МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«СИСТЕМА ПРОГНОЗУВАННЯ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ У ОНЛАЙН-ПЛАТФОРМАХ»**

Виконав: студент 4 курсу, групи 6.1261
спеціальності 126 Інформаційні системи та технології
(шифр і назва спеціальності)
освітньої програми Інформаційні системи та технології
(назва освітньої програми)
Д.К. Кондратенко
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, PhD, Калюжняк А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент професор кафедри програмної інженерії, доцент, к.ф.-м.н. Кудін О.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра комп'ютерних наук

Рівень вищої освіти бакалавр

Спеціальність 126 Інформаційні системи та технології
(шифр і назва)

Освітня програма Інформаційні системи та технології

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних наук,
д.т.н., професор

_____ Шило Г.М.
(підпис)

" ____ " _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Кондратенку Дмитру Костянтиновичу
(прізвище, ім'я та по-батькові)

1. Тема роботи Система прогнозування персоналізованих рекомендацій у онлайн-платформах

керівник роботи Калюжняк Анастасія Вікторівна, PhD
(прізвище, ім'я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 20 » грудня 2024 року № 2001-с

2. Строк подання студентом роботи 23.05.2025

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

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

1. Постановка задачі
2. Основні теоретичні відомості.
3. Система прогнозування персоналізованих рекомендацій у онлайн-платформах

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

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 23.12.2024

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	05.01.2025	
2.	Збір вихідних даних.	19.01.2025	
3.	Обробка методичних та теоретичних	21.02.2025	
	джерел.		
4.	Розробка першого та другого розділу.	06.04.2025	
5.	Розробка третього розділу.	12.05.2025	
6.	Оформлення та нормоконтроль		
	кваліфікаційної роботи бакалавра.	27.05.2025	
7.	Захист кваліфікаційної роботи.	21.06.2025	

Студент _____ Д.К. Кондратенко
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Калюжняк
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ О.Г. Спиця
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Система прогнозування персоналізованих рекомендацій у онлайн-платформах»: 70 с., 28 рис., 3 табл., 39 джерел.

КОЛАБОРАТИВНА ФІЛЬТРАЦІЯ, МАТРИЧНА ФАКТОРИЗАЦІЯ, ПЕРСОНАЛІЗАЦІЯ, РЕКОМЕНДАЦІЙНА СИСТЕМА, ХОЛОДНИЙ СТАРТ, FIREBASE, FLASK, PYTORCH, REACT, TMDB.

Об’єкт дослідження – процес проєктування та впровадження веб-сервісу, що формує індивідуальні рекомендації контенту на основі історії взаємодій користувачів.

Мета роботи: створення прототипу платформи, здатної прогнозувати ймовірну оцінку фільмів для відомого користувача та надавати релевантні рекомендації.

Методи дослідження – об’єктно-орієнтоване моделювання (UML), стратифікований відбір даних, темпоральне розділення вибірки, градієнтне навчання у PyTorch, порівняльний аналіз підходів до побудови рекомендаційних систем.

У роботі подано обґрунтування вибору матричної факторизації як ядра системи, описано скорочення датасету MovieLens 32M до 40 000 активних профілів, розроблено клієнт-серверну архітектуру на основі Flask (бекенд) та React (фронтенд). Реалізовано модель із 120 латентними факторами (RMSE = 0,87), модулі взаємодії з TMDB API й Firestore. Продемонстровано роботу платформи: автентифікація, головна сторінка з персоналізованими добірками, пошук і сторінка деталей фільму.

Отримані результати підтверджують практичну придатність запропонованого рішення для онлайн сервісів і створюють основу для подальшого розширення функціоналу системи.

SUMMARY

Bachelor’s Qualifying Thesis «System for Predicting Personalized Recommendations on Online Platforms»: 70 pages, 28 figures, 3 tables, 39 references.

COLLABORATIVE FILTERING, MATRIX FACTORISATION, PERSONALISATION, RECOMMENDER SYSTEM, COLD START, FIREBASE, FLASK, PYTORCH, REACT, TMDB.

Object of the study – the process of designing and implementing a web service that generates individual content recommendations based on the history of user interactions.

Aim of the study: to create a prototype platform capable of predicting the likely rating of movies for a known user and providing relevant recommendations.

Methods of research – object-oriented modeling (UML), stratified data selection, temporal sample splitting, gradient learning in PyTorch, comparative analysis of approaches to building recommender systems.

The paper presents the rationale for choosing matrix factorization as the core of the system, describes the reduction of the MovieLens 32M dataset to 40,000 active profiles, and develops a client-server architecture based on Flask (backend) and React (frontend). The model with 120 latent factors (RMSE = 0.87), modules for interaction with TMDB API and Firestore were implemented. The platform is demonstrated: authentication, the main page with personalized selections, search, and the movie details page.

The obtained results confirm the practical applicability of the proposed solution for online services and create a basis for further expansion of the system's functionality.

ЗМІСТ

Завдання на кваліфікаційну роботу……...2
Реферат 7
Summary 8
Вступ 9
1 Огляд методів побудови рекомендаційних систем 11
1.1 Аналіз задачі 11
1.1.1 Проблема інформаційного перевантаження 12
1.1.2 Визначення рекомендаційної системи та її основні задачі 13
1.1.3 Значення рекомендаційних систем для користувачів і бізнесу . 14
1.1.4 Практики побудови рекомендаційних систем на відомих платформах 14
1.2 Класифікація та гібридизація рекомендаційних підходів 15
1.2.1 Основні підходи до формування рекомендацій..... 15
1.2.2 Недоліки основних типів рекомендаційних підходів 16
1.2.3 Гібридні системи як відповідь на обмеження базових підходів 18
1.2.4 Ключова роль колаборативної фільтрації в сучасних системах 19
1.3 Методи колаборативної фільтрації (CF)..... 20
1.3.1 Методи на основі сусідства (memory-based collaborative filtering) 20
1.3.2 Матрична факторизація як базовий model-based підхід 22
1.3.3 Уточнення моделі зміщеннями 22
1.3.4 Навчання параметрів 23
1.3.5 Сильні й слабкі сторони підходу матричної факторизації 23
1.4 Опрацювання проблеми «холодного старту» 24
1.4.1 Критичність проблеми «холодного старту» (Cold Start) 24
1.4.2 Типові стратегії подолання проблеми «холодного старту» 25

1.4.3 Обрана стратегія для прототипу: зважений рейтинг (Bayesian Average) 26
2 Проєктування та архітектура прототипу рекомендаційної платформи 28
2.1 Опис функціональної та динамічної моделі системи 28
2.1.1 Діаграма варіантів використання 28
2.1.2 Діаграма розгортання 29
2.1.3 Діаграма діяльності процесу формування рекомендацій 31
2.1.4 Діаграми послідовностей додавання оцінки 33
2.1.4 Діаграми послідовностей надання рекомендацій 34
2.1.5 Діаграма класів 36
2.2 Опис вхідних даних для навчання нейромережі 37
3 Реалізація проєкту 39
3.1 Підготовка даних та навчання моделі матричної факторизації 39
3.1.1 Обґрунтування скорочення датасету та вибір методики 39
3.1.2 Підсумкова конфігурація робочого набору 40
3.1.3 Методологія валідації 41
3.1.4 Розділення даних на тренувальний і тестовий набори 42
3.1.5 Розробка структури нейронної мережі 43
3.1.6 Вибір гіперпараметрів моделі 44
3.1.7 Процес навчання та збереження моделі 45
3.2 Налаштування серверного середовища та завантаження артефактів (ml_setup.py) 46
3.2.1 Створення допоміжних словників 46
3.2.2 Формування списку популярних назв 47
3.2.3 Завантаження навченої моделі та словників індексації 47
3.2.4 Експорт готових об’єктів 49
3.3 Розробка серверного API на Flask (app.py) 49
3.3.1 Ініціалізація та конфігурація app.py 49
3.3.2 Механізм автентифікації та авторизації 50
3.3.3 Модуль взаємодії з TMDB API (tmdb_client.py) 51

3.3.4 Модуль взаємодії з базою оцінок (firestore_service.py) 53
3.3.5 Ключові REST-ендпоінти серверної частини 53
3.4 Розробка клієнтської частини (React) 55
3.4.1 Модуль взаємодії з API flaskApiClient.js 55
3.4.2 Клієнтський модуль firebase.js 55
3.4.3 Демонстрація функціоналу розробленої онлайн-платформи 56
Висновки 63
Перелік посилань 65

ВСТУП

Стрімке зростання онлайн-каталогів відеоконтенту перевищує людську можливість самостійно опрацьовувати інформаційні потоки: навіть за кілька хвилин перегляду стрічки користувач стикається з сотнями найменувань, що робить вибір випадковим і часто неефективним. На тлі такої «інформаційної перенасиченості» саме рекомендаційні системи забезпечують платформам конкурентоспроможність, а глядачам – відчутний приріст задоволення від сервісу. Отже, розроблення надійного алгоритму персоналізованого добору фільмів має як практичну, так і наукову цінність.

Об’єктом дослідження є процес формування індивідуальних рекомендацій у веб-середовищі.

Предметом є – методи колаборативної фільтрації, зокрема матрична факторизація, та програмні засоби їх реалізації.

Мета роботи – спроєктувати і створити прототип онлайн-платформи, що на основі скороченого набору MovieLens 32M надає списки рекомендованих фільмів для зареєстрованих користувачів.

Для досягнення мети поставлено такі завдання:

- виконати огляд існуючих підходів до рекомендаційних систем і обґрунтувати вибір статичної моделі матричної факторизації;
- підготувати репрезентативний піднабір MovieLens, дотримуючись балансу між обсягом і збереженням розподілу активності користувачів;
- спроєктувати архітектуру прототипу за схемою «React – Flask API – PyTorch», описати взаємодію модулів за допомогою UML;
- реалізувати навчання статичної моделі та інтеграцію з зовнішнім API TMDb для збагачення метаданих;
- розробити механізми автентифікації на базі Firebase Authentication і збереження оцінок у Firestore;

– оцінити точність рекомендацій на темпоральній вибірці та проаналізувати результати.

Структура роботи відповідає поставленим завданням: у першому розділі подано теоретичні передумови та вибір алгоритму; у другому – описано проєктні рішення, схему даних і програмні модулі; третій містить деталі реалізації та методику експериментальної перевірки. Завершує роботу висновок із рекомендаціями щодо подальшого розвитку системи.

1 ОГЛЯД МЕТОДІВ ПОБУДОВИ РЕКОМЕНДАЦІЙНИХ СИСТЕМ

1.1 Аналіз задачі

У даній кваліфікаційній роботі визначено завдання - створити веб-сервіс який буде не просто перелічувати наявні фільми, а підбирати кожному відомому відвідувачу ті стрічки, що, за досвідом попередніх глядачів, мають найбільший шанс сподобатися саме йому. На папері завдання виглядає нескладно; проте, щойно переходиш від загальної ідеї до конкретики, одразу з’являються кілька суттєвих перепон.

Перша з них стосується самих глядацьких вподобань. Дані, з якими доводиться працювати, зазвичай мають вигляд журналів подій, що фіксують ідентифікатор користувача, переглянутий об’єкт, часову позначку та виставлений рейтинг. Ці записи нагадують вибіркову мозаїку: кожен користувач оцінив зовсім невелику частку каталогу, а отже про решту його смаків доводиться судити опосередковано. У наборі, на який спиратиметься експериментальна частина роботи, типовий глядач залишив оцінки десь для трьох-чотирьох десятків назв, тоді як сам каталог налічує десятки тисяч позицій. Відновити «мовчазні» переваги, тобто здогадатися, чому людині сподобався один фільм і чи сподобається інший, виходячи з такої малочисельної вибірки, – є складною інженерною задачею.

Друга група труднощів випливає з опису самих стрічок. Якщо жанр або рік виходу ще можна занести в табличку без зайвих втрат, то, скажімо, атмосферу фільму, темп оповіді чи особливості операторської роботи одним рядком не зафіксуєш. Проте ці нюанси можуть виявитися визначальними для глядача. Отже, потрібен спосіб подати багатоманітну інформацію – текстову, візуальну, інколи й аудіо – у стандартизованій, «машинозчитуваній» формі, зберігши при цьому змістовну точність.

Третій фактор – розмір і нерівномірність самих даних. Матриця «користувач х фільм» у таких проєктах майже цілком порожня; це означає, що обраний алгоритм має вміти працювати в умовах розрідженості та все одно відповідати дуже швидко – буквально за частки секунди, аби не змушувати відвідувача чекати.

У підсумку основні виклики можна сформулювати так:

– навчитися витягувати з уривчастих оцінок стійке уявлення про смаки кожного глядача;
– подати різноманітні властивості фільмів у компактному та змістовному вигляді;
– побудувати модель, що лишається точною й оперативною попри розрідженість і великий масштаб даних.

Подальший теоретичний огляд методів-кандидатів і стане підґрунтям для вибору конкретного підходу, придатного для реалізації цих вимог.

1.1.1 Проблема інформаційного перевантаження

Проблема інформаційного перевантаження загострилася тоді, коли соцмережі перетворили кожного користувача на невпинного генератора контенту, а потік варіантів почав перевищувати межі усвідомленого вибору [1]. Коли Netflix чи YouTube завантажують тисячі годин відео щохвилини, класичний підхід «перегляну заголовки й оберу» перестає працювати. Користувач, не маючи внутрішніх «фільтрів», втомлюється, а втома породжує відмову від вибору. З погляду когнітивної психології це типова реакція на перевантаження стимулом, що підтримує правило Гікса [2]: що більше варіантів, то довше приймається рішення й тим більша ймовірність, що воно буде випадковим або взагалі не буде ухвалене.

Онлайн-платформа спостерігає це явище в метриках: зменшується тривалість сесії, зростає частка «порожніх» переглядів, падає конверсія. Рекомендаційна система тут виконує роль зовнішнього когнітивного протеза,

який у реальному часі стискує вектори можливостей до розміру, з яким людина ще здатна взаємодіяти без стресу.

1.1.2 Визначення рекомендаційної системи та її основні задачі

У технічних джерелах під рекомендаційною системою розуміють комплекс алгоритмів, що за спільними закономірностями у минулих взаємодіях користувачів з об’єктами передбачає ймовірну користь нової взаємодії. Формулювання звучить абстрактно, та на практиці воно зводиться до трьох фундаментальних проблем.

Перша – прогноз рейтингу. Якщо користувач уже оцінив кілька статей чи фільмів, то з якою точністю ми можемо передбачити його оцінку для ще не спробованого контенту? Матрична факторизація й латентні векторні простори стали кращою відповіддю на це запитання після знаменитого конкурсу Netflix Prize [3], де прості лінійні моделі поступилися місцем ансамблям зі сотень факторів.

Друга – формування топ-N списків. Тут дослідника цікавить не абсолютна величина уподобання, а перша десятка рекомендацій, бо саме стільки бачить око на екрані смартфона. Прецизійні метрики типу precision@N або NDCG винайдені лише тому, що класичні середньоквадратичні помилки гірше відбивають реальний досвід користувача.

Третя – ранжування у динамічному середовищі. Наприклад, домашня сторінка Amazon складається з модулів-«каруселей», кожен із яких мусить показати не просто те, що ймовірно сподобається, а й те, що додасть різноманіття, не повторюватиметься з іншими модулями і водночас забезпечить високу конверсію. Під капотом працюють стохастичні бандити й learning-to-rank-моделі, що переглядають тисячі варіантів у мілісекунди, аби знайти компроміс між експлуатацією відомого й дослідженням невідомого.

1.1.3 Значення рекомендаційних систем для користувачів і бізнесу

З позиції користувача рекомендаційна система виконує щонайменше три корисні функції. Перша – персоналізація. Досвід взаємодії стає продовженням власних смаків, а не анонімною вітриною супермаркету. Друга – відкриття нового. Парадоксально, але саме алгоритм, а не редактор, частіше приводить слухача до незнайомого жанру чи покупця до бренду, про який той ще вчора не чув. Третя – економія часу: рішення «що подивитися сьогодні ввечері» приймається за секунди, а не за полюванням по форумам і спискам IMDb.

Для платформи ці самі властивості конвертуються в метрики. Персоналізація підвищує ймовірність кліку і глибину перегляду, отже – дохід від реклами або передплати. Ефект несподіваних відкриттів зменшує ризик, що користувач знудиться й піде до конкурента.

1.1.4 Практики побудови рекомендаційних систем на відомих платформах

На відеохостингу YouTube щохвилини публікують сотні годин контенту [4], тому класичний підхід item-to-item, із яким платформа стартувала [5], виявився замалим. Сучасна архітектура побудована як двофазний конвеєр: швидка модель на основі ембеддингів зводить пошук до кількох сотень кандидатів, а потім глибинна нейронна мережа ранжує їх, комбінуючи сигнали переглядів, пошуку і вподобань [5, 6]. Така конфігурація дозволяє одночасно тримати масштаб каталогу й оновлювати персональні стрічки за частки секунди [5].

У Netflix еволюція йшла від rule-based Cinematch до ансамблів із матричної факторизації та глибоких мереж [7]. Після конкурсу Netflix Prize [3] латентні фактори лягли в основу системи, а пізніше їх поєднали з рекурентними й контекстними моделями, які враховують порядок переглядів, час доби й навіть генерують персоналізовані обкладинки [7, 8]. У підсумку

рекомендаційний блок став багатошаровим гібридом, що підтримує низький відтік передплатників і високу частку переглядів, ініційованих саме порадами системи.

Музичний сервіс Spotify робить ставку на поєднання аудіо-аналітики та поведінкових ембеддингів. Для кожного треку обчислюють десятки акустичних ознак (енергійність, танцювальність, валентність) [9], а паралельно навчають word2vec-подібну модель на послідовностях плейлистів [10]. Коли дві пісні часто трапляються поруч, їхні вектори зближуються; далі до цього простору додають контентні характеристики, отримуючи єдиний профіль треку, що працює і для популярних хітів, і для релізів-новачків [10].

В електронній комерції Amazon зробив промисловим стандартом алгоритм item-to-item collaborative filtering [11, 12]. Списки «схожих товарів» обчислюють офлайн за даними спільних покупок, а під час сесії система лише перетинає ці списки для кількох переглянутих позицій [12]. Такий дизайн відокремлює важкі обчислення від онлайнових запитів і масштабовано працює з багатомільйонним каталогом, одночасно підтримуючи персональний контекст користувача.

Хоча кожна платформа використовує власну суміш технік – від ембеддингів і двофазного ранжування до класичної item-to-item фільтрації – усі вони спираються на одні й ті самі базові родини методів та їхні комбінації. Саме ці родини – контентний, колаборативний та їхні гібриди – стануть предметом систематичної класифікації в наступному підрозділі.

1.2 Класифікація та гібридизація рекомендаційних підходів

1.2.1 Основні підходи до формування рекомендацій

Коли в джерелах розглядають «базові» підходи до побудови рекомендацій, зазвичай мають на увазі саме три окремі логіки пошуку

схожості: поведінкову, контентну й соціально-демографічну.

Колаборативна фільтрація(Collaborative Filtering, CF) спирається на гіпотезу, що користувачі з подібною історією взаємодій мають в майбутньому і подібні вподобання. Напростішим підходом такого варіанту алгоритму є пошук «сусідів» у матриці «користувач х фільм» і подальший перенос їхніх оцінок на невідомі клітинки. Для великих наборів даних цей пошук замінюють матричною факторизацією: історію переглядів стискають до векторів кількох десятків вимірів, у яких зберігаються приховані, але стійкі сигнали смаку. Практичний плюс підходу – ефект приємної несподіванки: система здатна запропонувати стрічку, що зовні нічим не нагадує попередні вподобання користувача, проте сподобалася тим, чиї перегляди з ним перетинаються.

Контентно-орієнтована фільтрація (Content-Based, CB), навпаки, розглядає кожний елемент каталогу як набір атрибутів: жанри, ключові слова, технічні характеристики тощо. Із переглянутих раніше назв система формує «профіль інтересів», після чого порівнює його з атрибутами інших стрічок, зазвичай через косинусну подібність TF-IDF-векторів. Перевага очевидна: новинка з докладним описом може потрапити у рекомендації відразу після релізу, навіть якщо її ще ніхто не оцінив. Недолік так само очевидний: алгоритм легко зациклюється, пропонуючи варіації вже відомого, і тим самим зменшує шанс натрапити на щось геть нетипове, але потенційно цікаве.

В свою чергу демографічна фільтрація використовує найпростіший, але інколи єдино доступний сигнал – соціально-демографічні атрибути. Якщо відомі вік, стать або регіон, платформа може стартувати з «колективних смаків» відповідної групи й пропонувати популярне серед однолітків чи мешканців тієї самої місцевості. Такий підхід швидко дає бодай якісь релевантні рекомендації новачкам, проте ціна за швидкість – грубе узагальнення та ризик упустити індивідуальні нюанси смаку.

1.2.2 Недоліки основних типів рекомендаційних підходів

Навіть найпоширеніші типи рекомендаційних систем мають свої слабкі місця. В академічних і промислових оглядах ці вади описують ледь не частіше, ніж самі алгоритми: саме вони підштовхують до пошуку змішаних рішень.

Почнемо розгляд з колаборативної фільтрації. Її головний недолік – «холодний старт». Коли до системи приходить новий користувач або з'являється новий елемент, у матриці «користувач х об'єкт» утворюється порожня колонка чи рядок, і модель просто не має з чого робити висновки. Стаття Cold start у Wikipedia [13] й технічна довідка IBM [14] узгоджено констатують: поки не накопичиться мінімальний обсяг взаємодій, персоналізація «працює всліпу». До цього додається розрідженість даних у великих каталогах – переважна більшість користувачів оцінює лише крихітну частку об'єктів, що робить пошук «сусідів» ненадійним . Навіть Netflix, який має істотно щільнішу матрицю відгуків, визнає у власному Tech Blog [15], що саме холодний старт змушував поєднувати факторні моделі з контентними та редакторськими правилами .

Контентно-орієнтовані алгоритми уникають проблеми нового елемента, адже працюють з описом об'єкта, однак потрапляють у пастку так званої надмірної спеціалізації. IBM Think у довіднику [16] пояснює: система бачить тільки явно задані атрибути й, внаслідок цього, наполегливо пропонує «більше того самого» – книги того ж жанру, товари тієї ж марки, фільми з тим самим актором. Користувацький досвід стає передбачуваним, а ефект новизни різко падає. Якщо ж опис об'єкта бідний або неструктурований, якість рекомендацій деградує ще швидше, бо алгоритм не має матеріалу для порівняння .

В свою чергу, демографічна фільтрація базується майже виключно на віці, статі чи географії, тому її головна вада – надмірне узагальнення. Коли система відштовхується лише від того, що «користувачеві 25 років і він із Запоріжжя», вона неминуче скочується до стереотипних порад і легко ігнорує індивідуальні нюанси смаку. Огляд [17] прямо вказує, що така стратегія підходить лише як початковий «якор» для новачків і швидко втрачає релевантність без тонших поведінкових сигналів .

У сумі ці обмеження пояснюють, чому провідні платформи рідко задовольняються одиничною методикою: аби нейтралізувати слабкі сторони кожної парадигми, інженери комбінують їх у багаторівневі гібриди.

1.2.3 Гібридні системи як відповідь на обмеження базових підходів

Практика великих сервісів показує, що жодна «чиста» стратегія не покриває всіх сценаріїв: колаборативна фільтрація буксує, коли бракує історії, контент-моделі надмірно обмежують горизонти новизни, а демографія грішить узагальненнями. Відтак інженери перейшли до комбінування методів, щоб сильні сторони одного алгоритму компенсували слабкі місця іншого.

Найпоширеніші способи гібридизації – це:

- зважене змішування результатів, де підсумковий бал кожного об’єкта розраховується як лінійна (або байєсова) комбінація кількох моделей;
- каскадні схеми, у яких «легший» модуль відсіює очевидно нерелевантні елементи, а «глибший» ранжує залишок;
- динамічне перемикання: система застосовує контент чи демографію для новачків і переходить на CF, щойно з’являються поведінкові дані [18].

Реалізації різняться деталями, але патерн один: колаборативне ядро підсилюють контентними та контекстними пластами. У Netflix латентна CF-серцевина, що виникла з уроків Netflix Prize, зараз працює пліч-о-пліч з рекурентними мережами й низкою бізнес-правил – саме такий «шаруватий» дизайн покриває й довгострокові смаки, і ситуативний контекст [8]. На YouTube проблему каталогу з сотень мільйонів відео розв’язують двоетапно: швидка модель на ембеддингах формує ядро кандидатів, а глибока нейронна сесійно ранжує їх, поєднуючи поведінкові й контентні сигнали [5]. Amazon, стартувавши з item-to-item CF, додав атрибутний шар, щоб нові або нішеві товари не випадали з поля зору, – такий гібрид мінімізує cold start без втрати масштабованості [11]. У Spotify колаборативні ембеддинги треків з плейлистів об’єднують зі спектральними та текстовими ознаками пісень у єдиний граф-

простір, що дозволяє одночасно ловити латентні зв’язки й рекомендувати свіжі релізи ще до перших прослуховувань [10].

Ці приклади показують: найбільший приріст якості дає саме зв’язка латентної колаборативної компоненти із зовнішніми описовими або контекстними даними. CF забезпечує неочевидні «стрибки» уподобань, а додаткові шари знімають проблеми холодного старту й вузької спеціалізації. Тому далі, фокус зміститься розгляді на колаборативній фільтрації як на несучій конструкції сучасних гібридних систем.

1.2.4 Ключова роль колаборативної фільтрації в сучасних системах

Попри те, що сучасні системи рекомендацій дедалі частіше мають багаторівневу архітектуру, їхнім «несучим каркасом» залишається саме колаборативна фільтрація. Річ не лише у тому, що цей підхід історично заклав основу перших великих успіхів – згадаймо, як після Netflix Prize методи матричної факторизації стали галузевим стандартом точності [19, 20]. Головне-теоретичне підґрунтя полягає в тому, що CF оперує не атрибутами об’єктів, а патернами колективної поведінки: система виявляє приховані зв’язки у матриці «користувач x елемент» і цим виходить за межі очевидної схожості контенту. Саме тому рекомендація, згенерована CF, часто виглядає «неочікуваною, але влучною» – вона базується на статистичних збігах смаків, які користувачеві самому важко помітити.

Друга причина стійкої популярності CF – її здатність масштабуватися разом із платформою. Алгоритм item-to-item у Amazon, наприклад, показав, що достатньо перенести обчислення «пошуку схожих» в офлайн-фазу, аби система практично лінійно зростала навіть із багатомільйонним каталогом товарів; при цьому якість рекомендацій лишалась стало високою [17]. Латентні факторні моделі, у свою чергу, стискають гігантські поведінкові матриці до кількох сотень вимірів, що робить їх придатними для реального часу без втрати здатності ловити слабкі сигнали вподобань [20].

Звичайно, CF не позбавлена слабких місць – проблеми «холодного старту» та розрідженості даних добре відомі. Однак саме ці вади й стимулювали перехід до гібридних архітектур: коли початкова рекомендація формується контентом чи демографією, але кінцеве тонке ранжування все одно виконує CF-ядро. У таких конфігураціях колаборативні вектори слугують точкою «збирання» інформації з усіх інших варіантів, забезпечуючи ефект серендипності, якого бракує чисто описовим підходам [17]. Інакше кажучи, навіть у складних гібридних ланцюжках саме CF приймає фінальне рішення про те, що користувач імовірно захоче побачити або придбати.

З огляду на ці аргументи подальший аналіз зосереджено саме на методах колаборативної фільтрації: від алгоритмів сусідства до матричної факторизації та їхніх сучасних нейронних розширень. Такий фокус дозволяє детальніше розкрити механізми, що фактично визначають якість рекомендацій у більшості промислових систем, і водночас служить логічним продовженням розділу.

1.3 Методи колаборативної фільтрації (CF)

1.3.1 Методи на основі сусідства (memory-based collaborative filtering)

Найперші системи рекомендацій обходилися без складних моделей: вони напряму працювали з вихідною матрицею «користувач x елемент» і шукали в ній повторювані патерни. Ідея була інтуїтивно проста: якщо історії двох користувачів подібні, то й невідомі оцінки одного з великою ймовірністю збігатимуться з уже відомими оцінками другого [21].

Пошук «подібності» реалізували двома симетричними способами. У user-based алгоритмі кожен рядок матриці розглядають як вектор; далі для активного користувача u обчислюється схожість з усіма іншими користувачами v. Найуживаніші метрики – косинусна та центрована кореляція Пірсона; щоб зменшити вплив випадкових збігів, їх часто поєднують із significance weighting,

коли коефіцієнт пропорційно знижується, якщо спільних оцінок замало [22]. Прогноз для нового елемента роблять через зважене середнє відгуків k найближчих сусідів:

$$\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in N_k(u)} \sin(u,v)(r_{vi} - \bar{r}_v)}{\sum_{v \in N_k(u)} |\sin(u,v)|}. \quad (1)$$

У дзеркальній item-based версії головною сутністю стає елемент: платформа попередньо формує таблицю «схожі товари» і під час сесії просто перетинає списки для кількох уже вподобаних позицій користувача. Саме такий механізм, названий item-to-item CF, зробився промисловим стандартом після його впровадження в Amazon, оскільки час відповіді перестав залежати від кількості клієнтів і масштабу каталогу [12]. Подальший аналіз показав, що item-based підхід зберігає точність user-based, але забезпечує суттєвий приріст продуктивності на великих даних [23].

Попри інтуїтивність і прозорі пояснення («користувачі, схожі на Вас, оцінили…»), методи сусідства мають дві принципові вади. По-перше, масштабованість. Повний перерахунок подібностей між усіма парами профілів – це O(MN) порівнянь, де M – число користувачів, N – число елементів. Щойно масштаб каталогу і аудиторії сягає десятків мільйонів, операція стає вузьким місцем і потребує інкрементних або офлайнових оптимізацій [12]. По-друге, розрідженість даних. У типовій продуктивній базі заповнено менш ніж 1 % клітинок; багато пар користувач-елемент не мають спільних оцінок, тому показники схожості спираються на мізерну статистику й стають нестійкими, особливо в умовах холодного старту [24].

Ці обмеження стимулювали пошук модельних альтернатив: узагальнити дані, скоротити розмірність і навчити параметри, що дають оцінку без виснажливого сканування сусідів. У моделі, що зберігає всі первинні оцінки, як було розглянуто складність зростає квадратично: це стало вирішальним аргументом на користь model-based CF, де взаємодії стискаються у компактні

параметри, а обчислення рейтингу перетворюється на швидку лінійну операцію [25].

1.3.2 Матрична факторизація як базовий model-based підхід

Найбільш впливовою модельною стратегією стала матрична факторизація (MF), чия ефективність була переконливо продемонстрована командою BellKor під час Netflix Prize 2009 р. [26]. Ідея полягає в апроксимації розрідженої матриці оцінок.

$$R \in \mathbb{R}^{M \times N}, \tag{2}$$

добутком двох щільних матриць латентних факторів

$$\hat{R} = PQ^{\mathrm{T}}, P \in \mathbb{R}^{M \times k}, Q \in \mathbb{R}^{N \times k}, k \ll min\{M, N\}, \tag{3}$$

де кожен рядок p_u та q_i є вбудовуванням (embedding) користувача й елемента у спільному k-вимірному просторі.

У цьому просторі геометрична близькість відбиває подібність уподобань: коли вектори спрямовані майже паралельно, скалярний добуток $p_u^{\mathrm{T}} q_i$ великий, що означає високу прогнозовану оцінку. Емпіричні дослідження показують, що окремі координати такого простору часто корелюють із жанрами, стилістикою або ціновим сегментом, хоча їхня семантика виводиться автоматично [27].

1.3.3 Уточнення моделі зміщеннями

Скалярний добуток не враховує систематичних відхилень: одні користувачі оцінюють щедріше, інші стриманіше; частина об’єктів отримує підвищені бали вже через свою загальну привабливість. Тому практична версія, відома як biased MF, доповнює факторизацію зміщеннями [28]:

$$\hat{r}_{ui} = \mu + b_u + b_i + p_u^{\top} q_i, \quad (4)$$

де μ– глобальне середнє, b_u і b_i моделюють відповідно індивідуальну суворість користувача та базову популярність елемента.

Додавання цих параметрів знижує систематичну похибку й особливо важливе для «довгого хвоста» малопопулярних позицій. Реалізація MfDotBias, що буде реалізовано в даній роботі, саме так і комбінує скалярний добуток векторів зі зміщеннями, а при потребі нормує результат через sigmoid, щоб гарантовано потрапити у діапазон шкали рейтингів.

1.3.4 Навчання параметрів

Модель навчається мінімізувати середньоквадратичну помилку наступним чином:

$$\mathcal{L} = \sum_{(u,i)\in\Omega} (r_{ui} - \hat{r}_{ui})^2 + \lambda(\| P \|_2^2 + \| Q \|_2^2 + \| b_u \|_2^2 + \| b_i \|_2^2). \quad (5)$$

На практиці для цього використовують одну з наступних двох процедур:

– стохастичний градієнтний спуск (SGD) оновлює параметри після кожної взаємодії; легко інкрементується та придатний для стримінгових сервісів [19];
– Alternating Least Square (ALS) почергово фіксує P або Q та розв’язує задачу найменших квадратів; алгоритм добре паралелізується й реалізований, зокрема, в Apache Spark MLlib [29].

1.3.5 Сильні й слабкі сторони підходу матричної факторизації

Було підсумовано, яку практичну поведінку демонструє модель у виробничому середовищі. У сучасній літературі ефективність факторизаційних

підходів оцінюють за трьома групами критеріїв – обчислювальною вартістю, стійкістю до структури даних і якістю прогнозу. У межах цих критеріїв MF метод має такі переваги:

- лінійна складність зберігання $(O((M+N)k))$ і швидкий прогноз $(O(k))$;
- стійкість до розрідженості, бо кожна взаємодія коригує усі k координат;
- висока точність, підтверджена експлуатацією в Amazon, Netflix та Spotify [12, 22].
- Та наступні обмеження:
- відсутність векторів для цілком нових елементів або користувачів (cold-start problem) [30];
- лінійна природа моделі і, відповідно, неможливість явно врахувати послідовний чи контекстний ефект;
- чутливість до вибору числа факторів k і коефіцієнта регуляризації λ.

На практиці зазначені вади компенсують шляхом подальшої гібридизації: до MF зверху додають контентні, графові або нейронно-послідовні компоненти. Така архітектура дозволяє зберегти обчислювальну економність MF та водночас адаптувати систему до холодного старту й контекстних сценаріїв, що й спостерігається у більшості сучасних промислових платформ.

1.4 Опрацювання проблеми «холодного старту»

1.4.1 Критичність проблеми «холодного старту» (Cold Start)

Коли новий користувач відкриває сервіс або в каталозі з’являється ще неоцінений елемент, система не має історичних даних, щоб розрахувати латентні вектори чи знайти надійних сусідів. Цю ситуацію називають **cold-start-**ефектом і поділяють на два різновиди [31]:

– user CS: бракує кліків, рейтингів чи переглядів, тому ані матрична факторизація, ані бандитні алгоритми не можуть відрізнити смаки новачка від загальної статистики [32];
– item CS: для щойно доданої позиції (фільм, трек, товар) ще немає взаємодій, отже її латентний вектор q_i невизначений [33].

Наслідки відчутні: зростає відтік (новий користувач не знаходить релевантного), просідає CTR/конверсія, а холодні елементи роками лишаються «невидимими» у довгому хвості каталогу [34].

1.4.2 Типові стратегії подолання проблеми «холодного старту»

Зазвичай виділяють наступні стратегії подолання проблеми:

– показ популярного, яка є простою, але перевіреною практикою – рекомендувати об’єкти з високою агрегованою популярністю (кількість переглядів, покупок). Вона не потребує персональних даних і дає «soft landing» під час онбордингу [35]. Недолік – посилення популярнісного перекосу, катастрофічний для нішевого контенту;
– швидке опитування вподобань, де платформа активно збирає сигнали: чек-лист жанрів, «лайк/дизлайк» прикладів, коротку вікторину. Цим шляхом пішли Netflix, Disney+ і TikTok; тривалість форми обмежують п’ятьма кліками, аби не підвищувати когнітивний бар’єр [36];
– контентні та мультимодальні моделі, де новому елементу призначають вектор, побудований із метаданих (жанри, актори, дескриптори) або навіть зображення/аудіо-ембеддингів. Нейронні підходи типу MARec (Amazon Science, 2024) узгоджують ці представлення з уже вивченим латентним простором [37]. Метод знімає item CS, але вимагає якісних і повних метаданих;
– демографічні/соціальні профілі за умови наявності віку, регіону або соц-зв’язків, системи формують початкові рекомендації за «середнім»

смаком когорти [38]. Перевага – швидкість; мінус – грубе узагальнення та ризик стереотипів.

1.4.3 Обрана стратегія для прототипу: зважений рейтинг (Bayesian Average)

Для реалізації прототипу обрано механізм відбору top N позицій за зваженим (байєсівським) рейтингом, що розраховується безпосередньо на основі наявної матриці оцінок. Рішення грунтується на наступних міркуваннях:

– повна доступність вихідних даних: алгоритм використовує ті самі оцінки, на яких навчалася модель матричної факторизації, і, отже, не вимагає залучення будь-яких зовнішніх метаданих;
– зменшення статистичного перекосу: байєсівське згладжування коригує середню оцінку елемента глобальним середнім каталогу, що запобігає виведенню в топ об’єктів із випадково завищеною популярністю за незначної кількості голосів.

Формула зваженого рейтингу виглядає наступним чином:

$$\text{BAvg(i)} = \frac{v_i R_i + mC}{v_i + m}, \quad (6)$$

де v_i – кількість оцінок елемента, R_i – його середнє, C – глобальне середнє рейтингу, m – гіперпараметр «мінімальної довіри».

При $v_i \gg m$ рейтинг прагне до власного середнього, а для рідкісних позицій – до C.

Такий підхід є доцільним оскільки:

– стабільний на відміну від «plain popularity», одиничний сплеск активності не підніме елемент у топ, поки не набереться достатньо голосів.

– надає релевантний каталог, оскільки використовуємо лише елементи з навчального датасета, cold-start – лист не включить зовнішніх чи видалених фільмів, уникнувши «битих» карток.

Планується що у межах майбутнього API реалізація працюватиме так, що після реєстрації клієнт отримувати top-N за BAvg з датасету.

У підсумку теоретичного огляду встановлено наступне. По-перше, в умовах розширених медіа-каталогів утримати відвідувача без адресних рекомендацій практично неможливо: надлишок вибору без персоналізації спричиняє швидкий відтік аудиторії. По-друге, традиційні алгоритми – як-от пошук найближчих сусідів чи ранжування за популярністю – виявилися малоефективними для великих розріджених матриць «користувач × елемент». По-третє, саме модель матричної факторизації зі зміщеннями продемонструвала найкраще співвідношення точності й обчислювальної складності, що зумовило її вибір як ядра прототипу. В свою чергу, для профілів без історії взаємодій запропоновано стартову добірку, сформовану за байєсівсько зваженим середнім рейтингом, яка забезпечує достатній рівень релевантності до накопичення особистих даних. Така комбінація методів є концептуальною основою подальшого проєктування веб-сервісу.

2 ПРОЄКТУВАННЯ ТА АРХІТЕКТУРА ПРОТОТИПУ РЕКОМЕНДАЦІЙНОЇ ПЛАТФОРМИ

2.1 Опис функціональної та динамічної моделі системи

2.1.1 Діаграма варіантів використання

Діаграма варіантів використання (use-case-діаграма) належить до поведінкових засобів UML і слугує початковою концептуальною моделлю, що фіксує зовнішні вимоги до системи на етапі її проєктування. На рис. 2.1 наведено усі сценарії взаємодії двох акторів – звичайного користувача та адміністратора – з прототипом стримінгової платформи рекомендацій.

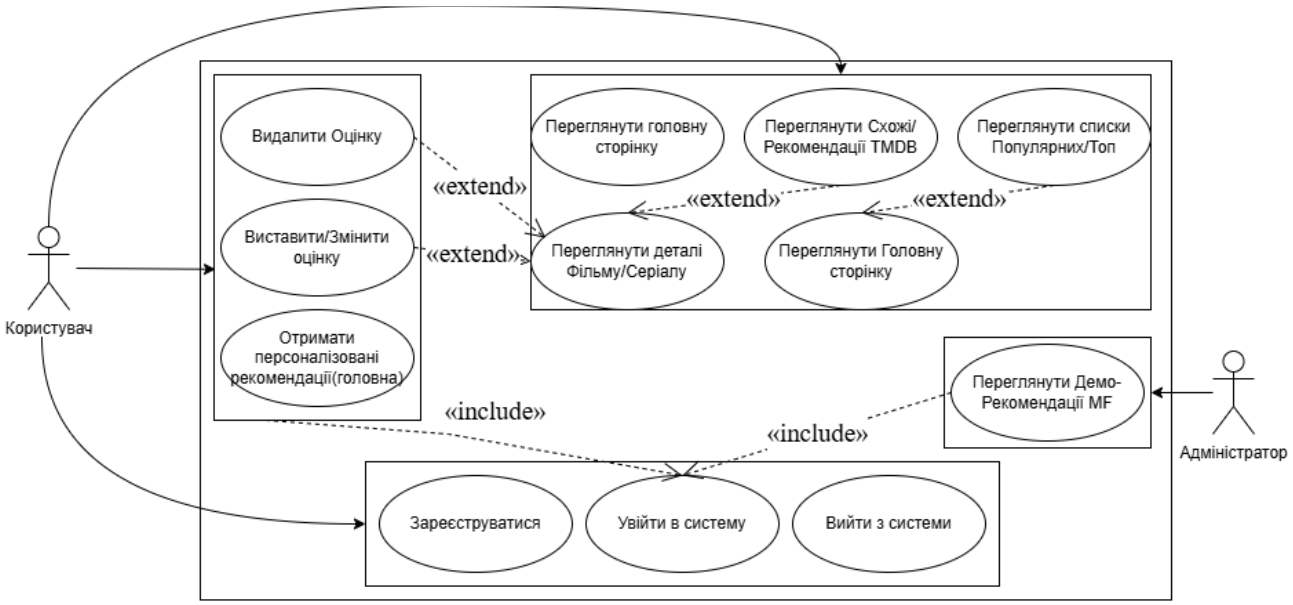


Рисунок 2.1 – Діаграма варіантів використання онлайн-платформи

На рисунку згруповано всі функції веб-платформи, поділивши їх на чотири логічні блоки.

Автентифікація:

– зареєструвати обліковий запис;

– увійти до системи;

– вийти із профілю;
– відновити пароль.

Персоналізація та взаємодія:

– отримати персоналізовані рекомендації для головної сторінки;
– виставити або відкоригувати оцінку;
– видалити раніше збережену оцінку;
– переглянути власні оцінки.

Перегляд контенту:

– переглянути головну сторінку з добірками;
– відкрити деталі вибраного фільму чи серіалу;
– переглянути схожі / рекомендовані TMDb назви;
– ознайомитися з глобальними списками «Популярне» та «Топ-рейтингове»;
– скористатися пошуком за ключовими словами.

Демонстраційний режим:

– показати результати MF-рекомендацій для будь-якого існуючого userId із навченого датасету(доступ лише адміністраторові).

Два основні актори – «Користувач» та «Адміністратор». «Користувач» взаємодіє з усіма чотирма блоками; права «Адміністратора» поширюються на демонстраційний підмодуль і, за необхідності, на загальнокористувацькі функції. Структуру діаграми побудовано так, щоб показати, які саме дії є обов’язковими передумовами (зв’язок include), а які виконуються опціонально або за ініціативою користувача (extend). Це дає цілісну картину функціональних можливостей прототипу та окреслює межі доступу різних категорій користувачів.

2.1.2 Діаграма розгортання

Діаграма розгортання належить до структурних засобів UML і фіксує, у якому саме середовищі виконуються окремі програмні модулі, а також через які

канали вони обмінюються даними. На рис. 2.2 подано фізичну конфігурацію прототипу рекомендаційної платформи.

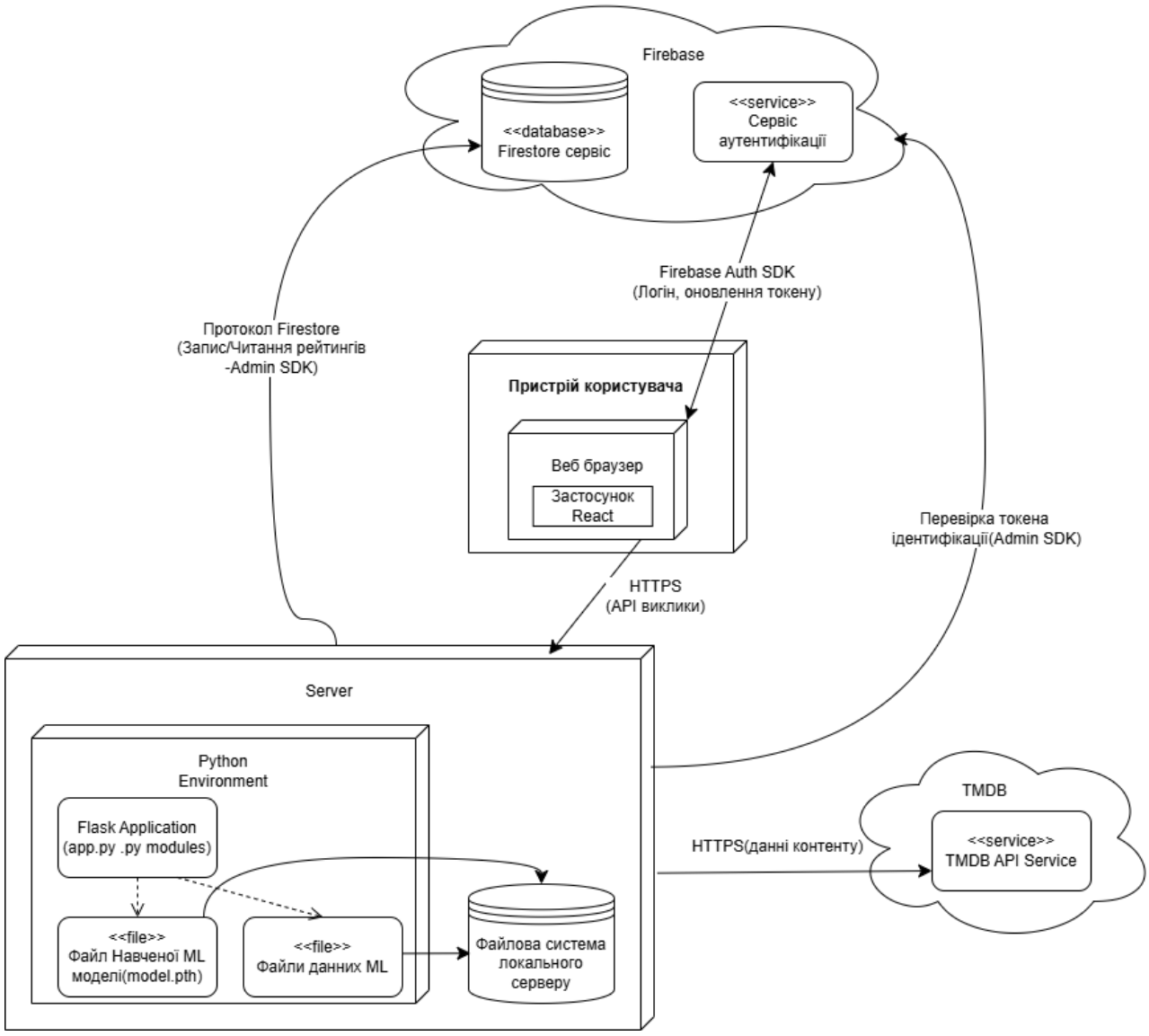


Рисунок 2.2 – Діаграма розгортання компонентів системи

У центрі взаємодії перебуває користувацький пристрій, на якому у вікні браузера завантажується односторінковий застосунок, реалізований засобами React. Саме тут ініціюються всі запити до серверної частини, а перевірку облікових даних виконує клієнтська бібліотека Firebase Auth SDK. Отриманий від сервісу автентифікації токен додається до кожного подальшого HTTP-звернення.

Серверна логіка розгорнута у середовищі Python. Основний процес – це застосунок на Flask, який приймає запити по HTTPS, здійснює перевірку токена

через Firebase Admin SDK і, залежно від типу звернення, передає керування моделі матричної факторизації. Навчена модель та супутні масиви параметрів зберігаються у локальній файловій системі сервера, що мінімізує латентність при формуванні відповіді.

Постійне сховище організовано у хмарі Firebase. Документно-орієнтована база Firestore містить оцінки, окремі профілі користувачів і службові налаштування. Той самий провайдер відповідає і за процедури реєстрації, входу та оновлення токенів. Застосунок на стороні браузера взаємодіє з Firebase напряму, тоді як сервер звертається до бази та сервісу автентифікації у привілейованому режимі, використовуючи Admin SDK.

Для збагачення картки фільму додатковими відомостями (афіша, жанрові мітки, короткий опис) сервер надсилає захищені запити до офіційного API The Movie Database. Таким чином, уся міжвузлова комунікація відбувається поверх TLS, що забезпечує цілісність і конфіденційність переданих даних.

2.1.3 Діаграма діяльності процесу формування рекомендацій

Діаграма діяльності відображає послідовність дій, які виконує система під час оброблення запиту на персональні рекомендації для уже зареєстрованого користувача.

На рис. 2.3 подано UML-діаграму діяльності, яка відтворює послідовність кроків під час опрацювання запиту GET /api/recommendations для авторизованого відвідувача. Сценарій розгортається у чотирьох вертикальних доріжках: Flask API, ML Logic/Data, PyTorch-модель та TMDb Client.

Після отримання запиту шлюз Flask перевіряє Firebase-токен і передає UID у підсистему роботи з даними. Якщо UID присутній у user_lookup, зберігається історія оцінок, формується множина кандидатів і передається в ядро матричної факторизації; за її відсутності підсистема одразу переходить до запасного сценарію й обирає першу N-ку з переліку популярних позицій popular_in_dataset_ids.

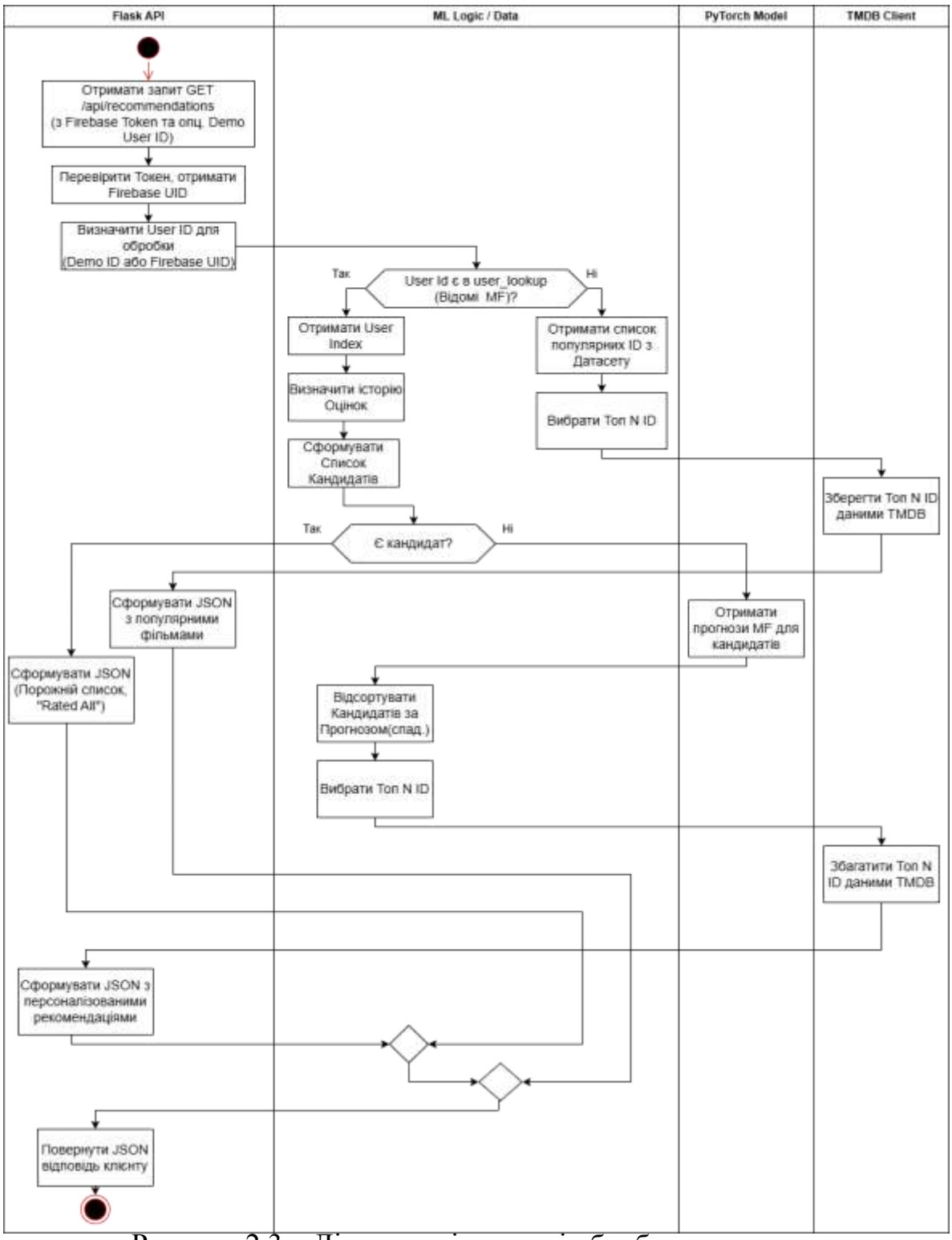


Рисунок 2.3 – Діаграма діяльності оброблення запиту

Модель, працюючи у середовищі PyTorch, повертає список ідентифікаторів із найвищими прогнозними балами. Далі ML-модуль доповнює ці ID метаданими, звертаючись до TMDb API для отримання назв, постерів та жанрів. У фіналі Flask API зводить збагачені записи в упорядкований JSON-масив і надсилає клієнтові.

Допустима розвилка «Є кандидати?» дозволяє чітко відокремити основний шлях (повноцінна персоналізація) від резервного, що гарантує повернення релевантних даних навіть за відсутності історії оцінок, тим самим підвищуючи надійність сервісу.

2.1.4 Діаграми послідовностей додавання оцінки

На рис. 2.4 подано діаграму послідовностей, що відображає повний цикл додавання або оновлення користувацької оцінк – від натискання кнопки на клієнтському інтерфейсі до фіксації змін у Firestore та повернення підтвердження у браузер.

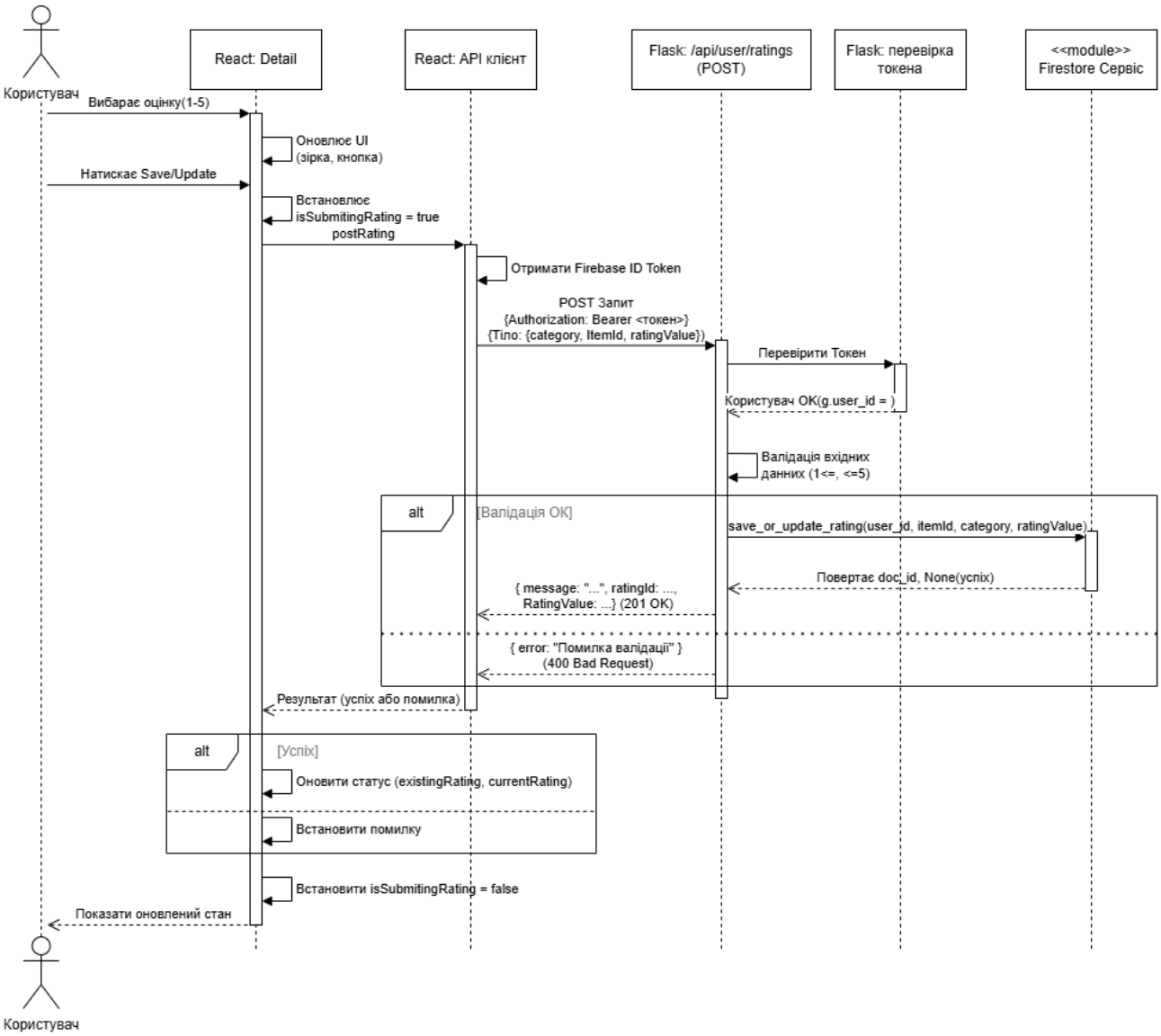


Рисунок 2.4 – Діаграма послідовностей додавання оцінки

Користувач натискає *Save/Update*; компонент **React Detail** блокує кнопку та передає обраний бал клієнтському модулю API. Далі формується запрос POST до /api/user/ratings, у заголовку – ID-токен Firebase, одержаний безпосередньо перед відправленням. На боці сервера маршрут Flask перевіряє токен, валідність даних і викликає функцію save_or_update_rating, що через Admin SDK записує документ у **Firestore**. За успіху повертається статус 201 Created; у разі помилки сервер відповідає 400 Bad Request. Отримавши підтвердження, **React Detail** оновлює локальний стан і відображає нове значення балу. У такий спосіб оцінка потрапляє до сховища без перезавантаження сторінки, а користувач одразу бачить результат дії.

2.1.5 Діаграми послідовностей надання рекомендацій

Діаграма послідовності на рис. 2.5 відтворює типовий перебіг запиту /api/recommendations для зареєстрованого користувача, дані якого вже присутні у навчальній вибірці. У правій частині рисунка наведено лише одну гілку алгоритму – ту, що задіює матричну факторизацію. Альтернативний сценарій, який активується для нових облікових записів (коли історія оцінок ще відсутня), також описано нижче текстом.

На першому етапі компонент React Home ініціює HTTP-запит і додає до заголовка токен автентифікації. У Flask API цей маркер перевіряється за допомогою Firebase Admin SDK; одночасно з токена видобувається внутрішній user_id. Далі сервер звертається до блоку ML Модель/Дані та отримує:

- список раніше оцінених фільмів;
- повний перелік ідентифікаторів, доступних у каталозі;
- підмножину кандидатів, що не перетинаються з історією.

Модель матричної факторизації (mf_predict) повертає прогнозні бали для кандидатів; сервер обирає першу N-ку позицій із найвищим рейтингом. Для кожного з обраних ідентифікаторів через клієнт TMDb добираються метадані (постер, назва, жанри), після чого формується відповідь JSON і надсилається фронт-енду. Клієнт оновлює інтерфейс, показуючи секцію «Recommended for you».

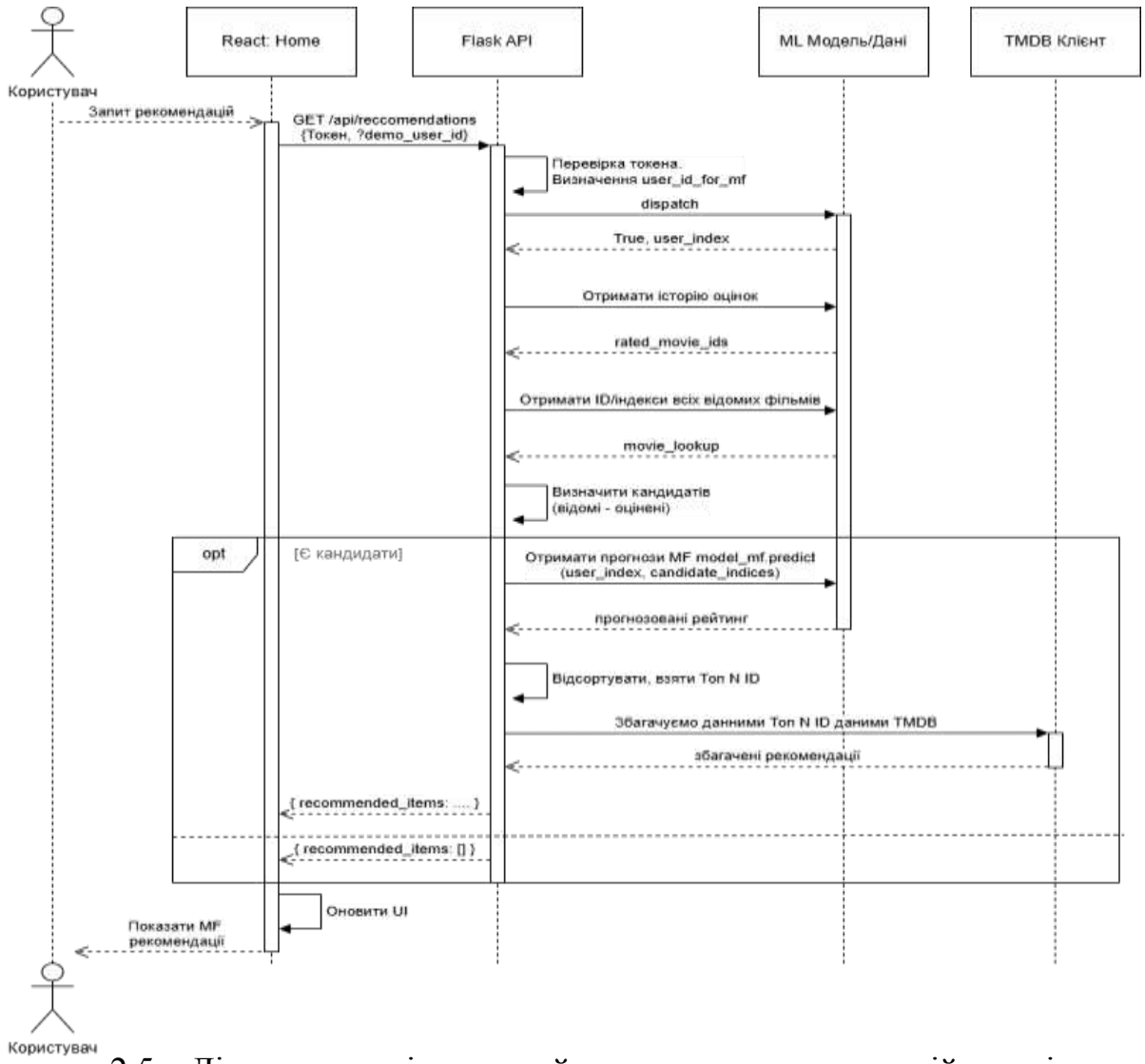


Рисунок 2.5 – Діаграма послідовностей отримання рекомендацій для відомого користувача

Якщо під час перевірки з’ясується, що user_id відсутній у довіднику (тобто користувач ще не залишив жодної оцінки), Flask API переходить на запасний шлях. У такому разі з блоку даних запитується список найпопулярніших назв, отриманих зі статистики каталогу («cold-start список»). Після збагачення метаданими цей перелік повертається клієнтові у тому самому форматі. Таким чином, обидва сценарії завершуються однаковою структурою відповіді, а різниця полягає лише у способі формування ранжованого списку.

2.1.6 Діаграма класів

UML-діаграма класів застосовується не лише для опису об’єктно-орієнтованого коду. У випадку документо-орієнтованого сховища, яким є Firebase Firestore, такий різновид діаграми дає змогу виразно подати структуру колекцій і атрибутів без надмірної деталізації зв’язків, характерної для класичних ER-схем. Відсутність жорсткої схеми зовнішніх ключів, що властива NoSQL-середовищу, робить саме UML-подання зручнішим для фіксації ієрархії документів та їхніх полів.

Діаграма класів показана на рис. 2.6.

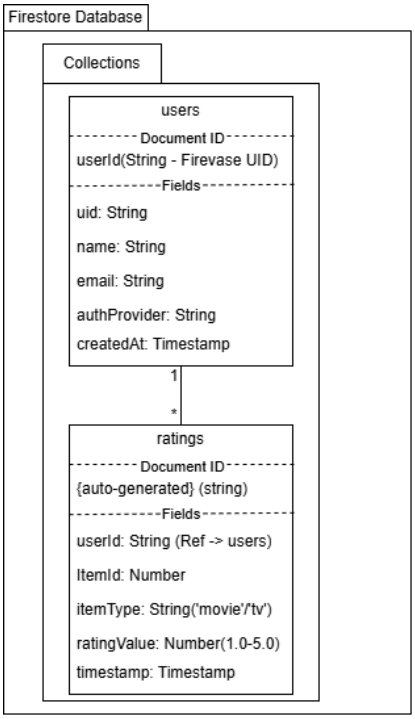


Рисунок 2.6 – Узагальнена структура колекцій Firestore

На діаграмі виділено дві колекції: users та ratings. Документи колекції users ідентифікуються значенням userId, що збігається з UID, виданим Firebase Authentication. Окрім технічного UID, документ містить ім’я, електронну адресу, спосіб реєстрації (authProvider) та мітку часу створення.

Колекція ratings призначена для зберігання індивідуальних оцінок. Кожен її документ має автоматично згенерований ідентифікатор і містить посилання

на користувача (userId), а також атрибути, що описують об’єкт оцінювання: itemId, тип об’єкта (itemType) і числове значення ratingValue. Поле timestamp фіксує момент останнього оновлення оцінки. В логіці системи зв’язок між users та ratings трактують як «один-до-багатьох», що на діаграмі відмічено відповідною нотацією.

Обраний спосіб моделювання сутностей відбиває особливості Firestore: дані зберігаються у вигляді плоских документів, а необхідні індекси створюються безпосередньо у хмарній консолі. Завдяки цьому доступ до оцінок конкретного користувача здійснюється єдиним фільтром за полем userId, що відповідає вимогам швидкодії у сценаріях формування персональних вибірок.

2.2 Опис вхідних даних для навчання нейромережі

Для побудови та випробування алгоритму персональних рекомендацій передбачається використати відкритий набір MovieLens-32M (реліз GroupLens, 05/2024). У ньому міститься близько 32 млн явних оцінок і понад 1,9 млн тегів, накладених на 87 585 аудіовізуальних творів. Кожен запис пов’язує три ключові сутності:

– userId – анонімізований ідентифікатор користувача;
– movieId – ідентифікатор фільму або серіалу;
– rating – числову оцінку в діапазоні 0,5 – 5,0.

Додаткові файли набору (зокрема links.csv) містять зіставлення внутрішніх ідентифікаторів із записами баз IMDb та TMDb, що надалі дозволить під’єднати зовнішні метадані (жанри, синопсиси, візуальні матеріали) без дублювання інформації.

Беручи до уваги обмеження апаратних ресурсів, повну вибірку планується скоротити. Передбачається застосувати стратифікований відбір користувачів за рівнем активності: буде сформовано підмножину приблизно сорока тисяч профілів, яка збереже початкові пропорції між високою та

низькою частотою оцінювання. До навчальної матриці потраплять лише оцінки цих користувачів та відповідні їм фільми. У результаті очікується зменшення обсягу даних приблизно на п’яту частину при збереженні репрезентативності розподілу.

Скорочений набір слугуватиме базою для навчання моделі матричної факторизації. Під час попередньої обробки будуть створені словники перетворення (user_lookup, movie_lookup), що забезпечать швидке зіставлення зовнішніх ідентифікаторів із внутрішніми під час роботи веб-сервісу.

3 РЕАЛІЗАЦІЯ ПРОЄКТУ

3.1 Підготовка даних та навчання моделі матричної факторизації

3.1.1 Обґрунтування скорочення датасету та вибір методики

Повна версія набору MovieLens-32M налічує понад двісті тисяч облікових записів і трохи більше тридцяти двох мільйонів оцінок. За таких масштабів навчання моделі на базовому робочому комп’ютері потребує значного обсягу часу та оперативної пам’яті. Очевидне рішення - випадково вилучити частину записів – виявилося нежиттєздатним, оскільки за такої процедури до вибірки нерідко потрапляють переважно дуже активні або, навпаки, майже пасивні користувачі, що спотворює реальний розподіл поведінки аудиторії. Щоб уникнути цього перекосу, було ухвалено рішення стратифікувати користувачів за кількістю виставлених оцінок.

Алгоритм має чотири кроки:

Для кожного userId підраховано сумарний обсяг виставлених балів.

Усі профілі впорядковано за зростанням активності і поділено на п’ять квінтилів однакової чисельності.

Із кожного квінтиля (за фіксованим RANDOM_SEED = 42) обрано таку кількість облікових записів, щоб сумарно отримати 40 000 користувачів.

В файлах ratings.csv, movies.csv, tags.csv та links.csv залишили лише ті записи, що пов’язані з відібраною підмножиною профілів.

Такий підхід дає дві переваги:

– у вибірці присутні як «важкі глядачі» (із сотнями оцінок), так і користувачі з мінімальною історією;
– чисельність даних зменшується приблизно в п’ять разів без суттєвого зменшення каталогу фільмів.

3.1.2 Підсумкова конфігурація робочого набору

Після виконання описаної процедури отримано наступні результати:

– користувачі: 40 000 профілів;

– фільми: ≈ 60387 позицій;

– рейтинги: ≈ 6,38 млн записів;

– теги: ≈ 272 тис. позначок.

Підсумкові значення підтверджує службовий логи скрипта, наведені на рис. 3.1.

```
==================================================
--- ЕТАП 0 (Скорочення) ЗАВЕРШЕНО ---
Скорочені дані повинні знаходитися у: /kaggle/working/ml-32m-reduced-users-stratified5
  Файл ratings.csv: Знайдено (Розмір: 160.78 MB)
  Файл movies.csv: Знайдено (Розмір: 2.74 MB)
  Файл links.csv: Знайдено (Розмір: 1.21 MB)
  Файл tags.csv: Знайдено (Розмір: 9.02 MB)

Підсумок: Користувачів: 40000, Фільмів: 60387, Рейтингів: 6381429, Тегів: 272713
==================================================
```

Рисунок 3.1 – Результат скорочення

Розмір файлів у стисненому вигляді суттєво скоротився, що дозволяє:

– запускати повний цикл навчання моделі матричної факторизації за менший час;

– легко зберігати дані у пам’яті під час експериментів із гіперпараметрами;

– не втратити жанрове різноманіття каталогу, оскільки фінальний перелік стрічок практично не скоротився.

Саме отриманий узгоджений та репрезентативний піднабір слугуватиме єдиним джерелом для попередньої обробки ознак і подальшого тренування моделі.

3.1.3 Методологія валідації

Ефективність рекомендаційної системи залежить не лише від обраної моделі, а й від того, як саме перевіряється її здатність узагальнювати дані. Для набору було розглянуто дві загальноприйняті схеми формування тестових вибірок. Їхні ключові характеристики зведено у табл. 3.1.

Таблиця 3.1 – Архітектурні компоненти моделі

Параметр оцінки	User-based split (нові користувачі)	Temporal split (теплі користувачі)
Об’єкт тесту	цілком невідомі профілі	остання дія відомого профілю
Наявність ембеддингів у тесті	відсутні	сформовані під час тренування
Симуляція «холодного старту»	так	ні
Відповідність щоденному сценарію платформи	низька	висока
Стабільність метрик (RMSE, nDCG)	середня	висока

Оскільки головна мета проєкту – поліпшити рекомендації для аудиторії з історією переглядів, було обрано темпоральний підхід.

Така методика:

– імітує реальний цикл роботи платформи «оцінка → наступна рекомендація»;
– зберігає стабільність метрик RMSE і nDCG, що полегшує порівняння гіперпараметрів;
– не вимагає додаткових процедур для генерації ембеддингів «з нуля».

Сценарій повного «холодного старту» залишено поза межами основного експерименту та вирішуватиметься за допомогою окремої, простішої стратегії (популярні фільми з байєсівським згладжуванням), яка не потребує навчання.

3.1.4 Розділення даних на тренувальний і тестовий набори

Валідована схема поділу реалізується трьома послідовними кроками:

– відфільтрувати профілі з менш ніж десятьма оцінками;
– упорядкувати записи кожного користувача за часом і позначити останній як валідаційний;
– зберегти отримані підмножини у форматі pickle.

Невеликий фрагмент коду, що виконує темпоральне розділення, наведено на рис. 3.2.

```
def temporal_split(df, min_r=10, ts="timestamp", uid="userId"):  1 usage
    active = df[uid].value_counts().loc[lambda x: x >= min_r].index
    df = df[df[uid].isin(active)].sort_values([uid, ts])
    df["is_valid"] = False
    df.loc[df.groupby(uid).tail(1).index, "is_valid"] = True
    return df


ratings = pd.read_pickle("ratings_df.pkl")
split = temporal_split(ratings)
split[~split.is_valid].to_pickle("train_df.pkl")
split[split.is_valid].to_pickle("valid_df.pkl")
```

Рисунок 3.2 – Невеликий фрагмент-із реалізації темпорального розділення

Отримані набори забезпечують незалежність перевірки: тренування бачить повну історію користувача до певного моменту, а тест перевіряє, чи правильно система відгукнеться на його найближчу дію. Формат pickle мінімізує час читання, що суттєво скорочує тривалість циклу «завантаження – навчання – оцінка» у подальших кроках.

3.1.5 Розробка структури нейронної мережі

У центрі прототипу лежить компактна нейронна архітектура, що реалізує матричну факторизацію зі зміщеннями. Її завдання – перенести вихідну розріджену матрицю «користувач – фільм» у два щільні векторні простори та відновити наближене значення оцінки через скалярну взаємодію цих представлень (табл. 3.2).

Таблиця 3.2 – Архітектурні компоненти моделі

Компонент	Розмірність	Призначення
$U \in \mathbb{R}^{(m+1)\times k}$	користувач × фактор	латентний профіль користувача
$\mathrm{V} \in \mathbb{R}^{(n+1)\times k}$	фільм × фактор	латентний опис фільму
$b_u \in \mathbb{R}^{(m+1)}$	зміщення користувача	індивідуальний нахил до вищих/нижчих балів
$b_i \in \mathbb{R}^{(n+1)}$	зміщення фільму	загальна популярність фільму

Нульові індекси резервуються під випадок некоректного або ще не зіставленого ідентифікатора; ваги цих рядків під час ініціалізації фіксовано занулюються, щоб вони не впливали на підсумковий бал.

Формула прямого проходу:

$$\hat{r}_{ui} = \mu + b_u + b_i + \langle U_u, V_i \rangle, \tag{7}$$

де μ – середнє значення у тренувальній вибірці, а внутрішній добуток обчислюється в FP16 для скорочення пам’яті.

Фрагмент реалізації моделі подано на рис. 3.3.

Ключові властивості наведеної моделі:

– швидкість прогнозу;

– стабільні параметри;

– легкість розширення.

```
# --- 5. Визначення Базової Моделі MF ---
print("\n--- 5. Визначення Базової Моделі MF (MfDotBias) ---")
class MfDotBias(nn.Module):
    def __init__(self, n_factors, n_users, n_items, ratings_range=None, use_biases=True):
        super().__init__(); self.bias = use_biases; self.y_range = ratings_range
        self.user_embedding = nn.Embedding(n_users + 1, n_factors, padding_idx=0)
        self.item_embedding = nn.Embedding(n_items + 1, n_factors, padding_idx=0)
        nn.init.normal_(self.user_embedding.weight, std=0.02)
        nn.init.normal_(self.item_embedding.weight, std=0.02)
        if use_biases:
            self.user_bias = nn.Embedding(n_users + 1, 1, padding_idx=0)
            self.item_bias = nn.Embedding(n_items + 1, 1, padding_idx=0)
            nn.init.zeros_(self.user_bias.weight); nn.init.zeros_(self.item_bias.weight)
    def forward(self, inputs):
        users, items = inputs; user_emb = self.user_embedding(users)
        item_emb = self.item_embedding(items)
        dot = (user_emb * item_emb).sum(dim=1)
        if self.bias: user_b = self.user_bias(users).squeeze(-1)
            item_b = self.item_bias(items).squeeze(-1); dot = dot + user_b + item_b
        if self.y_range is None: return dot
        else: return torch.sigmoid(dot) * (self.y_range[1] - self.y_range[0]) + self.y_range[0]
```

Рисунок 3.3 – Фрагмент коду, що реалізує шар MfDotBias у PyTorch

Головним недоліком наведеного підходу є лінійна природа взаємодії яка означає, що модель не враховує порядок переглядів чи зовнішні контекстні фактори. Але в розробленому прототипі це є виправданим, оскільки наведена конструкція демонструє компроміс між обчислювальною простотою та здатністю відтворювати індивідуальні вподобання, що є ключовою вимогою для інтеграції в онлайн-сервіс із обмеженим ресурсним бюджетом.

3.1.6 Вибір гіперпараметрів моделі

Гіперпараметри безпосередньо визначають, наскільки швидко сходиться модель матричної факторизації та чи здатна вона узагальнювати дані поза тренувальною вибіркою. Навчання моделі проводилося на безкоштовних обчислювальних ресурсах середовища Kaggle Notebooks (GPU Tesla P100). Для навчання моделі були застосовані наступні гіперпараметри:

- розмір латентного простору (k=120): забезпечує баланс між виразною здатністю та кількістю параметрів;
- використавує зміщення USE_BIASES=True;
- діапазон прогнозу [0,5; 5,5] – пом’якшує насичення на краях шкали оцінок;

– початковий крок оптимізації $\mathrm{lr} = 6 \times 10^{-4}$;
– пікове значення $\mathrm{lr}_{max} = 3 \times 10^{-3}$ (One-Cycle, pct_start=0,25);
– L2-штраф $\lambda = 2 \times 10^{-4}$ – знижує дисперсію ембеддингів рідко оцінених фільмів;
– батч 8192;
– максимум епох 30;
– рання зупинка (patience=2,RMSE=5×10^{-4}) – запобігає перенавчанню.

3.1.7 Процес навчання та збереження моделі

Навчання виконано з використанням наведених параметрів з минулого розділу. На практиці цикл тривав шість епох: після четвертої значення RMSE знизилося до 0,8705, а відсутність подальшого прогресу протягом двох перевірок активувала механізм ранньої зупинки. Фактичне навантаження становило 774 батчі на епоху.

Результат тренування моделі на ведено на рис. 3.4.

```
    Improvement! New best eval_rmse: 0.87049 (previous: 0.89508)
    Saving best model to /kaggle/working/base_mf_models_temporal_from_preprocessed/best_base_mf_temporal
_preprocessed_model.pth...
    Best model saved.
--- Epoch 4 finished in 203.12 seconds ---

--- Epoch 5/30 ---
Epoch 5 Train Summary: Avg Loss = 0.4707, Current LR = 0.002280
Epoch 5 Eval Summary - RMSE: 0.8704 (MAE: 0.6570, MSE: 0.7577)
    No significant improvement. Counter: 1/2
--- Epoch 5 finished in 203.41 seconds ---

--- Epoch 6/30 ---
Epoch 6 Train Summary: Avg Loss = 0.3755, Current LR = 0.002725
Epoch 6 Eval Summary - RMSE: 0.8919 (MAE: 0.6721, MSE: 0.7955)
    No significant improvement. Counter: 2/2
  Early stopping triggered after epoch 6 (Counter: 2)
------------------------------
Навчання Перервано (Early Stopping або Помилка).
Загальний час навчання: 20.26 хв.
Найкращий показник eval_rmse: 0.87049 (досягнуто на епосі 4)
   Найкращу модель збережено у: /kaggle/working/base_mf_models_temporal_from_preprocessed/best_base_mf_temp
oral_preprocessed_model.pth

Посилання для завантаження НАЙКРАЩОЇ моделі:
/kaggle/working/base_mf_models_temporal_from_preprocessed/best_base_mf_temporal_preprocessed_model.pth

Збереження стану ОСТАННЬОЇ моделі в /kaggle/working/base_mf_models_temporal_from_preprocessed/final_base
_mf_temporal_preprocessed_model.pth...
    Стан останньої моделі збережено.
```

Рисунок 3.4 – Процес та результат тренування моделі

По завершенні тренування створюються три артефакти(ML artifacts) необхідні для подальшої інтеграції моделі на серверній частині (табл. 3.3).

Таблиця 3.3 – Артефакти моделі, підготовлені до впровадження

Файл	Призначення
best_base_mf_temporal_model.pth	тензори факторів і зміщень, зафіксовані на епосі з найменшим RMSE
user_lookup.pkl	словник «рядковий UID → числовий індекс»
movie_lookup.pkl	словник «цілий movieId → числовий індекс»

3.2 Налаштування серверного середовища та завантаження артефактів (ml_setup.py)

Модуль ml_setup.py виконується лише під час старту сервісу Flask і готує всі об’єкти, потрібні для онлайнового інференсу. Його логіка поділена на чотири послідовні етапи.

3.2.1 Створення допоміжних словників

На підставі скорочених файлів **movies.csv** та **links.csv** формується мінімальний набір структуруючих словників:

– movielens_to_tmdb – відповідність числових MovieLens ID записам TMDB;
– movie_id_to_title – швидкий доступ до назви за MovieLens ID;
– movie_id_to_genres – список жанрових маркерів для кожної назви;

– movie_id_to_category_and_poster – посилання на постер і базова категорія («movie» за замовчуванням).

3.2.2 Формування списку популярних назв

З історичних оцінок ratings.csv обчислюється зважений рейтинг за формулою байєсівського середнього, яку було раніше розглянуто в теоретичній частині. Після сортування формується список popular_in_dataset_ids, який слугує резервним джерелом рекомендацій для профілів без історії.

Алгоритм реалізовано у модулі ml_setup.py; його ключові кроки показано на рис. 3.5.

```
if not ratings_df_final.empty:
    try:
        print("Calculating item popularity (Weighted Rating)...")
        C = ratings_df_final['rating'].mean();
        M = ratings_df_final['movieId'].value_counts().quantile(0.80)
        print(f"Popularity params: Global Mean (C)={C:.2f}, Min Votes (M)={M:.0f}")
        item_stats = ratings_df_final.groupby('movieId')['rating'].agg(['count', 'mean']).reset_index()
        def weighted_rating(row, m=M, c=C): v=row['count']; r=row['mean']; return (v/(v+m)*r)+(m/(v+m)*c)  1 usage
        item_stats['weighted_score'] = item_stats.apply(weighted_rating, axis=1)
        item_popularity_df = item_stats.sort_values(by='weighted_score', ascending=False)
        popular_in_dataset_ids = item_popularity_df['movieId'].tolist()
        print(f"Calculated weighted popularity for {len(item_popularity_df)} items.")
    except Exception as e_pop: print(f"Error calculating item popularity: {e_pop}"); popular_in_dataset_ids = []
else:
    print("Warning: No ratings data available to calculate popularity."); popular_in_dataset_ids = []
```

Рисунок 3.5 – Розрахунок Bayesian Average у модулі ml_setup.py

3.2.3 Завантаження навченої моделі та словників індексації

Щойно сформовано допоміжні словники, ml_setup.py виконує чотири логічні дії.

– файли user_lookup.pkl та movie_lookup.pkl розпаковуються в оперативну пам’ять. Їхній розмір одразу задає параметри N_USERS і N_ITEMS, що мають збігатися з конфігурацією під час навчання;

– на основі тих самих гіперпараметрів, що використовувалися в Kaggle-сесії (N_Factor=120, USE_BIASES=True, діапазон 0.5 – 5.5)

створюється об’єкт MfDotBias. Важливо ініціалізувати embeddings розміром N_USERS + 1 і N_ITEMS + 1, інакше завантаження ваг завершиться помилкою розміру тензора;

– відновлення ваг і переведення на пристрій: стан моделі зчитується з файлу best_base_mf_temporal_preprocessed_model.pth, мапується на device (cuda, якщо доступний, або cpu) і переводиться у режим eval().

Ініціалізацію та подальше завантаження моделі наведено на рис. 3.6–3.7.

```python
# --- Параметри НАВЧЕНОЇ моделі ---
try:
    N_FACTORS = 120
    RATINGS_RANGE = [0.5, 5.5]
    USE_BIASES = True
    print(f"Model parameters set: N_FACTORS={N_FACTORS}, USE_BIASES={USE_BIASES}, RANGE={RATINGS_RANGE}")
except Exception as e:
    print(f"Warning: Error defining model parameters - using defaults. Error: {e}")
    N_FACTORS = 100; RATINGS_RANGE = [0.5, 5.5]; USE_BIASES = True


# --- 1. Визначення Класу Моделі ---
class MfDotBias(nn.Module):  1 usage
    def __init__(self, n_factors, n_users, n_items, ratings_range=None, use_biases=True):
        super().__init__(); self.bias = use_biases; self.y_range = ratings_range
        self.user_embedding = nn.Embedding(n_users + 1, n_factors, padding_idx=0)
        self.item_embedding = nn.Embedding(n_items + 1, n_factors, padding_idx=0)
        if use_biases:
            self.user_bias = nn.Embedding(n_users + 1,  embedding_dim: 1, padding_idx=0)
            self.item_bias = nn.Embedding(n_items + 1,  embedding_dim: 1, padding_idx=0)
    def forward(self, inputs):
        users, items = inputs; user_emb = self.user_embedding(users); item_emb = self.item_embedding(items)
        dot = (user_emb * item_emb).sum(dim=1)
        if self.bias: user_b = self.user_bias(users).squeeze(-1); item_b = self.item_bias(items).squeeze(-1);
        dot = dot + user_b + item_b
        if self.y_range is None: return dot
        else: return torch.sigmoid(dot) * (self.y_range[1] - self.y_range[0]) + self.y_range[0]
```

Рисунок 3.6 – Фрагмент коду ініціалізації класу MfDotBias у модулі

```python
# --- 3. Завантаження Моделі MF ---
if N_USERS > 0 and N_ITEMS > 0 and MODEL_MF_PATH.exists():
    try:
        print(f"Loading MF model from {MODEL_MF_PATH}...")
        model_mf_instance = MfDotBias(n_factors=N_FACTORS, n_users=N_USERS, n_items=N_ITEMS, ratings_range=RATINGS_RANGE, use_biases=USE_BIASES)
        model_mf_instance.load_state_dict(torch.load(MODEL_MF_PATH, map_location=device))
        model_mf_instance.to(device); model_mf_instance.eval(); model_mf = model_mf_instance
        print(f"MF Model loaded successfully to {device}.")
    except Exception as e: print(f"Error loading MF model state dict: {e}"); import traceback; traceback.print_exc(); model_mf = None
elif not MODEL_MF_PATH.exists(): print(f"MF Model file not found at {MODEL_MF_PATH}.")
else: print("Cannot load MF model due to missing lookups (N_USERS or N_ITEMS is 0).")
```

Рисунок 3.7 – Завантаження збережених ваг (state_dict) і переведення моделі у режим eval()

3.2.4 Експорт готових об’єктів

Наприкінці роботи модуль форму список __all__, що надає app.py прямий доступ до:

– model_mf та device;
– основних словників (user_lookup, movie_lookup, movielens_to_tmdb, тощо);
– функції is_ready(), яка сигналізує про успішне завершення ініціалізації.

Такий поділ гарантує, що інші частини системи взаємодіють лише з уже перевіреними та готовими до роботи ресурсами, зменшуючи ризик помилок у продуктивному середовищі.

Повний перелік експортованих об’єктів наведено на рис. 3.8.

```
__all__ = [
    'model_mf', 'device', 'user_lookup', 'movie_lookup',
    'movie_id_to_title', 'movie_id_to_category_and_poster', 'movie_id_to_genres',
    'is_ready',
    'ratings_df_final',       # Історичні рейтинги (для демо MF)
    'KNOWN_MOVIE_IDS',        # Множина відомих MovieLens ID
    'popular_in_dataset_ids', # Відсортований список MovieLens ID за популярністю
    'movielens_to_tmdb'       # Мапінг MovieLens ID -> TMDB ID

]
```

Рисунок 3.8 – Перелік об’єктів, що експортуються з ml_setup.py

3.3 Розробка серверного API на Flask (app.py)

3.3.1 Ініціалізація та конфігурація app.py

Під час запуску серверний застосунок виконує п’ять взаємопов’язаних дій:

- завантаження змінних середовища: за допомогою python-dotenv зчитується файл .env; у процесі ініціалізуються ключі TMDB_API_KEY, адреса фронтенду та порт, на якому слухатиме Flask-сервер;
- під’єднання до Firebase: модуль firebase_admin ініціалізується сервісним ключем serviceAccountKey.json;
- створення екземпляра Flask та налаштування CORS: об’єкт Flask(__name__) розширюється пакетом Flask-CORS, який дозволяє запити з клієнтського React-застосунку (http://localhost:5173);
- імпорт машинно-навчальних артефактів: з модуля ml_setup імпортуються модель model_mf, словники індексації, допоміжні мапінги та список популярних назв; після імпорту викликається is_ready(). Якщо хоча б один із компонентів не завантажився, застосунок завершує роботу;
- ініціалізація кешу TMDB: Створюється об’єкт SimpleCache (бібліотека cachelib) з терміном життя запису 24 години.

3.3.2 Механізм автентифікації та авторизації

Для того щоб персоналізовані функції платформи були доступні виключно зареєстрованим відвідувачам, реалізовано двоступеневу схему «автентифікація клієнта – авторизація запиту». Центральним компонентом став сервіс Firebase Authentication, який, з одного боку, спрощує роботу з обліковими записами, а з іншого – усуває потребу зберігати паролі у власній базі.

На клієнтській стороні (React) автентифікація відбувається через бібліотеку Firebase Client SDK. Користувач може:

- створити обліковий запис або увійти за електронною адресою і паролем;
- скористатися спрощеним входом через Google чи Facebook;

– ініціювати процедуру відновлення забутого пароля.

Після успішної операції SDK формує підписаний ID-токен, чинний протягом однієї години, і автоматично поновлює його у фоновому режимі. Перед кожним зверненням до API актуальний токен додається до заголовка HTTP-запиту у формі Authorization: Bearer <token>.

Надалі перевірка вже відбувається у Flask-застосунку. Усі маршрути, що потребують автентифікації, помічено декоратором @check_token. На рис. 3.9 наведено ключовий фрагмент, що реалізує описану перевірку.

```python
def check_token(f):  5 usages
    @wraps(f)
    def wrap(*args,**kwargs):
        if request.method == 'OPTIONS': return app.make_default_options_response()
        auth_header = request.headers.get('Authorization')
        if not auth_header or not auth_header.startswith('Bearer '):
            return jsonify({"error": "Authorization Token is missing"}), 401
        id_token = auth_header.split('Bearer ')[1]
        try:
            decoded_token = admin_auth.verify_id_token(id_token)
            g.user_id = decoded_token['uid']
        except Exception as e:
            print(f"Token verification failed: {e}")
            message = "Invalid or expired token"
            if isinstance(e, admin_auth.ExpiredIdTokenError): message = "Token expired"
            elif isinstance(e, admin_auth.RevokedIdTokenError): message = "Token revoked"
            return jsonify({"error": message}), 401
        return f(*args, **kwargs)
    return wrap
```

Рисунок 3.9 – Функція перевірки ID-токена

Під час першого входу клієнтська частина, окрім отримання токена, створює у Firestore документ користувача з полями. Це дає можливість надалі пов’язувати оцінки з конкретним профілем без додаткових запитів.

3.3.3 Модуль взаємодії з TMDB API (tmdb_client.py)

Для відокремлення мережевої логіки також створено допоміжний модуль tmdb_client.py. Його завдання – уніфікувати доступ до сервісу The Movie

Database та мінімізувати кількість зовнішніх HTTP-звернень.

У серці модуля знаходиться функція _make_request(endpoint, params={}), яка зображена на рис. 3.10. Вона:

– автоматично підставляє ключ авторизації api_key (змінна середовища),
– формує повне посиланн BASE_URL/endpoint,
– виконує запит методом GET з тайм-аутом 5 с,
– перехоплює мережеві винятки, повертаючи кортеж (payload, status_code).

```
def _make_request(endpoint,params={}):  8 usages
    if not API_KEY:
        return {"error": "TMDB API Key not configured"}, 503
    all_params = {'api_key': API_KEY, **params}
    url = f"{BASE_URL}/{endpoint}"
    try:
        response = requests.get(url, params=all_params, timeout=5)
        response.raise_for_status()
        return response.json(), 200
    except requests.exceptions.RequestException as e:
        # ... (обробка помилок) ...
        status_code = e.response.status_code if e.response is not None else 503
        return {"error": f"TMDB API Error"}, status_code
```

Рисунок 3.10 Укорочена реалізація _make_request

На її основі реалізовано чотири спеціалізовані обгортки:

– get_list(category, list_type, page) – отримання стрічок popular, top_rated, upcoming;
– search(category, query, page) – пошук за ключовим словом;
– discover(category, page, genres, sort_by) – підбір списків із фільтрами;
– get_details_bundle(category, item_id) – розширені дані одного об’єкта через параметр append_to_response=credits,videos,similar,recommendations.

З огляду на те, що рекомендаційний алгоритм часто звертається до одних

і тих самих назв, у головному модулі app.py ввімкнено кешування результатів функції get_details_bundle за допомогою SimpleCache (бібліотека cachelib). При першому зверненні відповідь TMDB зберігається на 60 хвилин.

3.3.4 Модуль взаємодії з базою оцінок (firestore_service.py)

Оцінки користувачів зберігаються у хмарній NoSQL-базі Firebase Firestore в єдиній колекції ratings. Кожен документ містить поля userId, itemId, itemType, ratingValue та timestamp. Усі операції з цією колекцією зосереджено в модулі firestore_service.py, що працює через Firebase Admin SDK.

Основні методи модуля:

– get_user_rating_doc – повертає, якщо існує, документ оцінки, залишеної користувачем для заданого елемента;
– save_or_update_rating – створює новий запис або оновлює наявний;
– delete_rating_by_doc_id – видаляє оцінку за відомим doc_id;
– get_all_user_ratings – повертає повний список оцінок обраного користувача, відсортований за часом.

Записані дані слугують одразу двом завданням: вони в подальшому за бажанням дають змогу реалізувати інтерфейс («Мої оцінки») та накопичуються для подальшого перенавчання моделі рекомендацій. Завдяки використанню Admin SDK усі запити виконуються від імені сервера з перевіреним сервісним ключем, що відповідає вимогам до безпеки проєкту.

3.3.5 Ключові REST-ендпоінти серверної частини

Для взаємодії з клієнтською програмою Flask-бекенд використовує обмежений, але самодостатній набір маршрутів. Їх умовно поділено на два класи: публічні, що лише звертаються до каталогу TMDB, та захищені, які оперують даними конкретного профілю й потребують чинного Firebase-ID-токена. Усі захищені виклики автоматично перевіряються декоратором @check_token; послідовність його роботи наведено на діаграмі

послідовностей (рис. 2.4).

Публічний блок:

- GET/api/<cat>/list/<type> – повертає поточні списки popular, top_rated або upcoming для категорії movie / tv; додаткові фільтри не застосовуються, щоб показати реальний репертуар TMDB;
- GET/api/search/<cat> – виконує пошук за ключовим словом (query) й номером сторінки (page);
- GET/api/<cat>/<item_id>/details – агрегує розширену інформацію про фільм або серіал: базові дані, акторський склад, відеоматеріали, а також списки similar та recommendations від TMDB. За наявності токена до відповіді додається чинна оцінка користувача.

Захищений блок:

- POST/api/user/ratings – створює новий документ оцінки або оновлює наявний; значення ratingValue перевіряється на приналежність до діапазону 1–5;
- DELETE/api/user/ratings/<doc_id> – видаляє оцінку, якщо документ належить поточному UID; успішне виконання підтверджується кодом 204 No Content;
- GET/api/recommendations – головний маршрут системи. Якщо UID уже є в user_lookup, сервер користується матричною факторизацією: оцінює всі ще не переглянуті елементи й повертає топ-N. Коли ж профіль новий, застосовується резервний перелік popular_in_dataset_ids. Далі обидві вибірки збагачуються метаданими TMDB, і клієнт отримує впорядкований JSON-масив.

Запропоноване групування маршрутизатора наочно демонструє поділ відповідальності: публічні ендпоінти виконують роль «вікна» у зовнішній каталог, тоді як захищені операції обробляють персональні дані і взаємодіють із рекомендаційним ядром. Така структура спрощує підтримку, оскільки зміни в логіці однієї групи не впливають на роботу іншої, що відповідає принципам модульного проєктування.

3.4 Розробка клієнтської частини (React)

Клієнтська частина додатку, реалізована на бібліотеці React, відповідає за надання користувацького інтерфейсу, обробку взаємодій та відображення даних, отриманих від серверної частини.

3.4.1 Модуль взаємодії з API flaskApiClient.js

Щоб уникнути повторення коду HTTP-звернень у компонентах React, усю взаємодію з бекендом зосереджено в сервісному файлі flaskApiClient.js. У ньому створюється екземпляр Axios із базовою адресою http://localhost:5000/api. До клієнта під’єднано запит-інтерсептор: перед відправленням кожного запиту він отримує актуальний Firebase ID-токен (auth.currentUser.getIdToken()) і, якщо той наявний, підставляє його у заголовок Authorization: Bearer <token>. Завдяки цьому захищені маршрути, описані у 3.3.5, одержують коректно підписані запити, водночас публічні ендпоінти виконуються без додаткових накладних витрат.

Модуль експортує компактний набір функцій-обгорток: getItemDetails, postRating, fetchRecommendations та ін. – кожна відповідає окремому маршруту Flask API, тим самим спрощуючи код інтерфейсних компонентів

3.4.2 Клієнтський модуль firebase.js

Функції реєстрації, входу й відновлення пароля зосереджено у файлі firebase.js, який інсталює Firebase Client SDK, ініціалізує під’єднання до проєкту й експонує короткий перелік методів: signup, login, signInWithGoogle, signInWithFacebook, logout, resetPassword, а також ре-експорт слухача onAuthStateChanged. Після успішної операції SDK автоматично зберігає сеанс браузера й оновлює ID-токен, що спрощує клієнтську логіку та знімає потребу у власному механізмі сесій.

Обраний підхід виправданий з огляду на:

– готові засоби безпеки – хешування паролів та керування токенами виконує Firebase;
– просту інтеграцію OAuth – провайдери Google і Facebook підключаються двома викликами SDK;
– розділення відповідальності – фронтенд займається автентифікацією, тоді як бекенд виконує лише верифікацію отриманого токена.

Такий дизайн мінімізує повторюваний код, забезпечує масштабованість і відповідає стандартній практиці використання Firebase Authentication у веб-проєктах.

3.4.3 Демонстрація функціоналу розробленої онлайн-платформи

Клієнтський інтерфейс, реалізований за допомогою бібліотеки React, надає користувачеві інструменти для навігації по каталогу відеоконтенту, взаємодії з ним та отримання персоналізованих рекомендацій. Нижче детально розглянуто ключові функціональні можливості та екрани застосунку.

На першому кроці користувач знайомиться з формою входу(рис. 3.11), або реєстрації (рис. 3.12). Електронна адреса й пароль перевіряються на клієнтському боці засобами *react-hook-form*; за бажанням можна скористатися швидким входом через Google чи Facebook.

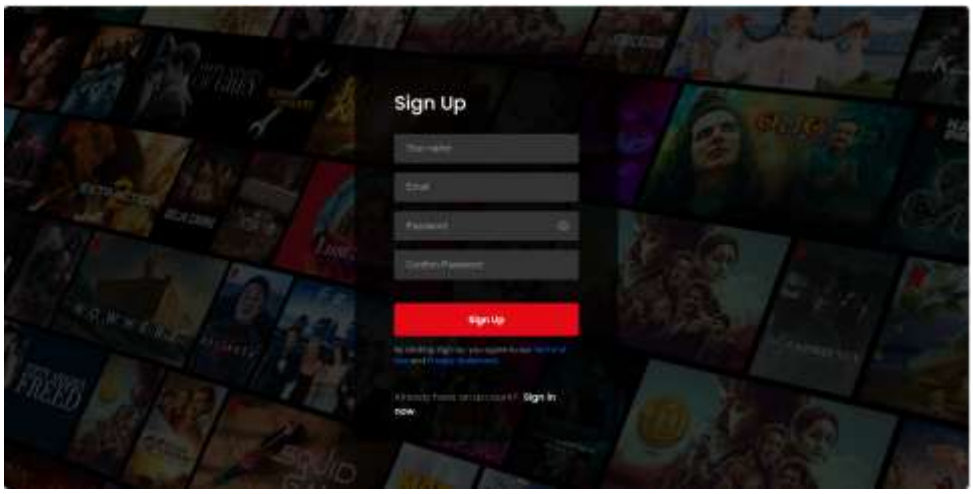


Рисунок 3.11 – Форма реєстрації нового облікового запису

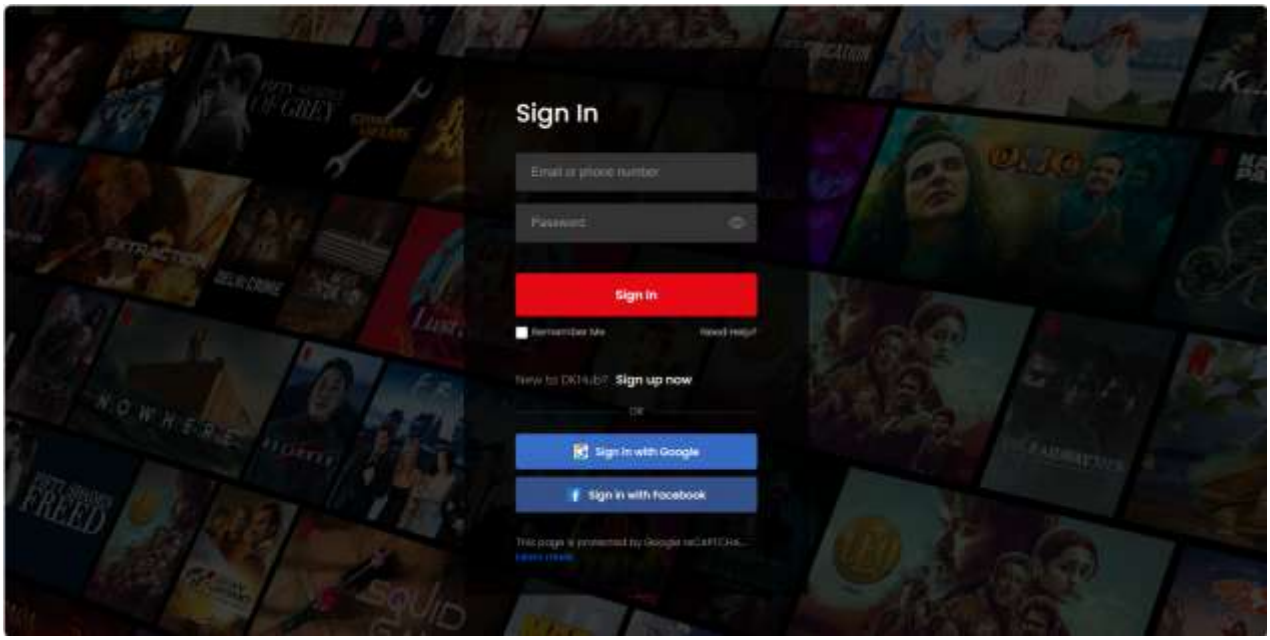


Рисунок 3.12 – Форма входу

Якщо пароль втрачено, натискання посилання *Need help?* відкриває діалог **Reset Password** (рис. 3.13). Сервіс надсилає одноразове посилання для відновлення доступу на вказану електронну адресу.

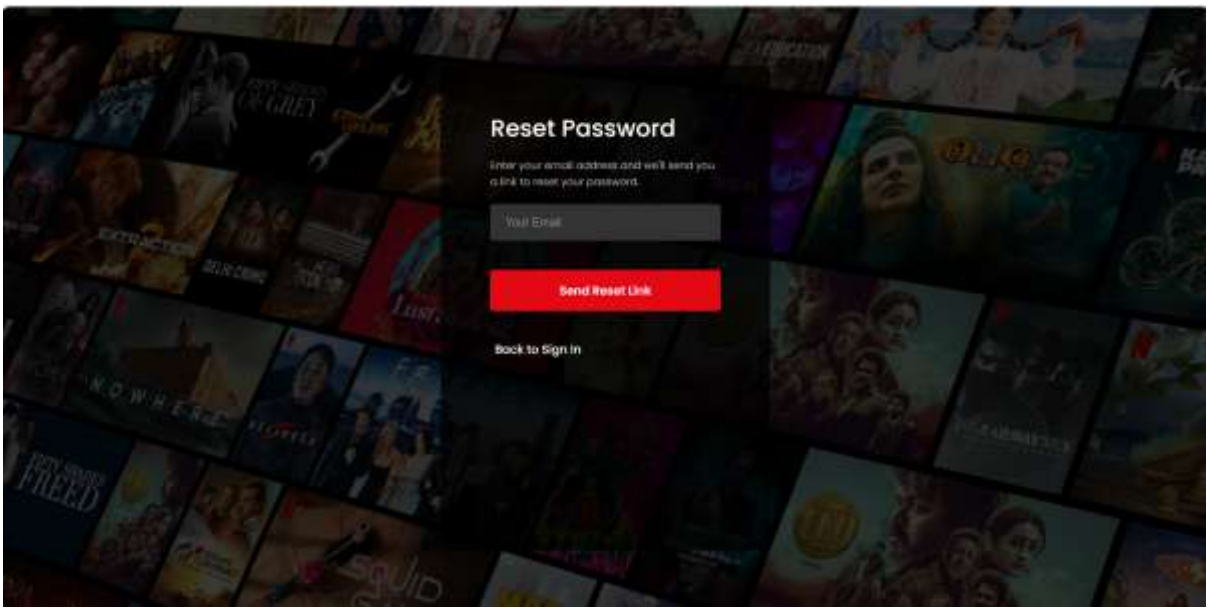


Рисунок 3.13 – Форма відновлення пароля

Після успішної автентифікації Firebase SDK створює сесію, а у верхній панелі з’являється пункт «Профіль» із можливістю завершити сеанс (рис. 3.14).

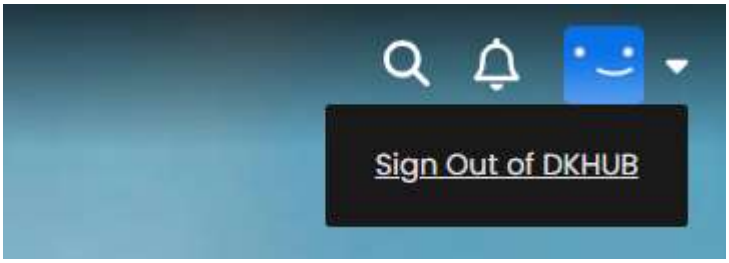


Рисунок 3.14 – Керування сесією у навігаційній панелі

Після входу клієнт автоматично потрапляє на маршрут *Home*. Загальний вигляд цієї сторінки подано на рис. 3.15. У верхній частині розташовано Hero-секцію(hero section), що динамічно формує слайдер із актуальних прем’єр, отриманих від бекенду одним запитом.



Рисунок 3.15 Головна сторінка розробленої онлайн платформи

Нижче розміщено кілька горизонтальних каруселей – «Популярні фільми», «Топ-рейтингові серіали», «Очікувані релізи» й інші добірки. Кожна карусель є екземпляром компонента *MovieList*, який отримує власний набір даних через асинхронний виклик до Flask API та відображається з поступовим завантаженням карток, щоб мінімізувати час рендеру першого екрана (рис. 3.16).

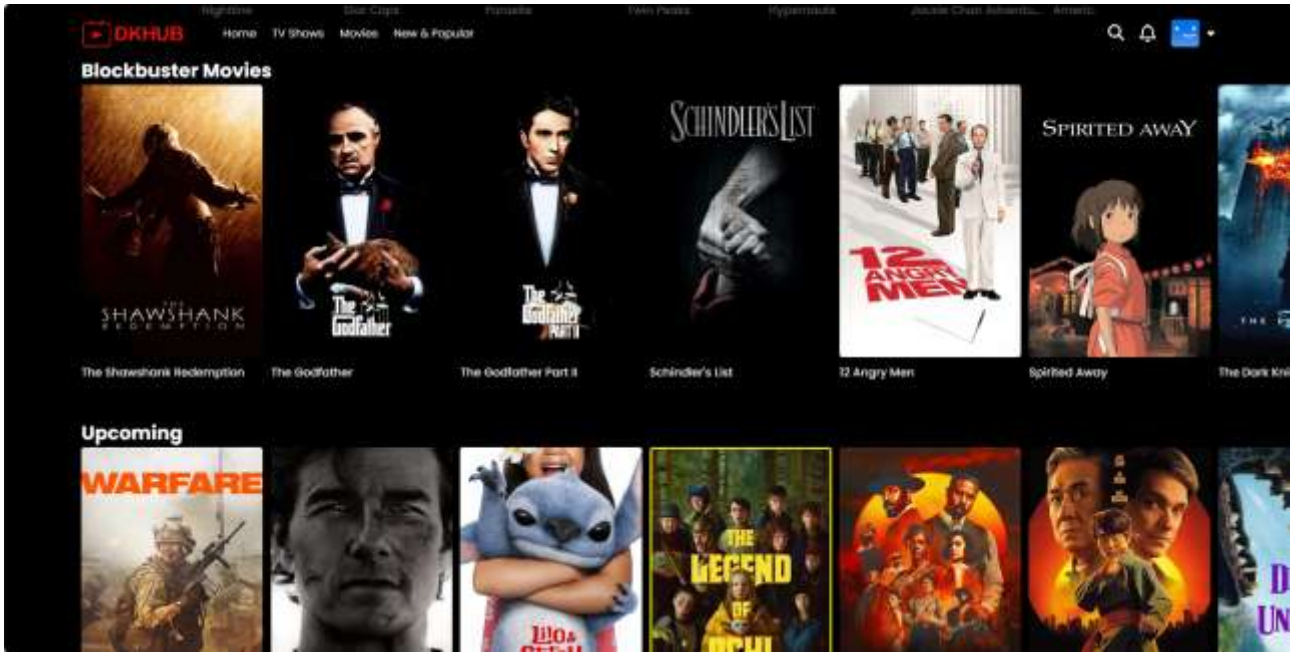


Рисунок 3.16 – Каруселі із контентом

Особливої уваги приділено блоку «Recommended for You». Якщо обліковий запис ще не зустрічався моделі, сервер повертає перелік фільмів із найвищим згладженим рейтингом у навчальному наборі; цей варіант інтерфейсу показано на рис. 3.17.



Рисунок 3.17 – Стартові рекомендації для нового користувача

В випадку коли UID користувача вже наявний у словнику *user_lookup*, Flask віддає персоналізований список, сформований статичною моделлю матричної факторизації. Для адміністратора передбачено демо-панель: для перевірки, де у спеціальному полі можна ввести умовний *demo_user_id* з

датасету на якому ми навчали модель і миттєво переглянути добірку, що була б показана цьому «старому» профілю, що зручно дозволяє продемонструвати можливості статичної MF(рис. 3.18).

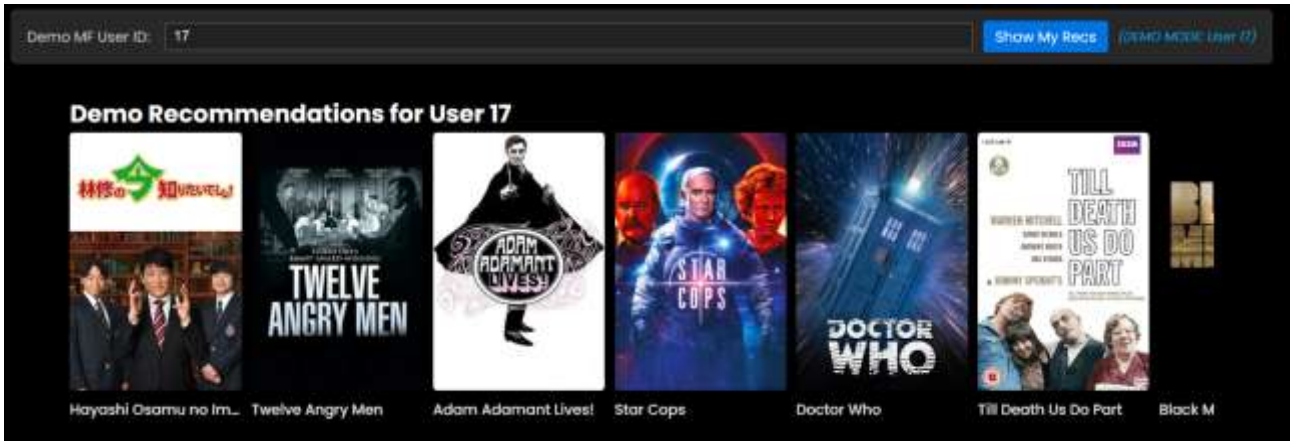


Рисунок 3.18 – Демо рекомендації для користувача з датасету

Клік по картці відкриває сторінку деталей (рис.3.19). Запит до бекенду виконується один раз і повертає агрегований JSON із базовими метаданими TMDb, списком акторів, доступними трейлерами, а також наборами «similar» і «recommendations». Зліва відображається постер, праворуч – назва, жанри, короткий опис і, якщо користувач авторизований, інтерактивна панель виставлення оцінки (рис. 3.20).



Рисунок 3.19 – Сторінка деталей

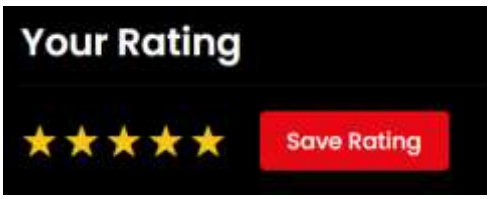


Рисунок 3.20 – інтерактивна панель виставлення оцінки

Оцінка відправляється в Firestore та оновлюється без перезавантаження сторінки; повторний натиск на ту саму кількість зірок очищує рейтинг, що автоматично ініціює запит на видалення.

Під основним описом та трейлерами розташовано дві каруселі: «Similar Titles» та «Recommendations», сформовані безпосередньо з відповіді TMDb (рис. 3.21).

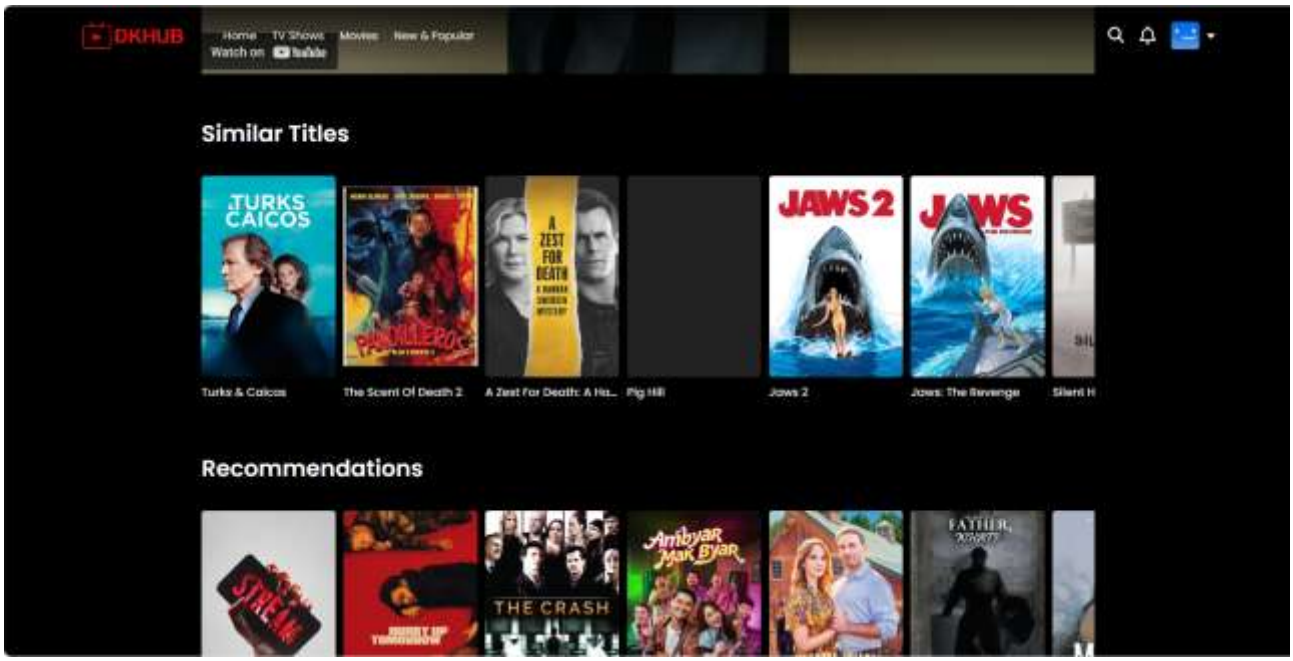


Рисунок 3.21 – Блоки «Similar Titles» та «Recommendations»

Пошуковий маршрут *Search* реалізовано єдиним компонентом MediaGrid, який підтримує безперервну підвантажувану сітку результатів (infinite scroll) (рис. 3.22).

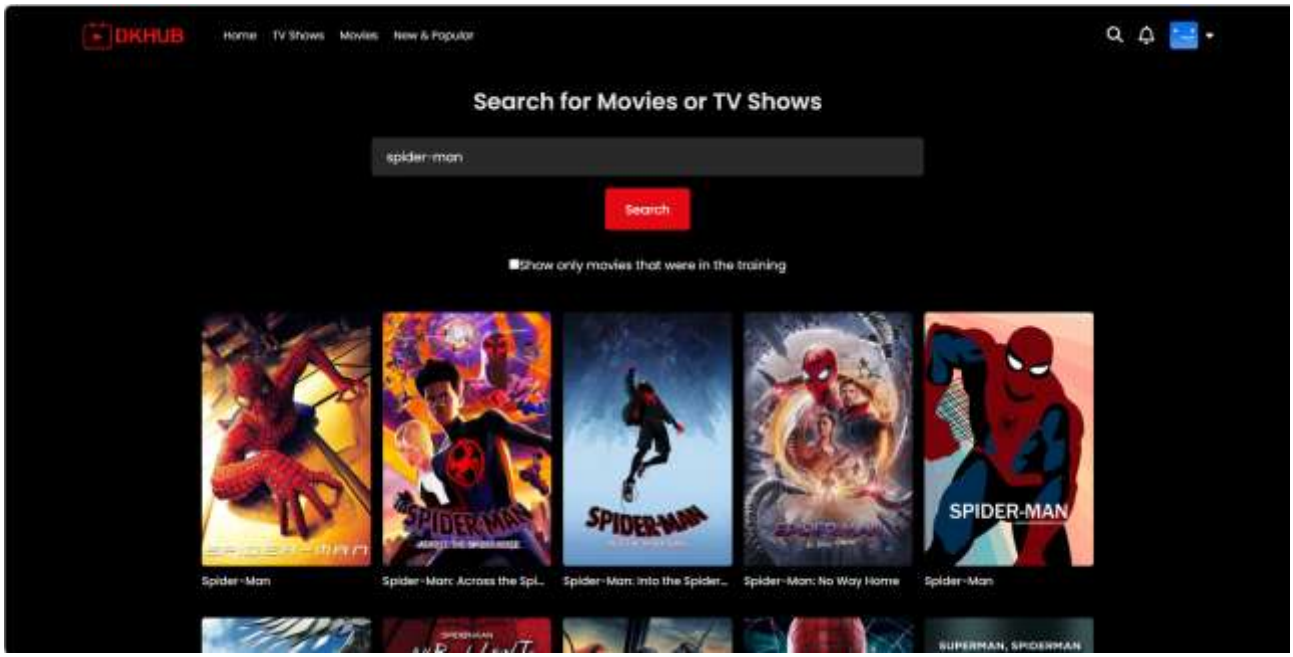


Рисунок 3.22 – Сторінка пошуку

Демонстровані екрани підтверджують, що прототип охоплює весь життєвий цикл взаємодії користувача з каталогом: від входу на платформу й перегляду тематичних добірок до отримання рекомендацій та виставлення індивідуальних оцінок, що відповідає вимогам типових стрімінг-сервісів.

ВИСНОВКИ

Метою дослідження було спроєктувати й реалізувати демонстраційний веб-сервіс, здатний пропонувати індивідуальні добірки фільмів на основі даних MovieLens 32M. На підставі виконаної роботи можна сформулювати такі підсумки:

- проведений огляд літератури довів доцільність використання матричної факторизації зі зміщеннями як базового алгоритму: модель зберігає лінійну складність прогнозу, добре працює з розрідженими матрицями та не потребує додаткової інформації про контент;
- проведено процедуру стратифікованої вибірки, що дає змогу зменшити обсяг MovieLens у кілька разів без викривлення розподілу користувацької активності. Таке скорочення суттєво зменшило час навчання та вимоги до пам’яті, залишивши при цьому майже повний перелік фільмів;
- на серверній стороні реалізовано Flask-API, який об’єднує модуль авторизації Firebase, підсистему роботи з оцінками у Firestore, модель PyTorch та клієнт TMDb. Клієнтський код побудовано у React з виділеним модулем flaskApiClient.js, що інкапсулює усі HTTP-запити й автоматично додає Firebase-токен;
- навчання статичної моделі на підготовленій вибірці показало середньоквадратичну помилку RMSE=0,87 при темпоральному розділі валідаційної вибірки;
- реалізовано простий механізм обробки холодного старту.

Практична значущість полягає у готовності прототипу до інтеграції у стрімінг-платформи малого та середнього масштабу, оскільки вся інфраструктура спирається на широко доступні хмарні сервіси.

Подальший розвиток проєкту доцільно спрямувати на:

– розширення статичної схеми гібридними або контекстними компонентами;
– автоматизацію офлайн-перенавчання та розгортання нових ваг;
– реалізувати онлайн-оновлення моделі;
– дослідження продуктивності у хмарних і розподілених середовищах.

Отримані результати доводять, що навіть статична матрична факторизація, реалізована з урахуванням сучасних веб-технологій і оптимізованої підмножини даних, здатна забезпечити релевантні персональні рекомендації й задовольнити вимоги до швидкодії в реальних умовах експлуатації.

ПЕРЕЛІК ПОСИЛАНЬ

1. Kuchar R. How recommendation systems transform information intake: Personalization, FOMO, and information bubbles : блог. – *ui42 Digital Blog*, 01.07.2024. URL : https://www.ui42.com/blog/how-recommendation-systems-transform-information-intake-personalization-fomo-and-information-bubbles (дата звернення : 20.01.2025).

2. Agarwal N. Understanding Hick's Law: How Can You Make Users Decide Quickly? *Alpha.One Blog*, 03.04.2024. URL : https://www.alpha.one/blog/understanding-hicks-law-how-can-you-make-users-decide-quickly (дата звернення : 20.01.2025).

3. What is the Netflix Prize competition and its relevance to recommender systems? *AI Quick Reference*. – *Milvus* / Zilliz, 2024. URL : https://milvus.io/ai-quick-reference/what-is-the-netflix-prize-competition-and-its-relevance-to-recommender-systems (дата звернення : 21.01.2025).

4. Flynn J. How YouTube's Recommendation System Works. *Particular Audience*, 04.01.2021. URL : https://particularaudience.com/youtube-recommendations/ (дата звернення : 21.02.2025).

5. Šimko Ľ., Felfernig A., Tautschnig M. An overview of video recommender systems: state-of-the-art and research issues. *Frontiers in Big Data*. 2023. Vol. 6. Article 1281614. DOI: 10.3389/fdata.2023.1281614. URL : https://www.frontiersin.org/articles/10.3389/fdata.2023.1281614/full (дата звернення : 22.01.2025).

6. Gutelle S. Reverse Engineering the YouTube Algorithm (Part II): How Engagement and Other Factors Drive Recommendations. *Tubefilter*. 16.02.2017. URL : https://www.tubefilter.com/2017/02/16/youtube-algorithm-reverse-engineering -part-ii/ (дата звернення : 23.01.2025).

7. Amatriain X., Basilico J. Netflix Recommendations: Beyond the 5 Stars (Part 1). *Netflix Technology Blog*. 06.04.2012. URL : https://netflixtechblog.com/

netflix-recommendations-beyond-the-5-stars-part-1-55838468f429 (дата звернення : 23.01.2025).

8. Steck H., Baltrunas L., Elahi E., Liang D., Raimond Y., Basilico J. Deep Learning for Recommender Systems : A Netflix Case Study. *AI Magazine*. 2021. Vol. 42, No 3. P. 7–18. URL : https://ojs.aaai.org/aimagazine/index.php/aimagazine/article/ view/18140 (дата звернення : 24.01.2025).

9. Inside Spotify's Recommender System: A Complete Guide to Spotify Recommendation Algorithms. Music Tomorrow, 09.02.2022. URL : https://www.music-tomorrow.com/blog/how-spotify-recommendation-system-works-a-complete-guide-2022 (дата звернення : 27.01.2025).

10. Spotify Recommendation System and User Engagement Strategies. TechAhead, 08.01.2024. URL : https://www.techaheadcorp.com/blog/spotify-recommendation-system/ (дата звернення : 27.01.2025).

11. Smith B., Linden G. The history of Amazon's recommendation algorithm. Amazon Science, 27.07.2021. URL : https://www.amazon.science/the-history-of-amazons-recommendation-algorithm (дата звернення : 28.01.2025).

12. Linden G., Smith B., York J. Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*. 2003. Vol. 7, No 1. P. 76–80. URL : https://www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf (дата звернення : 28.01.2025).

13. Cold start (recommender systems). Вільна енциклопедія Wikipedia. URL : https://en.wikipedia.org/wiki/Cold_start_(recommender_systems) (дата звернення : 29.01.2025).

14. What is collaborative filtering? IBM Think, 21.03.2024. URL : https://www.ibm.com/think/topics/collaborative-filtering (дата звернення : 29.01.2025).

15. Das S., Clark E., Goyal S., Basilico J. Foundation Model for Personalized Recommendation. *Netflix Technology Blog*. 07.03.2024. URL :

https://netflixtechblog.com/foundation-model-for-personalized-recommendation-1a0bd8e02d39 (дата звернення : 30.01.2025).

16. What is content-based filtering? IBM Think, 21.03.2024. URL : https://www.ibm.com/think/topics/content-based-filtering (дата звернення : 30.01.2025).

17. Adomavicius G., Tuzhilin A. Towards the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering*. 2005. Vol. 17, No 6. P. 734–749. URL : https://wiki.epfl.ch/edicpublic/documents/Candidacy%20exam/recommender-systems-survey-2005-2.pdf (дата звернення : 31.01.2025).

18. Collaborative filtering. BotPenguin Glossary. URL : https://botpenguin.com/glossary/collaborative-filtering (дата звернення : 03.02.2025).

19. Koren Y., Bell R., Volinsky C. Matrix Factorization Techniques for Recommender Systems. *Computer*. 2009. Vol. 42, No 8. P. 30–37. URL : https://datajobs.com/data-science-repo/Recommender-Systems-[Netflix].pdf (дата звернення : 03.02.2025).

20. Matrix Factorization. Dive into Deep Learning. URL : https://d2l.ai/chapter_recommender-systems/mf.html (дата звернення : 04.02.2025).

21. Dong Z., Wang Z., Xu J., Tang R., Wen J. A Brief History of Recommender Systems. *arXiv Preprint arXiv:2209.01860*. 2022. URL : https://arxiv.org/pdf/2209.01860 (дата звернення : 04.02.2025).

22. Fkih F. Similarity measures for Collaborative Filtering-based Recommender Systems : Review and experimental comparison. *Journal of King Saud University – Computer and Information Sciences*. 2022. Vol. 34, Issue 10B. P. 9635–9653. URL : https://www.sciencedirect.com/science/article/pii/S1319157821002652 (дата звернення : 05.02.2025).

23. Konstan J. A., Riedl J. Two Decades of Recommender Systems at Amazon.com. *IEEE Internet Computing*. 2022. Vol. 26, No 1. P. 10–17. URL :

https://assets.amazon.science/76/9e/7eac89c14a838746e91dde0a5e9f/two-decades-of-recommender-systems-at-amazon.pdf (дата звернення : 05.02.2025).

24. Herlocker J. L., Konstan J. A., Terveen L. G., Riedl J. T. Evaluating Collaborative Filtering Recommender Systems. *ACM Transactions on Information Systems*. 2004. Vol. 22, No 1. P. 5–53. URL : https://grouplens.org/site-content/uploads/evaluating-TOIS-20041.pdf (дата звернення : 06.02.2025).

25. Niitwork0921. Memory-Based vs. Model-Based Collaborative Filtering Techniques. *Medium*. 14.12.2021. URL : https://medium.com/@niitwork0921/memory-based-vs-model-based-collaborative-filtering-techniques-c0a7f6ec4f5f (дата звернення : 06.02.2025).

26. Koren Y. The BellKor Solution to the Netflix Grand Prize. Technical report, 08.2009. URL : https://www2.seas.gwu.edu/~simhaweb/champalg/cf/papers/KorenBellKor2009.pdf (дата звернення : 07.02.2025).

27. Alshbanat H. I., Benhidour H., Kerrache S. A Survey of Latent Factor Models in Recommender Systems. *arXiv Preprint arXiv:2405.18068*. 2024. URL : https://arxiv.org/pdf/2405.18068 (дата звернення : 07.02.2025).

28. Wan C., Dang P., Zhao T., Zang Y., Zhang C., Cao S. Bias-aware probabilistic Boolean matrix factorization. Proceedings of the 39th International *Conference on Machine Learning (ICML)*. 2022. Vol. 162. P. 22905–22918. URL : https://proceedings.mlr.press/v180/wan22a/wan22a.pdf (дата звернення : 10.02.2025).

29. Collaborative Filtering. Apache Spark Documentation 3.5.1. URL : https://spark.apache.org/docs/latest/mllib-collaborative-filtering.html (дата звернення : 10.02.2025).

30. Gope J., Jain S. K. A survey on solving cold-start problem in recommender systems. *2017 International Conference on Computing, Communication and Automation (ICCCA)* (Greater Noida, 05–06.05.2017). P. 133–138. DOI : 10.1109/CCAA.2017.8229786. URL : https://www.researchgate.net/publication/

322001810_A_survey_on_solving_cold_start_problem_in_recommender_systems (дата звернення : 11.02.2025).

31. David D. What is the Cold Start Problem in Recommender Systems? freeCodeCamp.org, 25.02.2023. URL : https://www.freecodecamp.org/news/cold-start-problem-in-recommender-systems (дата звернення : 11.02.2025).

32. Yuan H., Hernandez A. A. User Cold Start Problem in Recommendation Systems : A Systematic Review. *IEEE Access*. 2023. Vol. 11. P. 138038–138051. URL : https://www.researchgate.net/publication/376140792_User_Cold_Start_Problem_in_Recommendation_Systems_A_Systematic_Review (дата звернення : 12.02.2025).

33. Ma Y., Narayanaswamy B., Lin H., Ding H. Temporal-Contextual Recommendation in Real-Time. *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD ’20)* (Virtual Event, 23–27.08.2020). 9 p. DOI : 10.1145/3394486.3403278. URL : https://assets.amazon.science/96/71/ d1f25754497681133c7aa2b7eb05/temporal-contextual-recommendation-in-real-time.pdf (дата звернення : 12.02.2025).

34. Vassilakis C., та ін. Title Launch Observability at Netflix Scale. *Netflix Technology Blog*. 2024. URL : https://netflixtechblog.com/title-launch-observability-at-netflix-scale-c88c586629eb (дата звернення : 13.02.2025).

35. How to solve the cold start problem in recommender systems. *ThingSolver*. 19.01.2024. URL : https://thingsolver.com/blog/the-cold-start-problem/ (дата звернення : 13.02.2025).

36. In-App Surveys: Top Strategies to Boost User Engagement. UserGuiding. URL : https://userguiding.com/blog/in-app-surveys (дата звернення : 14.02.2025).

37. Monteil J., Vaskovych V., Majumder A., van den Hengel A., Lu W. MARec: Metadata Alignment for Cold-Start Recommendation. *Amazon*. 2024. URL : https://assets.amazon.science/5f/3e/d92eda32446891e927ccb0a63b06/marec-metadata-alignment-for-cold-start-recommendation.pdf (дата звернення : 14.02.2025).

38. Sharma M., Pant B., Singh V. Demographic profile building for cold start in recommender system: A social media fusion approach. Materials Today : Proceedings. 2021. Vol. 46, No 13. P. 11208–11212. DOI : 10.1016/j.matpr.2021.02.428. URL : https://www.sciencedirect.com/science/article/abs/pii/S2214785321015480 (дата звернення : 17.02.2025).

39. Harper F. M., Konstan J. A. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems*. 2015. Vol. 5, No 4. Article 19. DOI : 10.1145/2827872. URL : https://dl.acm.org/doi/10.1145/2827872 (дата звернення : 19.02.2025).