього відображення в HTML-шаблоні створюється менший проміжний масив currentTableData, який містить рівно 100 елементів для поточної активної сторінки. Ключовим, вирішальним інструментом оптимізації рендерингу стало застосування системного класу ChangeDetectorRef. Прямий виклик методу detectChanges() примусово наказує ядру Angular перемалювати виключно необхідні змінені елементи табличного віджета, ігноруючи решту важкого життєвого циклу програми. Це елегантне інженерне рішення забезпечило миттєву, бездоганну чуйність інтерфейсу при гортанні сторінок чи видаленні записів адміністратором. Крім того, імплементовано гнучкий try-catch

парсинг отриманих JSON-відповідей для коректної обробки різного регістру ключів, що надходили від різних мікросервісів.

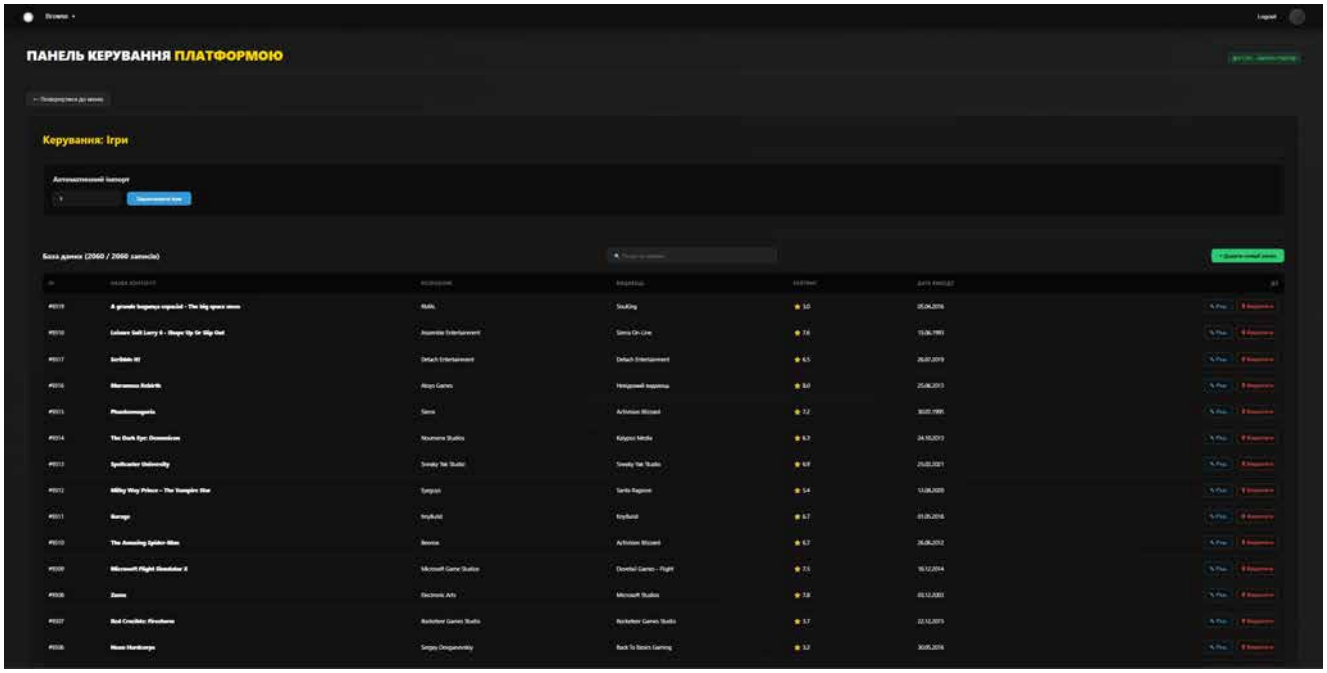

Рис. 3.1 Реалізація системи інтелектуальної клієнтської пагінації в адміністративній панелі

3.5 Програмна реалізація рекомендаційного ядра

Математичне ядро рекомендацій було архітектурно винесено в окремий незалежний мікросервіс, написаний мовою Python з використанням легковагового веб-фреймворку. Безпосередньо в класі «HybridRecommender» повністю програмно реалізовано описану у попередньому розділі складну математичну модель гібридної системи.

Алгоритм інтенсивно використовує клас TfidfVectorizer з популярної наукової бібліотеки scikit-learn для глибинної обробки сирих текстових фіч (жанрів, тегів, описів). Цікавою технічною особливістю розробленої програмної реалізації є свідоме штучне дублювання найбільш вагомих параметрів (наприклад, імені розробника гри або режисера фільму) у згенерованому рядку ознак. Це робиться для того, щоб надати їм штучно більшої математичної ваги при подальшій векторизації, оскільки користувачі частіше обирають контент

саме за іменем улюбленого автора. Двовимірна матриця взаємодій для колаборативної фільтрації надзвичайно швидко створюється за допомогою методу pivot_table з бібліотеки Pandas. Фінальна генерація гібридної рекомендації для конкретного користувача займає лише частки секунди завдяки ефективно оптимізованим матричним множенням бібліотеки NumPy під капотом.

3.6 Контейнеризація екосистеми за допомогою Docker

Для того, щоб назавжди уникнути класичної та болючої проблеми розробників «це працює на моєму комп'ютері, але не працює на сервері», уся без винятку розроблена IT-екосистема була професійно контейнеризована. Створено оптимізовані конфігураційні файли Dockerfile для кожного окремого мікросервісу з детальним покроковим описом необхідного середовища виконання та залежностей. За допомогою потужного інструменту оркестрації docker-compose.yml усі ці різнорідні контейнери, включаючи інфраструктурні бази даних PostgreSQL, логічно об'єднані в єдину захищену віртуальну мікромережу. Це революційне рішення дозволяє безболісно розгорнути весь складний проєкт рекомендаційної платформи на абсолютно будь-якому сервері чи хмарному провайдері лише однією простою консольною командою: docker-compose up, гарантуючи при цьому ідеальну ізоляцію залежностей та надійну, безперебійну роботу.

3.7 Опис графічного інтерфейсу та керівництво користувача

Для забезпечення зручної взаємодії користувачів з рекомендаційною системою було розроблено сучасний графічний інтерфейс у стилі «скляного мінімалізму». Інтерфейс адаптивний, інтуїтивно зрозумілий та розділений на клієнтську частину і панель адміністрування. Нижче наведено опис основних екранів та сценаріїв використання системи.

3.7.1 Каталог контенту та пошук

Основна взаємодія користувача з базою даних відбувається через сторінку каталогу (Browse). За допомогою випадаючого меню в навігаційній панелі користувач може обрати для перегляду весь контент або відфільтрувати його за конкретним типом: ігри, фільми, серіали чи книги.

На сторінці каталогу об'єкти відображаються у вигляді сітки інтерактивних карток, кожна з яких містить постер, назву та середній рейтинг. Контент відсортовано за алфавітом. Для швидкого знаходження потрібної інформації передбачено рядок текстового пошуку, а також випадаючий список для фільтрації результатів за жанрами. Оскільки база даних містить велику кількість записів, для зниження навантаження на мережу та зручності навігації реалізовано механізм пагінації.

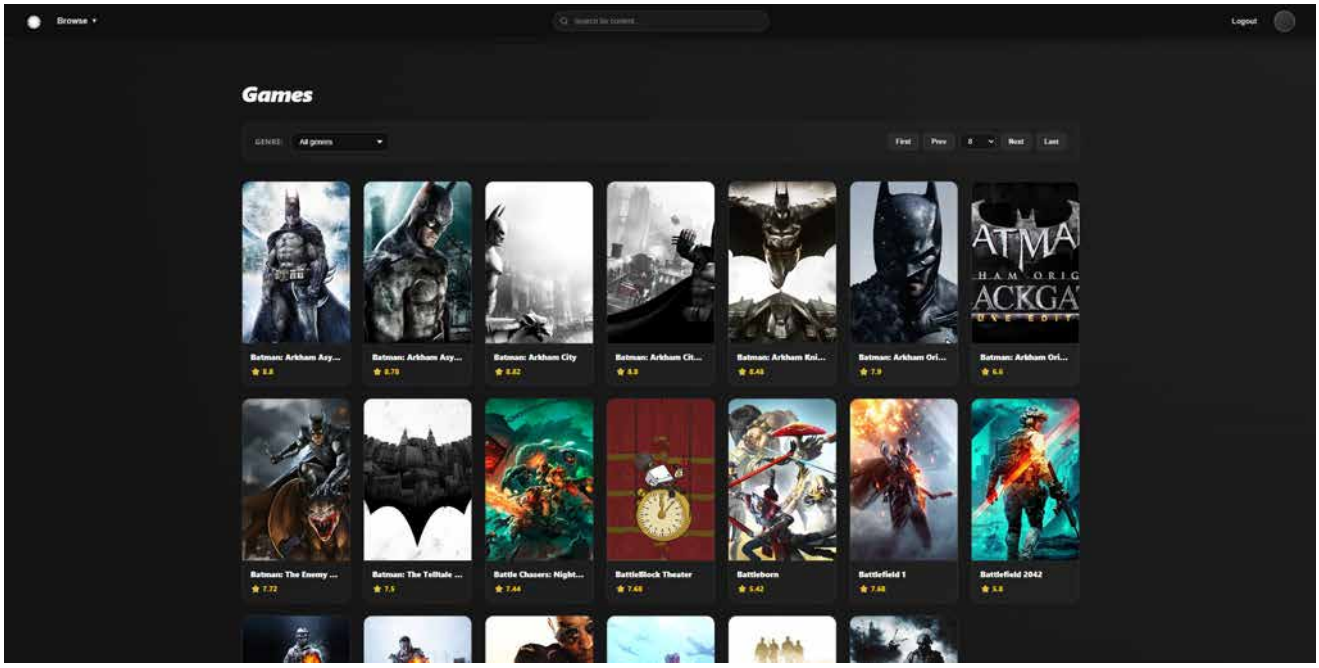

Рис. 3.2. Сторінка каталогу контенту з фільтрацією та пагінацією

3.7.2 Детальна інформація про об'єкт та рекомендації

При натисканні на картку контенту користувач переходить на сторінку детальної інформації. Структура цієї сторінки спроєктована таким чином, щоб

надати вичерпні дані: великий постер, текстовий опис, метадані та перелік жанрів.

Особлива увага приділена блоку рейтингів та взаємодії. На горизонтальній панелі відображаються:

1. Загальний рейтинг об'єкта, отриманий із зовнішніх джерел.
2. Внутрішній рейтинг платформи, сформований на основі оцінок зареєстрованих користувачів системи.
3. Інтерактивний блок з п'яти зірок для виставлення власної оцінки.

Також біля назви контенту розміщена кнопка у формі серця, яка дозволяє додати об'єкт до списку «Обране». Під основним блоком інформації розташована карусель «Схожий контент». Цей блок є прямим результатом роботи гібридного рекомендаційного ядра, яке підбирає найбільш релевантні об'єкти на основі контентної близькості.

Рис. 3.3. Сторінка детальної інформації про контент та блок рекомендацій

3.7.3 Особистий кабінет користувача

Для забезпечення максимальної персоналізації в системі передбачено сторінку профілю користувача. В особистому кабінеті користувач може переглядати та редагувати свої облікові дані.

Окрім налаштувань профілю, на цій сторінці акумулюється історія взаємодії клієнта із системою. Тут відображаються списки контенту, який користувач оцінив або додав до обраного. Цей функціонал не лише створює зручну персональну бібліотеку для користувача, але й дозволяє системі прозоро демонструвати, на основі яких саме даних будуються його майбутні колаборативні рекомендації.

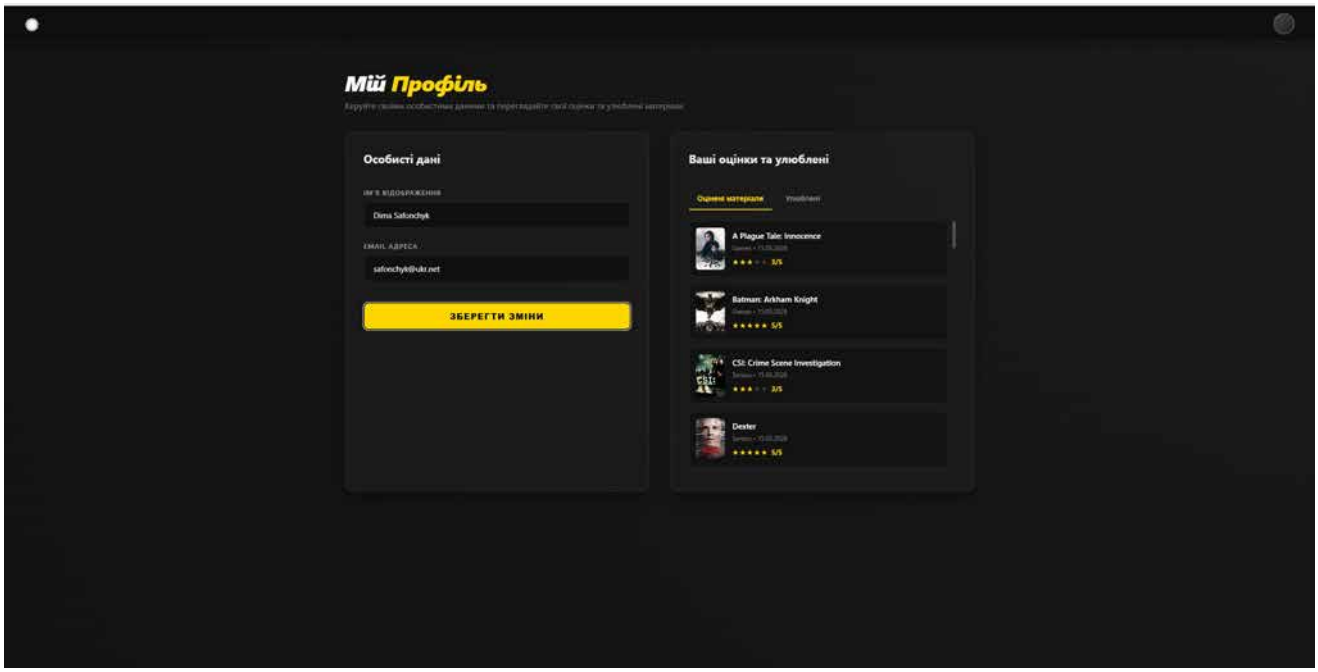

Рис. 3.4. Особистий кабінет користувача з історією взаємодій

3.7.4 Панель керування та аналітика

Для управління платформою створено захищену зону адміністрування, доступ до якої мають лише користувачі з відповідними привілеями.

Головне меню панелі керування складається з карток швидкого доступу до різних модулів системи: ігри, фільми, серіали, книги та статистика. Перейшовши у відповідний розділ контенту, адміністратор може вручну редагувати базу

даних, видаляти нерелевантні записи, а також запускати автоматичні бекграунд-парсери для завантаження нових даних із зовнішніх API.

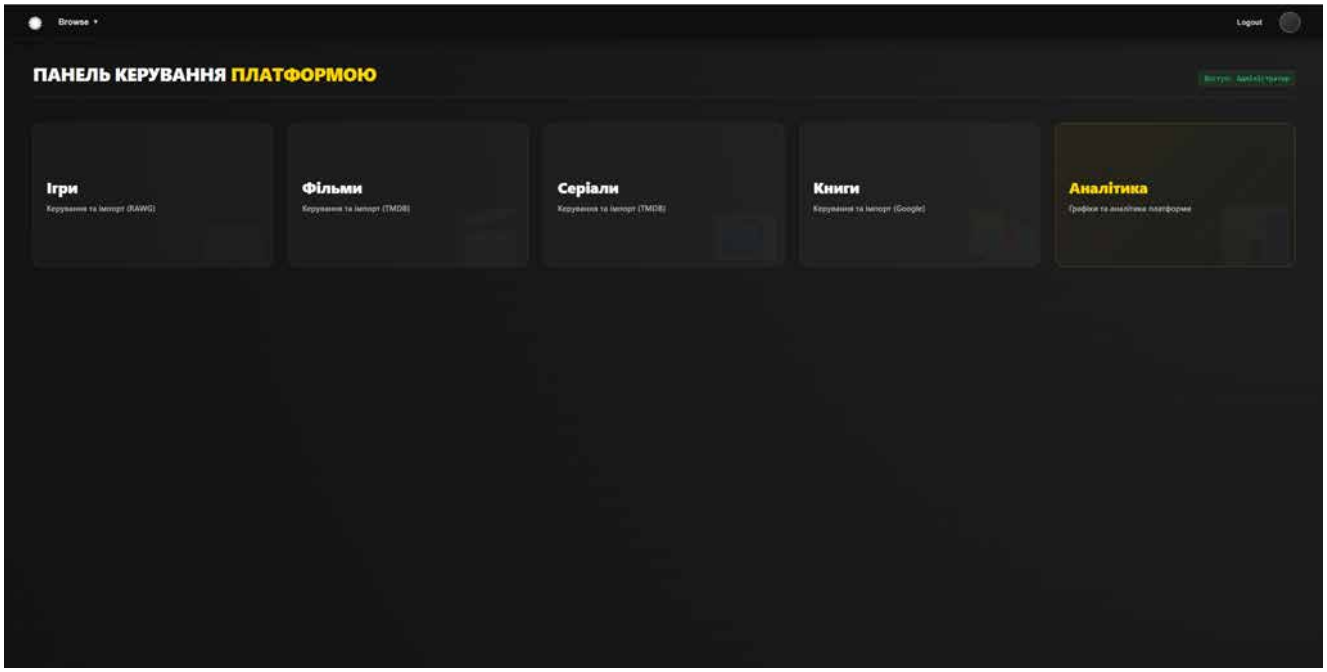

Рис. 3.5. Головне меню панелі адміністрування

Важливою складовою адміністрування є модуль «Аналітика». Ця сторінка містить дашборд з візуалізацією ключових показників роботи системи за допомогою графіків Chart.js. Наприклад, кругова діаграма показує відсоткове співвідношення контенту за жанрами у базі даних, а лінійні графіки демонструють активність користувачів (кількість виставлених оцінок та додавань в обране за певний період). Це дозволяє адміністраторам оцінювати залученість аудиторії та якість наповнення бази даних.

Рис. 3.6. Дашборд аналітики та статистики платформи

3.8 Склад інсталяційного пакету

Завершальним етапом реалізації інтелектуальної вебсистеми є підготовка інсталяційного пакету, що містить усі необхідні компоненти для розгортання в локальному або хмарному середовищі, а також створення інструкцій для адміністраторів та кінцевих користувачів.

3.8.1 Склад інсталяційного пакету

Програмний продукт поставляється у вигляді репозиторію з чітко визначеною структурою каталогів, що забезпечує автоматизоване збирання через Docker. До складу інсталяційного пакету входять:

1. **Каталог /src/backend**: містить вихідний код усієї серверної екосистеми. До його складу входять мікросервіси на базі .NET 8, конфігурації Entity Framework для автоматичного розгортання бази даних, а також внутрішній підкаталог RecommendationService, який містить скрипти на мові Python, програмну реалізацію

рекомендаційного алгоритму, файли векторизації та FastAPI-сервер для обслуговування запитів на рекомендації.

2. **Каталог /src/web-client**: містить вихідний код клієнтського застосунку на Angular, включаючи компоненти інтерфейсу, сервіси взаємодії з API та конфігураційні файли середовища.
3. **Файл docker-compose.yml**: головний файл маніфесту для оркестрації контейнерів, що визначає правила збирання та запуску серверних модулів, клієнтського додатку, баз даних PostgreSQL.
4. **Файл .env.example**: шаблон конфігураційних змінних середовища (рядки підключення, ключі зовнішніх API, налаштування секретів для JWT-токенів).

3.8.2 Інструкція з розгортання системи

Завдяки повноцінній контейнеризації, процес інсталяції зведено до декількох стандартизованих кроків, а структура бази даних генерується автоматично при першому запуску бекенд-сервісів (за допомогою механізму міграцій). Для успішного запуску на цільовій машині має бути встановлено Docker Desktop або Docker Engine з Docker Compose.

Послідовність дій для розгортання:

1. Завантажити інсталяційний пакет у робочу директорію.
2. Створити файл .env на основі шаблону .env.example та вказати персональні ключі доступу до зовнішніх сервісів (RAWG, TMDB API) і секрети для шифрування.
3. Відкрити термінал у кореневій папці проєкту та виконати команду збирання образів: docker-compose build.
4. Запустити систему командою: docker-compose up -d.

5. Після успішного старту та ініціалізації контейнерів, вебзастосунок стане доступним у браузері за адресою http://localhost:4200.

4 ТЕСТУВАННЯ ТА ОЦІНКА РЕЗУЛЬТАТІВ

4.1 Стратегія тестування та забезпечення якості програмного забезпечення

Для забезпечення високого рівня надійності, безперебійності та безпеки роботи спроєктованої розважальної системи було застосовано комплексну багатошарову стратегію забезпечення якості. Враховуючи мікросервісну архітектуру проєкту, що базується на технологіях .NET 8, Angular та Python, стратегія тестування охоплювала як окремі ізольовані модулі, так і складні ланцюжки взаємодії між ними у контейнеризованому середовищі Docker.

Процес верифікації програмного продукту структурно включав декілька ієрархічних рівнів:

1. **Модульне тестування.** На цьому рівні проводилася перевірка найбільш дрібних функцій та методів. У серверній частині фокус було зосереджено на валідації вхідних DTO-моделей та логіці контролерів. В Angular-клієнті перевірялася правильність роботи алгоритмів парсингу сирих JSON-відповідей, що надходять від API, та їх трансформація у внутрішні інтерфейси застосунку.
2. **Інтеграційне тестування.** Оцінювалася коректність взаємодії між розрізненими мікросервісами через API Gateway. Критично важливим етапом була перевірка стабільності зв'язку між .NET-сервісом контенту та Python-модулем рекомендаційного ядра, а також підключення до бази даних PostgreSQL.
3. **Системне тестування.** Проводилася повна перевірка комплексної поведінки інтелектуального алгоритму в реальних умовах. На цьому етапі аналізувалася відповідність системи всім функціональним вимогам, визначеним на етапі проєктування.

Основним інструментом верифікації функціональності було обрано метод «чорного ящика», де тестування проводиться з позиції кінцевого користувача без втручання в код під час перевірки. Результати виконання ключових тестових сценаріїв представлені в таблиці 4.1.

Таблиця 4.1

Специфікація та результати функціонального тестування

№	Назва сценарію (Test Case)	Кроки відтворення	Очікуваний результат	Статус (Результат)
1	Реєстрація нового користувача	1. Відкрити форму реєстрації. 2. Ввести дані. 3. Натиснути «Register».	Створення запису в БД PostgreSQL, повернення JWT-токена.	Успішно
2	Пошук контенту за назвою	1. Ввести назву в поле пошуку. 2. Натиснути Enter.	Відображення списку об'єктів, що відповідають запиту.	Успішно
3	Генерація рекомендацій	1. Перейти на сторінку об'єкта. 2. Переглянути блок «Схожий контент».	Python-ядро повертає список 5 найбільш релевантних об'єктів.	Успішно
4	Виставлення оцінки	1. Обрати зірковий рейтинг на сторінці. 2. Перевірити загальний бал.	Зміна середнього рейтингу об'єкта та збереження оцінки в БД.	Успішно
5	Робота кнопки «Surprise Me»	1. Натиснути кнопку «Surprise». 2. Очікувати редірект.	Система випадковим чином обирає об'єкт та відкриває його сторінку.	Успішно

Застосована стратегія дозволила виявити та усунути низку критичних помилок на ранніх етапах розробки та гарантувати високу якість кінцевого програмного продукту.

4.2 Оцінка ефективності та релевантності гібридних рекомендацій

Критично важливим етапом перевірки якості платформи стало тестування безпосередньо рекомендаційного математичного ядра. Це тестування було

проведено за допомогою спеціально змодельованих, штучно згенерованих профілів користувачів різного типу. Для перевірки крос-медійності було створено складний профіль гіпотетичного користувача, який дуже високо оцінив три відеоігри в улюбленому жанрі «рольові ігри» (RPG) та два фільми у специфічному жанрі «Похмура наукова фантастика».

При першому тестовому запуску генерації рекомендацій з екстремальними параметрами $w_{cb} = 1.0$, $w_{cf} = 0.0$ (тобто працював виключно чистий контентний алгоритм фільтрації), система цілком очікувано і передбачувано видала список об'єктів, що складалися переважно з інших популярних RPG та наукової фантастики, ігноруючи все інше. Проте, при зміні математичного балансу алгоритмів на повноцінний гібридний (де вагові коефіцієнти становили $w_{cb} = 0.6$, $w_{cf} = 0.4$, картина радикально змінилася: у фінальній видачі для цього ж користувача почали органічно з'являтися електронні книги та захоплюючі серіали суміжних або дотичних жанрів (таких як фентезі чи кіберпанк). Ці об'єкти гіпотетичний користувач безпосередньо ніколи не оцінював, але алгоритм визначив, що вони є надзвичайно популярними серед інших реальних людей у базі з подібним профілем та смаками. Це практичне випробування наочно і беззаперечно підтверджує високу працездатність, точність та необхідну гнучкість розробленої гібридної математичної моделі.

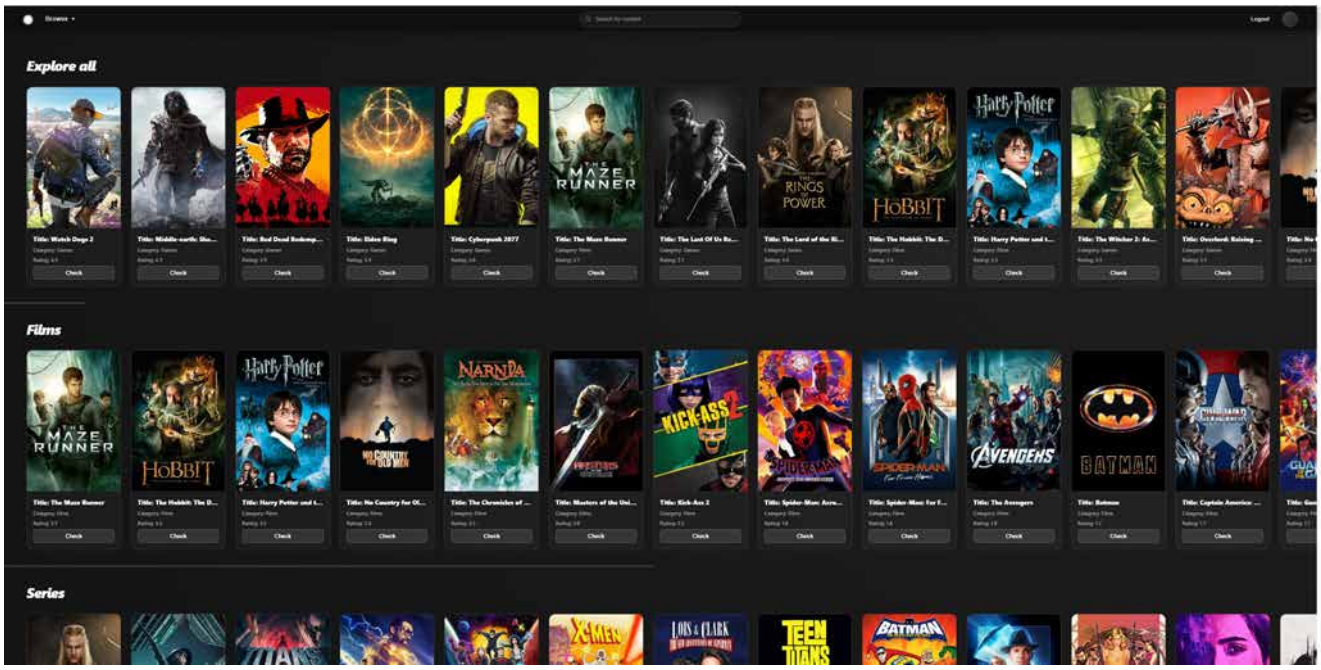

Рис. 4.1 Практичний результат роботи оптимізованого гібридного алгоритму рекомендацій

4.3 Навантажувальне тестування клієнтської та серверної частини

Останнім і не менш важливим етапом стало жорстке навантажувальне тестування продуктивності адміністративної панелі, розробленої на фреймворку Angular. З метою стрес-тесту у локальну тестову базу даних PostgreSQL за допомогою створених парсерів було швидко завантажено масивний набір даних, що складався з 1500 важких об'єктів контенту. Завдяки завчасній імплементації продуманого алгоритму клієнтської пагінації в пам'яті та правильному використанню класу ChangeDetectorRef, реальний час повного рендерингу HTML-компонента таблиці у браузері не перевищував 50-80 мілісекунд при гортанні сторінок. Такі блискучі метрики забезпечують винятково плавний і приємний користувацький досвід та гарантують абсолютну відсутність прикрого блокування головного потоку обчислень браузера навіть на слабких пристроях адміністраторів.

4.4 Перспективи масштабування та модернізації системи

Розроблена інтелектуальна вебсистема персоналізованих рекомендацій на поточному етапі повністю задовольняє визначеним функціональним та технологічним вимогам, демонструючи стабільну роботу в ізольованому контейнеризованому середовищі Docker на базі одного серверного вузла. Проте, у разі реального промислового запуску платформи, стрімке зростання кількості активних користувачів, обсягу їхніх оцінок та накопичення мільйонів одиниць розважального контенту неминуче призведе до підвищення навантаження на обчислювальні вузли. Завдяки первинному вибору мікросервісної архітектури, система має високий потенціал для вертикального та горизонтального масштабування, а також для подальшої функціональної модернізації.

Першочерговим напрямком технологічної модернізації екосистеми є перехід від базової оркестрації за допомогою Docker Compose до повноцінного промислового інструменту управління контейнерами Kubernetes. Це дозволить впровадити механізм автоматичного горизонтального масштабування подів на основі моніторингу утилізації центрального процесора та оперативної пам'яті. У періоди пікового навантаження. Kubernetes зможе автоматично реплікувати екземпляри шлюзу API Gateway та мікросервісу контенту, рівномірно розподіляючи трафік між ними за допомогою балансувальників навантаження. Після спаду активності зайві контейнери будуть автоматично знищуватися, що мінімізує витрати на хмарну інфраструктуру.

Суттєвого доопрацювання при масштабуванні потребуватиме шар збереження даних. Поточна реляційна СУБД PostgreSQL, розгорнута в одному контейнері, при мільйонних реєстраціях може стати «вузьким місцем» системи через обмеження дискових операцій читання та запису. Для вирішення цієї проблеми перспективним є впровадження архітектури реплікації типу «Master-Slave», де головний сервер обробляє виключно транзакції запису, а мережа синхронізованих реплік служить лише для читання та швидкого повернення списків контенту Angular-клієнту. Додатково, для оптимізації зберігання

гігантських матриць взаємодії користувачів, доцільно застосувати горизонтальне секціонування таблиці оцінок та впровадити пулер з'єднань PgBouncer для ефективного менеджменту активних підключень до бази даних.

Окремим вектором розвитку є еволюція рекомендаційного ядра, яке наразі реалізоване у вигляді синхронного FastAPI-сервера та пакетної обробки Scikit-learn. При масштабуванні платформи пряма HTTP-взаємодія між сервісом контенту та модулем машинного навчання може викликати затримки. Перспектива модернізації полягає в переході на подієво-орієнтовану архітектуру з використанням брокерів повідомлень Apache Kafka або RabbitMQ. У такому сценарії кожна дія користувача публікується як асинхронна подія в чергу, звідки Python-модуль забирає її для фонового перерахунку профілю вподобань, не блокуючи основний потік роботи вебзастосунку.

З погляду математичного апарату, у міру накопичення даних, поточний метод контентної фільтрації на основі міри TF-IDF доцільно модернізувати шляхом інтеграції нейромережевих підходів. Використання методів матричної факторизаці або архітектур на кшталт Two-Tower Neural Networks дозволить будувати значно складніші ембеддінги користувачів та об'єктів. Це допоможе враховувати приховані семантичні зв'язки та контекст споживання контенту, що кардинально підвищить релевантність видачі.

Нарешті, оптимізація доставки статичного контенту вимагатиме інтеграції системи з мережами доставки контенту. Кешування статичних ресурсів на географічно розподілених серверах CDN дозволить мінімізувати час завантаження інтерфейсу користувача в будь-якій точці світу, знижуючи пряме навантаження на Host-машину Docker. Таким чином, запропонований комплекс заходів з масштабування забезпечить перетворення розробленого прототипу на високонадійну, відмовостійку систему, здатну обслуговувати сотні тисяч користувачів одночасно без втрати продуктивності.

ВИСНОВКИ

У результаті сумлінного та комплексного виконання бакалаврської кваліфікаційної роботи було успішно, з дотриманням усіх сучасних стандартів інженерії, спроєктовано та програмно реалізовано високотехнологічну крос-медійну рекомендаційну систему розважального контенту.

1. Проведено глибокий, всебічний науковий аналіз предметної області та чітко виявлено ключову і найболючішу проблему сучасних розважальних сервісів – їхню жорстку доменну ізоляцію. Ця штучна ізоляція критично заважає формуванню комплексного, цілісного профілю інтересів сучасного користувача, який споживає різний контент.

2. Теоретично обґрунтовано та практично спроєктовано розподілену мікросервісну архітектуру з використанням передового архітектурного патерну Backend-For-Frontend. Це еталонне рішення забезпечило надзвичайно стійку, відмовостійку та гнучку комунікацію між клієнтським SPA-додатком на Angular та внутрішніми ізольованими сервісами, написаними на C# та Python.

3. Завдяки віртуозному використанню об'єктно-орієнтованого патерну Table-Per-Type у реляційній СУБД PostgreSQL було створено ідеально нормалізовану та розширювану ієрархію для безпечного зберігання масивів метаданих про відеоігри, фільми, серіали та електронні книги.

4. Вирішено проблему падіння продуктивності фронтенд-додатка при одночасній обробці та виведенні величезних обсягів інформації. Шляхом розробки інноваційного алгоритму клієнтської пагінації в оперативній пам'яті браузера з використанням потужних інструментів контролю виявлення змін у фреймворку Angular, вдалося досягти миттєвого рендерингу інтерфейсів.

5. Найважливішим і фундаментальним науковим здобутком роботи є математичне обґрунтування та повноцінна практична реалізація гібридного рекомендаційного ядра машинного навчання. Інтелектуальне об'єднання

контентної фільтрації (на базі алгоритмів векторизації TF-IDF та метрики косинусної подібності) та колаборативної Item-based фільтрації за допомогою математичного методу зваженої суми дозволило раз і назавжди подолати відому проблему «холодного старту». Дана модель гарантовано забезпечує надзвичайно високу персональну релевантність та бажану крос-медійну різноманітність фінальної видачі контенту.

Створений у межах роботи кінцевий програмний продукт повністю і беззастережно відповідає усім поставленим на початку функціональним та нефункціональним вимогам. Система є надійною, архітектурно готовою до масштабних навантажень, повністю розгорнута за допомогою сучасної технології контейнеризації Docker і абсолютно готова до повноцінного комерційного впровадження. У майбутньому платформа має величезний потенціал до вдосконалення, зокрема через інтеграцію ще більш складних нейромережевих моделей глибинного навчання для ще глибшого та точнішого аналізу прихованих поведінкових патернів своїх користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ricci F., Rokach L., Shapira B. Recommender Systems Handbook. 3rd ed. Springer US, 2022. 1003 p. SpringerLink : вебсайт. [Електронний ресурс]. – Режим доступу: https://link.springer.com/book/10.1007/978-1-0716-2197-4 – Назва з екрана. – Дата звернення: 25.04.2026.
2. Aggarwal C. C. Recommender Systems: The Textbook. Springer, 2016. 498 p. SpringerLink : вебсайт. [Електронний ресурс]. – Режим доступу: https://link.springer.com/book/10.1007/978-3-319-29659-3 – Назва з екрана. – Дата звернення: 25.04.2026.
3. Koren Y., Bell R., Volinsky C. Matrix Factorization Techniques for Recommender Systems. IEEE Computer, 2009. Vol. 42, No. 8. pp. 30–37. IEEE Xplore : цифрова бібліотека. [Електронний ресурс]. – Режим доступу: https://doi.org/10.1109/MC.2009.263 – Назва з екрана. – Дата звернення: 25.04.2026.
4. Newman S. Building Microservices: Designing Fine-Grained Systems. 2nd ed. O'Reilly Media, 2021. 612 p. O'Reilly Learning Platforms : вебсайт. [Електронний ресурс]. – Режим доступу: https://www.oreilly.com/library/view/building-microservices-2nd/9781492034018/ – Назва з екрана. – Дата звернення: 26.04.2026.
5. Freeman A. Pro ASP.NET Core 6: Develop Cloud-Ready Web Applications Using MVC, Blazor, and Razor Pages. Apress, 2022. 1256 p. SpringerLink : вебсайт. [Електронний ресурс]. – Режим доступу: https://link.springer.com/book/10.1007/978-1-4842-7957-1 – Назва з екрана. – Дата звернення: 26.04.2026.
6. Офіційна документація фреймворку Angular. Angular : офіційний сайт. [Електронний ресурс]. – Режим доступу: https://angular.io/docs – Назва з екрана. – Дата звернення: 20.04.2026.

7. Офіційна документація ASP.NET Core та Entity Framework Core. Microsoft Learn : портал для розробників. [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core/ – Назва з екрана. – Дата звернення: 22.04.2026.
8. Документація платформи Docker. Accelerate how you build, share, and run applications : офіційний сайт. [Електронний ресурс]. – Режим доступу: https://www.docker.com/ – Назва з екрана. – Дата звернення: 10.04.2026.
9. Документація бібліотеки Scikit-learn. Machine Learning in Python : офіційний сайт. [Електронний ресурс]. – Режим доступу: https://scikit-learn.org/stable/ – Назва з екрана. – Дата звернення: 21.04.2026.
10. Офіційний сайт СУБД PostgreSQL. The World's Most Advanced Open Source Relational Database : офіційний сайт. [Електронний ресурс]. – Режим доступу: https://www.postgresql.org/ – Назва з екрана. – Дата звернення: 15.04.2026.
11. Richards M., Ford N. Fundamentals of Software Architecture: An Engineering Approach. O'Reilly Media, 2020. 416 p. O'Reilly Learning Platforms : вебсайт. [Електронний ресурс]. – Режим доступу: https://www.oreilly.com/library/view/fundamentals-of-software/9781492043447/ – Назва з екрана. – Дата звернення: 27.04.2026.
12. Falk K. Practical Recommender Systems. Manning Publications, 2019. 432 p. Manning Platforms : вебсайт. [Електронний ресурс]. – Режим доступу: https://www.manning.com/books/practical-recommender-systems – Назва з екрана. – Дата звернення: 27.04.2026.
13. McKinney W. Python for Data Analysis: Data Wrangling with pandas, NumPy, and Jupyter. 3rd ed. O'Reilly Media, 2022. 547 p. O'Reilly Learning Platforms : вебсайт. [Електронний ресурс]. – Режим доступу: https://www.oreilly.com/library/view/python-for-data/9781098104023/ – Назва з екрана. – Дата звернення: 28.04.2026.

14. Obe R., Hsu L. PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. 3rd ed. O'Reilly Media, 2017. 352 p. O'Reilly Learning Platforms : вебсайт. [Електронний ресурс]. – Режим доступу: https://www.oreilly.com/library/view/postgresql-up-and/9781491963401/ – Назва з екрана. – Дата звернення: 29.04.2026.
15. Документація Entity Framework Core. Microsoft Learn : портал для розробників. [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/ef/core/ – Назва з екрана. – Дата звернення: 10.05.2026.
16. Офіційне керівництво бібліотеки RxJS. RxJS Overview : офіційний сайт. [Електронний ресурс]. – Режим доступу: https://rxjs.dev/guide/overview – Назва з екрана. – Дата звернення: 12.05.2026.
17. Документація API сервісу TMDB. The Movie Database (TMDB) API Documentation : портал розробників. [Електронний ресурс]. – Режим доступу: https://developer.themoviedb.org/docs – Назва з екрана. – Дата звернення: 05.05.2026.
18. Документація API бази даних відеоігор RAWG. RAWG Video Games Database API : портал розробників. [Електронний ресурс]. – Режим доступу: https://rawg.io/apidocs – Назва з екрана. – Дата звернення: 06.05.2026.
19. Керівництво по Docker Compose. Docker Documentation : офіційний сайт. [Електронний ресурс]. – Режим доступу: https://docs.docker.com/compose/ – Назва з екрана. – Дата звернення: 14.05.2026.

Додаток А

Декларація про академічну доброчесність та використання ШІ

Я, Сафончик Дмитро Олександрович, здобувач НУБіП України, усвідомлюючи відповідальність за порушення академічної доброчесності, цією заявою підтверджую, що:

1. Моя бакалаврська кваліфікаційна робота на тему «Інтелектуальна веб-система персоналізованих рекомендацій розважального контенту» є результатом моєї особистої праці.

2. Усі запозичення з текстів інших авторів мають відповідні посилання згідно з чинними стандартами.

3. Щодо використання інструментів ШІ зазначаю, що:

інструменти ШІ: Copilot та Gemini були використані для:

- перекладу іншомовних джерел;
- стилістичного редагування та перевірки граматики;
- допомоги у написанні/відлагодженні програмного коду;
- стилістичне редагування авторського тексту та його приведення до норм академічного письма;
- верифікація оформлення пояснювальної записки на відповідність методичним вказівкам університету;
- підготовка перекладу анотації англійською мовою.

Я розумію, що надання недостовірної інформації може призвести до скасування результатів захисту та відрахування з числа здобувачів НУБіП України. «21» травня 2026 р. _____ Дмитро Сафончик