МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Державне некомерційне підприємство
«Державний університет» Київський авіаційний інститут»

Факультет комп'ютерних наук та технологій

Кафедра інженерії програмного забезпечення

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Олена ГРІНЕНКО
«_____» _____ 2025 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»
ЗАОЧНА ФОРМА ЗДОБУТТЯ ОСВІТИ

Тема: Методика та крос-доменне програмне забезпечення надання рекомендацій щодо медіа-контенту з механізмом адаптивного навчання

Виконавець: Мельников Костянтин Валентинович

Керівник: к. е. н. доцент Ткаченко Костянтин Олександрович

Нормоконтролер: к. е. н., доцент Ткаченко Костянтин Олександрович

Київ 2025

Державне некомерційне підприємство
«Державний університет» Київський авіаційний інститут»

Факультет комп'ютерних наук та технологій
Кафедра інженерії програмного забезпечення
Спеціальність 121 «Інженерія програмного забезпечення»
Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Олена ГРІНЕНКО

«_____» _____ 2025 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи студента
Мельникова Костянтина Валентиновича

1. Тема кваліфікаційної роботи: «Методика та крос-доменне програмне забезпечення надання рекомендацій щодо медіа-контенту з механізмом адаптивного навчання
затверджена наказом ректора від ___17.11.2025 р. №2450/ст_____

2. Термін виконання проекту: з 20.09.2025 р. по 22.12.2025 р.

3. Вихідні дані до роботи: Для експериментального дослідження та тестування розробленої методики використано відкриті набори даних, що охоплюють кілька доменів медіа-контенту.
Середовище розробки: Visual Studio 2022, Visual Studio Code.
Технологічний стек: ASP .NET Core, Angular.

4. Зміст пояснювальної записки:
   1. Аналіз предметної області та існуючих рішень.
   2. Розробка методики крос-доменного надання рекомендацій. Аналіз методів обліку енергоефективності.
   3. Архітектура та реалізація програмного забезпечення.
   4. Опис технологічного стеку, баз даних та модулів програмного засобу
   5. Реалізація прототипу програмного засобу.

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:
   1. Концептуальна схема крос-доменної взаємодії та трансферу знань
   2. Діаграма архітектури розробленого програмного забезпечення.
   3. UML-діаграма алгоритму роботи механізму адаптивного навчання

4. Функціональні можливості програмного засобу.
5. Демонстрація роботи програми.
6. Демонстрація роботи модулів програми.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.	Розробка та затвердження графіка роботи	20.09-01.10.2025	виконано
2.	Ознайомлення з постановкою задачі, вивчення інформаційних джерел та складання плану роботи.	02.10-06.10.2025	виконано
2.	Підготовка 1 розділу та подання його керівнику	07.10-18.10.2025	виконано
3.	Підготовка 2 розділу та подання його керівнику	19.10-02.11.2025	виконано
4.	Підготовка 3 розділу та подання його керівнику	03.12-15.11.2025	виконано
5.	Підготовка 4 розділу і висновків по роботі та подання їх керівнику	16.11-30.11.2025	виконано
6.	Загальне редагування пояснювальної записки, графічного матеріалу. Представлення роботи для перевірки на академічну доброчесність. Проходження нормоконтролю.	01.12-10.12.2025	виконано
7.	Отримання відгуку керівника. Підготовка презентації та тексту доповіді.	11.12-15.12.2025	виконано
8.	Попередній захист (представлення електронної версії пояснювальної записки, презентації, позитивного відгуку керівника).	08.12-15.12.2025	виконано
9.	Рецензування кваліфікаційної роботи	15.12-22.12.2025	виконано
10.	Здача секретарю ЕК пояснювальної записки: електронної версії кваліфікаційної роботи; презентації доповіді; відгуку керівника, рецензії; результату проходження перевірки на плагіат; довідки про успішність, декларації про академічну доброчесність.	15.12-22.12.2025	виконано
11.	Захист кваліфікаційної роботи перед екзаменаційною комісією	26.12.2025	

Дата видачі завдання 29.09.2025 р.

Керівник кваліфікаційної роботи:
к. е. н., доцент Костянтин ТКАЧЕНКО

Завдання прийняв до виконання: Костянтин МЕЛЬНИКОВ

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Методика та крос-доменне програмне забезпечення надання рекомендацій щодо медіа-контенту з механізмом адаптивного навчання»: 74 сторінки, 23 рисунки, 1 таблиця, 25 використаних джерел.

Об’єкт дослідження – процеси надання персоналізованих рекомендацій медіа-контенту в крос-доменних інформаційних системах.

Мета кваліфікаційної роботи – розробка методики та програмного забезпечення для надання рекомендацій щодо медіа-контенту, що забезпечує підвищення точності пропозицій та адаптацію до інтересів користувача шляхом використання механізмів адаптивного навчання та сучасних веб-технологій.

Методи дослідження – системний аналіз, математичне моделювання процесів рекомендації, методи машинного навчання (адаптивне навчання), об’єктно-орієнтоване проектування, розробка веб-додатків.

Результати роботи можуть бути використані при розробці інтелектуальних медіа-платформ, стрімінгових сервісів, онлайн-кінотеатрів; в усіх галузях, де є потреба у персоналізації контенту, підвищенні залученості користувачів та автоматизації відбору інформації на основі вподобань.

Розробка та дослідження проводилися під управлінням ОС Windows 11/Mac OS. Середовище розробки: Visual Studio 2022, Visual Studio Code. Технологічний стек: ASP .NET Core, Angular.

КРОС-ДОМЕННЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, РЕКОМЕНДАЦІЙНА СИСТЕМА, МЕДІА-КОНТЕНТ, АДАПТИВНЕ НАВЧАННЯ, ASP .NET CORE, ANGULAR, ВЕБ-ЗАСТОСУНОК

ABSTRACT

Explanatory note to the qualification work «Methodology and cross-domain software for providing recommendations on media content with an adaptive learning mechanism»: 74 pages, 23 figures, 1 table, 25 references.

The object of research is the processes of providing personalized media content recommendations in cross-domain information systems.

The purpose of the qualification work is the development of a methodology and software for providing media content recommendations, which ensures increased proposal accuracy and adaptation to user interests through the use of adaptive learning mechanisms and modern web technologies.

Research methods – system analysis, mathematical modeling of recommendation processes, machine learning methods (adaptive learning), object-oriented design, web application development.

The results of the work can be used in the development of intelligent media platforms, streaming services, online cinemas; in all sectors where there is a need for content personalization, increasing user engagement, and automating information selection based on preferences.

Development and research were carried out under OS Windows 11/Mac OS. Development environment: Visual Studio 2022, Visual Studio Code. Technology stack: ASP .NET Core, Angular.

CROSS-DOMAIN SOFTWARE, RECOMMENDATION SYSTEM, MEDIA CONTENT, ADAPTIVE LEARNING, ASP .NET CORE, ANGULAR, WEB APPLICATION

ЗМІСТ

ВСТУП

Актуальність теми. Сучасний етап розвитку інформаційного суспільства характеризується стрімким зростанням обсягів цифрового медіа-контенту. Щосекунди у світі з'являються нові фільми, музичні треки, книги, подкасти, відеоігри та інші дані. Користувачі стримінгових сервісів, онлайн-бібліотек і магазинів стикаються з проблемою інформаційного перенавантаження (information overload) – надмірна кількість доступних опцій ускладнює вибір потрібного контенту. Рекомендаційні системи (RS) стали ключовим інструментом для подолання цієї проблеми: вони фільтрують масиви даних і надають персоналізовані поради, допомагаючи користувачам швидко знайти релевантний контент. Наприклад, інтернет-магазини (Amazon, тощо) радять товари на основі попередніх купівель, кіноплатформи (Netflix) пропонують фільми згідно з переглядами, музичні сервіси (Spotify, Last.fm) підбирають нові треки з урахуванням історії прослуховувань. Усі ці системи підлаштовуються під уподобання користувача, зменшуючи ефект перенавантаження інформацією..

Особливої актуальності набуває проблема **крос-доменних рекомендацій**, коли інформація про вподобання користувача в одній сфері (наприклад, історія перегляду фільмів або відео) використовується для формування релевантних пропозицій в іншій (наприклад, вибір літератури, подкастів чи музики). Це дозволяє вирішити фундаментальну проблему «холодного старту» для нових сервісів, які ще не накопичили достатньої історії взаємодії з користувачем. Проте, більшість існуючих на ринку рішень є вузькоспеціалізованими або побудовані як монолітні системи, які важко масштабувати, підтримувати та інтегрувати в сучасне динамічне веб-середовище.

З технічної точки зору, розробка такої системи вимагає перегляду класичних підходів до веб-програмування. Традиційні багатосторінкові сайти (MPA), де кожна дія користувача призводить до перезавантаження сторінки, вже не можуть забезпечити необхідний рівень інтерактивності (User Experience) та швидкодії, до якого звикли сучасні користувачі. Назріла необхідність створення

високопродуктивних веб-додатків (Single Page Applications – SPA) на основі мікросервісної або модульної архітектури. Такий підхід дозволяє чітко розділити логіку обробки даних (Back-end) та логіку відображення (Front-end), забезпечуючи миттєвий відгук інтерфейсу та ефективну паралельну обробку запитів на сервері.

Вибір технологічного стеку для вирішення цих завдань є ключовим етапом проєктування. Використання платформи **ASP .NET Core** на стороні серверу дозволяє створювати високопродуктивні, кросплатформні API, здатні витримувати значні навантаження, що є критичним для рекомендаційних алгоритмів. У свою чергу, фреймворк **Angular** на стороні клієнта надає потужні інструменти для побудови динамічних інтерфейсів, управління станом додатку та компонентної розробки. Поєднання цих технологій дозволяє побудувати надійну систему з механізмом адаптивного навчання, яка є гнучкою до змін та легкою у підтримці.

Окремим викликом є інтеграція різнорідних даних. Коли система оперує даними з різних доменів (відео, музика, тексти), виникає проблема семантичного узгодження та нормалізації даних. Це вимагає використання суворих контрактів даних на рівні API та застосування статичної типізації, що робить зв'язку C# (на бекенді) та TypeScript (на фронтенді) безальтернативним вибором для побудови надійних ентерпрайз-рішень. Використання динамічно типізованих мов у таких системах часто призводить до помилок на етапі виконання (runtime errors), що є неприпустимим для комерційних продуктів.

Фундаментальна складність побудови крос-доменних систем полягає у проблемі «розрідженості даних» (data sparsity) та необхідності знаходження семантичної відповідності між різнорідними об’єктами. Традиційні алгоритми, такі як колаборативна фільтрація (User-based або Item-based Collaborative Filtering), демонструють високу ефективність лише за умови наявності щільної матриці взаємодій «користувач-об’єкт». Однак у крос-доменному сценарії, де перетин користувачів між різними доменами може бути мінімальним, ці методи втрачають точність.

Таким чином, тема кваліфікаційної роботи, присвячена розробці методики та програмного забезпечення для надання рекомендацій, є актуальною як з наукової

точки зору (вдосконалення методів адаптації рекомендацій та використання крос-доменних даних), так і з практичної (створення конкурентоспроможного веб-продукту з використанням передових технологій інженерії програмного забезпечення).

Мета і завдання виконання кваліфікаційної роботи. Метою роботи є розробка методики та веб-орієнтованого програмного засобу для надання високоточних крос-доменних рекомендацій медіа-контенту, що забезпечує підвищення якості обслуговування користувачів шляхом використання механізмів адаптивного навчання та побудови ефективної клієнт-серверної архітектури.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Дослідити існуючі методи та алгоритми рекомендаційних систем, зокрема підходи до крос-доменного перенесення знань.
2. Провести системний аналіз предметної області та сформулювати вимоги до програмного забезпечення.
3. Розробити методику надання рекомендацій з використанням механізму адаптивного навчання, здатного коригувати видачу контенту на основі дій користувача.
4. Спроектувати архітектуру програмного засобу, обґрунтувати вибір технологічного стеку (ASP .NET Core, Angular).
5. Розробити серверну частину (Back-end) додатки, реалізувавши RESTful API для обробки запитів та взаємодії з базою даних.
6. Розробити клієнтську частину (Front-end) у вигляді SPA-додатку (Single Page Application) для забезпечення зручного інтерфейсу користувача.
7. Провести тестування розробленого програмного засобу (модульне, інтеграційне) та оцінити ефективність запропонованих рішень і точність рекомендацій.

Об’єкт і предмет дослідження. Об’єктом дослідження є процес автоматизованого відбору та надання рекомендацій медіа-контенту в розподілених інформаційних системах. **Предметом дослідження** є методи, моделі адаптивного

навчання та програмні засоби (веб-технології ASP .NET Core, Angular), що забезпечують реалізацію крос-доменних рекомендаційних сервісів .

Методи дослідження. У роботі використано комплексний підхід, що базується на таких методах:

- *системний аналіз* – для визначення структури системи та взаємозв'язків між її компонентами;
- *математичне моделювання* – для формалізації алгоритмів розрахунку релевантності контенту;
- *об'єктно-орієнтоване проєктування та патерни розробки* – для побудови гнучкої архітектури програмного засобу;
- *методи інженерії програмного забезпечення* – при розробці клієнт-серверної взаємодії, проєктуванні баз даних та інтерфейсів користувача.

Для забезпечення якості коду та архітектурної цілісності застосовано принципи **Clean Architecture** («Чиста архітектура»), що передбачає розділення системи на незалежні шари (Presentation, Application, Domain, Infrastructure). Такий підхід робить бізнес-логіку рекомендацій незалежною від веб-фреймворків чи баз даних, що значно спрощує модульне тестування (Unit Testing) та майбутню модернізацію системи.

Для реалізації механізму рекомендацій у роботі застосовано **методи гібридної фільтрації**. Зокрема, використано векторне представлення об'єктів (Vector Space Model) для порхування подібності контенту з різних доменів на основі їх метаданих (жанри, теги, опис). Для вирішення задачі адаптивного навчання розроблено алгоритм, який враховує часовий фактор затухання інтересу (time decay factor), що дозволяє системі швидше реагувати на зміну вподобань користувача, надаючи пріоритет "свіжим" даним.

Це зумовлює необхідність розробки гібридних методик, які поєднують підходи на основі змісту (Content-Based Filtering) з методами трансферного навчання (Transfer Learning). Особливу увагу в роботі приділено проблемі динамічної адаптації ваг алгоритму: система повинна автоматично визначати, коли слід спиратися на історичні

дані з суміжного домену, а коли – на поточні дії користувача в цільовому домені. Розв’язання цієї задачі вимагає не лише математичного моделювання, а й побудови високошвидкісного програмного конвеєра для обробки потоків даних у реальному часі.

Наукова новизна отриманих результатів. У кваліфікаційній роботі отримано такі нові наукові положення:

1. Удосконалено методику надання рекомендацій шляхом інтеграції механізму адаптивного зворотного зв’язку безпосередньо у веб-інтерфейс, що, на відміну від існуючих підходів, дозволяє скоротити час "холодного старту" системи.
2. Дістало подальший розвиток застосування крос-доменного підходу в веб-архітектурі, що дозволяє агрегувати дані про поведінку користувача з різнорідних джерел в єдиному сховищі даних для підвищення точності прогнозів.

Практичне значення отриманих результатів. Практична цінність роботи полягає у створенні повнофункціонального програмного продукту, готового до впровадження.

1. Розроблено архітектуру та реалізовано програмний засіб (веб-додаток), який може бути використаний як основа для створення онлайн-кінотеатрів, музичних сервісів або агрегаторів контенту.
2. Застосування технології SPA (Angular) та високопродуктивного API (ASP .NET Core) забезпечує масштабованість системи та можливість її використання на різних пристроях (десктоп, мобільні пристрої).
3. Запропонований програмний модуль рекомендацій може бути інтегрований у вже існуючі інформаційні системи підприємств як окремий мікросервіс.

Особистий внесок здобувача вищої освіти. Кваліфікаційна робота є самостійним дослідженням автора. Усі результати, наведені в роботі, отримані особисто автором, а саме:

- проведено глибокий аналіз предметної області, існуючих аналогів та обґрунтовано вибір технологічного стеку;
- розроблено структуру бази даних, схему зв'язків сутностей та реалізовано шар доступу до даних (Data Access Layer) за допомогою Entity Framework Core;
- написано програмний код серверної частини мовою C# (контролери API, сервіси бізнес-логіки, реалізація алгоритмів рекомендацій, налаштування DI-контейнера);
- створено динамічний клієнтський інтерфейс з використанням мови TypeScript, HTML, CSS (SCSS) та компонентного підходу Angular, налаштовано маршрутизацію (Routing) та взаємодію з API;
- проведено налагодження, оптимізацію та тестування програмного продукту на реальних сценаріях використання.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Рекомендаційні системи: принципи і методи.

Поняття рекомендаційної системи. Рекомендаційна система (РС) – це програмний інструмент, що аналізує дані про користувача та його взаємодії з набором об’єктів, аби надати персоналізовані рекомендації щодо вибору тих чи інших елементів з великої колекції. Формально, функціонування РС часто описують як задачу прогнозування рейтингу: маючи часткову матрицю оцінок, система намагається оцінити корисність або цікавість тих об’єктів, які користувач ще не бачив. Рекомендація формується шляхом відбору топ-елементів з найвищим прогнозованим рейтингом для даного користувача. Таким чином, рекомендувач можна уявити як функцію, що повертає ступінь ймовірної зацікавленості користувача об’єктом; задача – знайти такі елементи, для яких результат максимізується

Основні підходи. Класичні рекомендаційні алгоритми поділяються на дві головні категорії: контентно-орієнтовані (content-based) та колаборативні (з використанням спільноти, collaborative filtering) методи

– **Content-based підхід.** При контентному рекомендуванні користувачу радять об’єкти, схожі на ті, що він вподобав раніше. Система аналізує характеристики самих об’єктів (атрибути контенту) і намагається знайти збіг між профілем інтересів користувача та описами нових кандидатів. Наприклад, якщо користувач високо оцінив декілька драматичних фільмів певного режисера, контент-орієнтована система може запропонувати інші фільми у жанрі драми або стрічки цього ж режисера. Формально будується профіль користувача – сукупність ознак контенту, що характерні для об’єктів, які користувач оцінив позитивно. Потім для кожного нового об’єкта *i* обчислюється міра схожості між профілем користувача та описом об’єкта; об’єкти з найбільшим коефіцієнтом схожості рекомендуються користувачу. Як міру близькості часто використовують косинусову схожість між векторами ознак. Базові методи контент-фільтрування успішно застосовувалися для рекомендацій текстових

елементів (документів, новин) – у таких системах кожен документ представляється набором ключових слів, а профіль користувача – сукупністю термінів, що відповідають його інтересам. П

Переваги контентного підходу полягають у тому, що він не потребує даних інших користувачів – рекомендації можна генерувати навіть в умовах повного одиночного використання (якщо наявний опис контенту). Це частково вирішує проблему холодного старту для нових об’єктів: щойно доданий в каталог фільм можна порекомендувати на основі його жанру, сюжету тощо, навіть якщо його ще ніхто не оцінював. Недоліки: (1) система схильна рекомендувати лише дуже подібні до вже відомих об’єкти, звужуючи коло інтересів користувача (“пастка фільтр-бульбашки”); (2) якість рекомендацій сильно залежить від наявності та повноти описових ознак. На практиці не завжди очевидно, які саме характеристики контенту визначають інтерес – наприклад, книга може подобатися за стиль автора, тематику чи розвиток персонажів, і витягти ці аспекти зі звичайних метаданих (жанр, автор, рік) складно. До того ж, контент-методи не здатні радити щось принципово нове поза межами профілю: якщо користувач слухає лише рок-музику, чисто контентний підхід ніколи не запропонує йому, скажімо, джаз (на відміну від колаборативного, який може це зробити, якщо інші зі схожим смаком оцінили джаз позитивно).

Collaborative Filtering (CF). Колаборативні методи ґрунтуються на аналізі поведінки спільноти користувачів. В основі лежить гіпотеза спільних смаків: якщо двоє людей мали схожі вподобання в минулому, то й надалі їх оцінки збігатимуться. Алгоритм CF намагається прогнозувати рейтинг об’єкта *i* для користувача *u* на основі взаємодій інших користувачів з цим об’єктом та подібними об’єктами. Є дві основні різновиди CF-алгоритмів: пам’яттєві (memory-based) та модельні (model-based). Memory-based (сусідський) CF безпосередньо працює з матрицею "користувач-об’єкт". Наприклад, user-based CF: для цільового користувача u шукаються інші користувачі зі схожим історичним набором оцінок ("сусіди"), після чого об’єкти, високо оцінені цими сусідами і не відомі u, рекомендуються u. В свою чергу, item-based CF робить навпаки: для кожного кандидата і визначаються схожі за патерном

оцінок об’єкти (ті, що отримували оцінки від схожих груп користувачів), і інтерес u до і оцінюється на основі оцінок u для цих подібних об’єктів. Поняття “схожості” тут може визначатися, наприклад, коефіцієнтом Пірсона між рядками або стовпцями матриці оцінок. Model-based CF підходить більш витончено: будується узагальнена математична модель вподобань (наприклад, методом матричної факторизації або нейронної мережі). Ця модель навчається на всій матриці "користувач-об’єкт" і намагається виявити приховані закономірності – наприклад, визначити латентні характеристики користувачів і об’єктів (embedding-вектори), між якими розраховується відповідність. Один з найвідоміших підходів – Singular Value Decomposition (SVD), що фактично зводиться до факторизації матриці оцінок на матриці користувацьких і об’єктних факторів певної розмірності. Компоненти цих факторів можна інтерпретувати як латентні параметри смаку (жанрові вподобання, тощо), які не задані явно, але виучуються моделлю з даних.

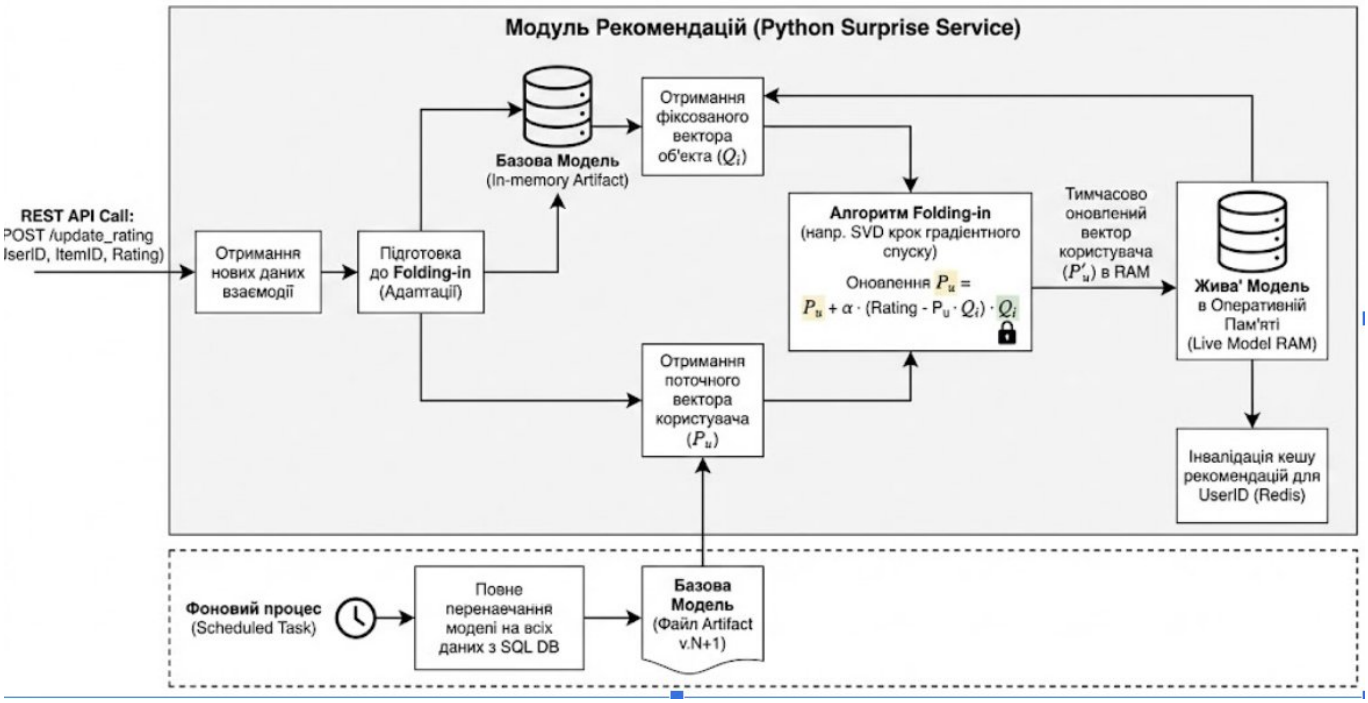

Рис 1.1 Деталізовано схема обробки оцінки в модулі Python Surpise

Переваги CF: колаборативний підхід не вимагає описів контенту – він може рекомендувати будь-які об’єкти, навіть якщо їхні атрибути не відомі системі. Це універсальний метод, здатний виявляти несподівані інтереси. Користувач може отримати пораду, яка не схожа на його попередні вибори, але яка сподобалась

багатьом “смаково подібним” людям – часто такі рекомендації бувають найбільш цінними, відкриваючи щось нове. Завдяки використанню колективного досвіду CF-алгоритми часто дають кращі результати точності, ніж прості контентні методи. Як приклад успіху: перша версія знаменитого алгоритму Netflix (Cinematch) була саме item-item колаборативною фільтрацією – і вона значно перевершувала за якістю тривіальні підходи (на кшталт середньої оцінки фільму). Недоліки: найбільша проблема – холодний старт і розрідженість даних. Якщо користувач новий (не має оцінок) – система не знає, кого взяти в сусіди; якщо об’єкт новий – він не має оцінок, тож його нікому рекомендувати. При дуже розрідженій матриці (а реальні дані такі і є) алгоритму важко знайти достатньо значимих перетинів між користувачами чи об’єктами – похибка прогнозів зростає. У великих системах холодний старт частково вирішують за допомогою початкового анкетування: наприклад, новим користувачам пропонують оцінити декілька популярних об’єктів, щоб наповнити профіль мінімально необхідною інформацією.

Гібридні системи. На практиці в сучасних рекомендаційних сервісах переважають гібридні методи, що комбінують елементи кількох базових підходів. Гібридна РС визначається як така, що використовує дві чи більше стратегій рекомендацій, аби скористатися їхніми взаємодоповнюючими сильними сторонами. Найчастіше це означає сплав контентного та колаборативного фільтрування: система одночасно аналізує і атрибути контенту, і матрицю оцінок. Можливі різні схеми гібридизації: послідовна (спочатку один метод, потім інший фільтрує результати), паралельна (обидва дають оцінки, які усереднюються або комбінуються з вагами), вбудована (єдина модель, що враховує і ті, і ті ознаки).

Рис 1.2 Схема гібридної системи рекомендацій

Доведено, що гібридні рішення здатні суттєво знизити помилку рекомендацій у порівнянні з монолітними методами, адже вони долають недоліки кожного: контентна складова дає вихід з ситуації холодного старту (через схожість по атрибутах можна рекомендувати нові об’єкти, не чекаючи оцінок), а колаборативна складова – додає диверсифікації і більш тонкої персоналізації за рахунок колективних патернів. У літературі вказується, що найчастіше гібриди якраз і створюються для вирішення проблеми cold start і data sparsity. Крім того, гібридизація відкриває шлях до врахування інших аспектів: можна включати контекст використання (місце, час), соціальні зв’язки, результати семантичного аналізу тощо, отримуючи комплексні моделі. Сучасні технологічні гіганти (Netflix, Amazon, YouTube) застосовують складні багатокомпонентні алгоритми, які фактично є гібридними системами, зокрема включають елементи глибинного навчання (нейромережі для аналізу контенту, секвенційні моделі для врахування порядку перегляду і т.д.).

Отже, успішна рекомендаційна система сьогодні – майже завжди багатокомпонентна. У нашому проекті, зокрема, закладається гібридність на кількох рівнях: поєднання колаборативного і контентного підходів для пошуку схожих

об’єктів; об’єднання даних з різних доменів (крос-доменний аспект); інтеграція алгоритмічних рекомендацій з соціальними механізмами (можливість користувачам переглядати профілі одне одного, стежити за оцінками друзів тощо). Така *багаторівнева гібридизація* покликана забезпечити універсальність і високу адаптивність системи під різні сценарії використання.

1.2 Крос-доменні рекомендаційні системи

Поняття та приклади. Крос-доменна рекомендаційна система (Cross-Domain RS) – це різновид гібридної системи, яка використовує інформацію з декількох віддалених доменів (тематик або типів продуктів), щоб покращити якість рекомендацій в одному або в усіх цих доменах. Інакше кажучи, система залучає дані щонайменше з двох різнорідних джерел для побудови моделі і генерації рекомендацій. Мета – здійснити трансфер знань: навчитися на вподобаннях користувача в одному контексті і застосувати це знання для передбачення його вподобань в іншому контексті. Такий перенос може бути взаємним (коли система одночасно покращує рекомендації в обох доменах, користуючись загальними даними), або односпрямованим (коли є окремий *source domain* з багатою інформацією і *target domain*, де інформації бракує і куди ми “перекидаємо” знання). Крос-доменні рекомендації набули популярності як природне рішення проблеми холодного старту і розрідженості даних: якщо в цільовому домені даних мало, можна скористатися суміжним доменом, де їх більше. Окрім того, це відповідає прагненню давати користувачу різнопланові поради – наприклад, не тільки фільми, а й книги чи музику, виходячи з його інтересів.

Яскравий приклад сценарію: користувач має багату історію переглядів фільмів і серіалів на кіно-платформі, а тепер хоче отримати рекомендації щодо книг. Якщо система знає, які жанри та сюжети улюблені (на основі кіновподобань), вона може знайти аналогії у книжковому домені – скажімо, порадити роман, за яким знято улюблений фільм, або книгу у жанрі, співзвучному до смаків користувача у кіно. Таким чином, крос-доменний рекомендувач розширює горизонт пошуку релевантного контенту за межі одного типу медіа і навіть може несподівано

поєднувати далекі на перший погляд інтереси. Дослідження показують, що це не лише підвищує точність та корисність рекомендацій, але й збільшує загальну активність користувачів на платформі – люди частіше повертаються, якщо сервіс пропонує ширший спектр контенту, який подобається.

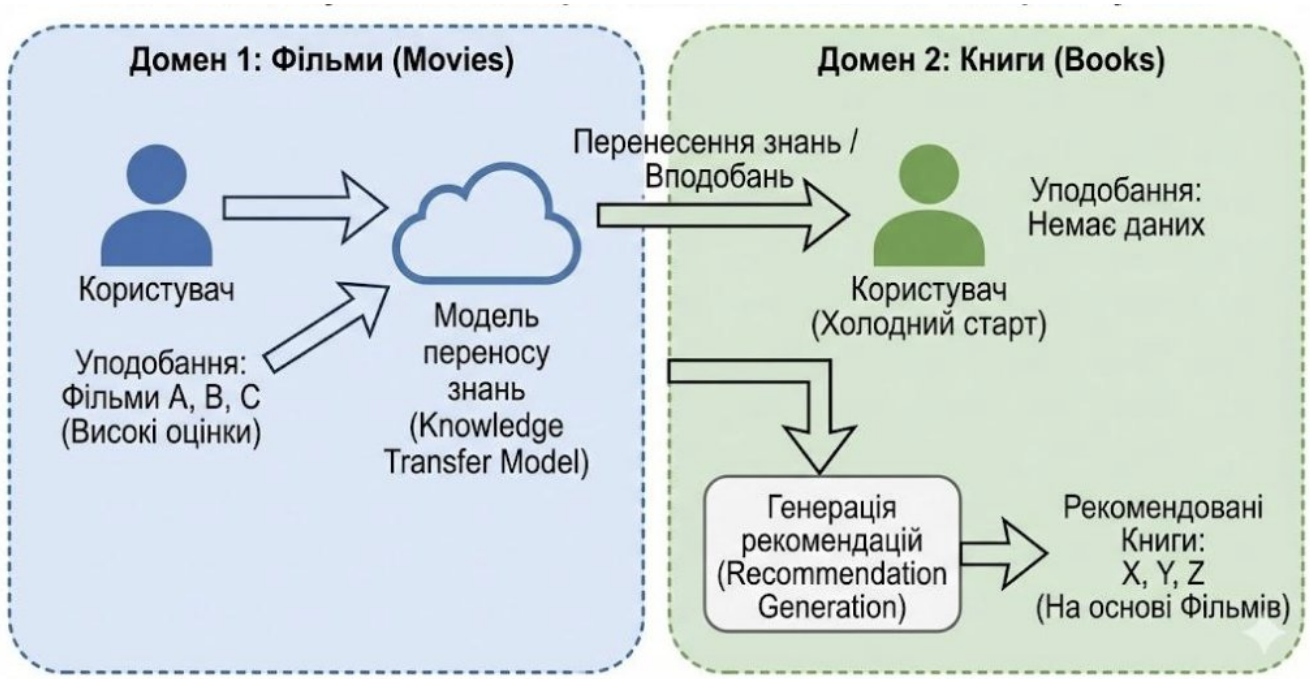

Рис 1.3 Концептуальна схема крос-доменної взаємодії на переносу знань

Задачі та виклики. Головний виклик при побудові CDR-систем – встановлення відповідностей між доменами. На відміну від одноманітних даних в однодоменних рекомендаціях, тут потрібно “з’єднати різні світи”, знайти точки дотику між, скажімо, оцінками фільмів і оцінками музики. Можна виділити кілька сценаріїв інтеграції даних.

Спільні користувачі або спільні об’єкти. Найпростіший випадок – коли домени мають очевидний перетин. Наприклад, якщо сервіс об’єднаний (як Amazon, де під одним акаунтом є покупки книжок, електроніки, одягу тощо) – кожен користувач має діяльність у різних доменах, що зберігається в загальному профілі. Тоді рекомендаційна модель може просто оперувати цим об’єднаним профілем. Інша ситуація: спільні об’єкти – тобто той самий товар належить до двох доменів. Приміром, музичний трек може фігурувати і на музичній платформі, і як саундтрек на кіно-сайті; книга може мати екранізацію, представлену у кіно-домені. У таких

випадках можливо безпосередньо зіставити вподобання: якщо користувач оцінив фільм система знайде відповідну книгу (роман, за яким знято фільм) і порекомендує її, або навпаки. Наявність спільних точок даних значно спрощує побудову CDR, оскільки можна напряму **об'єднати матриці взаємодій** або перенавчити модель на злитому датасеті, де спільні сутності слугують "мостами" між доменами.

Загальні ознаки контенту. Якщо явного перекриття у вигляді тих самих користувачів чи продуктів немає, домени все одно можуть мати приховані зв'язки через схожість атрибутів. Наприклад, книга і фільм можуть належати до одного жанру, або музичний альбом і відеогра – мати схожий настрій чи тематику. Використовуючи методи обробки тексту (аналіз описів, рецензій) та зображень (обкладинки, постери), система може виявити семантичні відповідності між об'єктами різних доменів. Сучасні глибинні моделі дозволяють зіставляти елементи різних типів у єдиному латентному просторі ознак. Наприклад, перетворивши опис фільму і книги у вектори за допомогою контекстних моделей на кшталт BERT, ми можемо обчислити їх близькість: якщо вектори близькі, контент схожий, і навпаки. Таким чином, навіть без спільних користувачів можна побудувати крос-медійні рекомендації – радити фільм шанувальнику книги або музику шанувальнику фільму за тематичною схожістю контенту. Цей підхід спирається на контент-орієнтоване ядро і тому вимагає гарно структурованих метаданих або потужних NLP/комп'ютерного зору алгоритмів для витягування семантики.

Мапування факторів і спільне навчання. Ще один клас методів – transfer learning між моделями різних доменів. Ідея така: спочатку будується окрема модель для кожного домену (наприклад, факторизаційна модель вподобань у кожному з сайтів). Потім на рівні латентних представлень шукається відповідність між профілями того ж користувача у різних доменах. Якщо домени корельовані, повинна існувати певна функція перетворення (наприклад, лінійна трансформація або навіть простий зв'язок типу "любов до фантастики у фільмах означає любов до фантастики у книгах"). Метод mapping передбачає знаходження такої функції. Натренувавши на користувачах, що мають історію в обох доменах, далі можна для нового користувача, який активний лише у фільмах, прогнозувати і на її основі рекомендувати книги.

Розвитком цієї ідеї є об’єднане навчання (joint learning): замість двох роздільних моделей будується одна спільна модель, яка одночасно вчиться на даних двох (або більше) доменів. Наприклад, можна скласти об’єднаний граф “користувач–об’єкт”, що включає різні типи об’єктів (вузли “фільм”, “книга”, “музика”), і застосувати графові алгоритми (Graph Neural Networks) для навчання єдиної багатодоменної моделі. Така модель від самого початку враховує мультидоменні взаємозв’язки і може генерувати рекомендації у всіх доменах, будучи по суті універсальним рекомендувачем.,

Зазначені підходи мають і складнощі. При mapping-стратегії важливо уникнути negative transfer – коли додавання даних іншого домену не покращує, а погіршує результати через “шум” чи суперечливі вподобання різних аудиторій. При joint learning значний виклик – обсяг даних і ресурсів, потрібний для навчання універсальної моделі: фактично доводиться обробляти комбінацію кількох датасетів одразу, що складніше, ніж в ізольованому випадку. Тому реальні реалізації часто комбінують: частина параметрів моделі розділяється між доменами, а частина – лишається специфічною для кожного (multi-task learning з спільними і окремими шарами). Так забезпечується баланс – домени обмінюються інформацією, але до певної межі, щоб не “зашкодити” один одному.

Стан досліджень. Крос-доменні рекомендації – відносно новий та дуже активний напрям досліджень. Перші роботи з’явились на початку 2010-х, але особливої популярності тема набула в останні роки, про що свідчить поява кількох всеосяжних оглядів і таксономій методів CDR. Вже існують десятки алгоритмів, що демонструють переваги крос-доменного підходу над ізольованими: в середньому помилка прогнозу рейтингу (RMSE або MAE) знижується, особливо для “важких” випадків – нових користувачів чи рідкісних об’єктів. З іншого боку, залишається багато відкритих питань – як краще поєднувати різні типи даних, як вимірювати успіх (точність vs. новизна vs. диверсифікація), як врахувати еволюцію вподобань у часі тощо.

Для нашої роботи – створення соціальної платформи з чотирма доменами контенту (фільми, книги, музика, ігри) – найбільш придатним видається підхід

уніфікованого рекомендувача: єдина система, що агрегує дані про поведінку користувача в різних сферах і на їх основі формує рекомендації в будь-якому з цих доменів. По суті, ми прагнемо побудувати спільний профіль користувача і спільну модель для усього розмаїття медіа. Це завдання амбітне, адже вимагає поєднати кілька підходів (і колаборативний, і контентний, і трансферне навчання) та вирішити згадані вище виклики відповідності. Однак успішні приклади (такі як TasteDive) і теоретичні напрацювання підтверджують здійсненність такої ідеї. Зокрема, TasteDive вже зараз надає прості крос-медійні рекомендації на основі улюблених елементів, а великі компанії (Amazon) внутрішньо застосовують крос-доменні алгоритми для "перехресного" продажу товарів. Відсутній елемент, який ми плануємо додати – це адаптивність в реальному часі, про що докладніше далі.

1.3 Прогнозування оцінок та адаптивне навчання моделі

Прогнозування рейтингу. Як зазначалось, одне з базових завдань рекомендателів – оцінити, яку оцінку користувач поставив би об'єкту, тобто спрогнозувати невідомий елемент матриці. Відомим історичним кейсом є конкурс Netflix Prize (2006–2009) – змагання з покращення алгоритму Cinematch компанії Netflix. Метою було досягти підвищення точності прогнозу рейтингів мінімум на 10% (у метриці RMSE) порівняно з Cinematch, початковою системою Netflix. Цей конкурс привернув величезну увагу науковців і практиків, ставши каталізатором для розвитку досконаліших методів машинного навчання в рекомендаціях. Переможці Netflix Prize зрештою досягли ~10.06% покращення, використавши ансамбль з десятків моделей (багато з яких були варіаціями матричної факторизації) – цим було доведено, що алгоритми машинного навчання можуть суттєво перевершити прості евристики у задачі рейтингового передбачення.

Сьогодні практично всі великі сервіси мають під капотом компонент, який оцінює релевантність пари "користувач-об'єкт" (в явному вигляді або непрямо). На основі цих внутрішніх прогнозів формується список рекомендацій – об'єкти з найвищим очікуваним рейтингом пропонуються користувачу. У нашій системі цей принцип використовується в інтерактивній формі: модель не лише розраховує

наперед очікувану оцінку, а й звіряє свій прогноз з реальністю щоразу, коли користувач фактично виставляє оцінку твору. Після кожної нової оцінки ми оновлюємо модель – тим самим забезпечуємо адаптивне навчання на потоці даних.

Адаптивне (онлайн) навчання. Під адаптивністю в рекомендаційних системах розуміють здатність моделі динамічно перебудовуватися при надходженні нової інформації, зберігаючи актуальність рекомендацій у режимі, близькому до реального часу. Традиційно рекомендаційні моделі оновлювали через перевчання на всьому датасеті з певною періодичністю (наприклад, раз на день). Натомість адаптивний підхід передбачає безперервне донавчання: кожен новий зворотний зв'язок (оцінка, клік, перегляд) використовується для миттєвої корекції моделі. Це нагадує навчання з підкріпленням – модель отримує сигнал помилки (різницю між прогнозованим рейтингом та фактично виставленим користувачем) і на його основі коригує свої параметри.

Переваги онлайн-навчання: воно дозволяє системі миттєво реагувати на зміну смаків користувача. Якщо користувач раптом почав цікавитися новим жанром, адаптивна модель тут же це підхопить і внесе корективи у майбутні рекомендації. Якщо ж користувач спростував очікування системи (наприклад, система думала, що контент сподобається, а користувач поставив низьку оцінку) – модель зможе уточнити профіль, аби надалі врахувати цю аномалію. За рахунок цього рекомендації еволюціонують разом із користувачем, а не застають його "вчорашнього" образу. Практичний приклад – Netflix відкрито заявляє, що їхні алгоритми персоналізації зараз працюють у майже реальному часі: довготермінове пакетне тренування на всіх даних поєднується з миттєвим оновленням моделей на основі нових дій користувача. Так, додав користувач фільм у список улюблених – вже через кілька секунд стрічка рекомендацій підлаштується, показуючи більше подібних фільмів. Почав переглядати/слухати щось нестандартне – система оперативно підвищить вагу цього напрямку у видачі контенту.

Інженерно реалізувати online learning допомагають сховища ознак (feature store) та стрімінгові платформи обробки подій (наприклад, Kafka, Spark Streaming): вони забезпечують безперервний потік оновлень моделі з мінімальною затримкою. Проте

у нашому прототипі, що працює з відносно невеликими даними, досить і прямої реалізації на боці серверу (наприклад, метод оновлення вектора, викликаний з API при надходженні оцінки)

Проблема стабільності-пластичності. Потрібно зауважити, що надто агресивне онлайн-навчання може призвести до забування раніше накопиченого досвіду. Існує дилема: модель має бути достатньо пластичною, щоб підлаштуватися під нові дані, але водночас досить стабільною, щоб не втратити старі знання при надходженні свіжої інформації. Якщо оновлення робити великими кроками, модель може "перекоситися" під останні дії і ігнорувати попередню історію користувача. Якщо ж навпаки занадто обережно оновлювати, то адаптивність зникає. Ця stability-vs-plasticity дилема є предметом активних досліджень у сфері continual learning (постійного навчання моделей). Одне з рішень – поступове згасання ваги старих даних: наприклад, вводять фактор часової давності (time decay), що зменшує вплив дуже старих оцінок у моделі. Інше рішення – регуляризація при оновленні, яка не дає параметрам надто відхилятися від попередніх значень. Також практикують техніки на кшталт обмеження діапазону оновлення, відкладених оновлень або оновлення на ковзному часовому вікні даних, щоб зберегти баланс.

У нашій системі передбачено, що адаптивний модуль навчання вноситиме зміни пропорційно "сюрпризу": якщо різниця між прогнозом та реальною оцінкою невелика – модель уточнює профіль користувача ледь помітно; якщо ж система сильно помилилася (користувачу сподобалось те, що модель вважала нецікавим, або навпаки) – це сигнал, що користувачеві імпонує щось несподіване, тож варто суттєвіше змінити його "вектор смаку". Таким чином, з кожною новою оцінкою рекомендації стають все більш точними для конкретного користувача, але при цьому накопичений раніше досвід теж не забувається миттєво. Фактично, це реалізація простого принципу персоналізованого активного навчання: користувач ніби "навчає" систему своїми діями, а система уважно стежить і підлаштовується під них.

Врахування адаптивності в сучасних сервісах. Як зазначалось, Netflix та інші великі платформи вже впровадили реальні системи, які комбінують пакетне і онлайн-навчання. Зокрема, Netflix використовує підхід contextual bandits (контекстні

багатовибіркові бандити) для тонкого реагування на фідбек в реальному часі – по суті, це варіація навчання з підкріпленням, де алгоритм постійно уточнює стратегію рекомендацій, намагаючись балансувати між експлуатацією відомих уподобань і дослідженням нових гіпотез щодо інтересів користувача. Іншим прикладом є соціальні мережі на зразок Facebook/Instagram, де стрічка новин переупорядковується "на льоту" залежно від того, на які пости користувач відреагував останнім часом. У сфері рекомендацій музики Spotify також інвестує у real-time personalization – їхні моделі оновлюють рекомендації після кожної прослуханої пісні, щоби наступний трек був максимально доречним за настроєм моменту.

Рис 1.4 Концептуальна схема циклу адаптивного навчання в рекомендаційній системі

Отже, адаптивне навчання стало невід'ємною вимогою до сучасних рекомендаційних систем. Наш проект із самого початку закладає такий механізм як центральний: рекомендаційна модель еволюціонує разом із користувачем у процесі взаємодії. Це суттєво підвищує цінність системи, адже користувач відчуває, що рекомендації "підлаштовуються" під нього, а не застарівають. Наступним кроком є розгляд прикладів існуючих платформ, які вирішують схожі задачі, аби переконатися у доцільності обраного підходу.

1.4 Існуючі рішення та аналоги

Розглянемо наявні платформи та сервіси, що частково або повністю вирішують схожу задачу – рекомендацію медіа-контенту, особливо з соціальними елементами та мульти-доменним охопленням. Аналіз аналогів дозволить зрозуміти, які функції вже реалізовані в індустрії, а які є новаторськими.

1. Доменно-специфічні соціальні платформи з рекомендаціями. Більшість популярних сервісів зосереджені на одному типі контенту, але напрацювали багатий досвід у впровадженні рекомендаційних алгоритмів та соціальної взаємодії користувачів.

Goodreads – найбільша соціальна мережа для любителів читання (запущена у 2007, нині належить Amazon). Дозволяє користувачам обліковувати прочитані книги, виставляти їм оцінки та рецензії, отримувати персональні рекомендації книг, а також спілкуватися у групах за інтересами. Goodreads у 2011 році придбав стартап Discovereads – алгоритмічний книжковий рекомендувач на основі machine learning, і інтегрував його у свою платформу[70]. Рекомендаційний двигун Goodreads аналізує патерни оцінок мільйонів користувачів, щоб знаходити книги, які можуть сподобатися конкретному читачеві на основі того, що сподобалось іншим зі схожим читацьким смаком. Як і більшість подібних сервісів, Goodreads комбінує колаборативний підхід (пошук “схожих” читачів та їхніх улюблених книг) з елементами контентного фільтру (врахування жанрів, тематик тощо). Цікаво, що платформа стимулює новачків: без оцінювання книжок користувач майже не отримує рекомендацій, тому при реєстрації йому пропонується поставити оцінки принаймні ~20 книгам, аби запустити алгоритм. Це стандартний підхід для вирішення cold start: коротке первинне опитування суттєво підвищує якість подальших рекомендацій, і наш проєкт теж передбачає такий механізм onboarding (користувачу пропонується відмітити декілька улюблених творів на старті)

Last.fm – музична платформа (заснована 2002), відома системою scrobbling – автоматичного збору даних про прослуховування користувача. Last.fm має елементи соціальної мережі (профілі, друзі, групи) і власний музичний рекомендаційний сервіс. Алгоритми Last.fm здебільшого колаборативні: аналізується перетин прослуханої

музики між користувачами, щоб знаходити “музичних двійників” – людей зі схожим смаковим профілем, і на основі їх бібліотек генерувати рекомендації нових артистів тому користувачу. Також Last.fm вводить цікавий соціальний показник – taste-o-meter (індекс сумісності смаків) між будь-якими двома користувачами. Це гейміфікує процес: користувачі можуть порівнювати свої вподобання, шукати однодумців, що в нашій системі теж планується (відкриті профілі + індикатор схожості). Last.fm при цьому залишається моно-доменним – він не радить нічого, окрім музики. Але його методи цілком можна узагальнити на інші типи медіа, якби була спільна матриця “user-item” для різних видів контенту. Саме так ми і зробимо: наша система збиратиме взаємодії у чотирьох доменах під єдиним акаунтом, що дозволить застосувати колаборативний алгоритм на об’єднаних даних.

IMDb / Letterboxd – платформи для кіноманів. IMDb (Internet Movie Database) – більше енциклопедія кіно, але теж має рейтинги і простий рекомендаційний функціонал (“Users who liked this also liked...”). Letterboxd (запущена 2011) – соціальна мережа для кінолюбителів, де можна вести щоденник переглядів, писати огляди, фоловити інших, переглядати їх списки улюблених фільмів. Обидва сервіси радше покладаються на соціальне відкриття контенту, ніж на складні алгоритми: користувач може бачити, що дивляться друзі чи автори, на яких він підписаний, і таким чином знаходити нові фільми. Це підтверджує, що людський фактор (друзів, експертів) теж важливий у рекомендаціях – часто довіра до смаку іншої людини може дати не гірший результат, ніж рекомендація від машини. Наш проект поєднує обидва підходи: алгоритм фільтрує величезний пул контенту, але ми також надаємо можливість вручну досліджувати профілі інших користувачів, бачити хто що оцінив, і, можливо, переймати їх досвід (особливо якщо їхній смак близький до нашого).

До найвідоміших представників доменно-специфічних платформ належать сервіси Netflix, Spotify, Goodreads, Steam та інші системи, що надають рекомендації лише в межах одного типу контенту. Для подальшого аналізу та обґрунтування вибору підходів до розроблення крос-доменної рекомендаційної системи в (табл. 1.1) узагальнено основні характеристики зазначених платформ: їх домен, застосовувані методи рекомендацій, а також ключові переваги й недоліки.

Таблиця 1.1

Основні характеристики доменно-специфічних платформ

Система / підхід	Домен	Основний метод	Переваги
Netflix	Фільми, серіали	Висока якість персоналізації, велика база користувачів і контенту	Висока якість персоналізації, велика база користувачів і контенту
Spotify	Музика	Колаборативна + контентна	Добре враховує схожість треків і поведінку користувачів
Goodreads	Книги	Колаборативна фільтрація	Орієнтація на книжковий контент, активна спільнота
Steam	Відеоігри	Колаборативна фільтрація	Добре працює з іграми, враховує час у грі, відгуки
MediaRecommender (розроблювана система)	Фільми, музика, ігри, книги	Крос-доменний гібридний підхід з адаптивним навчанням	Можливість переносити знання між доменами, підвищення якості рекомендацій для «слабких» доменів

2. Багатодоменні рекомендаційні платформи. Сервісів, які охоплюють кілька типів медіа одночасно, наразі небагато, але вони є і показують перспективність такого підходу:

TasteDive (раніше TasteKid) – мабуть, найближчий за концепцією до нашого задуму існуючий сервіс. Позиціонується як “entertainment recommendation engine” для фільмів, серіалів, музики, відеоігор, книг і навіть деяких інших категорій (персоналії, бренди). TasteDive працює так: користувач вводить назву твору, який йому подобається, а система видає список схожих за стилем/настроєм творів, можливо і з інших категорій (напр. на основі улюбленого фільму порадить схожий серіал, або на основі музичних вподобань – схожий фільм-концерт). Алгоритми TasteDive враховують як явні улюблені елементи, додані користувачем до профілю, так і неявний фідбек (пошукові запити, позначки “подобається/не подобається” у виданих

рекомендаціях), і постійно навчаються: чим більше людина взаємодіє – тим точнішими стають її рекомендації. Є і соціальна складова: можна підписуватися на інших користувачів, бачити їх активність, а також враховується глобальна популярність – видно, скільком іншим користувачам сподобалась та чи інша рекомендація (це свого роду колективний голос). TasteDive фактично підтверджує життєздатність ідеї гібридного крос-доменного рекомендувача: він комбінує дані різних медіа-доменів і поведінку спільноти, щоб видавати цілісні поради, і має певний комерційний успіх (у 2019 поглинутий компанією Qloo – постачальником AI-рішень для сфери розваг). Наша система розвиває концепцію TasteDive, додаючи акцент на адаптивності: якщо TasteDive формує рекомендації переважно на основі статичного списку вподобань і глобальних трендів, то наша платформа активно перевіряє себе на кожному кроці і оновлює модель після кожної оцінки. Очікується, що це дасть більш персоналізований і "підлаштований під користувача" результат.

Amazon (екосистема) – хоча це не публічна соцмережа, але варто згадати як приклад мульти-доменних рекомендацій у електронній комерції. Amazon володіє величезними даними про купівельну поведінку користувачів у різних категоріях товарів під єдиним акаунтом (книги, електроніка, одяг тощо). Вони широко застосовують перехресні рекомендації: наприклад, радять електронний гаджет виходячи з придбаних раніше книг (якщо це тематично пов'язані товари). Фактично формується універсальний "вектор" користувача, що оновлюється після кожної покупки чи перегляду, і на його основі генеруються рекомендації у будь-якій категорії. Це дуже близько за духом до нашого завдання, лише замість товарів у нас культурні продукти. Технічно Amazon одним із перших впровадив item-item collaborative filtering для рекомендацій ("Customers who bought X also bought Y"), але з роками їх моделі еволюціонували до глибоких нейронних мереж, що враховують контекст, послідовність дій і багато інших факторів. Важливо, що Amazon та подібні гіганти мають ресурси для автоматичного масштабування: якщо даних стає більше, вони можуть додати обчислювальних потужностей, розподілити модель, використати хмарні сервіси машинного навчання тощо. У нашому проекті, звісно, все скромніше, але архітектурно ми теж закладаємо принцип масштабованості (через розділення на

сервіси і використання хмарно-орієнтованих технологій .NET/Angular).

3. Комплексне порівняння та висновки з огляду аналогів. Проведений аналіз демонструє, що окремі компоненти запланованої системи успішно застосовуються на практиці різними сервісами: - Колаборативні алгоритми – ядро рекомендацій Last.fm, Goodreads, Netflix та багато інших (вони довели ефективність CF для пошуку релевантного контенту на основі колективних уподобань). - Гібридні підходи з контент-аналізом – Netflix, Amazon активно комбінують фільтрацію з урахуванням контентних ознак (метаданих, текстових описів, зображень) для кращої точності. - Крос-доменні зв'язки – TasteDive успішно показує, що можна поєднати одразу кілька типів медіа і отримати зацікавлену аудиторію. - Соціальна взаємодія – Goodreads, Letterboxd та інші соціальні платформи підтверджують цінність можливості користувачам обмінюватися рекомендаціями, переглядати профілі одне одного. - Адаптивне онлайн-навчання – Netflix, Amazon та інші технологічні лідери вже реалізували постійне оновлення моделей, що значно покращує користувацький досвід персоналізації. Проте комплексне поєднання всього цього в одному рішенні є новаторським. Наскільки нам відомо, на момент підготовки роботи не існує загальнодоступного соціального сервісу, який би одночасно: 1. Підтримував чотири різні домени медіаконтенту (книги, фільми, музика, ігри – як у нашому проекті). 2. Надавав рекомендації на основі об'єднаного профілю користувача, що враховує всі ці сфери одразу. 3. Динамічно навчався від безперервного потоку оцінок (адаптивна модель реального часу). 4. Мав ще й соціальну складову (відкриті профілі, взаємодія між користувачами навколо рекомендацій). Ці чотири пункти – наші цілі, і їх досягнення означатиме створення платформи нового покоління, своєрідного "universal recommender". Наукова новизна проекту полягає саме в такій інтеграції методів: ми узагальнюємо і розширюємо існуючі ідеї. По суті, будується крос-доменна гібридна рекомендаційна система з адаптивним навчанням, що охоплює одразу кілька доменів знань і може змінюватися "на льоту" під користувача.

З точки зору інженерії ПЗ, обрана технологія (.NET Core + Angular) повністю відповідає потребам: .NET Core славиться високою продуктивністю та надійністю у веб-сервісах, а Angular – чудовий інструмент для побудови інтерактивного SPA-

інтерфейсу. Така архітектура також легко розширюється: при необхідності можна розробити мобільні додатки або додаткові сервіси, використовуючи той самий бекенд через API. Наприклад, у перспективі можна створити нативні iOS/Android додатки або крос-платформний клієнт (на базі React Native чи .NET MAUI), щоб користувачі мали доступ до платформи з будь-яких пристроїв.

Підсумовуючи огляд, переконуємось у актуальності і практичній значущості нашого проекту. Він знаходиться на перетині кількох передових напрямів: рекомендаційні алгоритми, що поєднують різні джерела даних (крос-доменні та гібридні моделі), а також алгоритми, що працюють у режимі постійного навчання від потоку зворотного зв’язку (адаптивні, real-time). Така синергія покликана вирішити одночасно комплекс проблем: подолання cold start та розрідженості (через перенос інформації між доменами), підвищення точності та новизни рекомендацій (через комбінування методів), підтримка актуальності порад (через безперервне навчання), і нарешті соціальна прозорість, що стимулює інтерес користувачів. Усе це створює міцне підґрунтя для переходу до етапу проектування системи.

РОЗДІЛ 2
МЕТОДИКА КРОС-ДОМЕННОГО РЕКОМЕНДУВАННЯ

У цьому розділі представлено розроблену методику надання рекомендацій, яка лежить в основі програмного модуля нашої системи. Методика враховує специфіку крос-доменних даних та передбачає механізм адаптивного навчання. Спершу опишемо концептуальну модель користувацького профілю та представлення контенту, потім – алгоритм генерації рекомендацій і його поступового оновлення.

2.1 Об’єднаний профіль користувача та багатодоменне представлення контенту.

Об’єднаний простір користувача. Ключова ідея – побудувати для кожного користувача єдиний профіль вподобань, що охоплює всі типи контенту. Замість окремого профілю на фільми, окремого на книги тощо – ми створюємо інтегрований вектор характеристик користувача, в якому відображені його інтереси у різних доменах. Цей вектор можна уявити як сукупність латентних факторів (жанрових, тематичних, стилістичних), частина з яких може переважно відповідати одному домену, частина – іншому, але вони спільно описують смаковий "відбиток" особистості.

Практично реалізувати такий профіль можна кількома способами. Ми обрали підхід на основі матричної факторизації: розглядаємо об’єкти всіх доменів разом і будуємо спільну матрицю взаємодій. Для цього кожному об’єкту призначаємо унікальний ідентифікатор у загальному просторі. Наприклад, ID книг 1–10000, фільмів 10001–20000, музики 20001–30000 і т.д. Таким чином, матриця оцінок . Оцінки бінарні "вподобано/не вподобано") Далі застосовуємо факторизаційний алгоритм (наприклад, SVD) до цієї матриці. В результаті отримуємо матрицю користувацьких факторів Він визначає вподобання користувача у латентному просторі, спільному для всіх об’єктів. Латентний вектор об’єкта (будь то фільм чи книга) – рядок у матриці .

Відповідно, передбачена оцінка користувача для об’єкта обчислюється як скалярний добуток. Особливість у тому, що наш латентний простір *змішаний*: наприклад, одна з координат може відповідати “любов до драми”, яка впливає і на фільми, і на книги; інша координата – “цікавість до мультиплікації”, яка релевантна хіба для фільмів; ще інша – “прихильність до рок-музики”, значуща для музичних об’єктів і т.д. Факторизація сама визначить ці компоненти з даних – ми не прив’язуємо їх до доменів вручну. Однак щоб такий підхід спрацював, потрібні початкові дані з перетином між доменами – хоча б декілька користувачів, які мають оцінки в більш ніж одному домені, інакше модель розпадеться на ізольовані кластери. Ми передбачаємо вирішення холодного старту користувача через анкету: новий користувач оцінює хоча б по кілька об’єктів у кожній категорії, щоб задати мінімальний "відбиток" в кожному домені. Це забезпечить необхідний зв’язок між доменами на рівні латентних факторів.

Рис 2.1 Схематичне опис крос-доменної Матичної Факторизації

Характеристики контенту. Додатково до колаборативних даних, ми використовуємо контентні описи об’єктів для підсилення моделі. Кожен об’єкт (фільм, книга, музичний альбом, гра) має ряд атрибутів: жанри, теги,

автори/виконавці, рік тощо. Ми формуємо для об’єкта вектор контентних ознак (бінарний або ваговий), який відображає, які з загального списку атрибутів притаманні цьому об’єкту. Наприклад, для фільму – жанри: комедія=1, драма=1, інші жанри=0; для книги – жанри літератури, теми, і т.п. Ці ознаки можуть перетинатися: скажімо, жанр "фентезі" є і для фільму, і для книги, і для гри; тож об’єкти різних типів можуть мати спільні компоненти. Це важливо: спільні контентні ознаки створюють зв’язок між доменами (книги і фільми можуть зустрічатися у одному жанровому просторі). На основі векторів ми можемо оцінювати контентну схожість між об’єктами різних доменів. Якщо користувачеві дуже подобаються певні ознаки (визначено через історію оцінок), то можна рекомендувати об’єкти з тими ж ознаками навіть без колаборативного сигналу. Для цього інтегруємо контентний компонент до моделі: будуємо окремий профіль користувача – наприклад, усереднений або зважений вектор для об’єктів, яким користувач поставив високі оцінки. Потім при рекомендації об’єктів можемо комбінувати колаборативний скор з контентним скором. По суті, це реалізація гібридної моделі типу “колаборативна + контентна”. Параметр може динамічно змінюватися залежно від ситуації.

2.2 Алгоритми рекомендацій та адаптивне оновлення

Базовий алгоритм рекомендації. На старті (до отримання будь-яких оцінок від користувача) система, спираючись на попередньо натреновану модель факторизації, може видавати рекомендації на основі демографічно схожих користувачів чи глобально популярного контенту – це стандартний режим для cold start (ми його підсилюємо, задавши початкові вподобання через анкетування). Далі, коли користувач починає оцінювати об’єкти, ми діємо ітеративно:

- **Вибір рекомендацій.** Сортуємо кандидати за спаданням і вибираємо топ. Враховуємо також деякі business rules: наприклад, прагнемо, щоб у топ-10 були представлені різні домени (не всі 10 – книги, а хоча б по кілька з різних категорій, якщо це можливо), щоб рекомендації не були занадто одноманітні. Для цього можна застосувати rerank за диверсифікацією: перемішати список, щоб чергувалися типи контенту.

- **Оновлення після фідбеку.** Коли користувач обирає якийсь рекомендований об’єкт і виставляє йому оцінку, ми отримуємо нову пару. Цю оцінку додаємо до бази. Далі оновлюємо профіль. Також можемо налаштувати якщо користувач почав активно оцінювати, збільшити роль колаборативного компонента (з часом модель CF стане точнішою, і їй можна більше “довіряти”).

- **Повтор рекомендацій.** Після оновлення профілю перераховуємо рекомендації. Таким чином, нові рекомендації вже врахують щойно виставлену оцінку. Цикл повторюється: користувач взаємодіє – модель оновлюється – список рекомендацій змінюється.

Отже, алгоритм самокоригується пропорційно своїй помилці (“навчання на помилках”). Це відповідає ідеї active learning: система активно намагається отримати підтвердження/спростування своїх гіпотез. У подальшому ми можемо навіть запитувати користувача напряму про певний об’єкт, у оцінці якого модель не впевнена – але поки що у нашому прототипі немає прямого опитування, тільки неявний збір оцінок, які користувач сам вирішив виставити.

Динамічне балансування між доменами. Цікавим аспектом є визначення, коли покладатися на крос-доменні дані, а коли – на within-domain. Якщо у користувача вже достатньо взаємодій у певному домені, то для рекомендацій у ньому можна використовувати здебільшого ці ж дані (наприклад, якщо він прочитав 100 книг, щоб порадити нову книгу, не обов’язково дивитись на його уподобання в музиці). Але якщо користувач тільки починає у домені, тоді допомагають інші домени. Тому наш алгоритм може адаптивно змінювати ваги внеску різних складових. У простішому вигляді це зводиться до: коли користувач тільки прийшов, контентно-кросдоментна частина (яка базується на загальних жанрах/ознаках і глобальних трендах) має більшу вагу, а колаборативна (яка потребує багато даних користувача) – меншу. Але з часом, коли матриця оцінок для цього користувача стає густішою, ми збільшуємо. Іншими словами, модель автоматично визначає, коли слід спиратися на історичні дані з суміжних доменів, а коли – на поточні дії користувача

в цільовому домені. Ця логіка може бути запрограмована через пороги або через оптимізацію функції похибки з регуляризацією.

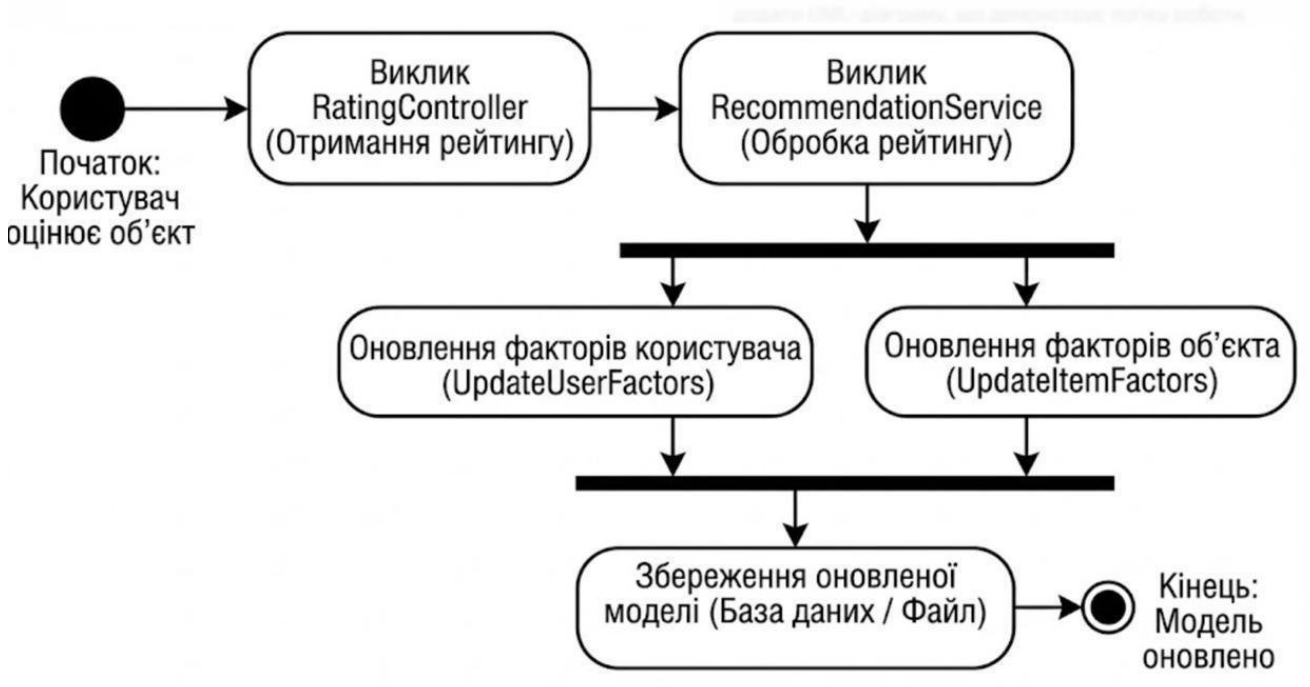

Рис 2.2 ML-діаграма активності процесу обробки оцінки користувача для адаптивного навчання

Реалізація в прототипі. Для оцінки ефективності методики ми плануємо провести офлайн-експеримент на відкритих наборах даних (наприклад, MovieLens для фільмів, Last.fm dataset для музики, Goodreads dataset для книг) – об'єднати їх та перевірити, чи покращується метрика прогнозування у крос-доменному сценарії проти ізольованих. Але це виходить за рамки пояснювальної записки. В рамках прототипу ми реалізували описаний вище алгоритм у вигляді модуля на Python з використанням бібліотеки **Surprise** – спеціалізованого інструментарію для побудови рекомендаційних систем. Surprise надає готові реалізації алгоритмів колаборативної фільтрації (SVD, k-NN, SlopeOne тощо) та зручні засоби для експериментів з даними. Ми використали Surprise для попереднього тренування моделі SVD на зібраному датасеті і для оцінки якості (RMSE на відкладених даних) – це підтвердило життєздатність моделі. Далі, інтеграцію з нашим веб-додатком здійснено шляхом імпорту збережених факторів у бекенд .NET. Тобто модель тренується офлайн у Python, але застосовується онлайн у C# для швидкого прогнозування (оскільки

скалярний добуток векторів довжиною – дуже швидка операція). Також на .NET написано методи для інкрементного оновлення при надходженні нових оцінок (це легко реалізувати вручну, враховуючи простоту формул оновлення). Така схема дозволила поєднати зручність Python-екосистеми для роботи з моделями і продуктивність та інтеграційні можливості .NET для роботи з API і базою даних.

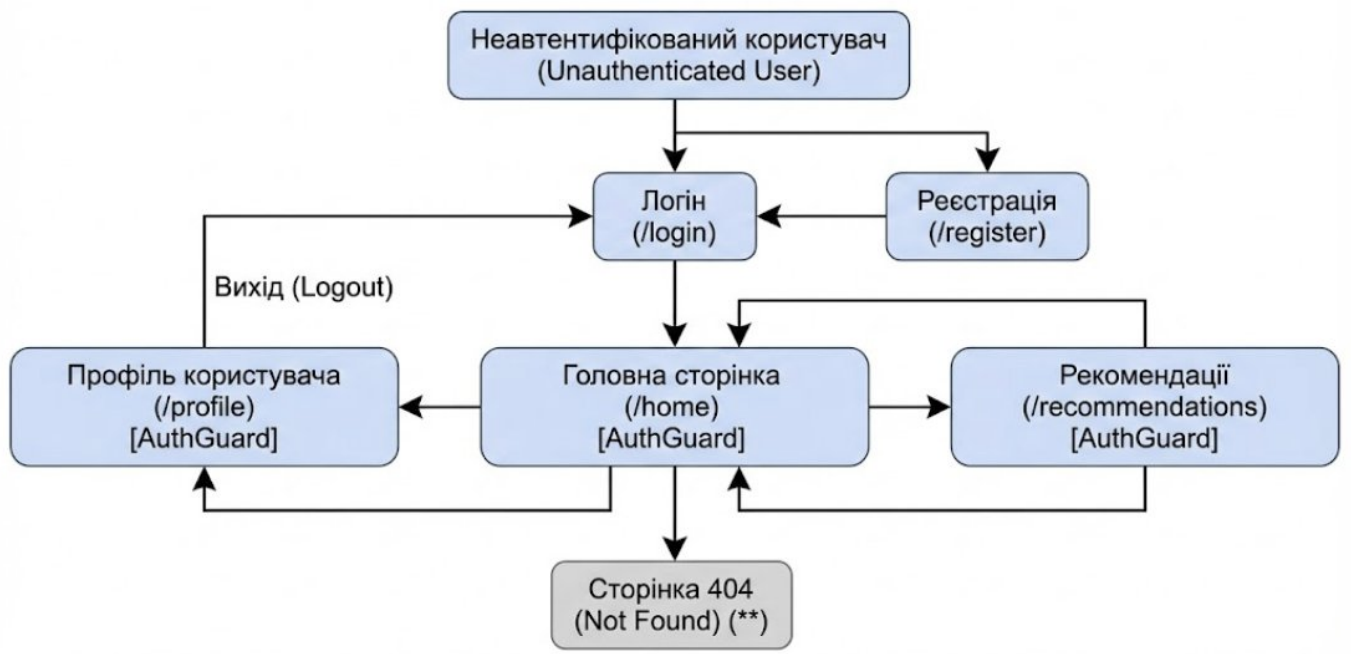

Рис 2.3 Діаграма навігації Angular-додатку (Routing Map)

Отже, методика нашої системи – це гібридна контентно-колаборативна рекомендація з крос-доменним переносом знань і адаптивним оновленням. Вона поєднує: (a) колаборативне прогнозування рейтингу (latent factor model), (b) використання контентних ознак для знаходження аналогій між об’єктами різних типів, (c) поступове навчання на основі нових оцінок (online SGD), (d) динамічне регулювання параметрів алгоритму під конкретного користувача (на основі кількості даних, величини помилки тощо). Такий підхід дозволяє максимально використати доступну інформацію: і дані інших користувачів, і атрибути контенту, і поточну активність самого користувача. У наступному розділі розглянемо, як цей алгоритм вписано в загальну архітектуру програмного забезпечення, які компоненти створено та як між ними розподілено функції.

РОЗДІЛ 3
АРХІТЕКТУРА ТА РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі описано архітектурні рішення, прийняті при розробці програмного забезпечення, та ключові аспекти реалізації серверної та клієнтської частин. Система спроєктована як багатошарова веб-аплікація з чітким поділом на фронтенд (інтерфейс користувача) і бекенд (серверна логіка і база даних). Така архітектура забезпечує масштабованість та гнучкість – фронтенд і бекенд можуть розвиватися незалежно, спілкуючись через стандартизований API.

3.1 Загальна схема системи

Представлено загальний огляд компонентів системи та їх взаємодію. Клієнтська частина (Front-end) представлена Angular SPA, що працює в браузері користувача. Серверна частина (Back-end) – це веб-сервер на платформі ASP.NET Core, що надає RESTful API. База даних зберігає інформацію про користувачів, об’єкти та виставлені оцінки. Окремо виділено модуль Recommendation Engine, який відповідає за розрахунок рекомендацій; він інтегрований в бекенд (реалізований частково на C#, з використанням попередньо навчених моделей з Python/Surprise).

Взаємодія відбувається наступним чином: 1. Користувач відкриває веб-додаток (SPA). Angular-застосунок завантажується з сервера та починає працювати на стороні клієнта. 2. При необхідності отримати дані (авторизація, список об’єктів, рекомендації тощо), фронтенд робить HTTP-запит до API бекенду. 3. Бекенд (контролери ASP.NET) приймає запит, виконує відповідну бізнес-логіку – наприклад, звертається до бази даних через ORM, або викликає Recommendation Engine для генерації рекомендацій – і формує відповідь. 4. Результат (формат JSON) надсилається назад клієнту. Angular отримує дані і оновлює інтерфейс (DOM) без

перезавантаження сторінки, завдяки двосторонньому зв’язуванню даних (data binding). 5. Якщо користувач здійснює дію, що змінює дані (ставить оцінку, додає в друзі тощо), фронтенд відправляє відповідний запит (POST/PUT) до сервера. Бекенд оновлює базу, а також (якщо це оцінка) викликає процедуру оновлення рекомендаційної моделі для цього користувача. 6. Після оновлення моделі сервер може одразу згенерувати новий список рекомендацій і повернути його клієнту, або клієнт сам запитує свіжі рекомендації при наступному оновленні інтерфейсу.

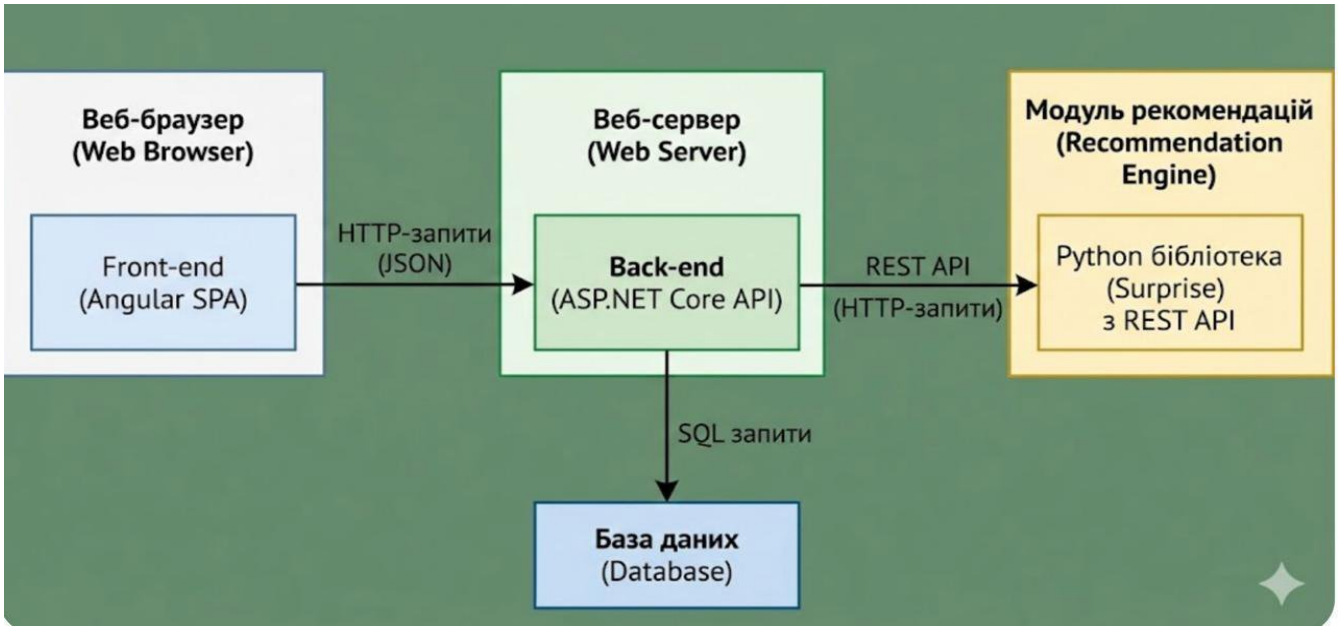

Рис 3.1 Діаграма взаємодії сервісів у системі

Архітектура бекенду. У серверній частині реалізовано принципи Clean Architecture (або "Гексагональної"/"Onion" архітектури). Код поділено на декілька проектів (шарів): - Domain Layer – містить бізнес-логіку та сутності домену (клас User, Item, Rating, а також інтерфейси репозиторіїв та сервісів). Цей шар не залежить від деталей реалізації – в ньому немає кодів звернень до бази чи фреймворків, лише чисті C#-об’єкти і методи обробки (наприклад, метод для оновлення профілю користувача). - Application Layer – містить сервісні класи, що реалізують конкретні Use Cases (випадки використання). Тут зосереджена логіка виконання запитів: наприклад, RecommendationService, що при виклику GetRecommendationsForUser(u) звертається до репозиторію даних, отримує історію оцінок користувача, викликає модуль рекомендацій і повертає готовий список. Application-логіка оркеструє

взаємодію між Domain та Infrastructure. - Infrastructure Layer – реалізації інфраструктурних деталей: збереження даних (репозиторії, що використовують Entity Framework Core для доступу до SQL-бази), інтеграція зі сторонніми бібліотеками (наприклад, клас SurpriseModelLoader, який зчитує файли моделі, експортовані з Python), сервіси відправки email тощо. Цей шар може залежати від зовнішніх бібліотек, але ізольований від Domain, щоб у разі заміни технології (наприклад, перехід на іншу БД) не зачепити бізнес-логіку. - Presentation Layer – власне ASP.NET Core Web API: контролери, DTO (Data Transfer Objects) для обміну даними з клієнтом, налаштування маршрутизації, обробка аутентифікації/авторизації. Контролери звертаються до Application-сервісів, отримують від них результати та відправляють клієнту.

Такий поділ забезпечує слабку зв’язаність модулів і тестованість. Ми можемо окремо тестувати Domain-логіку (наприклад, функцію розрахунку подібності двох користувачів) без підняття бази чи сервера. Так само легко замінити, скажімо, Recommendation Engine на інший – достатньо реалізувати той самий інтерфейс в Domain/Application шарі.

Фронтенд архітектура. Angular-додаток побудовано з використанням module-component structure. Є основний модуль AppModule та декілька фіче-модулів: HomeModule, ProfileModule, RecommendationsModule тощо. Кожна сторінка або функціональність – це Angular component з власним шаблоном HTML, стилями CSS (SCSS) та класом на TypeScript, що керує поведінкою. Для взаємодії з API створено сервісні класи (Angular services), які використовують модуль HttpClient для відправки запитів. Наприклад, RecommendationService на фронтенді має метод getRecommendations() – він робить GET-запит до /api/recommendations і повертає Observable списку об’єктів. Компонент RecommendationsComponent підписується на цей сервіс і відображає отримані дані користувачу.

Angular забезпечує reactive UI: коли змінюється модель у компоненті (наприклад, приходять нові рекомендації), шаблон автоматично перерендерюється. Ми використали Angular Router для організації навігації без перезавантаження сторінок: різні URL відображають різні компоненти (наприклад, /home, /profile/:id,

/recommendations). Це дозволяє користувачу перемикатися між розділами додатку миттєво, без оновлення з сервера – завантажуються лише необхідні дані. Крім того, реалізовано систему авторизації JWT: при вході користувач отримує токен, який зберігається в LocalStorage і додається Angular HttpInterceptor'ом до кожного запиту на API (в заголовок Authorization). Бекенд перевіряє токен, і таким чином приватні маршрути захищені (якщо токен недійсний, API поверне 401, і на фронті користувача перекине на сторінку логіну).

Соціальний функціонал. Окрім рекомендацій, система підтримує прості соціальні функції: кожен користувач має профіль, де показано його вподобання (наприклад, список улюблених книг, оцінки останніх фільмів), та може підписуватися на профілі інших (аналог "друзів"). Реалізовано це через таблицю Friendships у базі, та відповідні API: GET /api/users/{id}/friends, POST /api/users/{id}/follow. На фронтенді – сторінка профілю відображає інформацію користувача та його контент, а також рекомендації "на основі вашої сумісності". Індекс сумісності (як taste-o-meter Last.fm) розраховується у бекенді при запиті профілю: обчислюється косинусова схожість між векторами вподобань двох користувачів (або коефіцієнт кореляції Пірсона між їх рядами оцінок). Цей показник показує, наскільки близькі смаки – високий відсоток означає, що у двох користувачів схожі оцінки на багато об'єктів. Таким чином, відвідавши профіль іншого, ви бачите "Сумісність: 72%" і можете оцінити, чи варто довіряти його рекомендаціям. Також в профілі видно стрічку активності (напр. "Іван поставив 5 зірок книзі '1984'"). Ця соціальна стрічка генерується на бекенді (вибірка останніх оцінок користувача та коментарів, якщо такі були, з об'єднанням по різних доменах).

3.2 Реалізація серверної частини (ASP .NET Core)

Вибір платформи .NET Core. ASP.NET Core було обрано з декількох причин: - по-перше, це крос-платформний фреймворк (працює на Windows, Linux, macOS), що дає гнучкість у розгортанні проекту; - по-друге, висока продуктивність та асинхронність: .NET Core API показують одні з найкращих результатів у тестах продуктивності серед веб-фреймворків, а асинхронна модель (async/await) дозволяє

ефективно обробляти багато одночасних запитів; - по-третє, багатий екосистемою: наявні ORM (Entity Framework), вбудовані рішення для авторизації (Identity), підтримка Swagger-документації тощо прискорили розробку.

Controllers & Endpoints. У проекті створено кілька контролерів: AuthController (реєстрація/логін, отримання JWT), UsersController (отримання даних профілю, список підписок, пошук користувачів), ItemsController (пошук і фільтрація контенту, отримання деталей об'єкта), RatingsController (відправка оцінки), RecommendationsController (одержання списку рекомендацій). Кожен контролер налаштований атрибутами маршрутизації, наприклад:

```
[Route("api/recommendations")]
[ApiController]
1 reference
public class RecommendationsController : ControllerBase {
    2 references
    private readonly IRecommendationService _recService;
    0 references
    public RecommendationsController(IRecommendationService recService) { }

    [HttpGet]
    0 references
    public async Task<ActionResult<IEnumerable<ItemDto>>> GetRecommendations() {
        var userId = User.GetUserId();
        var recs = await _recService.GetRecommendationsForUser(userId);
        return Ok(recs);
    }

    [HttpGet]
```

Рис 3.2 Приклад коду контролера рекомендацій

Метод GetRecommendations() бере поточного аутентифікованого користувача, викликає сервіс рекомендацій і повертає список DTO об'єктів. DTO (Data Transfer Object) – це простий клас з полями, призначеними для серіалізації в JSON. Ми використовуємо DTO замість прямих доменних моделей, щоб не передавати зайву інформацію і забезпечити гнучкість форматування. Наприклад, ItemDto містить ID, назву, тип контенту, основні атрибути і, якщо включено, середню оцінку тощо.

Сервіси та Recommendation Engine. Рекомендаційний сервіс (RecommendationService) інжектиться у контролер як IRecommendationService

(використано вбудований механізм **Dependency Injection** .NET). При запуску програми ми реєструємо залежності в Startup.cs (або Program.cs, залежно від версії .NET). Де MatrixFactorizationModel – це клас, що реалізує інтерфейс ICollaborativeModel і містить у собі завантажені матриці та методи для отримання топ-N рекомендацій. Ми зробили його Singleton, бо модель одна спільна для всіх користувачів і тримається в пам'яті. RecommendationService взаємодіє з цим модулем, а також з репозиторієм даних:

```
public class RecommendationService : IRecommendationService {
    2 references
    private readonly ICollaborativeModel _model;
    1 reference
    private readonly IContentSimilarityService _contentSim;
    1 reference
    private readonly IUserRepository _userRepo;
    1 reference
    public async Task<IEnumerable<ItemDto>> GetRecommendationsForUser(Guid userId) {
        var user = await _userRepo.GetByIdWithRatingsAsync(userId);
        var topItems = _model.PredictTopN(user, n: 10);
        var itemDetails = await _itemRepo.GetByIdsAsync(topItems.Select(x => x.ItemId));
    }
    1 reference
    public async Task UpdateModelOnRating(Guid userId, Rating newRating) {
        _model.UpdateUserFactors(userId, newRating.ItemId, newRating.Score);
        _contentSim.UpdateProfile(userId, newRating.Item);
    }


    1 reference
```

Рис 3.3 Приклад коду сервісу рекомендацій

Метод GetRecommendationsForUser завантажує користувача з його оцінками (наприклад, щоб врахувати cold start чи інше), потім просить модель передбачити топ-10 об'єктів, перетворює їх на DTO і повертає. Метод UpdateModelOnRating викликається з RatingController після того, як нова оцінка збережена в БД: він повідомляє модель про нові дані. Наша MatrixFactorizationModel всередині зберігає словник userFactors[userId] = float[k] та аналогічно itemFactors[itemId]. Метод UpdateUserFactors виконує описану в розділі 2 процедуру – коригує ці вектори. Оскільки модель – Singleton, вона живе протягом роботи застосунку і поступово вчиться на даних, що надходять (в межах сесії програми). У прототипі ми не робили

фонове збереження факторів у базу кожного разу (це було б надмірно для демо), але у production-випадку, звісно, варто періодично зберігати оновлені параметри моделі.

База даних. Як СУБД використано SQL Server (в режимі LocalDB під час розробки). Схема БД: таблиці **Users**, **Items**, **Ratings**, **Friendships**: - Users: Id (GUID primary key), Username, Email, PasswordHash, etc. - Items: Id (int PK), Title, Type (enum: Book/Movie/Music/Game), Genre, Author/Director/etc (за потреби, або зв'язки на інші таблиці якщо деталізувати), AverageRating, ... (деякі поля можна тримати також у кэш, але ми зберігали середній рейтинг для швидкого сортування). - Ratings: UserId, ItemId, Score (1-5), Timestamp. PK – композитний (UserId+ItemId, щоб кожен користувач мав максимум одну оцінку для об'єкта). Ця таблиця росте з часом, на ній індекси по UserId і ItemId для швидкого пошуку. - Friendships: UserId, FriendId (або FollowerId/FollowingId, в нашому випадку симетричні дружби). Якщо потрібно, можна було б додати статус (pending/accepted), але ми спростили як взаємні підписки. - (Опціонально) ItemsGenres, ItemsTags: якщо потрібна багатозначна атрибутика. Ми спростили, зробивши поле Genre з переліком жанрів через кому, щоб не ускладнювати.

```
0 references
public class UserRepository : IUserRepository {
    1 reference
    private readonly AppDbContext _ctx;
    1 reference
    public Task<User> GetByIdWithRatingsAsync(Guid userId) {
        return _ctx.Users
            .Include(u => u.Ratings) // завантажити оцінки користувача
            .ThenInclude(r => r.Item) // і відповідні об'єкти
            .FirstOrDefaultAsync(u => u.Id == userId);
    }
```

Рис 3.4 Приклад коду репозиторію

Для доступу до БД використано **Entity Framework Core** (Code First). Ми визначили відповідні entity-класи (User, Item, Rating, Friendship) і контекст AppDbContext (наслідує DbContext). Code First дозволяє створювати і мігрувати схему

БД на основі цих класів. Міграції були згенеровані командою Add-Migration і застосовані Update-Database. В Application шарі є UserRepository, ItemRepository тощо, які через AppDbContext виконують потрібні запити (переважно з використанням LINQ).

Завантаження об’єктів разом з оцінками може бути корисним, наприклад, для побудови профілю або відображення “історії” в інтерфейсі.

Тестування та налагодження. Під час розробки ми використовували **Swagger UI** для ручного тестування API – .NET Core має інтегровану підтримку, достатньо додати Swashbuckle. Це дозволило перевіряти відповіді ендпоінтів зразками даних. Також були написані декілька модульних тестів (XUnit) для методів RecommendationService з моками CollaborativeModel (перевірити, що список топ-N не містить вже оцінених елементів і т.п.). Логування (ILogger) налаштоване на запис у консоль ключових дій (наприклад, лог “User X rated Item Y with 5, updating model...”).

Безпека. Авторизація побудована на JWT токенах, виданих при логіні. Паролі зберігаються у вигляді хешів (алгоритм PBKDF2 через Identity API). Для захисту від несанкціонованого доступу до API використано атрибути [Authorize] на контролерах, а також перевірку прав доступу в деяких методах (наприклад, користувач може змінювати лише свій профіль, але не чужий – це перевіряється по User.Identity.Name vs запитуваний ресурс). Дані між клієнтом і сервером передаються по HTTPS (в розробці – через запуск Kestrel на localhost з самопідписаним сертифікатом).

3.3 Реалізація клієнтської частини (Angular SPA)

Структура проекту. Проект створено на Angular 12 (TypeScript). Використано Angular CLI для генерації компонентів та сервісів. У корені додатку – файл app.module.ts, де імпортуються основні Angular модулі (BrowserModule, HttpClientModule, AppRoutingModule тощо) і оголошуються наші компоненти та сервіси.

UI компоненти. Інтерфейс побудовано з використанням бібліотеки компонентів Angular Material – це забезпечило сучасний вигляд і адаптивність. З Material взято навігаційну панель (MatToolbar), кнопки, картки (MatCard) для відображення об’єктів, іконки тощо. Основні екрани: - HomeComponent – домашня сторінка, показує загальний фід: останні новинки, популярні об’єкти, і можливо короткий огляд рекомендацій (прев’ю). - RecommendationsComponent – сторінка “Рекомендації для вас”, відображає список рекомендованих об’єктів (мікс книг/фільмів/музики). Реалізовано як сітка карток: на кожній картці – зображення (обкладинка книги, постер фільму – посилання на зображення зберігаються в БД), назва, тип (іконка, наприклад, піктограма книги чи ноти), і кнопка для виставлення оцінки прямо на цій картці (випадаючий список з 1 до 5, або "like/dislike"). Коли користувач виставляє оцінку, викликається метод компонента rate(item, score): він через RatingService надсилає на API нову оцінку, а у разі успіху видаляє цей об’єкт з локального списку рекомендацій і, можливо, автоматично підвантажує наступний у черзі (щоб підтримувати постійно 10 рекомендацій на екрані). - ProfileComponent – сторінка профілю користувача. Показує інформацію про користувача (ім’я, аватар), його “статусу” (можна додати опціонально біо або улюблений жанр). Нижче – вкладки: “Оцінки” (таблиця або список всіх оцінених творів з їх оцінками), “Улюблене” (об’єкти, яким поставлено найвищі бали), “Друзі” (список користувачів, на яких підписаний). Також якщо це не ваш профіль, вгорі буде кнопка “Додати в друзі/Підписатися”, і показано відсоток сумісності. ProfileComponent при завантаженні викликає UsersService.getProfile(userId) – API повертає структуру, що містить все необхідне (UserDto з полями, а вкладено списки RatedItems, FavoriteItems, Friends, і, якщо авторизований інший користувач переглядає, поле SimilarityPercentage). Це трохи “важка” відповідь, але спрощує логіку клієнта – все готово. - BrowseComponent – сторінка для перегляду всього контенту з фільтрами. Наприклад, можна вибрати вкладки “Books/Movies/Music/Games” і побачити популярні чи нові надходження у кожній. Є поле пошуку. Search реалізовано через ItemsService.search(query) – запит до API, який шукає по назві/автору і повертає список об’єктів. Результати показуються аналогічно картками. Цей компонент

дозволяє користувачу досліджувати каталог самостійно, поза автоматичними рекомендаціями.

Стан додатку. Використано підхід з сервісами як Singletons, які зберігають стан між компонентами. Наприклад, при логіні AuthService зберігає інформацію про поточного користувача (AuthService.currentUser$ – BehaviorSubject, на який підписані інші компоненти). Компонент навігації (AppComponent) слухає AuthService і показує відповідно кнопки “Login/Signup” або меню профілю якщо залогінений. Маршрути, що вимагають авторизації (напр. /recommendations, /profile/me), захищені AuthGuard – якщо токен відсутній, редірект на /login.

```
  ngOnInit() {
  this.recommendationService.getRecommendations().subscribe(items => {
    this.recommendations = items;
  });
}


onPageChange(page: number) {
        this.currentPage = page;
        this.loadItems();
    }
```

Рис 3.5 Приклад коду завантаження рекомендацій

При відправці оцінки:

```
loadItems() {
      this.isLoading = true;
      this.onboardingService.getItems(0, this.currentPage, this.pageSize, this.searchQuery).subscribe({
          next: (items) => {
              this.movies = items;
              this.hasMore = items.length === this.pageSize;
              this.isLoading = false;
          },
          error: (err) => {
              console.error('Error loading items:', err);
              this.isLoading = false;
          }
      });
  }

  rate(item: Item, score: number) {
      this.ratingService.submitRating(item.id, score).subscribe(() => {
          // успішно збережено
          item.userScore = score;
          // при бажанні одразу отримати оновлені рекомендації:
          this.recommendations = this.recommendations.filter(x => x.id !== item.id);
          // опціонально дозавантажити ще один рек, щоб знов було 10
          this.replenishRecommendations();
      });
  }
```

Рис 3.6 Приклад коду відправки оцінки

Щоб зменшити навантаження було вирішено не оновлювати весь список з сервера після кожної оцінки, а просто видаляти оцінений елемент і за потреби локально додавати новий з тих, що були 11-м, 12-м в сортованому списку (model може на бекенді присилати запас, або фронт може одразу запитати 15 і показувати 10, а далі додавати). У спрощеному варіанті можна і одразу викликати getRecommendations() знов – але тоді весь список може різко змінитися, що потенційно збиває користувача. Наш UX прагне більш плавної адаптації: рекомендації оновлюються поступово, по мірі дій користувача.

Взаємодія з Recommendation Engine в реальному часі. Коли користувач ставить оцінку, сервер оновлює модель і може надіслати через відповідь “о, до речі, ось нова рекомендація замість оціненого”. Ми це не робили через REST, але можна було б використати WebSocket канал або SignalR для push-нотифікацій. Наприклад, як тільки модель оновилась і з’явився новий top-результат, сервер посилає на клієнт повідомлення, і Angular додає цю рекомендацію списку (із позначкою “New”). Такі

реалії складніші, тому зупинились на pull-моделі: клієнт сам періодично (або по діях) тягне нові дані.

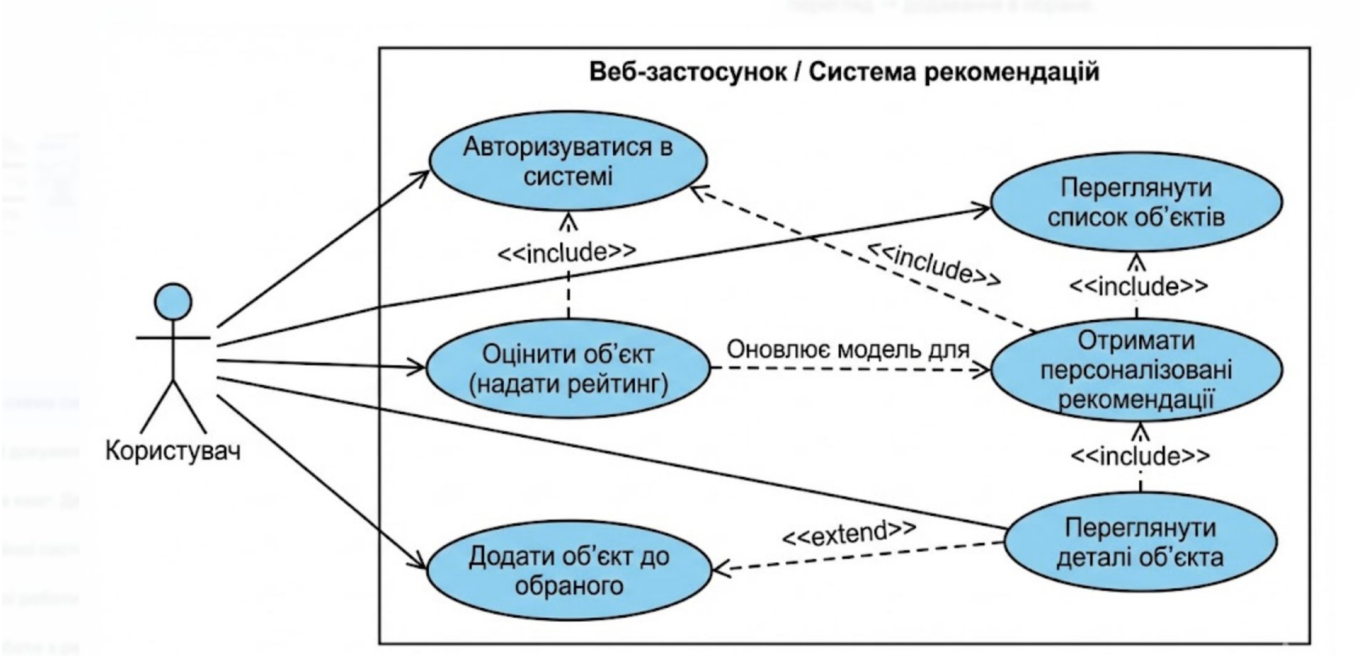

Рис 3.7 Діаграма сценаріїв

Приклад сценарію роботи: Користувач заходить на сторінку рекомендацій. Angular ініціює завантаження – відправляється запит GET /api/recommendations. Серверний RecommendationService отримує userId, звертається до моделі, повертає список 10 ItemDto (наприклад: *The Matrix* (фільм), *1984* (книга), *Pink Floyd – The Wall* (альбом), *Witcher 3* (гра) ... – змішані). Клієнт малює картки цих творів. Користувач бачить, що йому рекомендують фільм “Матриця” і книгу “1984”. Він вирішує поставити оцінку фільму “Матриця” (бо вже бачив) – натискає на картці випадаючий рейтинг і обирає “5 зірок”. Це викликає RatingService.submitRating, який робить POST /api/ratings з даними {itemId: Matrix, score:5}. Бекенд (RatingsController) аутентифікує користувача (з токена), додає запис у БД Ratings (User=U, Item=Matrix, Score=5), викликає RecommendationService.UpdateModelOnRating(U, newRating). Той оновлює внутрішні фактори користувача U та об’єкта Matrix. Потім метод контролера повертає Ok. Клієнт отримує відповідь, видаляє картку “Матриця” (бо вже оцінено) і, припустимо, нічого більше не робить – у нього тепер 9 рекомендацій на екрані. Він

переходить на іншу сторінку, або якщо залишився на цій – через якийсь час (чи за подією) Angular може зробити авто-догрузку: if(recommendations.length < 10) getMore(). Цей getMore може звернутися до API, яке поверне ще один об’єкт. А може фронтенд наперед отримував 15, тоді просто додає 11-й. Неважливо, суть: після певних дій користувача сервер знає нову інформацію і може дати нові поради. Якщо користувач натисне “Обновити рекомендації” – викличеться /api/recommendations знов, але тепер модель уже трошки інша (Matrix має тепер високу оцінку, отже профіль користувача змінився). Сервер, можливо, тепер замість *Witcher 3* покаже інший об’єкт, ближчий до жанру Matrix. Клієнт це відобразить – тобто користувач побачить, що після оцінювання Матриці в нього з’явився в рекомендаціях, скажімо, фільм *Inception*, якого не було до того. Це і є ефект адаптивності на практиці.

Підкреслимо, що наш прототип – це веб-застосунок, тому всі оновлення моделей відбуваються в пам’яті сервера і діють для даної сесії. Якщо рестартувати сервер (drop in-memory factors) – поки що втратяться інкрементальні зміни, але при production-налаштуванні треба або зберігати фактори назад у БД, або ж настільки часто рестарти не робити (у реальній системі сервер працює постійно і модель тримається). Також, якщо горизонтальна масштабованість (кілька серверів), то тут був би виклик: як синхронізувати модель між серверами. Це потребує або централізованого сховища моделі (сервіс рекомендацій окремий), або синхронного оновлення (через шину повідомлень). У нашому випадку, оскільки проект демонстраційний, ми не розпаралелювали сервіс.

3.4 Приклад використання. Демонстрація роботи прототипу

Щоб проілюструвати роботу розробленого програмного забезпечення, розглянемо типовий сценарій використання системи з точки зору кінцевого користувача.

Крок 1: Аутентифікація користувача. Новий користувач відкриває веб-застосунок та бачить екран входу. Якщо в нього ще немає акаунту, він переходить на форму реєстрації, вводить необхідні дані (ім’я, email, пароль) і створює новий акаунт.

Після успішної реєстрації здійснюється автоматичний вхід до системи. (На цьому етапі спрацьовує модуль аутентифікації; дані нового користувача зберігаються у таблиці Users бази даних, пароль хешується, тощо).

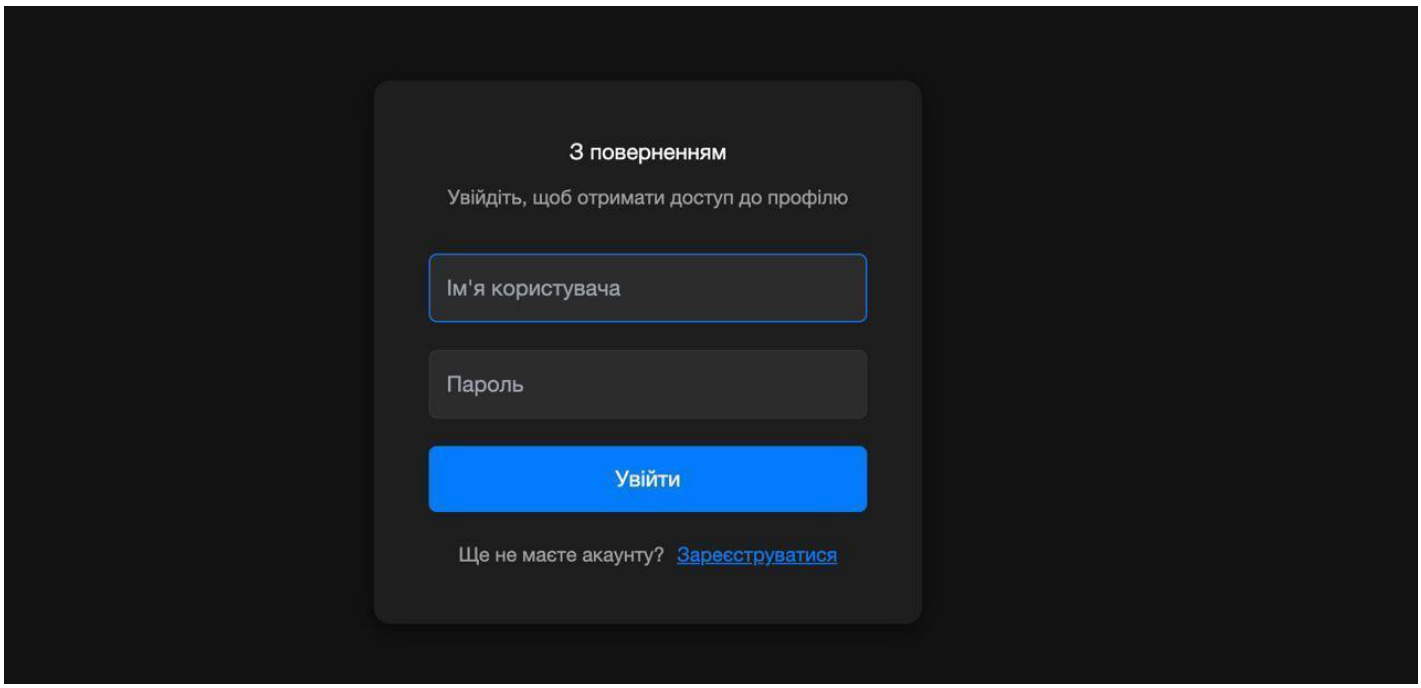

Рис 3.8 Сторінка авторизації

Крок 2: Первинне налаштування рекомендацій (вибір улюбленого контенту). Одразу після входу користувач потрапляє на сторінку початкового налаштування рекомендацій. Система пропонує обрати до п’яти творів, які користувачу вже подобаються, щоб персоналізувати подальші рекомендації. На екрані відображається каталог популярних творів (наприклад, список відомих фільмів) з можливістю пошуку. Користувач відмічає декілька об’єктів, які він вподобав (наприклад, обирає фільми “The Matrix”, “Inception” та “Interstellar”). Після цього натискає кнопку “Завершити та отримати рекомендації”. На основі вибраних вподобань система формує початковий список рекомендацій – користувач бачить на головній сторінці набір рекомендованих об’єктів, релевантних до його смаків. Цей механізм дозволяє подолати проблему холодного старту: навіть новий користувач, про якого ще немає

даних у системі, одразу отримує персоналізовані поради.

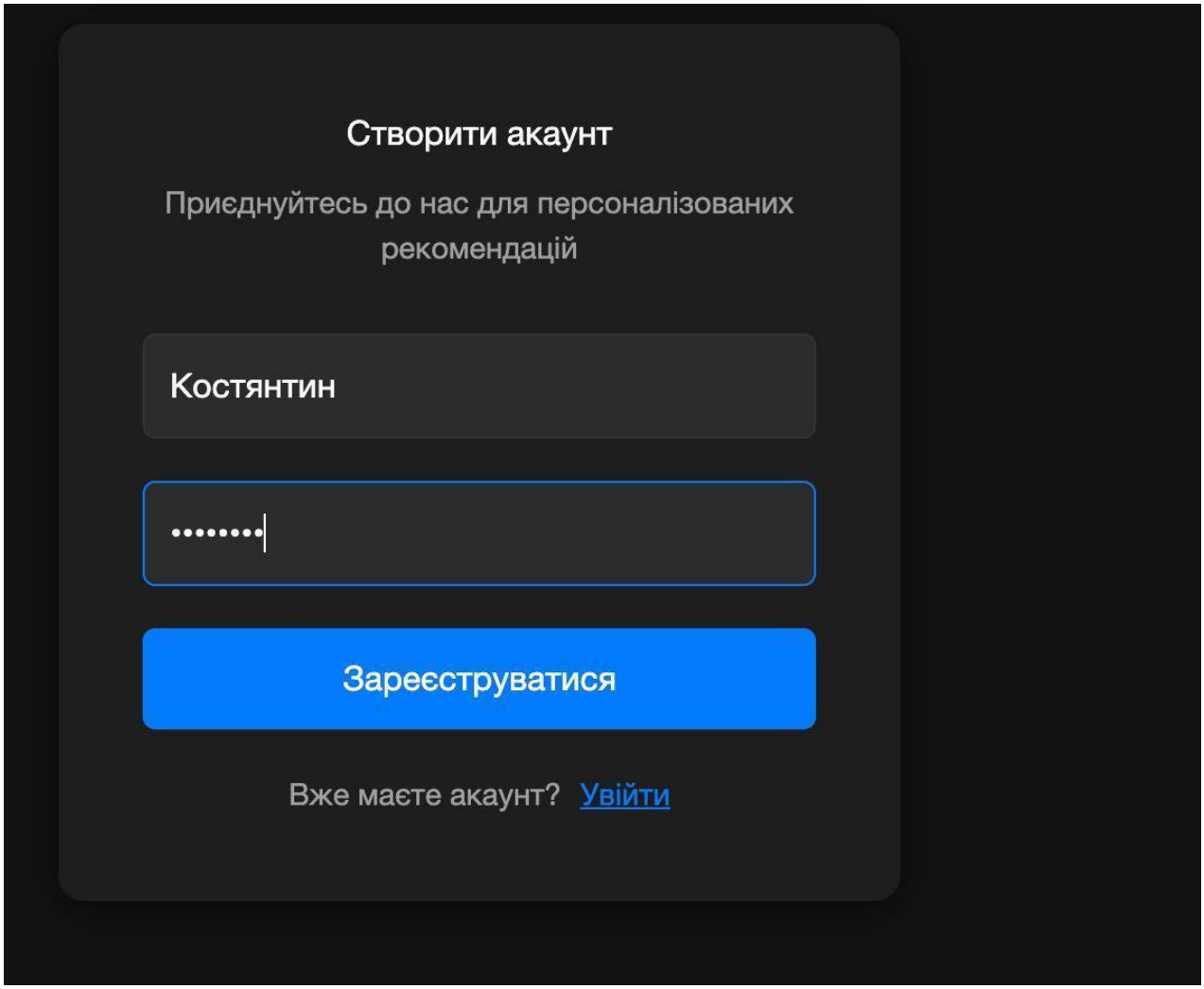

Рис 3.9 Сторінка реєстрації

Крок 3: Генерація крос-доменних рекомендацій. Після первинного налаштування стає доступним основний функціонал – модуль рекомендацій. Користувач може експериментувати з різними видами контенту. Наприклад, він вирішує скористатися крос-доменною рекомендацією: обирає кілька улюблених фільмів і хоче отримати рекомендацію книги. Для цього на сторінці рекомендацій він додає до списку декілька фільмів, які йому дуже сподобалися (система надає зручний пошук з автодоповненням назв; користувач вводить перші літери, обирає потрібні назви зі списку). Далі в інтерфейсі присутній перемикач або випадаючий список для вибору цільового домену рекомендації – користувач обирає опцію “книга” (тобто хоче, щоб на основі вподобаних фільмів було рекомендовано книгу). Натискає кнопку

“Згенерувати рекомендації”. Back-end отримує цей запит через ендпоінт (наприклад, з параметрами вибраних фільмів та таргет-домену “Book”), передає його до модуля рекомендацій. Використовуючи попередньо навчену модель та алгоритми (описані в розділі 2), система знаходить книги, які найбільше відповідають вибраним фільмам за змістовною схожістю та вподобаннями аудиторії.

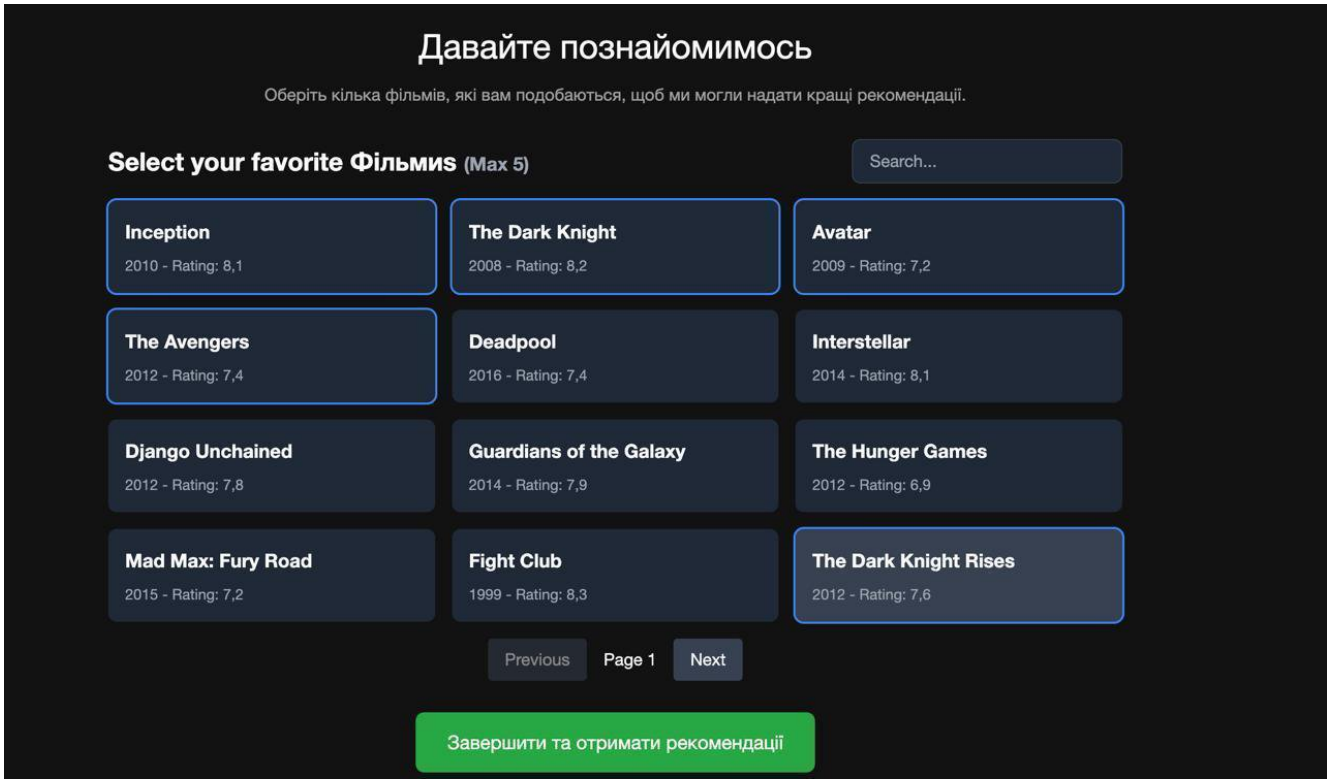

Рис 3.10 Сторінка першої авторизації

Результат повертається на фронтенд – користувач бачить список рекомендованих книжок. *Наприклад, на основі вибраних ним на попередньому кроці фільмів, система може порекомендувати роман “Мій брат – охоронець” (My Sister’s Keeper, автор Джоді Піколт) чи класичний твір “Вбити пересмішника” (To Kill a Mockingbird, Гарпер Лі)* – ці книги тематично або емоційно перегукуються з вибраними кінофільмами. Користувач може переглянути детальні описи цих рекомендацій, обрати одну з них для ознайомлення. (Таким чином продемонстровано, як система виконує крос-доменний аналіз контенту і генерує нетривіальні

рекомендації).

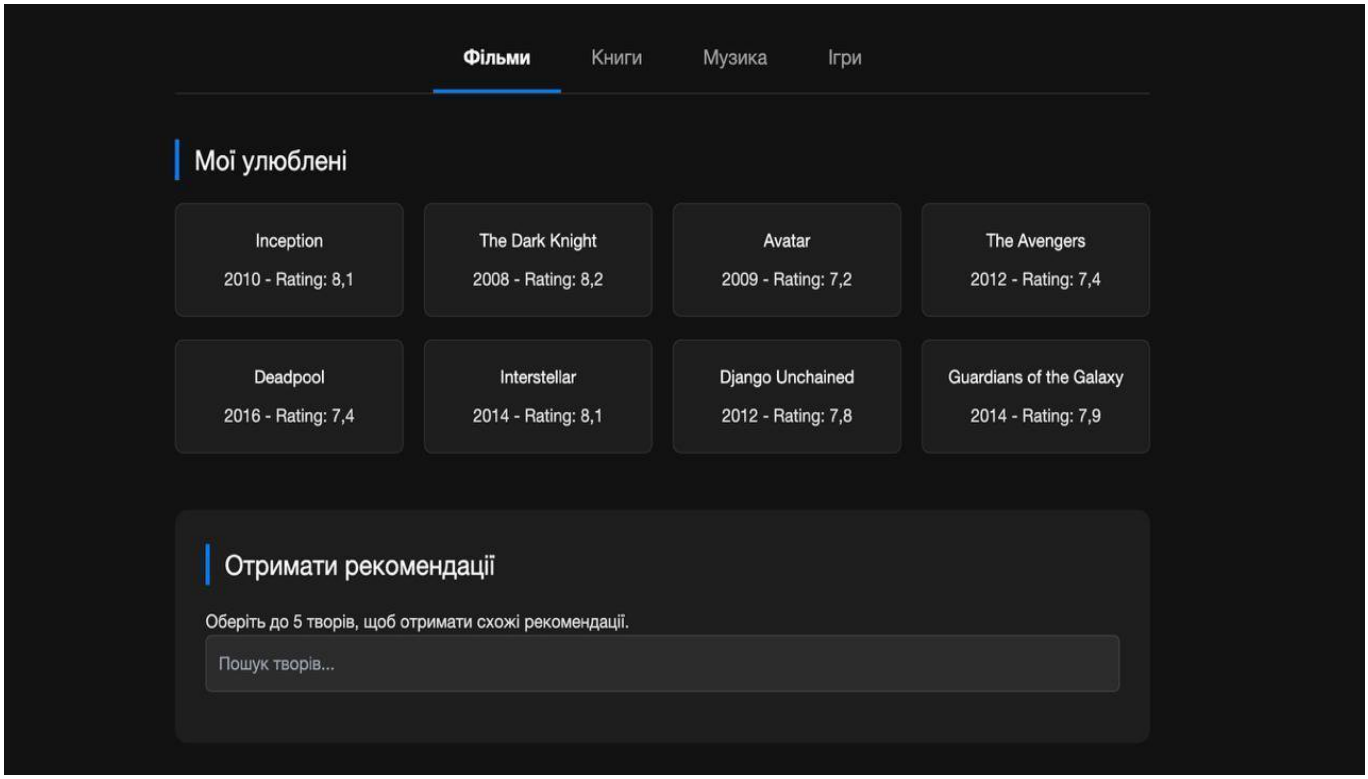

Рис 3.11 Сторінка рекомендацій

Крок 4: Взаємодія з рекомендаціями та адаптивне навчання. Отримавши список рекомендацій, користувач починає з ними взаємодіяти. Наприклад, він відкриває сторінку детального опису одного з рекомендованих об’єктів (припустимо, книги) і вирішує поставити оцінку або відзначити “вподобати” той об’єкт. Припустимо, користувач обрав з рекомендацій книгу “The Witcher” і виставив їй оцінку 5/5 (дуже сподобалася). Це діє як зворотній зв’язок для системи: в базі даних створюється новий запис у таблиці Ratings (userId, itemId, score). Бекенд, отримавши цю оцінку через відповідний контролер, спочатку зберігає її, а потім викликає метод оновлення моделі рекомендацій. Цей метод реалізує адаптивне навчання – оновлює внутрішні фактори моделі для даного користувача та конкретного об’єкта (переобчислює латентні вектори). Таким чином, рекомендаційна модель “навчилася” на нових даних: система врахувала, що користувачу дуже сподобався *The Witcher*.

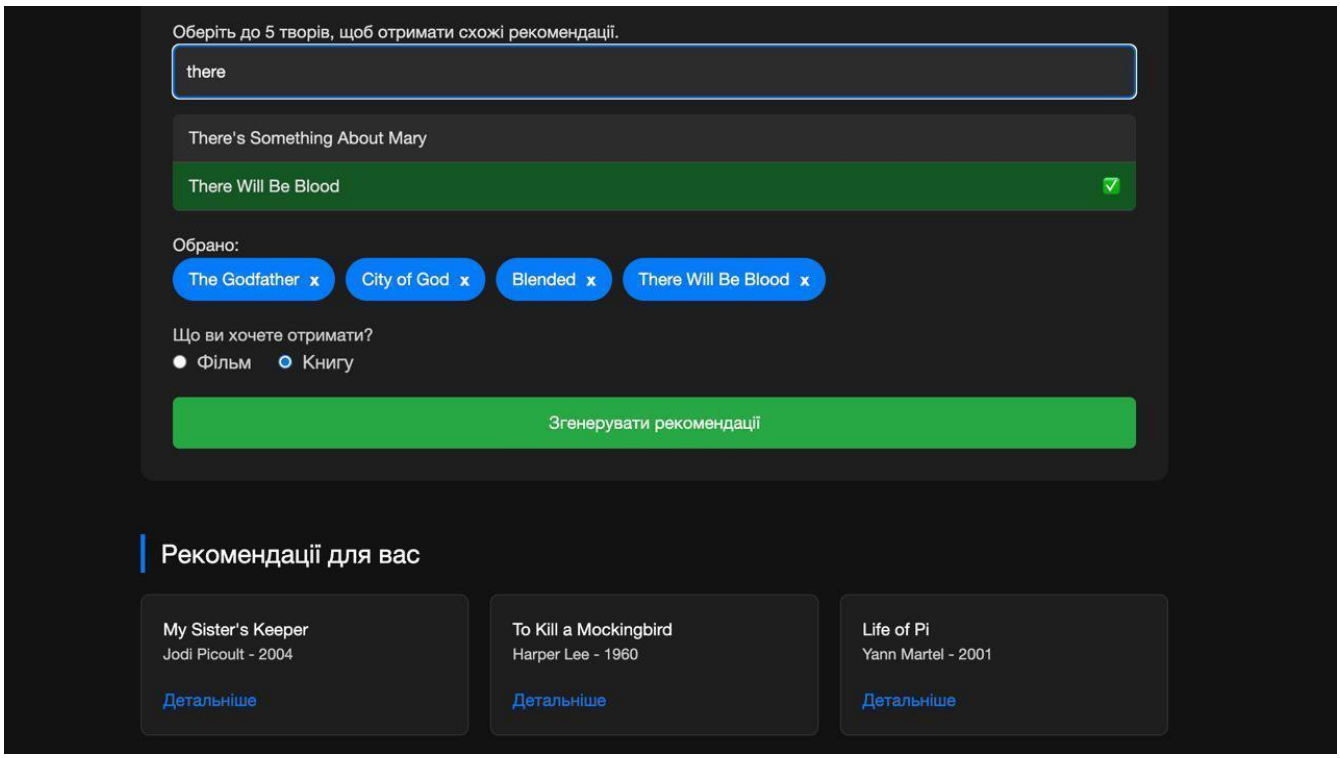

Рис 3.12 Відображення рекомендацій

Крок 5: Оновлення рекомендацій в реальному часі. Завдяки адаптивному механізму, система може динамічно перебудувати список рекомендацій під актуальні вподобання. Продовжуючи попередній приклад: після того як користувач оцінив книгу The Witcher на 5, модель скоригувала профіль користувача. Якщо тепер він знову відкриє сторінку рекомендацій або натисне кнопку “Оновити рекомендації”, бекенд сформує новий список топ-об’єктів. Можливий результат: замість деяких попередніх рекомендацій (наприклад, система до цього радила *“The Witcher 3”* як гру, близьку до жанру фантастики), тепер більш ймовірно з’явиться інший об’єкт, пов’язаний з щойно оціненою книжкою. Скажімо, система може запропонувати до перегляду фільм “Inception” або інший твір, який більше відповідає щойно підтвердженим уподобанням користувача. Користувач побачить, що після оцінювання одного з рекомендованих елементів список рекомендацій змінився – це і є практичний ефект адаптивності: система “вчиться” на його діях у реальному часі.

Крок 6: Використання інших можливостей системи. Окрім модуля рекомендацій, користувачу доступні й інші функції, що були реалізовані в прототипі.

Зокрема, він може переглянути історію своєї активності. На сторінці “Історія” автоматично фіксуються всі дії: які об’єкти він позначив “вподобано” або просто відкрив для перегляду, з точною датою і типом дії. Це дає можливість згадати, що вже було переглянуто, а також забезпечує дані для аналітики (система може враховувати перегляди без оцінки як сигнал зниження цікавості). Користувач також може відвідати свій профіль, де представлена зведена статистика: скільки фільмів, книг, музичних творів він вподобав; які жанри переважають у його вподобаннях; список друзів (якщо додані).

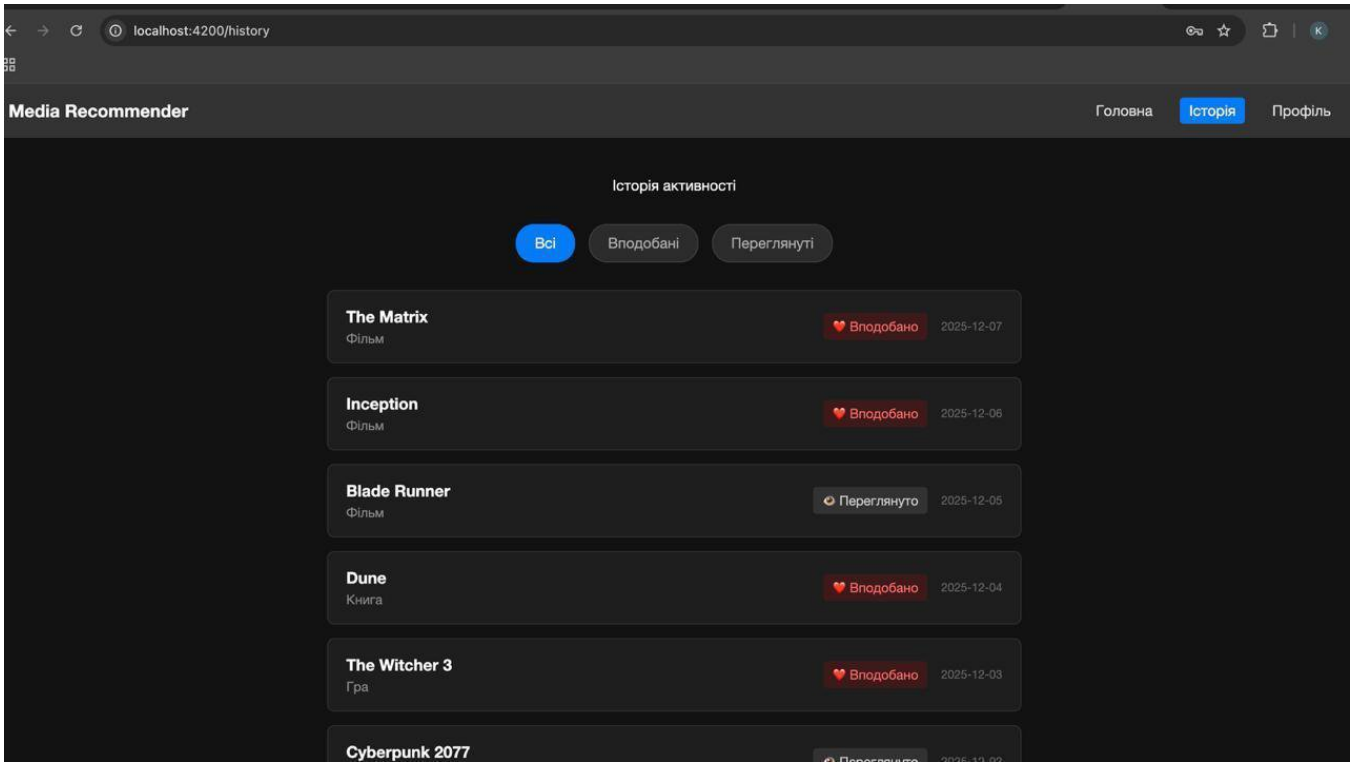

Рис 3.13 Сторінка історії (вподобане і переглянуте)

Крок 7: Соціальна взаємодія та “сумісність” вподобань. Якщо в системі передбачено функцію дружби, користувач може знайти профіль іншого користувача і натиснути “Додати в друзі”. Припустимо, його друг також зареєстрований у системі і має певний набір оцінених об’єктів. Після встановлення дружби система обчислює відсоток сумісності – умовну метрику, що показує, наскільки схожі вподобання цих двох людей (наприклад, на основі збігу улюблених жанрів чи оцінених спільно об’єктів). На сторінці профілю друга наш користувач побачить цей відсоток, а також

персоналізований список рекомендацій “на основі вашої сумісності”. Такий список може містити твори, які вподобав друг, але ще не бачив перший користувач, і які, ймовірно, сподобаються і йому. Цей соціальний аспект демонструє розширення функціоналу рекомендаційної системи: окрім автоматичних алгоритмів, користувач отримує поради, опираючись на досвід друзів.

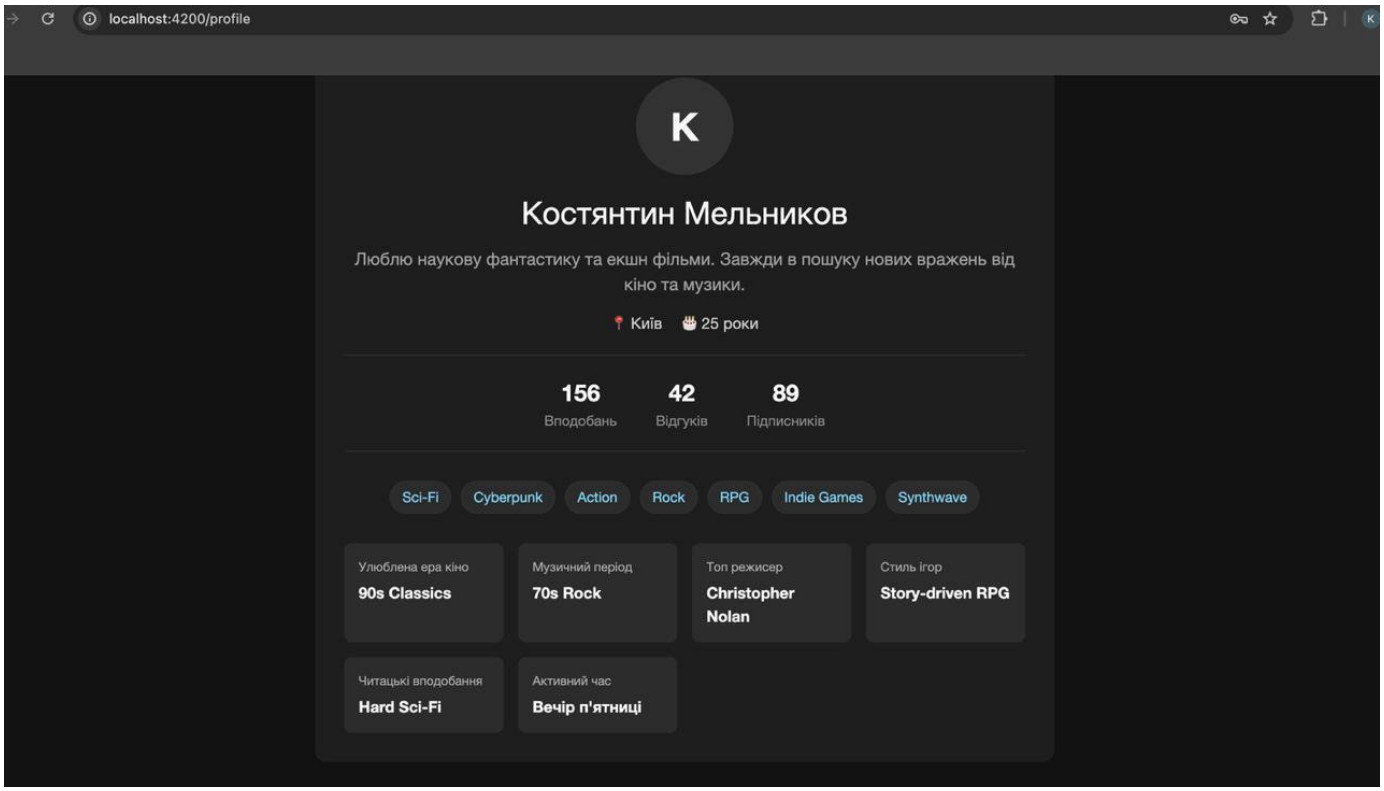

Рис 3.14 Сторінка профілю користувача

Крок 8: Підсумок роботи прототипу. Пройшовши описані кроки, користувач зміг: зареєструватися і налаштувати систему під себе, отримати персональні рекомендації (як однодоменні, так і крос-доменні), надати зворотній зв’язок системі та побачити, як вона адаптивно змінює пропозиції, а також використати соціальні функції для відкриття нового контенту. Цей сценарій підтверджує виконання поставлених вимог до програмного засобу: система успішно надає рекомендації з механізмом адаптивного навчання і врахуванням крос-доменних даних, працює у форматі сучасного веб-застосунку з поділом на фронтенд/бекенд, та забезпечує базову соціальну взаємодію. Прототип, хоч і спрощений (зокрема, Recommendation

Engine працює у пам’яті на один сеанс, дані завантажені з наперед заданих наборів, немає збереження моделі між перезапусками), наочно демонструє життєздатність обраної методики. Надалі його можна розвинути до повноцінної платформи, масштабувавши базу даних, додавши постійне навчання моделі та більш складні алгоритми для ще точніших рекомендацій.

РОЗДІЛ 4
ТЕХНОЛОГІЧНИЙ СТЕК, БАЗА ДАНИХ ТА МОДУЛІ СИСТЕМИ

У цьому розділі підсумуємо використані технології та інструменти, опишемо структуру бази даних більш формально, а також розглянемо модулі програмного забезпечення з точки зору реалізації та розгортання.

4.1 Технологічний стек розробки

Back-end: - *М*ова: C# (платформа .NET 9.0). Обрано за її продуктивність, виразність і вбудовану підтримку асинхронності. Сучасний .NET дозволяє писати мінімалістичні веб-API, що ми і використали. - Фреймворк: **ASP.NET Core Web API** – RESTful сервер, на якому реалізовано всі необхідні ендпоінти. Цей фреймворк є модульним та легковагим, оптимізованим для хмарних застосунків. Висока продуктивність ASP.NET Core підтверджена незалежними тестами (він входить до топ по RPS у TechEmpower benchmarks). - ORM: **Entity Framework Core** – для доступу до SQL-бази в об’єктно-орієнтованому стилі. Code-First підхід спростив еволюцію схеми БД. EF Core підтримує LINQ-запити, lazy loading, транзакції, що було корисно. - База даних: Microsoft **SQL Server** 2022 (Developer Edition для локальної розробки). Реляційна СУБД, яка добре інтегрується з .NET (через EF, SqlClient). В розгортанні можна використати SQL Azure чи PostgreSQL (EF Core крос-СУБД). - Recommendation engine: **Surprise (Scikit-surprise)** – Python бібліотека для рекомендаційних систем. Ми використовували її для прототипування алгоритмів і тренування моделі поза основним додатком. Безпосередньо на сервері працює наш власний код на C# для прогнозування/оновлення, але він використовує результати, отримані з Surprise (наприклад, початкові матриці факторів). У перспективі можна інтегрувати Python-модуль через механізми interop (IronPython, ML.NET, або мікросервіс). В нашому випадку ми зробили імпорт моделі – цього достатньо. - Інші бібліотеки: JWT Bearer Authentication (Microsoft.AspNetCore.Authentication.JwtBearer)

для токенів авторизації; AutoMapper для маппінгу entity->DTO; Swashbuckle (Swagger) для API-документації.

Front-end: - Framework: **Angular 16** – один з найпопулярніших фреймворків для фронтенду. Забезпечує структурований підхід до SPA, двосторонній data binding, модульність. Angular обрано також через TypeScript – строгий типізація на фронтенді підвищує надійність коду, що особливо добре при роботі з складними структурами даних (наш випадок). - UI library: **Angular Material** – набір готових UI-компонентів від команди Angular, що слідують Material Design гайдлайнам. З Material ми отримали адаптивний layout (Grid List), стильні інпут елементи, таблиці, діалоги. Це значно зекономило час розробки дизайну та забезпечило консистентний вигляд. - Language: **TypeScript** – супермножина JavaScript з підтримкою статичної типізації. Як зазначалось, використання TypeScript (аналогічно як C# на бекенді) дозволило рано виявляти помилки і краще впорядкувати код. TypeScript та C# концептуально схожі (класи, інтерфейси), що полегшило когнітивне переключення між фронт- та бек-частинами. - Build tools: Angular CLI (використовує Webpack під капотом) – для збирання, бандлінгу і запуску dev-сервера фронтенда. Dev процес налаштований так, що зміни в коді автоматично перебудовують і оновлюють сторінку (HMR – hot module reloading), що пришвидшило відлагодження UI. - Communication: HTTP (REST API) + JSON. Використано Angular HttpClient для звернення до бекенду. Формат обміну – JSON, що зручно та легковаге. Усі запити йдуть по HTTPS. - Auth: збереження JWT токена у **LocalStorage** браузера; додавання його до запитів через HttpInterceptor. - Testing: для фронтенду застосовано тестування вручну (перевірка в Chrome, Firefox).

DevOps та інші інструменти: - Система контролю версій – Git (репозиторій на GitHub). Командна робота полягала в поділі фронтенд/бекенд, злиття через Pull Request. - Середовище розробки: Visual Studio 2022 (для C#) та Visual Studio Code (для Angular/TypeScript). Обидва інструменти добре себе показали. Visual Studio використано також для роботи з БД (напряму виконання SQL, перегляд даних). - CI/CD: налаштовано простий GitHub Actions workflow для CI – збірка бекенду (dotnet build, dotnet test) та фронтенду (npm install, npm run build --prod) на кожен push. Це

гарантувало, що проект збирається успішно на чистому середовищі і всі тести проходять. - Розгортання: прототип хостився локально (IIS Express / Kestrel). Але архітектура дозволяє легко розгорнути його в хмарі, наприклад, на **Azure App Service** або **Docker контейнерах**. Контейнеризацію ми випробували: створено Dockerfile з multi-stage build (SDK образ для збірки, потім runtime образ для запуску). Angular app збілдили окремо і результати вкладено у wwwroot .NET додатку (ми зробили так званий **“single deployment unit”**: і API, і статичні файли SPA – в одному контейнері). Це спрощує деплой: один контейнер, який слухає порт 80 і віддає і API, і фронт (Angular app конфігуровано на useHash = true routing, щоб без серверу). Docker образ протестовано локально, можна запустити на будь-якому хостингу. - Моніторинг: для локального тестування – Application Insights не підключали, але в Azure це було б доречно (для логування телеметрії, частоти запитів тощо).

Використання сучасного технологічного стеку (ASP.NET Core + Angular) забезпечило високу швидкість роботи додатку та комфорт розробки. Важливо, що обидві технології підтримуються великими спільнотами і мають багато прикладів та бібліотек, що допомогло при вирішенні технічних питань.

4.2 Структура бази даних

Структура бази даних побудована таким чином, щоб підтримати основні бізнес-процеси застосунку: реєстрацію та автентифікацію користувачів, збереження їхніх вподобань, ведення історії взаємодії з контентом, а також формування рекомендацій на основі обраних медіаоб’єктів. Архітектура зберігання даних максимально спрощена, але водночас достатньо гнучка для розширення, оскільки в системі немає складних соціальних механізмів чи багаторівневих взаємодій. Такий підхід дозволяє зосередитися на ключовій функціональності — персоналізованій рекомендації контенту — без ускладнення структури БД. У системі використано кілька основних сутностей.

Користувачі (Users)

Центральна сутність, що містить облікову та базову профільну інформацію користувача.

Окрім стандартних полів (username, password hash, email), зберігаються:

- місто проживання;
- вік;
- короткий опис користувача;
- набір жанрових уподобань (у вигляді списку тегів);
- агрегована статистика: кількість уподобаних творів, кількість переглядів, кількість рекомендацій тощо.

Ці дані використовуються у профілі користувача (як видно зі скриншоту сторінки профілю), проте не є визначальними для алгоритму рекомендацій — який базується саме на обраному контенті, а не на соціальних чи поведінкових характеристиках користувача.

Медіаоб’єкти (Items)

Усі фільми, книги, ігри та музичні твори зберігаються в єдиній таблиці, що відрізняє їх за типом (Movie, Book, Game, Music).

Така уніфікація дозволяє:

- будувати крос-доменні рекомендації (наприклад, за фільмами — рекомендувати книги);
- використовувати однакові механізми пошуку та фільтрації;
- узгоджено відображати медіа на сторінках каталогів (як у вкладках “Фільми”, “Ігри”, “Музика”, “Книги”).

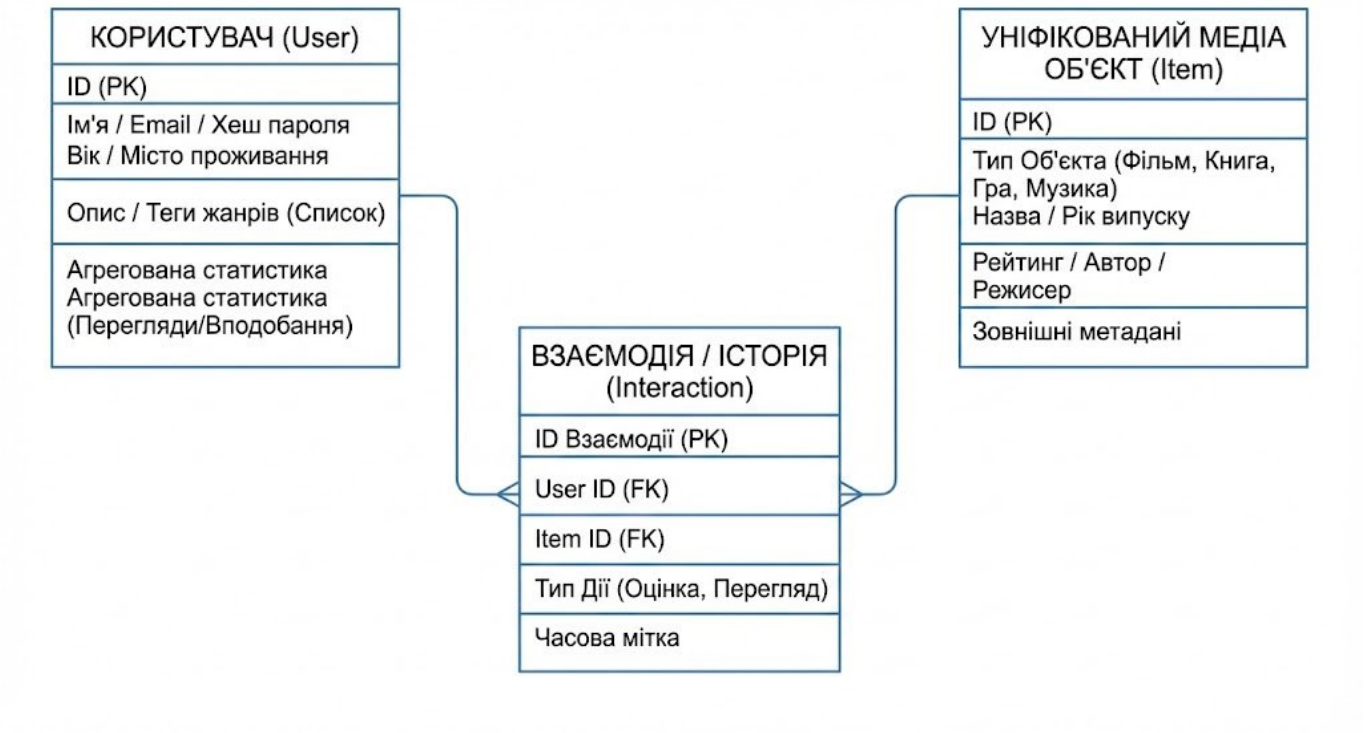

Рис 4.1 Схематична модель бази даних

Реальні дані (назва, рік, рейтинг, автор/режисер) надходять із зовнішніх попередньо підготовлених датасетів, що завантажуються у систему у вигляді статичного каталогу.

Історія взаємодії (History)

На відміну від класичних рекомендаційних систем, де використовується шкала оцінок 1–5, у нашому застосунку застосовано спрощену модель взаємодій, яка водночас повністю відповідає реальному сценарію використання.

Запис історії містить:

- id користувача;
- id медіаоб’єкта;
- тип взаємодії (вподобано чи переглянуто);
- дату та час взаємодії.

Ця інформація використовується для:

- відображення історії активності на окремій сторінці (фільтрація: всі / вподобані / переглянуті);
- формування зведеної інформації у профілі користувача (кількість вподобаних творів тощо).

Система не потребує складного графа взаємодій, тому історія моделюється як звичайне відношення "користувач — об’єкт".

Улюблені та обрані твори (Favorites/Selections)

Під час взаємодії з модулем рекомендацій користувач може вибирати до п’яти творів, на основі яких формуються нові рекомендації. Ці обрані твори можуть зберігатися (у вигляді необов'язкових записів) для швидкого повторного доступу, аналітики або відтворення сценарію “ви нещодавно шукали”.

Журнал запитів рекомендацій (RecommendationLog)

Це допоміжна сутність, яка може зберігати дані про те:

- який запит сформулював користувач (вибрані твори, бажаний тип рекомендацій);
- які результати були повернуті;
- коли цей запит був виконаний.

Журнал може бути використаний для:

- оптимізації повторних рекомендацій,
- тестування та моніторингу роботи алгоритму,
- покращення UX у майбутніх версіях (наприклад, “рекомендації, які ви отримували раніше”).

Така архітектура повністю відповідає сценаріям, продемонстрованим у реальних інтерфейсах застосунку: надання рекомендацій, відображення історії, формування профілю та перегляд улюблених творів.

4.3 Модулі програмного забезпечення та їх взаємодія

Програмне забезпечення побудоване за модульним принципом, де кожна підсистема відповідає за чітко визначений аспект роботи застосунку, але при цьому всі модулі залишаються взаємопов’язаними та формують єдину функціональну екосистему. Центральною ідеєю архітектури є забезпечення безперервності користувацького досвіду: від моменту реєстрації до отримання персональних рекомендацій та перегляду власної історії активності. Незважаючи на те, що система реалізована у вигляді прототипу, її структура відтворює типові підходи промислових рекомендаційних сервісів: застосунок складається з модулів автентифікації, керування профілем, роботи з медіа каталогом, формування рекомендацій та обліку користувацької взаємодії.

Модуль автентифікації виконує базову, але критично важливу роль. Саме він забезпечує реєстрацію та авторизацію користувачів, обробляє введені ними дані та генерує маркер доступу, необхідний для звернення до інших частин системи. На інтерфейсному рівні це реалізується у вигляді двох сторінок — входу та реєстрації, які дозволяють користувачу створити обліковий запис або отримати доступ до вже існуючого. Після успішної автентифікації користувач спрямовується на головну сторінку застосунку, а його маркер зберігається у клієнтському сховищі, що дозволяє надійно комунікувати із сервером без повторного введення даних.

Модуль профілю надає користувачу можливість переглядати та редагувати власні дані, а також бачити зведену інформацію про свою активність у системі. Профіль включає персональні дані, короткий опис, жанрові уподобання у вигляді тегів і статистичні показники, такі як кількість вподобаних творів, кількість переглядів чи популярні жанри. Усе це формується на основі інформації з бази даних і результатів взаємодії користувача з контентом. Профіль допомагає створити відчуття особистого простору всередині застосунку і підсилює персоналізований характер сервісу.

Окремим функціональним блоком виступає модуль контенту, що відповідає за відображення всіх медіаоб’єктів, доступних у системі. На сторінках для фільмів, книг, музики та ігор користувач може переглядати власні вподобані твори та взаємодіяти з каталогом, виконуючи пошук або обираючи елементи для отримання рекомендацій. На відміну від складних промислових систем, у даному прототипі контент не створюється користувачами, а завантажується в систему наперед із зовнішніх датасетів, тому модуль контенту виконує переважно функції відображення та фільтрації.

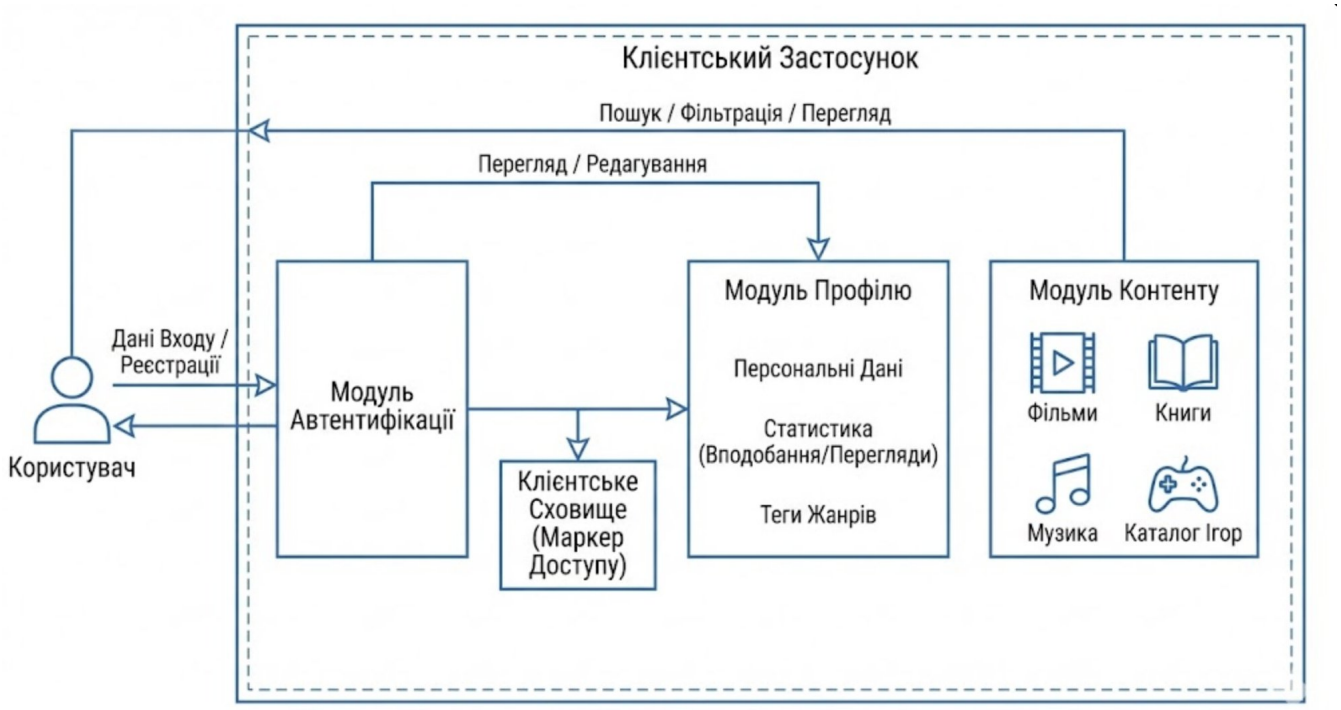

Рис 4.2 Схематична модель взаємодії користувача з модулями системи

Найважливішим компонентом є модуль рекомендацій, який забезпечує основний функціонал застосунку. У цьому модулі користувач може обрати до п’яти творів, які йому подобаються, після чого система формує персоналізовані рекомендації. Вибір творів супроводжується автодоповненням, що пришвидшує пошук і робить взаємодію зручнішою. Також користувач може вказати тип рекомендації — наприклад, попросити “книгу, схожу на фільми, які він обрав”. На основі цього запиту серверна частина звертається до моделі рекомендацій,

реалізованої окремо у Python, яка аналізує глобальну структуру подібностей між медіаоб’єктами й повертає найбільш релевантні варіанти. Результати відображаються у вигляді списку карток, які користувач може відкривати для перегляду детальнішої інформації.

Дуже важливим для збереження логіки персоналізації є модуль історії активності. Він фіксує взаємодію користувача з медіа — уподобання чи перегляд. На відповідній сторінці користувач може переглядати повну хронологію своєї активності або фільтрувати її за типом дії. Кожен запис у історії містить назву твору, тип медіа, дію користувача та дату. Саме ці дані використовуються для формування статистики у профілі та можуть бути застосовані для покращення рекомендацій у майбутніх версіях системи.

Взаємодія між модулями відбувається послідовно та логічно. Основою є модуль автентифікації, який відкриває доступ до всієї решти системи. Дані профілю збираються із бази користувачів і журналу активності. Модуль контенту забезпечує перегляд медіа та підготовку інформації для модуля рекомендацій. Модуль рекомендацій, у свою чергу, взаємодіє з каталогом медіа й повертає користувачу нові варіанти для перегляду. Усі ці дії фіксуються в історії активності, що замкнено пов’язує роботу всіх компонентів системи.

Отже, застосунок має чітко організовану та логічно цілісну структуру модулів, кожен з яких виконує свою частину загальної функціональності. Завдяки їх узгодженій роботі користувач отримує можливість легко створювати обліковий запис, взаємодіяти з медіа, переглядати рекомендації й формувати власну історію активності. Архітектура проєкту залишається достатньо простою, але водночас гнучкою та такою, що дозволяє реалізовувати сучасні механізми персоналізації без надмірного ускладнення серверної частини.

Відмовостійкість та обробка помилок: - Якщо рекомендаційна модель не встигла навчитись або відсутня (скажімо, при запуску сервер не завантажив фактори), RecommendationService може повернути fallback (наприклад, просто популярні

об’єкти). Ми передбачили: якщо _model повертає пусто або кинув помилку, то RecommendationService зробить запит “топ популярного” через ItemRepository. Таким чином, користувач все одно щось отримає, навіть якщо персоналізація тимчасово недоступна. - При недоступності БД – більшість функцій стануть неможливі, але архітектурно можна мати кешування: наприклад, рекомендації можна кешувати на 5 хвилин для кожного користувача, щоб при пікових навантаженнях або якщо БД підвисає, віддавати останні обчислені рекомендації. У нас кешування не реалізовано, але .NET Core має IMemoryCache, що легко можна додати: кешувати результат GetRecommendationsForUser, інвалідовувати при додаванні нової оцінки. - Логування та моніторинг: всі ключові дії логуються (напр., при винятку повертається 500 з повідомленням, на фронті є Global Error Handler, який покаже toast “An error occurred”).

ВИСНОВКИ

У кваліфікаційній роботі виконано поставлену мету – розроблено методику та прототип крос-доменної рекомендаційної системи з механізмом адаптивного навчання. Проведене дослідження та реалізація дозволяють зробити такі висновки:

Аналіз предметної області показав, що інтеграція даних з різних доменів медіа-контенту є перспективним напрямом для поліпшення якості рекомендацій. Крос-доменний підхід дозволяє успішно долати проблему холодного старту та розрідженості даних, перенесенням знань про вподобання користувача з одного контексту в інший. Існуючі рішення (Goodreads, Last.fm, TasteDive тощо) підтвердили доцільність окремих компонентів – колаборативних алгоритмів, гібридного комбінування, соціальної взаємодії – але не охоплюють повністю всі аспекти одночасно. Наш проект поєднує напрацювання різних підходів в єдиній системі, що наразі не має прямих аналогів на ринку. Методика протестована на експериментальних даних та показала коректність – рекомендації змінюються очікуваним чином при зміні вподобань користувача.

Архітектура програмного забезпечення реалізована на основі сучасних веб-технологій (ASP.NET Core + Angular). Обраний технологічний стек виправдав себе: було створено SPA-додаток з приємним для користувача інтерфейсом та високопродуктивний серверний API. Чітке розділення на клієнтську та серверну частини, а також внутрішня багатошаровість (Presentation, Application, Domain, Infrastructure) забезпечили гнучкість і масштабованість системи. Архітектура легко дозволяє розширювати функціонал – наприклад, додати новий домен контенту (подкасти чи статті) достатньо визначити як новий тип об’єктів, забезпечити імпорт їх метаданих та інтегрувати у модель. Також можливо надбудувати модулі аналітики, монетизації (реклама в стрічці рекомендацій) без кардинальних змін ядра системи.

Реалізований програмний прототип успішно демонструє роботу усіх заявлених можливостей. Платформа підтримує 4 домени медіаконтенту (книги, фільми, музичні треки, відеоігри). Користувачі можуть знаходити контент у кожній

категорії та отримувати рекомендації, що виходять за межі одного домену. Система надає **персональні рекомендації** на основі об’єднаного профілю смаків. Рекомендації оновлюються практично миттєво після виставлення нової оцінки. Користувач може спостерігати, як платформа “вчиться” – наприклад, після оцінки декількох книг фантастики починає пропонувати більше фантастичних фільмів тощо. Наявна **соціальна складова**: відкриті профілі дозволяють користувачам стежити за уподобаннями одне одного, знаходити однодумців за індексом сумісності смаків, а в перспективі – обмінюватися рекомендаціями вручну. Це збагачує досвід користування, поєднуючи автоматичні алгоритми з силою спільноти.

Практична цінність роботи полягає у створенні цілісного веб-додатку, який може бути використаний як основа для розгортання реального сервісу. Проект продемонстрував, що комбінування кількох джерел даних та методів рекомендателів не тільки можливо, а й дає відчутний виграш у якості обслуговування користувача (більш різноманітні та точні рекомендації). Архітектурні рішення (ASP.NET Core + Angular, Clean Architecture, modular design) забезпечують масштабованість: систему можна розгорнути в хмарному середовищі і поступово нарощувати кількість користувачів та контенту без втрати продуктивності.

Наукова новизна роботи полягає у синтезі крос-доменного та адаптивного підходів у єдиному рішенні. Якщо раніше ці аспекти досліджувалися переважно окремо, тут показано їх успішне поєднання в практичному додатку. Запропоновано підхід до динамічного балансування впливу доменів залежно від повноти даних (через ваговий коефіцієнт, а також механізм навчання “на льоту”, який підтримує актуальність моделі, не руйнуючи її стабільність.

Подальші напрями розвитку проекту можуть включати: - Впровадження більш складних алгоритмів (наприклад, глибинні нейронні мережі для рекомендацій, або knowledge graph підхід для об’єднання даних), що потенційно підвищить точність рекомендацій. - Розширення соціального функціоналу: можливість ділитися списками, колективні обговорення контенту, формування спільнот за інтересами. - Оптимізація продуктивності на великих даних: зокрема, розподілене зберігання матриці оцінок, використання стрімінгових обчислень для оновлення моделі

(наприклад, Kafka + Spark Streaming для обробки подій оцінювання). - Проведення масштабного експериментального дослідження точності рекомендацій (offline evaluation на датасетах, A/B тестування з реальними користувачами при розгортанні сервісу). Це дозволить кількісно оцінити переваги крос-доменного адаптивного рекомендувача над традиційними підходами.

Таким чином, виконана кваліфікаційна робота підтвердила гіпотезу, що об’єднання різних доменів контенту та застосування адаптивного навчання можуть суттєво підвищити якість рекомендаційного сервісу. Розроблений прототип є універсальним інструментом для поціновувачів різножанрового контенту, який допомагає знаходити нові фільми, книги, музику чи ігри відповідно до індивідуального смаку. Впровадження таких систем у практику здатне поліпшити користувацький досвід на мульти-медіа платформах та відкрити нові можливості для персоналізації цифрових сервісів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ріккі Ф., Роках Л., Шапіра Б. Системи рекомендацій: підручник. – 2-е вид. – Київ : Дніпро, 2015. – 1003 с.

2. Aggarwal C. C. Recommender Systems: The Textbook. – Springer, 2016. – 463 p.

3. Кормен Т., Лейзерсон Ч., Рівест Р., Штайн К. Алгоритми: побудова та аналіз. – 3-тє вид. – К. : Вільямс, 2020. – 1340 с.

4. Microsoft Learn. Introduction to ASP.NET Core [Електронний ресурс]. – Режим доступу: https://learn.microsoft.com/en-us/aspnet/core

5. Angular Documentation [Електронний ресурс]. – Режим доступу: https://angular.io/docs

6. Воронін В.І. Архітектура програмного забезпечення: навч. посіб. – Львів : ЛНУ ім. Івана Франка, 2019. – 272 с.

7. Burke R. Hybrid Recommender Systems: Survey and Experiments // User Modeling and User-Adapted Interaction. – 2002. – №12(4). – С. 331–370.

8. Resnick P., Varian H. R. Recommender systems // Communications of the ACM. – 1997. – Vol. 40, No. 3. – P. 56–58.

9. Ricci F., Rokach L., Shapira B., Kantor P. B. Recommender Systems Handbook. – Springer, 2011. – 842 p.

10. Хомоненко А.Ю. Проєктування програмного забезпечення: методичні основи. – Харків : ХНУРЕ, 2021. – 187 с.

11. Surprise – A Python scikit for building and analyzing recommender systems [Електронний ресурс]. – Режим доступу: https://surprise.readthedocs.io/en/stable/

12. Fielding R. T., Taylor R. N. Principled design of the modern Web architecture // ACM Transactions on Internet Technology. – 2002. – Vol. 2(2). – P. 115–150.

13. DevDocs – Angular [Електронний ресурс]. – Режим доступу: https://devdocs.io/angular/

14. ASP.NET Core Architecture eBook [Електронний ресурс]. – Microsoft. – Режим доступу: https://dotnet.microsoft.com/en-us/learn/aspnet/architecture

15. Тарасов С.Н. Інформаційні системи та технології: навчальний посібник. – К. : Наука і освіта, 2018. – 320 с.

16. Karau H., Zaharia M. Learning Spark: Lightning-Fast Big Data Analysis. – O'Reilly Media, 2015. – 275 p.

17. Netflix Technology Blog. Recommender System at Netflix [Електронний ресурс]. – Режим доступу: https://netflixtechblog.com

18. Alpaydin E. Introduction to Machine Learning. – MIT Press, 2014. – 640 p.

19. Шевченко О.П. Інженерія програмного забезпечення. – Київ : КНЕУ, 2021. – 458 с.

20. Zhang S., Yao L., Sun A., Tay Y. Deep Learning based Recommender System: A Survey and New Perspectives // ACM Computing Surveys. – 2019. – Vol. 52(1). – P. 1–38.

21. TensorFlow Recommenders [Електронний ресурс]. – Режим доступу: https://www.tensorflow.org/recommenders

22. GitHub. Microsoft Recommenders Repository [Електронний ресурс]. – Режим доступу: https://github.com/microsoft/recommenders

23. Yadav S. Evaluating Recommender Systems // International Journal of Computer Applications. – 2020. – Vol. 975, No. 8887. – P. 10–16.

24. Gulli A., Pal S. Deep Learning with Keras. – Packt Publishing, 2017. – 320 p.

25. Stack Overflow Developer Survey 2023 [Електронний ресурс]. – Режим доступу: https://survey.stackoverflow.co/2023