Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук

Бакалаврська кваліфікаційна робота на тему:

ПРОГРАМНИЙ МОДУЛЬ ПІДБОРУ КІНЕМАТОГРАФІЧНОГО ТВОРУ НА ОСНОВІ УПОДОБАНЬ КОРИСТУВАЧА

Виконала: студентка 4 курсу, групи 2КН-21б
спеціальності 122 – Комп'ютерні науки
(шифр і назва напряму підготовки, спеціальності)
Христина КУЛЬБІДА
(прізвище та ініціали)
Керівник БКР: д.т.н., проф. каф. КН
Ярослав ІВАНЧУК
(прізвище та ініціали)
« 04 » червня 2025 р.
Рецензент: д.т.н., проф., зав. каф. КСУ
Вячеслав КОВТУН
(прізвище та ініціали)
« 05 » червня 2025 р.

Допущено до захисту
Зав. кафедри проф., д.т.н. Андрій ЯРОВИЙ
« 06 » червня 2025 р.

Вінниця ВНТУ - 2025 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних наук
Рівень вищої освіти перший (бакалаврський)
Галузь знань – 12 Інформаційні технології
Спеціальність – 122 Комп'ютерні науки
Освітньо - професійна програма – Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
проф., д.т.н. Андрій ЯРОВИЙ
«05» травня 2025 р.

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Кульбіді Христині Олександрівні

1. Тема роботи: Програмний модуль підбору кінематографічного твору на основі уподобань користувача.
керівник роботи: Іванчук Ярослав Володимирович, д.т.н, професор кафедри КН
затверджені наказом вищого навчального закладу «20» березня 2025 року №97
2. Строк подання студентом роботи 30 травня 2025 р.
3. Вихідні дані до роботи: датасет кінематографічних творів; список рекомендованих фільмів – не менше 5; датасет із записами про результати рекомендацій – 9 характеристик, датасет із записами про зареєстрованих користувачів – 2 характеристики.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): визначення об'єкту, предмета, мети та задач подальшого дослідження; аналіз предметної області, існуючих методів, алгоритмів та інструментів для створення програмного модуля підбору кінематографічного твору на основі уподобань користувача; проєктування програмного модуля підбору кінематографічного твору на основі уподобань користувача; програмна реалізація програмного модуля підбору кінематографічного твору на основі уподобань користувача
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень): архітектура програмного модуля; структурна схема програмного модуля; UML-діаграма класів; схема алгоритму взаємодії розроблених модулів;

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-3	Ярослав ІВАНЧУК, д.т.н., проф. каф. КН		

7. Дата видачі завдання 05 травня 2025 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітка
		початок	закінчення	
1	Визначення об'єкта, предмета, мети та задачі подальшого дослідження	05.05.2025	07.05.2025	
2	Аналіз предметної області, існуючих методів, алгоритмів та інструментів для створення програмного модуля підбору кінематографічних творів на основі уподобань користувача;	08.05.2025	13.05.2025	
3	Проєктування програмного модуля підбору кінематографічних творів на основі уподобань користувача;	14.05.2025	19.05.2025	
4	Програмна реалізація програмного модуля підбору кінематографічних творів на основі уподобань користувача	20.05.2025	30.05.2025	
5	Оформлення матеріалів до захисту БКР	31.05.2025	04.06.2025	

Студент _____ Христина КУЛЬБІДА
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ярослав ІВАНЧУК
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська кваліфікаційна робота складається з 93 сторінок формату А4, на яких представлено 21 рисунок, 2 таблиці, список використаної літератури містить 24 найменування.

Ця бакалаврська робота присвячена розробці програмного модуля підбору кінематографічного твору на основі вподобань користувача. Головною метою роботи є підвищення точності та ефективності індивідуального підбору кінематографічних творів шляхом розробки та впровадження програмного модуля рекомендаційної системи, що базується на вподобаннях користувача з використанням сучасних методів фільтрації та інтерактивного інтерфейсу і дозволяє користувачам швидко знаходити фільми відповідно до заданих жанрових, мовних, часових та інших вподобань. У ході реалізації було використано графічний інтерфейс на основі бібліотеки Tkinter, а також реалізовано логіку обробки даних з використанням Pandas. Система базується на даних набору TMDB, який містить ключову інформацію про фільми. У розробці реалізовано інтерфейс опитування користувача, обробку введених параметрів, генерацію рекомендацій та логування результатів.

Результатом роботи є надійний і зручний у використанні програмний модуль, що дозволяє підбирати фільми за індивідуальними критеріями та покращує взаємодію користувача з кінематографічним контентом.

Ключові слова: рекомендаційна система, підбір кінематографічних творів, уподобання користувача, Python, датасет фільмів.

ABSTRACT

The bachelor's qualification thesis consists of 93 A4 pages, including 21 figures, 2 tables, and a list of 24 references.

This bachelor's thesis is dedicated to the development of a software module for selecting cinematographic works based on user preferences. The main goal of the work is to improve the accuracy and efficiency of personalized film selection by developing and implementing a recommendation system module that leverages user preferences. The system utilizes modern filtering techniques and an interactive interface, enabling users to quickly find films that match specified genre, language, time period, and other preferences.

The implementation features a graphical user interface built with the Tkinter library, and data processing logic developed using the Pandas library. The system is based on the TMDB dataset, which provides key information about movies. The software includes a user survey interface, input parameter processing, recommendation generation, and result logging.

The outcome of the work is a reliable and user-friendly software module that allows for film selection based on individual criteria and enhances user interaction with cinematic content.

Keywords: recommendation system, film selection, user preferences, Python, movie dataset.

ЗМІСТ

ВСТУП

Актуальність досліджень. У сучасному цифровому середовищі користувачі щодня взаємодіють з великою кількістю інформаційного та мультимедійного контенту, зокрема фільмів, серіалів і відеоматеріалів. Такий стрімкий ріст обсягу доступних даних створює нові виклики у сфері зручного та релевантного доступу до контенту. Одним із найбільш ефективних інструментів вирішення цієї проблеми є рекомендаційні системи – програмні засоби, що аналізують поведінку, інтереси та вподобання користувачів з метою надання персоналізованих пропозицій. Особливої актуальності такі системи набувають у сфері кінематографу, де вибір фільму часто залежить від індивідуальних смаків, жанрових уподобань, настрою чи попереднього досвіду перегляду. Саме тому створення інтелектуального модуля, здатного враховувати ці особливості та формувати точні рекомендації, є важливим напрямом розвитку сучасних інформаційних технологій.

Зараз, коли у житті людей велику частину займає цифровий контент, що стрімко розвивається, постає проблема його ефективного відбору. Щодня користувачі стикаються з величезною кількістю музики, фільмів та інших матеріалів, що ускладнює процес вибору. Традиційні пошукові інструменти вже не в змозі задовольнити потребу користувача в швидкому і точному підборі контенту, що відповідає індивідуальним уподобанням. Тому виникає потреба в розробці систем, здатних аналізувати вподобання користувачів і автоматично генерувати персоналізовані рекомендації [1].

Рекомендаційні системи є важливою частиною сучасних платформ цифрового контенту, таких як Netflix, YouTube, Megogo та інші. Вони підвищують задоволеність користувачів та збільшують залученість аудиторії. Водночас розробка рішень, орієнтованих на рекомендацію кінематографічних творів, набуває особливого значення через їхню культурну, розважальну та соціальну значущість [2].

Мета дослідження: підвищити точність індивідуального підбору кінематографічних творів шляхом розробки та впровадження програмного модуля

рекомендаційної системи, що базується на вподобаннях користувача.

Предметом дослідження є програмний модуль підбору кінематографічного твору на основі уподобань користувача та методи й програмні засоби побудови рекомендаційних систем.

Об’єктом дослідження є процес розробки та функціонування рекомендаційних систем у сфері цифрових медіа, зокрема у кіноіндустрії.

Задачі подальшого дослідження:

1. Аналіз актуальності розробки рекомендаційних систем в сфері кінематографу та розгляд основних проблем в цій сфері.
2. Розробка загального алгоритму програмного модуля підбору кінематографічних творів на основі вподобань користувача.
3. Виконати програмну реалізацію модуля підбору кінематографічних творів на основі вподобань користувача.
4. Провести тестування програмного модулю підбору кінематографічних творів на основі вподобань користувача.
5. Розробка інструкції користувача програмного модуля підбору кінематографічних творів на основі вподобань користувача.

Апробація роботи. Робота апробувалася на конференції «LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2025)».

Публікації: електронні тези на тему «Рекомендаційний метод підбору музичних композицій» [3].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз актуальності розробки рекомендаційних систем в сфері кінематографу та розгляд основних проблем в цій сфері

В епоху цифрових технологій кіно відіграє важливу роль у культурному житті суспільства. Щороку у світі створюються тисячі нових фільмів, серіалів, анімаційних стрічок та інших відеопродуктів, поповнюючи і без того вражаючу бібліотеку кіноконтенту. Завдяки стрімкому технологічному розвитку виробництво кінематографічного матеріалу стало доступнішим, а отже, більш поширеним. Це призвело до надзвичайно швидкого збільшення кількості фільмів, що випускаються в кінотеатрах, на телебаченні і, перш за все, на онлайн-платформах.

З появою численних потокових сервісів, таких як Netflix, Amazon Prime Video, Hulu, Disney+, Apple TV+ та інших, користувачі отримали необмежений доступ до найрізноманітніших фільмів і серіалів з усього світу. З одного боку, це дозволяє користувачам дивитися контент у зручний для них час і без прив'язки до телевізійної програми. З іншого боку, користувачі стикаються з проблемою вибору: серед тисяч найменувань важко знайти твір, який найбільше відповідає їхнім інтересам, настрою чи культурним уподобанням. Таке «інформаційне перевантаження» не лише створює практичні труднощі у виборі контенту, але й призводить до марнування часу та зниження задоволення від перегляду.

Зростаюча кількість фільмів, що виходять щороку, ускладнює глядачам вибір фільму, який найбільше відповідає їхнім уподобанням. Завдяки всім сучасним потоковим сервісам споживачі мають доступ до великої бібліотеки контенту, проте це також значно ускладнює вибір кінематографічного твору для перегляду. Саме тому з’являється потреба системах рекомендацій, які можуть запропонувати персоналізований контент і покращити користувацький досвід, як наслідок рекомендуючи лише ті фільми, які, найімовірніше, викличуть інтерес і відповідатимуть індивідуальним смакам глядача [4].

Окрім цього, інтерес глядачів до кіно стрімко зростає з кожним роком. Все більше людей дивляться відеоконтент, щоб розслабитися, відчути нові емоції, отримати нові знання або навіть самовиразитися. Цей інтерес підсилюється стрімким розвитком кіноіндустрії, появою нових кіножанрів, форм і технологій, а також зростанням ролі візуального мистецтва в соціальних мережах. Тому створення інструментів, які допоможуть орієнтуватися в цьому інформаційному просторі, є не лише бажаним, а й необхідним.

Ці сучасні виклики можна вирішити, розробивши програмний модуль з функціями інтелектуального відбору відеоконтенту на основі вподобань користувача. Такий модуль зможе не тільки спростити процес пошуку релевантного контенту, але й підвищити ефективність взаємодії з медіа-платформами, зменшити втрати часу та покращити загальний досвід перегляду. Враховуючи раніше описані тези можемо зробити висновок, що тема даної дипломної роботи є дуже актуальною в контексті інформаційних технологій, індустрії розваг, персоналізованих сервісів та їх розвитку.

Системи рекомендацій в сфері кінематографу можна ефективно використовувати на різних платформах, які пропонують користувачам доступ до відеоконтенту. Перш за все це стосується популярних стрімінгових сервісів, таких як Netflix, Amazon Prime, Disney+, MEGOGO, Sweet.tv тощо. На таких платформах інтелектуальні системи підбору контенту діють як персоналізований навігатор: вони аналізують ваші глядацькі звички, уподобання, рейтинги фільмів, жанрові вподобання та поведінкові фактори (наприклад, скільки часу ви витрачаєте на певний контент). На основі такого типу даних система формує рекомендації - добірку фільмів і серіалів, які можуть бути цікавими для конкретного користувача. Це не лише економить час, але й покращує користувацький досвід на платформі та підвищує лояльність користувачів [5].

Окрім стрімінгових сервісів, такі системи також можуть бути ефективно інтегровані у вебсайти кінотеатрів. Проте, у цьому випадку мета рекомендаційної системи дещо інша: вона може допомогти користувачеві вибрати фільм для перегляду в кінотеатрі враховуючи попередні відвідування, жанрові уподобання і

навіть актуальних трендів серед друзів (якщо вони зареєстровані й авторизовані в соціальних мережах пов’язаних з цим вебсайтом чи на самій вебсторінці). Наприклад, користувач, який часто відвідує кінотеатр для перегляду фільмів у жанрі фентезі або наукової фантастики, може отримувати рекомендації для цього жанру, коли виходять нові фільми. Такого типу системи також можуть запропонувати квитки на основі особистої історії переглядів користувача або заохочувати соціальну взаємодію, рекомендуючи фільм, який друзі/знайомі вже переглянули [6].

Незважаючи на очевидні переваги, розробники рекомендаційних систем у кіноіндустрії стикаються з низкою проблем. Однією з найважливіших є проблема «холодного старту», коли система не має достатньо інформації про нового користувача або новий фільм, щоб надати адекватні рекомендації. Це особливо часто трапляється у випадках, коли користувач ще не здійснював жодних дій на платформі, і системі доводиться робити певні припущення щодо його вподобань.

Інша основна проблема полягає в тому, що вибір рекомендацій дуже обмежений, оскільки система пропонує лише той контент, який дуже схожий на раніше переглянутий. Це призводить до того, що користувач «застрягає» в певному жанрі чи стилі і втрачає можливість відкрити для себе нові типи фільмів. У таких випадках користувач позбавлений різноманітності і, зрештою, втрачає цікавість до платформи.

Якість даних також є проблемою, якщо система не має доступу до повної достовірної інформації про користувача, або якщо рейтинги фільмів не є справедливими (до прикладу, через ботів або упереджені відгуки). У таких випадках результати рекомендацій можуть бути невідповідними або небажаними для користувача.

Останнім, та не менш важливим викликом є знаходження балансу між популярністю та персоналізацією. Системи часто рекомендують найпопулярніші фільми, які мають високі позиції в загальних рейтингах. Проте, такі рекомендації не завжди враховують індивідуальні смаки користувача і призводять до того, що контент стає загальнодоступним, а не персоналізованим [7].

Одним із основних завдань цієї дипломної роботи є дослідження вищезазначених проблем та пошук ефективних рішень. Особливу увагу буде приділено створенню програмного модуля, який зможе не лише рекомендувати релевантні фільми, але й адаптуватися до нових користувачів, підтримувати різні рекомендації та надавати якісний персоналізований досвід. Це зробить рекомендаційний механізм гнучким, адаптивним і простим у використанні для стрімінгових сервісів, кінотеатрів та інших платформ кіноконтенту.

1.2 Дослідження основних методів розробки систем рекомендацій та визначення найбільш оптимального методу

Метою будь-якої рекомендаційної системи є створення ефективної взаємодії з користувачем, перед яким стоїть завдання вибрати найбільш підходящий товар об’єкт з великої кількості доступних варіантів. Часто користувачеві може не вистачати знань, досвіду або часу, щоб знайти контент, який найкраще відповідає його інтересам або поточним потребам. У процесі взаємодії користувач надає системі інформацію про свої смаки та преференції прямо, за допомогою оцінок чи вподобань або опосередковано, за допомогою переглядів, часу взаємодії чи поведінкових патернів. На основі цієї інформації рекомендаційна система за допомогою певних алгоритмів обробки даних генерує персоналізований список елементів, в нашому випадку фільмів, серіалів, відео тощо, які скоріш за все будуть корисними чи цікавими для конкретного користувача.

Тобто, узагальнюючи, можемо сказати, що система рекомендацій — це програмне рішення, створене для надання персоналізованих пропозицій чи рекомендацій стосовно продуктів, послуг або інформаційного контенту, з урахуванням індивідуальних особливостей користувачів. Вона аналізує вподобання та поведінкові дані користувачів, щоб забезпечити максимально релевантні рекомендації.

Рекомендаційні системи широко використовуються в різних сферах, таких як електронна комерція, пошукові системи, освітні платформи, медіа-сервіси та

стрімінгові платформи. Їх використання покращує якість взаємодії користувача з інформаційною системою, робить ефективнішим пошук релевантного контенту та підвищує задоволеність користувачів від досвіду роботи з платформою.

У сучасних умовах для побудови рекомендаційних систем використовуються різні підходи, зокрема фільтрація на основі контенту, колаборативна фільтрація, гібридні підходи, системи, засновані на знаннях, та підходи, засновані на машинному навчанні. Кожен з цих методів має свої особливості, сфери застосування, сильні та слабкі сторони [8].

Розглянемо детальніше кожен із вказаних методів, проаналізуємо їхні переваги та недоліки, а також оцінимо доцільність використання кожного з них для розробки ефективного програмного модуля підбору кінематографічного твору.

Системи рекомендацій, що працюють за методом фільтрації на основі контенту створюють персоналізовані пропозиції на основі індивідуальних інтересів кожного користувача. Вони покладаються як на явні дані, які надає користувач до прикладу, рейтинги або відгуки, так і на неявні, як перегляди, прослуховування чи збереження. На основі цієї інформації поступово створюється унікальний профіль користувача, який адаптується і покращується відповідно до взаємодії користувача з системою. Чим більш активний користувач при використанні платформи, тим точніше система може оцінити його смаки і запропонувати йому цікавий для нього контент.

На рисунку 1.1 зображено загальний принцип роботи системи рекомендацій, що використовує контентну фільтрацію.

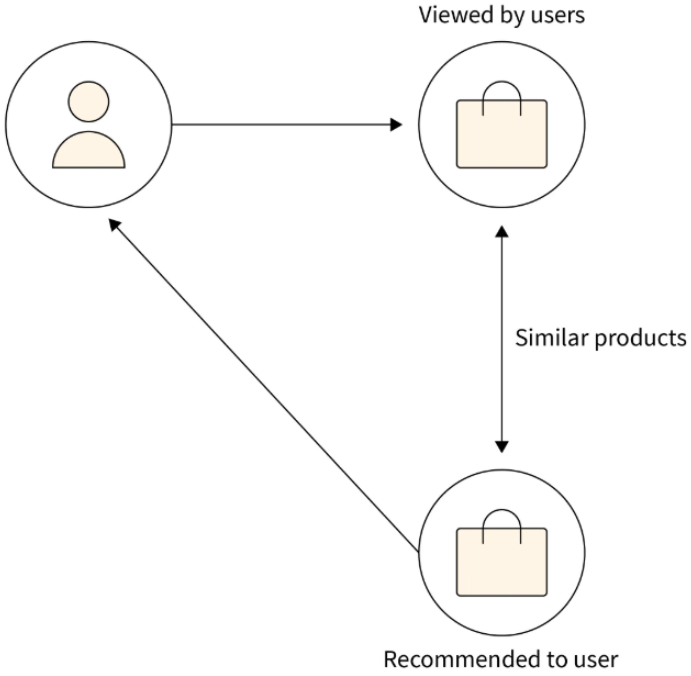

Рисунок 1.1 – Схема роботи рекомендаційної системи, що використовує фільтрацію на основі контенту

Профіль користувача зазвичай представляється як вектор, що відображає його вподобання. Цей підхід базується на матриці корисності – структурі, яка показує зв'язок між користувачем і певними об'єктами, такими як певні продукти, фільми чи музика. Таким чином, на основі загальних характеристик контенту, який вже викликав позитивну реакцію у користувача, можна оцінити ймовірність того, що йому сподобається новий об'єкт.

Заповнювати цю матрицю повністю є абсолютно не обов’язковим, оскільки часткових даних достатньо, щоб система зробила правильні висновки. Наприклад, якщо кінематографічний твір отримав багато негативних оцінок та рецензій від користувачів зі схожими смаками, система скоріш за все не запропонує його іншому користувачеві зі схожим профілем [9].

Для оцінки схожості між користувачем і певним елементом найчастіше використовується метод обчислення косинусної відстані між відповідними векторами. Це дає можливість визначити, наскільки близькі їхні характеристики один до одного. Наприклад, якщо користувач часто дивиться фільми та серіали з глибоким психологічним змістом, система надаватиме перевагу творам зі схожими

характеристиками, оскільки їхні вектори «ближчі» до профілю користувача.

Однією з головних переваг фільтрації на основі контенту є можливість створювати індивідуальні рекомендації, навіть якщо система ще не має великої бази даних стосовно інших користувачів. Це особливо важливо при роботі з новим або менш популярним контентом, який ще не отримав достатньої кількості відгуків. Тому, контент-фільтрація є ефективним підходом у випадках, коли необхідно швидко і точно адаптувати рекомендації до конкретного користувача при цьому, не маючи достатньо інформації про вподобання інших, подібних йому користувачів [10].

Ідея рекомендаційних систем, що базуються на колаборативній фільтрації полягає в тому, щоб знайти користувачів зі схожими смаками. Замість того, щоб аналізувати характеристики самого контенту, ці системи зосереджуються на вподобаннях інших користувачів. Інакше кажучи, якщо кілька людей мають схожі інтереси, то те, що подобається одній людині, швидше за все, сподобається й іншій.

Такий підхід дозволяє об'єднувати користувачів у групи за інтересами і на основі вподобань інших членів групи генерувати персоналізовані рекомендації.

Існує два основних типи реалізацій колаборативної фільтрації. Перший – user-based (тобто на основі користувачів), коли система шукає користувачів зі схожими рейтингами і на основі дій певної групи користувачів прогнозує, що може сподобатися конкретній людині. Другий – item-based, заснований на схожості між самими елементами контенту, тобто якщо користувачеві подобається певний фільм, йому можуть запропонувати інший фільм, який має схожі рейтинги серед інших глядачів [11].

На рисунку 1.2 подано спрощену схему функціонування обох типів колаборативної фільтрації — user-based та item-based.

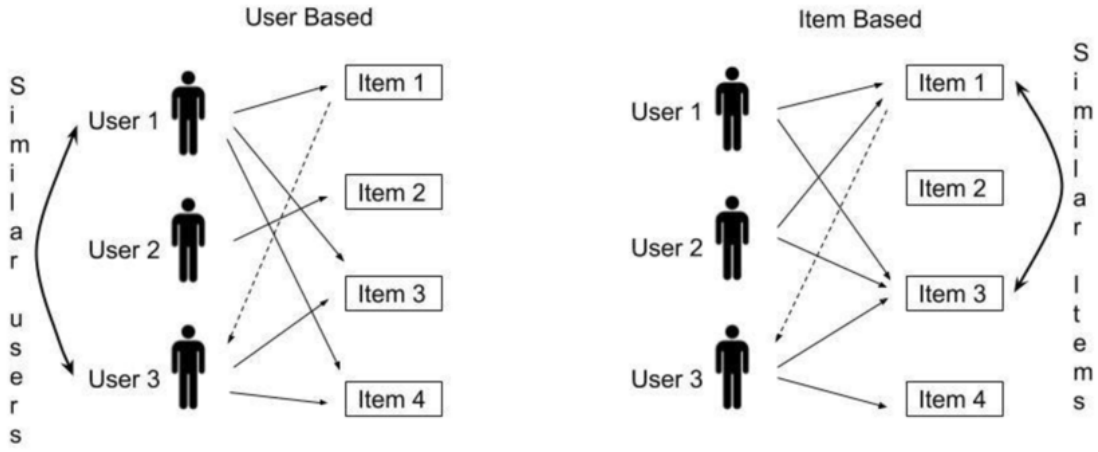

Рисунок 1.2 - Візуалізація user-based та item-based колаборативної фільтрації

Одна з головних переваг цього підходу полягає в тому, що за допомогою нього можна формувати персоналізовані рекомендації, які можна гнучко адаптувати з часом відповідно до динаміки інтересів користувачів. Зокрема, колаборативна фільтрація особливо добре працює з великими наборами даних, оскільки фокусується на реальних діях користувачів, а не на формальних характеристиках контенту.

Для того щоб визначити, наскільки користувачі або елементи схожі один на одного, система часто використовує косинусну подібність: чим менший кут між векторами інтересів, тим більше схожі ці вектори, а отже, і самі користувачі або об'єкти. Щоб підвищити точність розрахунків, використовується нормалізація оцінок, щоб зменшити вплив індивідуальних тенденцій до завищення або заниження оцінок.

Косинусна подібність обчислюється за формулою 1.1:

$$sim(i,j) = \frac{\sum_{u \in U_{ij}} r_{ui} \cdot r_{uj}}{\sqrt{\sum_{u \in U_{ij}} r_{ui}^2} \cdot \sqrt{\sum_{u \in U_{ij}} r_{uj}^2}} \quad . \tag{1.1}$$

Проте, колаборативна фільтрація також має свої недоліки. Одним з найбільших викликів є складність масштабування зі збільшенням кількості користувачів та контенту, а також тенденція до обмеженого вибору рекомендацій. Це пов'язано з тим, що система значною мірою покладається на історичні дані і не обов'язково враховує новий або унікальний контент [12].

У таких випадках хорошим доповненням є фільтрація на основі контенту, що базується безпосередньо на характеристиках самих об'єктів: жанрі, акторах, категорії тощо. Це дозволяє надавати рекомендації навіть тоді, коли про вподобання інших користувачів відомо небагато, підвищуючи персоналізацію та різноманітність результатів.

Гібридні рекомендаційні системи поєднують різні підходи до формування рекомендацій, щоб надати користувачеві більш точні, актуальні та різноманітні результати. Найбільш поширеним варіантом є поєднання фільтрації на основі контенту та колаборативної фільтрації. Такий підхід спирається на сильні сторони кожного з методів і компенсує їхні недоліки.

До прикладу, якщо система не має достатньої кількості даних про нових користувачів або новий контент (раніше описана проблема холодного старту), інформація про характеристики елементів – жанр, акторів, режисера, категорію тощо може бути використана, щоб допомогти сформулювати хоча б базовий список рекомендацій. І навпаки, коли система збирає достатньо історичних даних, вступає в дію колаборативна фільтрація, яка враховує поведінку подібних користувачів і підвищує рівень персоналізації.

Основною перевагою гібридних систем є гнучкість, тобто вони не обмежуються одним підходом, а можуть адаптуватися до різних ситуацій. Як наслідок, ці системи демонструють більшу точність, ширше охоплення інтересів користувачів і кращу здатність знаходити новий і нетривіальний контент. Це особливо актуально для кіноіндустрії, де кількість фільмів, серіалів, анімацій, тощо постійно зростає, а смаки аудиторії можуть бути дуже специфічними і динамічними.

Залежно від викликів і ресурсів, гібридизація може бути реалізована по-різному. Наприклад, певні алгоритми можна застосовувати послідовно – спочатку використати контентну фільтрацію, а потім доопрацьовувати її за допомогою колаборативної фільтрації. Інший варіант – об'єднати результати двох методів і створити остаточний список рекомендацій на основі вагових коефіцієнтів. Глибша інтеграція також можлива, якщо методи об'єднати в єдину модель або нейронну

мережу.

У висновку, гібридні системи краще справляються з кастомізацією, адаптуються до нових умов і демонструють стійкість до проблем, характерних для вузькоспеціалізованих підходів [13].

Системи рекомендацій на основі знань є важливим доповненням до класичних підходів до колаборативної та контентної фільтрації. Вони особливо корисні, коли «традиційні» методи стикаються з раніше згаданою проблемою «холодного старту», тобто коли майже немає даних про користувача або контент. На відміну від колаборативної фільтрації та фільтрації на основі контенту, системи, засновані на знаннях, не потребують великої історії взаємодій. Натомість вони зосереджуються на безпосередньому зборі вподобань користувачів, найчастіше через діалог чи опитування.

Ці системи працюють за принципом, що користувач вказує свої вимоги, вподобання або критерії, такі як бажаний жанр, тривалість фільму, мова тощо, а система підбирає відповідні варіанти на основі попередньо визначених правил або алгоритмів підбору. Такого типу механізм дозволяє надавати персоналізовані рекомендації навіть у випадках, коли інші методи безсилі через відсутність або нестачу даних.

Таким чином, для цього підходу можна виділити два основні типи систем:

- системи засновані на обмеженнях, в яких кожен вибір користувача порівнюється з певними правилами (наприклад, дитячі фільми не повинні містити сцен насильства) й рекомендації формуються відповідно до цього;
- системи, засновані на прикладах, в яких метрики подібності використовуються для пошуку елементів, схожих на вже відомі вподобання користувача.

Спільним для обох варіантів є те, що вони активно взаємодіють з користувачем під час надання рекомендацій. Цей діалог дозволяє краще зрозуміти індивідуальні вподобання і, перш за все, адаптуватися до смаків, що змінюються з часом. Це особливо корисно в контексті кінематографу, де люди можуть віддавати перевагу абсолютно різному контенту залежно від настрою, обставин і навіть пори

року.

Ще однією перевагою рекомендаційних систем на основі знань є можливість пояснити користувачеві, чому було рекомендовано той чи інший варіант, що підвищує довіру до рекомендацій. Завдяки своїй гнучкості ці системи широко використовуються у складних та ризикованих секторах, таких як нерухомість, інвестиції, цифрові технології, тощо і можуть бути ефективно змінені для використання у кіносервісах, де користувачі мають нестандартні та швидкозмінні запити [14].

Однак важливо пам'ятати, що розробка таких систем часто є ресурсомісткою: для формалізації знань і визначення правил необхідно залучати експертів у цій галузі (до прикладу, кінокритиків чи аналітиків). Тим не менше, завдяки своєму інтелектуальному діалоговому підходу, гнучкості та високій якості рекомендацій, системи, засновані на знаннях, є потужним інструментом для розробки сучасних рекомендаційних модулів для рекомендацій фільмів чи серіалів.

Підходи на основі машинного навчання відіграють ключову роль у сучасних рекомендаційних системах, оскільки вони не лише аналізують великі обсяги даних, але й виявляють приховані зв'язки та закономірності у вподобаннях користувачів. До таких методів належать класифікація, регресія, дерева рішень, нейронні мережі, кластеризація тощо. Одним із найпоширеніших і найефективніших методів є кластеризація, тобто процес, у якому користувачі або об'єкти (наприклад, фільми чи серіали) групуються в кластери на основі схожих характеристик або поведінкових особливостей.

У кіноіндустрії кластеризація може значно покращити якість рекомендацій. Наприклад, система може згрупувати користувачів, які віддають перевагу драмам, фентезі або трилерам і якщо новий користувач виявляє інтерес до певного жанру, алгоритм автоматично «призначає» його до відповідної групи і пропонує фільми, які вже сподобалися іншим членам цього кластера. Так само система працюватиме і для об’єктів – фільми також можна згрупувати за жанром, режисером або рейтингом, таким чином рекомендації можна давати, навіть якщо користувач ще не залишив багато оцінок.

Одним з найбільш відомих алгоритмів кластеризації є алгоритм k-середніх. Його ідея полягає в тому, щоб розбити всі об'єкти на k кластерів, кожен з яких матиме власний центр, тобто середнє значення об'єктів у кластері. Алгоритм поступово «пересуває» ці центри так, щоб кожен об'єкт належав до найближчого до нього центру. Відстань між об'єктом x і центром кластера μ зазвичай обчислюється за допомогою евклідової метрики:

$$d(x, \mu) = \sqrt{\sum_{i=1}^{n}(x_i - \mu_i)^2}\,. \qquad (1.2)$$

Кластеризація дозволяє системам надавати точні рекомендації навіть тоді, коли детальна інформація про вподобання користувача недоступна. Це дозволяє ефективно персоналізувати дані, особливо за наявності великих масивів даних і різних уподобань аудиторії [15].

Проаналізувавши частину методів для побудови рекомендаційних систем, можемо зробити висновок, що для розробки програмного модуля варто обрати фільтрацію на основі контенту тому, що вона дозволяє створювати персоналізовані рекомендації, враховуючи виключно характеристики самих фільмів та вподобання користувача. Така методика не потребує великої кількості даних про інших користувачів, що особливо корисно на початковому етапі розробки системи чи при відсутності взаємодій. Завдяки детальним характеристикам, як жанр, опис, рейтинг чи тривалість, система зможе точно знаходити подібні фільми, формуючи релевантні та зрозумілі рекомендації навіть для нових або унікальних фільмів.

1.3 Аналіз відомих інформаційних рекомендаційних систем

У сучасному цифровому середовищі існує багато рекомендаційних систем, які допомагають користувачам швидко знаходити потрібний контент. У музичній сфері, зокрема, Spotify, YouTube Music та Apple Music мають такі системи, рекомендації яких ґрунтуються на вподобаннях, прослуханих піснях та поведінці схожих користувачів. Для соціальних мереж алгоритми рекомендацій активно використовуються для відображення контенту, друзів або груп у Facebook, Instagram і TikTok, при цьому вирішальну роль відіграють взаємодія користувачів,

їхні вподобання та часові патерни. У кіноіндустрії Netflix та Amazon Prime Video – дві найвідоміші платформи з потужними рекомендаційними системами. Розглянемо докладніше, як працюють системи рекомендацій на цих двох платформах.

Система рекомендацій Netflix – це надзвичайно складна багаторівнева архітектура, що базується на поєднанні багатьох підходів від класичної фільтрації до машинного навчання. Основна мета – надати кожному користувачу максимально персоналізований контент, збільшуючи таким чином час перегляду та загального задоволення відвідувачів від користування сервісом. Для цього система використовує величезну кількість даних, які постійно оновлюються і обробляються в режимі реального часу.

На першому рівні аналізується історія взаємодії користувача з платформою: які фільми чи телешоу були переглянуті, які були скасовані або додані до списку «до перегляду», які отримали позитивні чи негативні оцінки. Це доповнюється непрямими підказками: коли користувач дивиться контент (наприклад, у вихідні або ввечері), на якому пристрої (смартфон, телевізор, планшет), в якому місці (геолокація), скільки часу витрачає на пошук нового фільму до перегляду тощо.

При цьому система враховує інформацію про сам контент: жанр, акторський склад, мову, країну виробництва, зміст, режисерів, рік випуску. На основі цих характеристик створюється векторний простір ознак для користувачів і фільмів. Це дозволяє робити фільтрацію на основі контенту, тобто система порівнює вподобання користувачів з характеристиками нових фільмів, щоб передбачити ймовірність зацікавленості.

Паралельно працює колаборативна фільтрація, якщо інші користувачі зі схожими вподобаннями дивилися певні фільми, вони, ймовірно, зацікавлять поточного користувача. Цей підхід особливо ефективний, коли користувач ще не бачив багато контенту і його інтереси ще формуються.

Ключовою частиною системи є використання машинного навчання – складні моделі (в тому числі нейронні мережі) прогнозують ймовірність того, що користувач натисне на фільм або почне його дивитися, а також визначають, які

елементи відображати на головній сторінці. Також Netflix активно використовує A/B-тестування – різним групам користувачів пропонуються різні варіанти рекомендацій, дизайну і навіть постерів фільмів, а система фіксує, які варіанти оцінюються найкраще.

Важливою особливістю рекомендаційної системи Netflix є те, що вона динамічна, тобто щоразу, коли користувач взаємодіє з сайтом, його профіль оновлюється, а набір рекомендацій змінюється. Це дозволяє не тільки точно передбачити, що саме зацікавить користувача зараз, але й адаптуватися до змін у його смаках з часом. Таким чином, Netflix є не просто платформою для перегляду фільмів, а інтелектуальним сервісом, який «розуміє» своїх глядачів [16].

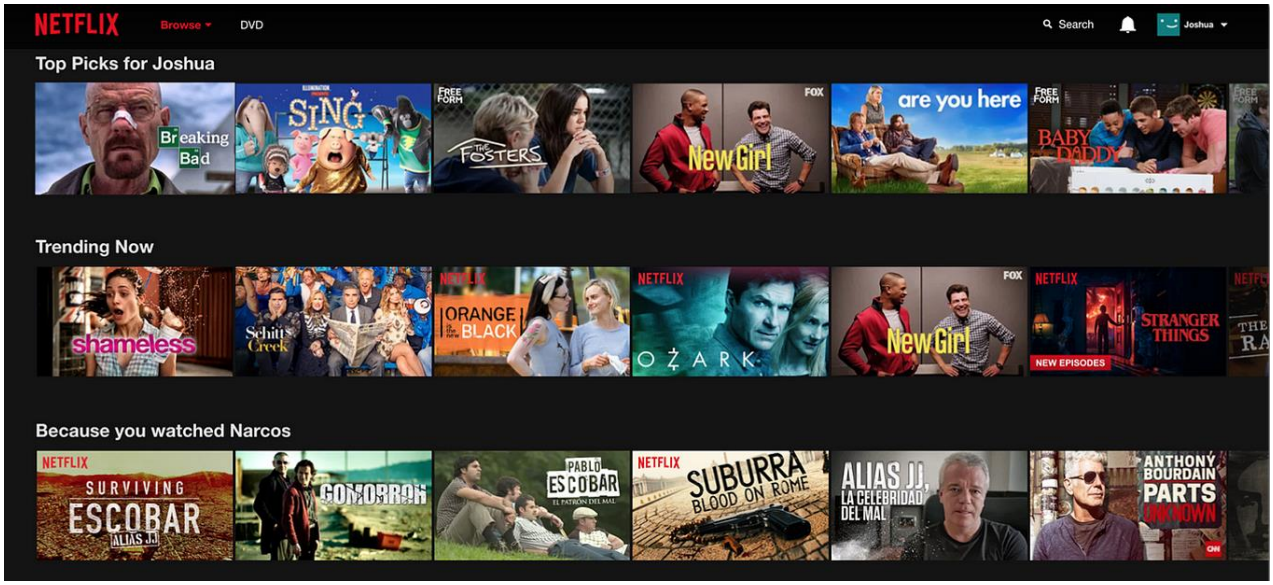

Рисунок 1.3 – Приклад вигляду персоналізованих рекомендацій на стрімінговій платформі Netflix

Аналізуючи систему рекомендацій Amazon Prime Video, можемо сказати, що це складна та адаптивна технологія, яка тісно інтегрована в екосистему Amazon і спирається на поєднання різних технологій, таких як фільтрація контенту, колаборативна фільтрація, моделі машинного навчання та знання користувачів з усього середовища Amazon. Основною метою системи є запропонувати кожному користувачеві найбільш актуальний для нього відеоконтент, беручи до уваги його попередній досвід, інтереси та поведінкові моделі.

Як і Netflix, Amazon Prime Video збирає різні типи даних про користувачів, зокрема історію переглядів, рейтинги фільмів і серіалів, збереження, час перегляду та час витрачений на пошук контенту, повторні перегляди та взаємодію з іншими сервісами Amazon. Особливо унікальним є те, що Amazon враховує не лише активність стосовно відео контенту, але й дані з основного ринку, до прикладу, що користувач купує, які книги читає на своєму Kindle і чи використовує Alexa. Це створює дуже детальний профіль користувача, який набагато краще прогнозує його інтереси.

Дуже важливу роль в системі рекомендацій Amazon Prime Video відіграє фільтрація за контентом. Система аналізує фільми за жанром, тривалістю, акторами, режисером, ключовими словами та іншими атрибутами, щоб знайти фільми, схожі на контент, який користувач вже позитивно оцінив. Водночас колаборативна фільтрація дає змогу виявляти спільноти користувачів зі схожими інтересами.

Також Amazon активно використовує машинне навчання для моделювання поведінки користувачів. Використовуючи методи класифікації, кластеризації та регресії, система може передбачити ймовірність того, що певний фрагмент контенту буде переглянутий або зупинений. Система також враховує контекст, наприклад, як час доби, день тижня або сезон, коли користувач зазвичай дивиться відео, щоб надати ще більш точні рекомендації. Однією з функцій є персональний порядок, тобто система визначає порядок, в якому фільми з'являються на першій сторінці, тому перші варіанти, швидше за все, будуть найцікавішими.

Крім того, Amazon Prime Video об’єднує в рекомендаціях рейтинги, відгуки, оцінки IMDb і внутрішні показники якості, щоб краще зрозуміти, наскільки цінним є контент для спільноти. Важливо, що система також реагує на поведінку в режимі реального часу, тобто якщо користувач раптом починає дивитися більше трилерів або документальних фільмів, рекомендації швидко зміняться відповідно до нових інтересів [17].

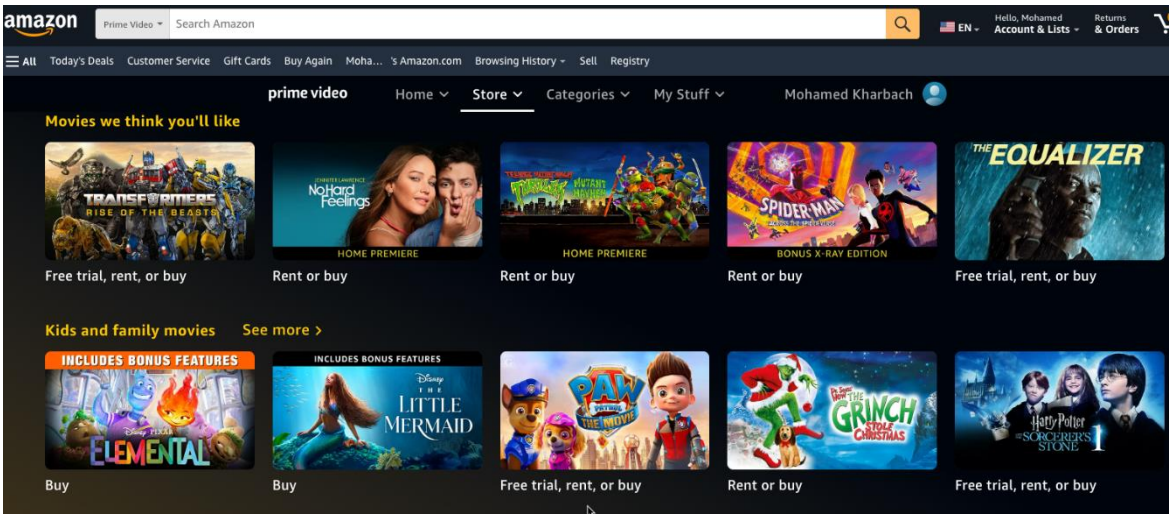

Рисунок 1.4 – Приклад вигляду персоналізованих рекомендацій на стрімінговій платформі Amazon Prime Video

Загалом система рекомендацій Amazon Prime Video є унікальною, оскільки вона є частиною набагато більшої екосистеми Amazon, яка дозволяє дуже точно передбачати вподобання користувачів. Система є динамічною, диференційованою та гнучкою, що дозволяє платформі обслуговувати глядачів різного віку, національності та стилю за допомогою широкого та постійно зростаючого вибору відеовмісту.

1.4 Висновок до розділу 1

Під час дослідження було встановлено, що створення програмного модуля рекомендаційної системи є надзвичайно актуальним у сфері кінематографу. Це пов'язано з швидким зростанням доступного відеоконтенту, що ускладнює користувачам самостійний вибір фільмів, які відповідають їх смакам. Персоналізовані рекомендації не тільки підвищують доступність платформи, але і підвищують рівень задоволеності аудиторії, зберігають її увагу і забезпечують триваліший перегляд. Саме тому впровадження розумної системи маршрутизації є важливим кроком для сучасних онлайн-кінотеатрів.

У цьому розділі розглядаються основні методи створення рекомендаційних систем, а саме фільтрація на основі контенту, колаборативна фільтрація, гібридні

підходи, системи на основі знань та алгоритми машинного навчання. У кожного методу є свої переваги і обмеження. Наприклад, колаборативна фільтрація добре працює, коли вона має велику історичну базу даних, але менш ефективна з новими користувачами або фільмами. Натомість, методи машинного навчання можуть запропонувати гнучкість і точність, але вони вимагають високого рівня якісних даних. Враховуючи структуру поточного набору даних, який буде використовуватись при розробці, було обрано фільтрацію на основі контенту, оскільки наявні описові властивості фільму (типи, опис, рік випуску, мова, рейтинги тощо), які можуть бути використані для аналізу та пошуку подібності між ними.

Також були розглянуті приклади реальних систем рекомендацій – зокрема, як вони реалізуються на популярних платформах, таких як Netflix і Amazon Prime Video. Досвід цих платформ показує, наскільки важливо постійно вдосконалювати підходи до персоналізації, поєднувати різні методи, враховувати зміни в налаштуваннях користувачів і підтримувати релевантні рекомендації. Ці приклади допомогли краще зрозуміти, як працюють складні системи рекомендацій в реальному середовищі і які ідеї можуть бути адаптовані для власної розробки. У висновку, отримані знання є надійною основою для створення ефективного програмного модуля, яка відповідає сучасним потребам і очікуванням користувачів.

2 РОЗРОБКА ТА ПРОЄКТУВАННЯ ПРОГРАМОГО МОДУЛЯ ПІДБОРУ КІНЕМАТОГРАФІЧНОГО ТВОРУ

2.1 Розробка архітектури та взаємодії програмних модулів

Розробка архітектури програмних модулів є одним з ключових етапів при створенні рекомендаційної системи кінематографічних творів. Архітектура визначає загальну логіку взаємодії компонентів, розподіляє обов'язки між ними та забезпечує структурованість і масштабованість системи. В умовах сучасного програмного забезпечення, особливо у сфері персоналізованих рекомендацій, правильне проєктування архітектури дозволяє не лише підвищити ефективність роботи системи, а й спростити процес її розширення, обслуговування та адаптації до нових вимог.

Поділ програми на окремі модулі дозволяє уникнути монолітної структури, яка ускладнює підтримку та масштабування. Завдяки модульному підходу кожен компонент має чітко визначену зону відповідальності: один модуль займається збором вподобань користувача, інший – обробкою набору даних, ще інший – формуванням рекомендацій тощо. Такий підхід дозволяє швидко виявляти та виправляти помилки, замінювати або допрацьовувати окремі частини системи без впливу на інші компоненти [18].

Також варто враховувати, що рекомендаційна система буде працювати з реальним набором даних, тому побудова чіткої архітектури забезпечує коректну та ефективну обробку великого обсягу інформації. Крім цього, правильно побудована архітектура забезпечує простоту тестування кожного окремого модуля, що особливо важливо для перевірки якості алгоритму рекомендацій.

Отже, архітектура системи включає такі модулі: модуль графічного інтерфейсу (MovieAppGUI), модуль збору вподобань користувача (UserPreferences), модуль роботи з датасетом (MovieDataset) та модуль рекомендацій (Recommender). Кожен з цих модулів виконує окрему функцію, але водночас взаємодіє з іншими. Такий підхід дозволяє досягти високої гнучкості,

полегшує налагодження і забезпечує зрозумілість структури всієї програми для стороннього читача або розробника.

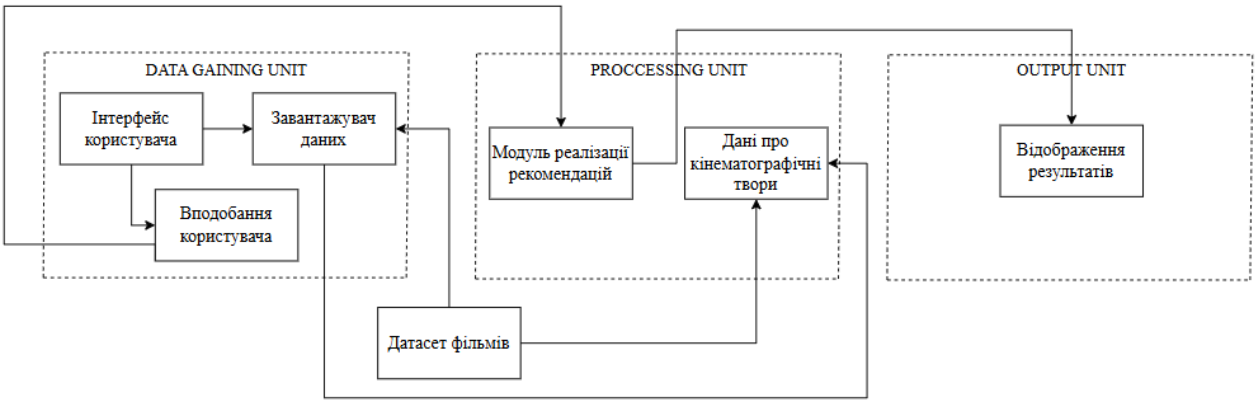

Рисунок 2.1 – Схема архітектури програмного модуля підбору кінематографічного твору

Розробка моделі взаємодії програмних модулів є наступним важливим кроком після побудови архітектури. Важливо не лише знати, які модулі існують, а й розуміти, як саме вони обмінюються інформацією, в якому порядку відбувається взаємодія, і які дані передаються від одного модуля до іншого. Це дозволяє сформувати цілісне уявлення про роботу системи та забезпечити її стабільне функціонування [19].

Початковим модулем у системі є MovieAppGUI. Саме з нього починається взаємодія користувача із програмою. Графічний інтерфейс ініціює послідовність запитань до користувача, після чого отримані відповіді передаються у модуль UserPreferences. UserPreferences отримує введені користувачем дані, перевіряє їх на валідність та перетворює у структуру, з якою може працювати система.

Далі підключається модуль MovieDataset. Він завантажує та обробляє набір фільмів, очищає та структурує дані, щоб вони відповідали критеріям пошуку. Цей модуль не взаємодіє з користувачем безпосередньо, але є базовим джерелом інформації для Recommender модуля.

Модуль Recommender приймає дані вподобань користувача з UserPreferences і поєднує їх з обробленим датасетом з MovieDataset. На основі встановлених

критеріїв система виконує фільтрацію, сортує фільми за релевантністю або рейтингом, після чого формує кінцеву підбірку рекомендованих фільмів. Ця інформація знову повертається у MovieAppGUI для відображення результату користувачеві.

Таким чином, кожен модуль виконує свою чітко визначену роль і взаємодіє з іншими у строго визначеній послідовності. Такий підхід дозволяє створити стабільну, зрозумілу та ефективну систему.

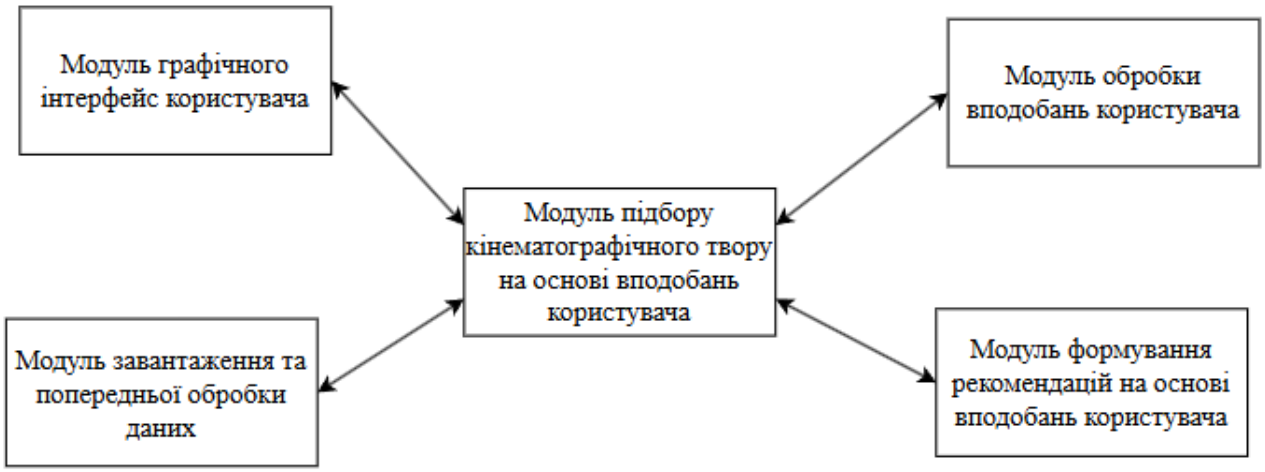

Рисунок 2.2 – Структурна схема програмного модуля підбору кінематографічного твору на основі вподобань користувача

Коротко охарактеризуємо кожен з основних модулів системи. MovieAppGUI – це модуль, який відповідає за інтерфейс користувача. Він забезпечує проведення опитування та виведення результатів.

UserPreferences збирає, зберігає та обробляє вподобання користувача, готуючи їх до передачі іншим модулям.

MovieDataset відповідає за завантаження та обробку набору фільмів, формуючи готовий до фільтрації список.

Recommender – головний аналітичний модуль, що виконує логіку відбору фільмів відповідно до вподобань користувача.

У підсумку можна зробити висновок, що після аналізу принципів побудови архітектури та модульної взаємодії було визначено оптимальну структуру системи.

Такий підхід дозволяє створити ефективну, масштабовану та придатну для подальшого розвитку рекомендаційну систему кінематографічних творів на основі вподобань користувача.

2.2 Розробка UML-діаграми класів системи рекомендацій

Розробка UML-діаграми класів є одним із ключових етапів у створенні будь-якої програмної системи, зокрема і програмного модуля рекомендаційної системи кінематографічних творів на основі вподобань користувача. Така діаграма дозволяє сформувати чітке уявлення про структуру майбутньої програми ще до написання коду, адже наочно демонструє, які класи буде використано, які атрибути та методи вони міститимуть, як вони взаємодіють між собою та які залежності між ними існують. Це не лише спрощує розробку, але й забезпечує зручність підтримки, модифікації й масштабування проєкту в майбутньому.

UML-діаграма класів також сприяє стандартизації проєктування. У великих або середніх проєктах, де над однією системою працює команда розробників, наявність діаграми класів дозволяє кожному з учасників розуміти загальну структуру та логіку взаємодії між компонентами системи. Це знижує ризик помилок, дублювання функціональностей та суперечностей між модулями. Крім того, така діаграма може використовуватись для спілкування з іншими учасниками розробки — аналітиками, дизайнерами, тестувальниками чи навіть замовниками — адже вона є універсальним візуальним засобом передачі інформації [20].

У випадку розробки рекомендаційної системи фільмів, UML-діаграма класів допоможе структурувати логіку системи, що складається з кількох функціональних блоків. Вона дозволяє розподілити обов’язки між класами таким чином, щоб код залишався чистим, зрозумілим та легко масштабованим. Наприклад, один клас відповідальний лише за інтерфейс користувача, інший – за зберігання та обробку даних, ще один – за логіку рекомендацій, а четвертий – за зберігання вподобань користувача. Такий підхід дозволяє ізолювати функціональність кожного модуля і забезпечити правильну взаємодію між ними.

У межах нашого програмного модуля варто представити як мінімум чотири основні класи. Перший – MovieAppGUI, який відповідатиме за взаємодію з користувачем, збір інформації та відображення результатів. Він містить методи для запуску вікна додатку, ініціалізації опитування, збору вибраних параметрів, а також відображення списку рекомендованих фільмів. Цей клас є центральною точкою взаємодії користувача з системою.

Другий клас – UserPreferences – буде зберігати усі вибрані користувачем параметри: жанри, мову, країну, тривалість тощо. Він працює як контейнер даних, який передається до системи рекомендацій для формування персоналізованого списку фільмів. Клас має поля для всіх необхідних характеристик і методи для встановлення та зчитування цих значень.

Третій клас MovieDataset – клас, який відповідає за завантаження, зберігання та попередню обробку датасету з фільмами. Він реалізує фільтрацію даних за ключовими параметрами, підготовку датасету до пошуку рекомендацій, а також завантаження необхідної інформації (включно з постерами, назвами, рейтингами тощо). Це – модуль, що працює безпосередньо з даними, які є основою для рекомендацій.

Останній клас – Recommender – відповідає за безпосереднє формування рекомендацій. Він приймає як вхідні дані з UserPreferences, обробляє їх, виконує фільтрацію та формує остаточний список фільмів, які найбільше відповідають вподобанням користувача. Це «мозок» системи, де закладено логіку аналізу даних та формування списку результатів.

Завдяки чітко побудованій UML-діаграмі класів стає можливим легко і послідовно реалізувати систему згідно з поставленими вимогами. Вона дозволяє ще на етапі проєктування виявити недоліки, оптимізувати структуру, уникнути дублювання функцій та правильно розподілити логіку між класами.

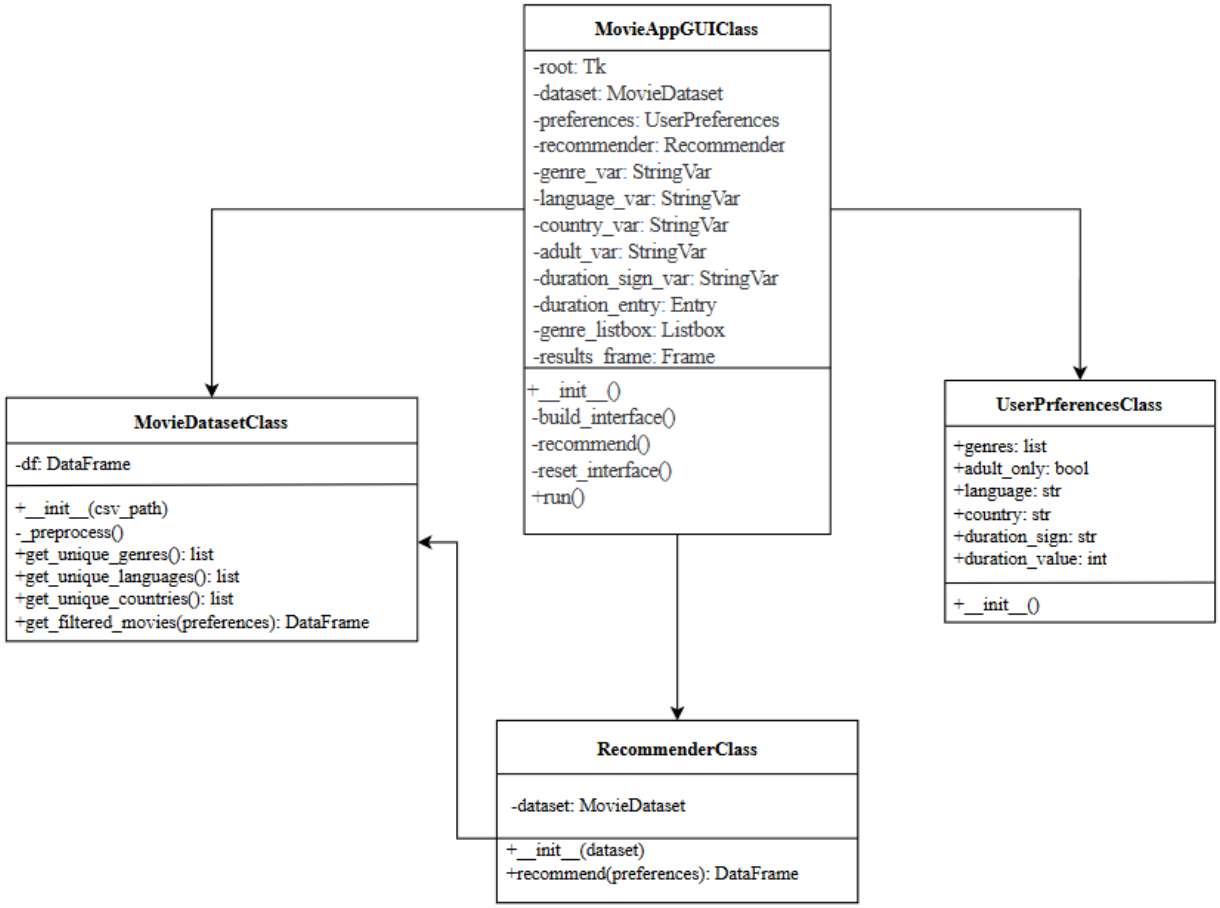

Рисунок 2.3 – UML-діаграма класів програмного модулю підбору кінематографічного твору на онові вподобань користувача

Діаграма класів системи рекомендації фільмів складається з чотирьох основних компонентів, кожен з яких виконує специфічні функції та має чітко визначені обов'язки в архітектурі додатку.

Клас MovieDataset є фундаментальним компонентом системи, відповідальним за роботу з даними про фільми. Цей клас інкапсулює всю логіку завантаження, обробки та фільтрації інформації про кінематографічні твори. При ініціалізації клас приймає шлях до CSV-файлу та завантажує дані у внутрішню структуру pandas DataFrame. Приватний метод препроцесингу забезпечує очищення даних від записів з відсутніми критично важливими полями та перетворює текстові списки жанрів і країн виробництва у структуровані колекції. Клас надає публічні методи для отримання унікальних значень жанрів, мов та

країн, що використовуються для побудови елементів інтерфейсу. Найважливішим методом є фільтрація фільмів за користувацькими вподобаннями, який повертає DataFrame з п'ятьма найкращими фільмами, відсортованими за рейтингом.

Клас UserPreferences представляє модель даних для зберігання критеріїв пошуку користувача. Цей простий клас-контейнер містить атрибути для всіх можливих параметрів фільтрації: список обраних жанрів, булеве значення для фільмів з віковим обмеженням, мову оригіналу, країну виробництва, а також параметри тривалості фільму у вигляді знаку порівняння та числового значення. Простота структури цього класу забезпечує легкість його використання та модифікації, дозволяючи легко додавати нові критерії фільтрації без значних змін в архітектурі системи.

Клас Recommender реалізує основну логіку системи рекомендацій. Він служить посередником між даними про фільми та вподобаннями користувача, забезпечуючи абстракцію рекомендаційного процесу. Клас містить посилання на екземпляр MovieDataset та надає єдиний публічний метод recommend, який приймає об'єкт користувацьких вподобань та повертає відфільтровані рекомендації. Така архітектура дозволяє легко розширювати логіку рекомендацій, наприклад, додаванням алгоритмів машинного навчання або більш складних методів фільтрації, не впливаючи на інші компоненти системи.

Клас MovieAppGUI є найбільш складним компонентом системи, що відповідає за графічний інтерфейс користувача та координацію роботи всіх інших класів. Цей клас містить численні атрибути для зберігання посилань на елементи інтерфейсу Tkinter, включаючи змінні для зберігання значень комбобоксів, поля введення та списки вибору. Клас також утримує екземпляри всіх інших класів системи: MovieDataset для роботи з даними, UserPreferences для зберігання поточних налаштувань користувача та Recommender для отримання рекомендацій. Метод побудови інтерфейсу створює всі необхідні віджети та розміщує їх у вікні програми, використовуючи дані з MovieDataset для заповнення списків вибору. Метод рекомендації збирає дані з інтерфейсу, заповнює об'єкт вподобань, отримує результати від рекомендаційного модуля та відображає їх користувачу.

Діаграма класів демонструє різні типи відносин між компонентами системи. MovieAppGUI має композиційні зв'язки з усіма іншими класами, створюючи та утримуючи їх екземпляри протягом життєвого циклу додатку. Клас Recommender має агрегаційний зв'язок з MovieDataset, використовуючи його для доступу до даних про фільми.

Створення діаграми класів значно допомогло у розробці системи рекомендації фільмів, надавши візуальне представлення архітектури та взаємозв'язків між компонентами. Діаграма дозволила чітко визначити обов'язки кожного класу та уникнути дублювання функціональності. Вона служила своєрідним планом розробки, допомагаючи структурувати код згідно з принципами об'єктно-орієнтованого програмування. Візуалізація залежностей між класами полегшила розуміння потоку даних в системі та допомогла передбачити потенційні проблеми інтеграції компонентів.

Діаграма також виявилася корисною для документування системи, надаючи майбутнім розробникам швидкий огляд архітектури без необхідності детального вивчення коду. Вона допомогла ідентифікувати можливості для рефакторингу та оптимізації, показуючи, які класи мають занадто багато обов'язків або складні взаємозв'язки.

Аналіз діаграми класів дозволяє зробити кілька важливих висновків щодо архітектури системи. По-перше, система демонструє гарне розділення обов'язків, де кожен клас має чітко визначену роль: MovieDataset відповідає за дані, UserPreferences за модель вподобань, Recommender за логіку, а MovieAppGUI за презентацію. Це відповідає принципу єдиної відповідальності та робить систему більш модульною та підтримуваною.

По-друге, архітектура демонструє низький рівень зв'язаності між класами даних та логіки, що є позитивним аспектом дизайну. MovieDataset та Recommender можуть функціонувати незалежно від графічного інтерфейсу, що дозволяє легко тестувати їх окремо або використовувати в інших контекстах.

Загалом, діаграма класів показує добре структуровану систему з чіткою архітектурою, що забезпечує гнучкість для майбутніх розширень та модифікацій

при збереженні простоти розуміння та підтримки коду.

2.3 Розробка алгоритму програмного модуля підбору кінематографічних творів

Одним з найважливіших етапів розробки програмного модуля для рекомендаційної системи в кіноіндустрії є проектування відповідної архітектури та визначення ефективного алгоритму його роботи. Це надзвичайно важливо, адже від якості архітектурних рішень і логіки взаємодії модулів залежить не лише точність генерації рекомендацій, а й стабільність, масштабованість і зручність використання самої системи. Добре організована архітектура дозволяє чітко розподілити обов'язки між окремими компонентами і гарантує, що функціональність може бути легко модифікована і розширена в майбутньому. Водночас, заздалегідь визначений алгоритм роботи забезпечує послідовність процесів, оптимальне використання ресурсів і високий рівень адаптивності системи до зміни вхідних параметрів, визначених користувачем. Такий підхід дозволяє створити інтегровану, логічно завершену систему, яка за своїми функціональними можливостями відповідає сучасним вимогам до розробки інтелектуальних рекомендаційних рішень [21].

Система рекомендацій буде базуватися на принципі взаємодії з користувачем, за допомогою короткого, але інформативного опитування, в якій користувач крок за кроком вводить свої вподобання: бажаний жанр, вікові обмеження, мову оригіналу, країну виробництва та бажану тривалість перегляду. Кожне з цих рішень буде враховуватися при фільтрації записів у датасеті фільмів. Отримані дані передаватимуться до блоку обробки, де система автоматично аналізує обрані параметри, порівнює їх з даними про фільми, наявними в датасеті, і формує добірку з п'яти фільмів, які найбільше відповідають смакам користувача. Ознайомившись з рекомендаціями, користувач може вибрати, чи задоволений він результатом і у випадку позитивної відповіді програма завершує свою роботу. Якщо користувач вважає, що вибір не є вдалим, система дає йому можливість або

завершити сеанс, або повернутися до початкового етапу, де він може запустити запит ще раз і згенерувати нові рекомендації. На основі цієї циклічної логіки буде побудовано загальний алгоритм роботи модуля системи рекомендацій кінематографічних творів. Такий підхід дозволяє врахувати мінливість вподобань користувача, забезпечити гнучкість у формуванні результатів та створити зрозумілий, зручний та ефективний механізм взаємодії між людиною та програмою.

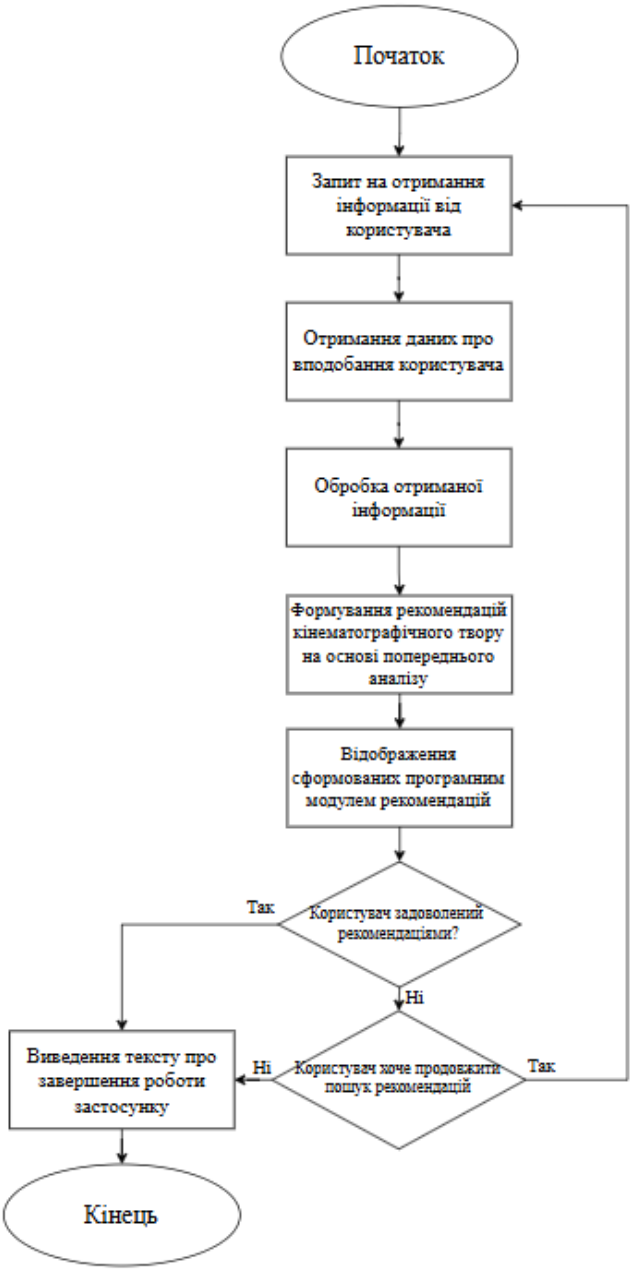

Рисунок 2.4 – Схема загального алгоритму роботи програмного модулю підбору кінематографічного твору на основі вподобань користувача

Розглянемо детальніше модулі, які потрібно реалізувати при розробці програмного модуля рекомендаційної системи фільмів.

Першим і основним модулем є MovieAppGUI, який відповідає за графічний інтерфейс користувача. Цей модуль забезпечує візуальну взаємодію з користувачем, створює головне вікно застосунку та організовує послідовність екранів, що з'являються під час опитування. Він ставить запитання про вподобання користувача, такі як улюблені жанри, мова оригіналу, країна виробництва, наявність вікових обмежень та бажана тривалість фільму. Отримані відповіді передаються для подальшої обробки, після завершення опитування MovieAppGUI запитує результати з модуля рекомендацій і представляє їх користувачеві у зручний і привабливий спосіб. Якщо користувач не задоволений результатом, система дозволяє йому вийти чи повторити опитування.

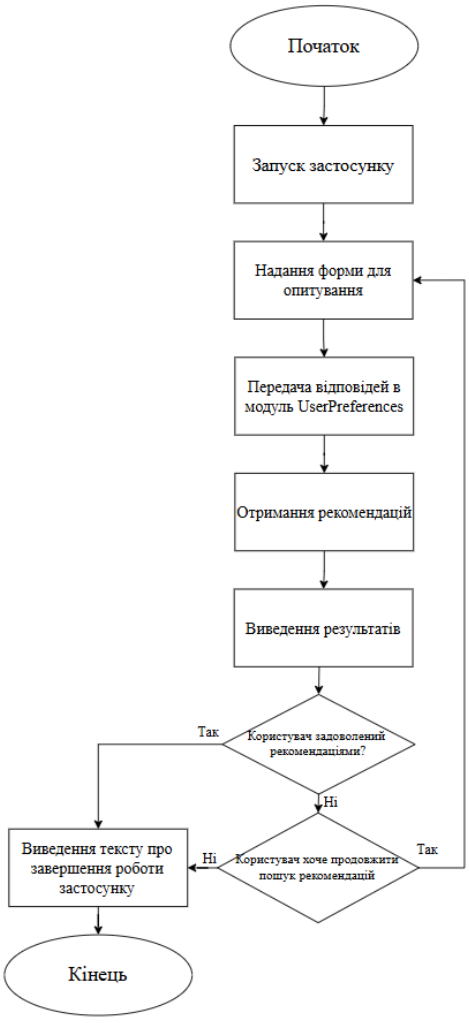

Рисунок 2.5 – Схема алгоритму роботи MovieAppGUI модуля

Модуль UserPreferences збирає, обробляє та зберігає інформацію про вподобання користувачів, отриману від графічного інтерфейсу користувача. Цей модуль відповідає за перетворення даних у формат, придатний для фільтрації фільмів, включаючи створення структури даних, яка чітко описує вподобання з точки зору жанру, мови, країни, часу перегляду і т.д. Цей модуль також виконує базову перевірку введених даних, наприклад, перевіряє, чи відповідає жанр жанрам в датасеті. Після того, як структура вподобань завершена, дані передаються до модуля рекомендацій для подальшої роботи.

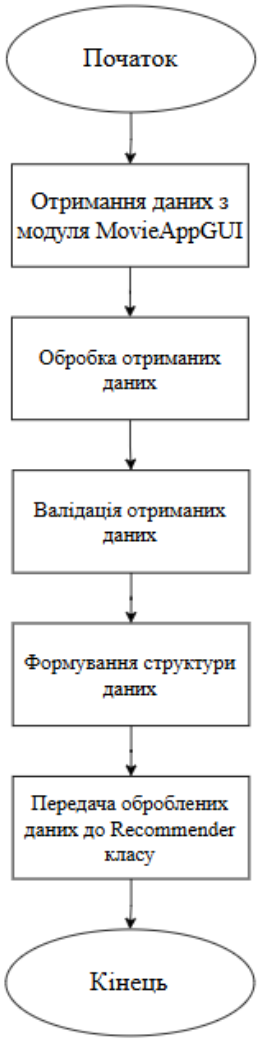

Рисунок 2.6 – Схема алгоритму роботи UserPreferences модуля

Іншим важливим компонентом є MovieDataset модуль, який відповідає за завантаження, очищення та підготовку набору даних. Цей модуль працює з файлом

датасетом, який містить дані про фільми з відкритої бази даних TMDB: назви, описи, жанри, мову, країну виробництва, рейтинг, постери тощо. Після завантаження даних MovieDataset очищає їх – видаляє порожні значення, нормалізує формати та адаптує дані для подальшого пошуку. Результатом є набір фільмів, який можна обробити в системі рекомендацій за допомогою наступного модуля.

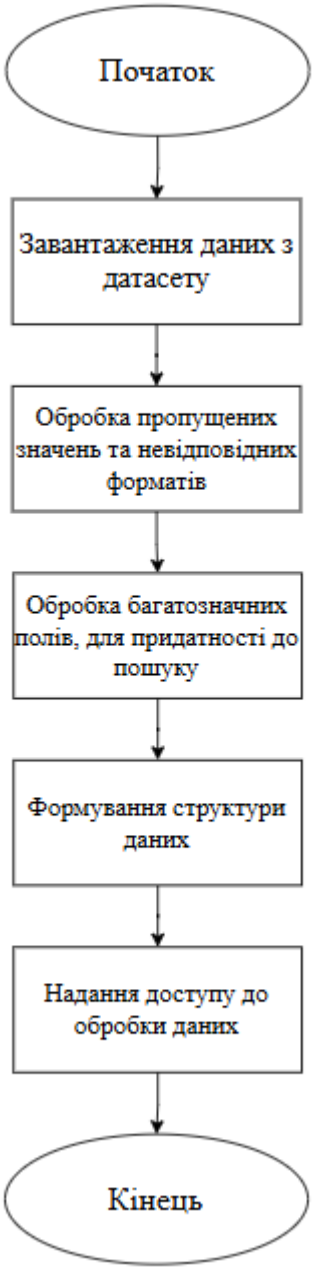

Рисунок 2.7 – Схема алгоритму роботи MovieDataset модуля

Останнім, проте найбільш важливим логічним блоком є модуль

рекомендацій Recommender, який реалізує основну логіку формування індивідуальних рекомендацій. Отримавши вподобання користувача від модуля UserPreferences, модуль рекомендацій послідовно фільтрує весь набір фільмів за певними ознаками: жанр, тривалість, мова, країна-виробник та рівень прийнятності (наприклад, вікові обмеження). Після цього отриманий список сортується, за рейтингом або популярністю. Результатом є добірка з п'яти найбільш підходящих фільмів, які повертаються в інтерфейс для перегляду користувачем.

Рисунок 2.8 – Схема алгоритму роботи Recommender модуля

Така структура застосунку забезпечить високий рівень модульності, що полегшить подальший розвиток системи, тестування та побудову чіткої діаграми

класів.

2.4 Висновок до розділу 2

У другому розділі було детально розглянуто процес розробки програмного модуля рекомендаційної системи кінематографічних творів. З метою забезпечення ефективного функціонування системи було спроєктовано архітектуру, що відображає взаємозв’язки між ключовими програмними модулями та дозволяє забезпечити модульність, масштабованість і легкість супроводу розробки.

Також була створена структурна схема програмного модуля, яка наочно демонструє логіку взаємодії між окремими компонентами системи в процесі її роботи. Окрему увагу приділено побудові UML-діаграми класів, яка відображає основні об’єкти, їх властивості та взаємозв’язки, що забезпечує зручність у подальшій реалізації та супроводі програмного забезпечення. Крім того, було описано загальні алгоритми роботи для кожного з модулів, що формують логічну основу функціонування всієї системи.

Ці результати закладають надійну основу для реалізації програмного модуля та забезпечують розуміння принципів його побудови як з технічної, так і з функціональної точки зору.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ТЕСТУВАННЯ МОДУЛЯ ПІДБОРУ КІНЕМАТОГРАФІЧНОГО ТВОРУ

3.1 Обґрунтування вибору інструментів розробки програмного модуля

Вибір мови програмування є одним з найбільш важливих кроків у розробці рекомендаційної системи, оскільки це рішення впливає не лише на швидкість та ефективність розробки, але й на масштабованість, продуктивність, підтримку спільноти, доступ до бібліотек та інструментів машинного навчання, простоту реалізації алгоритмів обробки даних, тощо. Саме тому важливо правильно вибрати мову програмування для розробки рекомендаційних систем.

Для розробки рекомендаційних систем можна використовувати низку мов програмування. Найпопулярнішими з них є Python, Java, C++, JavaScript, R та Scala. Кожна з цих мов має свої особливості, переваги та обмеження для розробки таких систем.

У цьому розділі ми розглянемо дві найпопулярніші мови програмування для розробки рекомендаційних систем – Python та Java та на основі аналізу їхніх сильних і слабких сторін виберемо найбільш підходящу мову для даного проєкту. Розглянемо переваги та недоліки мови програмування Python.

Переваги:

- Python має велику кількість бібліотек для роботи з машинним навчанням, обробкою даних та побудовою рекомендаційних систем, наприклад, NumPy, Pandas, Scikit-learn, TensorFlow, PyTorch та інші. Ці інструменти значно полегшують розробку складних алгоритмів, оптимізацію моделей та обробку великих даних.

- простий і зрозумілий синтаксис Python пришвидшує написання та розуміння коду, що полегшує розробку та підтримку великих проєктів. Це важливо для розробки та оптимізації алгоритмів;

- Python має активну та велику спільноту розробників, що забезпечує легкий доступ до великої кількості ресурсів, документації та підтримки;
- Python добре інтегрується з мовами C/C++, завдяки чому можна оптимізувати ключові частини вашого коду для підвищення продуктивності;
- Python підтримує нові алгоритми та дозволяє швидко створювати прототипи.

Недоліки:

- Python інтерпретується, тому її продуктивність може бути нижчою, ніж у компільованих мов, таких як Java;
- Python може використовувати більше оперативної пам'яті, що може бути критичним при роботі з дуже великими наборами даних;
- Python має обмеження через Global Interpreter Lock, що ускладнює реалізацію багатопоточності в певних випадках [22].

Розглянемо також переваги та недоліки використання Java в розробці рекомендаційних систем.

Переваги:

- Java пропонує високу продуктивність і масштабованість, оскільки вона компілюється в байт-код, який виконується в JVM;
- Java надає можливості багатопотоковості, що може бути корисним для паралельної обробки великих обсягів даних;
- оскільки Java широко використовується у великих корпоративних системах, вона добре підходить для розробки масштабованих і надійних систем;
- JVM дозволяє легко переносити додатки, написані на Java, на інші платформи.

Недоліки:

- код на Java, як правило, довший і складніший, ніж код на Python, що може уповільнити розробку;
- хоч і існують бібліотеки для роботи з машинним навчанням (наприклад, Weka, Deeplearning4j), їх набагато менше ніж у Python;

- Java менше використовується в дослідженнях і розробках ШІ, тому нові рішення часто з'являються спочатку на Python [23].

Для кращого представлення даних, на основі проаналізованої інформації побудуємо порівняльну таблицю 3.1 для розглянутих мов програмування:

Таблиця 3.1 – Порівняльний аналіз мов програмування Python та Java

Характеристика	Python	Java
Простота синтаксису	+	-
Швидкість виконання	-	+
Кількість бібліотек	+	-
Підтримка ML/AI	+	-
Активність спільноти	+	+
Підтримка багатопоточності	-	+
Швидкість розробки	+	-

На основі цього аналізу було обрано мову програмування Python. Основними причинами цього є її орієнтованість на дані, велика кількість бібліотек, доступних для реалізації рекомендаційних систем, і простота реалізації алгоритмів машинного навчання. Як наслідок, Python значно полегшує створення прототипів, експерименти з алгоритмами та налагодження системи.

Варто відзначити, що обраний метод підбору рекомендацій – фільтрація на основі контенту, вимагає роботи з атрибутами елементів та використання векторного представлення, що легко реалізується за допомогою бібліотек Python, таких як pandas та scikit-learn. Реалізація векторизації TF-IDF, косинусної подібності або кластеризації є стандартною практикою в Python, що робить її ідеальним вибором для реалізації цього методу.

В якості середовища розробки було обрано PyCharm – інтегроване середовище розробки, спеціально розроблене для Python. Розглянемо також переваги та недоліки обраного IDE.

Переваги PyCharm:

- зручна навігація по коду;
- вбудований відлагоджувач;
- інтеграція з системами контролю версій;
- можливість швидкого запуску скриптів і тестування функціоналу.

Недоліки PyCharm:

- високе споживання ресурсів (таких як ОЗП);
- повільний запуск на слабких комп'ютерах.

Отже, можна зробити висновок, що PyCharm є чудовим вибором для розробки рекомендаційної системи на Python, завдяки багатьом інструментам аналізу, налагодження та легкому управлінню кодом.

Обрана мова програмування Python та середовище розробки PyCharm дозволить ефективно спроектувати та реалізувати механізм рекомендацій. Python надає доступ до потужних інструментів для обчислень, машинного навчання та реалізації алгоритмів, а PyCharm забезпечить зручність, продуктивність та якісний процес розробки. Цей вибір є оптимальним для проектів, пов'язаних з побудовою інтелектуальних рекомендаційних систем.

3.2 Програмна реалізація модулів рекомендаційної системи

Розроблена система складається з чотирьох основних модулів, кожен з яких виконує окремі, але взаємопов’язані функції. До складу системи входять такі модулі: MovieAppGUI (інтерфейс користувача), MovieDataset (завантаження та обробка даних), UserPreferences (формування переваг користувача) та recommender (рекомендаційний модуль).

- Інтерфейс користувача (MovieAppGUI) — відповідає за взаємодію з користувачем, збір вхідних даних, виклик рекомендаційного механізму та відображення результатів. Побудований з використанням бібліотеки tkinter.

- Модуль завантаження та обробки даних (MovieDataset) — відповідає за завантаження датасету фільмів, обробку полів (жанри, країни, мови тощо), а також за надання списків унікальних значень для фільтрації.

- Модуль переваг користувача (UserPreferences) — зберігає обрані користувачем параметри для подальшого використання в алгоритмі формування рекомендацій.

- Рекомендаційний модуль (Recommender) — реалізує основну логіку підбору кінематографічних творів на основі методу контентної фільтрації.

Розглянемо детальніше всі функції, які виконує модуль MovieAppGUI.

Функція __init__ є конструктором класу MovieAppGUI, який ініціалізує головне вікно додатку, встановлюючи його розмір, назву та графічний вигляд. За допомогою бібліотеки tkinter створюється головне вікно інтерфейсу з назвою «Movie Recommender» та розмірами 1200x800 пікселів. Потім створюються екземпляри класів MovieDataset, UserPreferences і Recommender, які відповідають за завантаження фільмів, зберігання налаштувань користувача і створення рекомендацій. На завершення, викликається метод build_interface для побудови графічного інтерфейсу користувача.

```
def __init__(self):
    self.root = tk.Tk()
    self.root.title("Movie Recommender")
    self.root.geometry("1200x800")

    self.style = ttk.Style()
    self.style.configure('TCombobox', padding=8, font=('Roboto', 10))
    self.style.configure('TButton', padding=8, font=('Roboto', 10, 'bold'))

    self.bg_image = None
    try:
        response = requests.get(
            "https://images.unsplash.com/photo-1489599849927-
2ee91cede3ba?auto=format&fit=crop&w=1200")
        img = Image.open(BytesIO(response.content))
        img = img.resize((1200, 800), Image.Resampling.LANCZOS)
        img = img.convert('RGBA')
        alpha = Image.new('RGBA', img.size, (255, 255, 255, 80))
        img = Image.blend(img, alpha, 0.2)
```

```
        self.bg_image = ImageTk.PhotoImage(img)
    except:
        pass

    self.dataset = MovieDataset("movies.csv")
    self.preferences = UserPreferences()
    self.recommender = Recommender(self.dataset)

    self.build_interface().
```

Функція build_interface відповідає за побудову графічного інтерфейсу користувача рекомендаційної системи. Першим кроком є створення об'єкта Canvas, який буде тлом для всього інтерфейсу. Якщо під час ініціалізації було завантажено фонове зображення, воно додається до полотна у верхній лівій частині вікна (anchor="nw"). Це створює візуально привабливу основу для додаткових елементів інтерфейсу.

```
  def build_interface(self):
    bg_canvas = tk.Canvas(self.root, highlightthickness=0)
    bg_canvas.pack(fill='both', expand=True)
    if self.bg_image:
        bg_canvas.create_image(0, 0, image=self.bg_image, anchor='nw').
```

У центрі вікна створюється основна рамка (main_frame) темного кольору, що виступає контейнером для всіх подальших елементів. У верхній частині цієї рамки додається заголовок системи "Movie Recommendation System" з великим шрифтом для наочності.

```
  main_frame = tk.Frame(bg_canvas, bg='#2A2A2E')
main_frame.place(relx=0.5, rely=0.5, anchor='center', width=1000, height=700)

gradient_canvas = tk.Canvas(main_frame, highlightthickness=0)
gradient_canvas.place(relwidth=1, relheight=1)
gradient_canvas.create_rectangle(0, 0, 1000, 700, fill='', outline='')

title = tk.Label(main_frame, text="Movie Recommendation System",
                 font=('Roboto', 24, 'bold'), bg='#2A2A2E', fg='#FFFFFF')
title.pack(pady=20).
```

Далі створюється окремий блок filters_frame, який містить всі елементи для збору користувацьких опцій. Фрейм має вирівнювання по стовпчиках, щоб дозволити компонентам адаптуватися до розміру вікна.

```
self.filters_frame = tk.Frame(main_frame, bg='#3A3A3E', bd=2, relief='flat')
self.filters_frame.pack(pady=20, padx=20, fill='x')

self.filters_frame.columnconfigure((0, 1, 2, 3, 4, 5), weight=1).
```

Користувач може вибрати до трьох улюблених жанрів з динамічного списку. Також можна вибрати вікові обмеження (так/ні), мову оригіналу (відображається назва та код мови) і країну виробництва.

```
self.genre_combo1 = ttk.Combobox(self.filters_frame, values=[""] +
list(self.dataset.get_unique_genres()),
                                 style='TCombobox')
...
self.age_limit = ttk.Combobox(self.filters_frame, values=["Yes", "No"],
                              style='TCombobox', font=('Roboto', 10))
...
self.lang_combo = ttk.Combobox(self.filters_frame,
                               values=[f"{LANGUAGE_MAP.get(code, code)}({code})"
                                       for code in
self.dataset.get_unique_languages()],
                               style='TCombobox')
...
self.country_combo = ttk.Combobox(self.filters_frame,
                                  values=self.dataset.get_unique_countries(),
                                  style='TCombobox').
```

Для більшої гнучкості у виборі фільмів можна ввести тривалість і рік випуску. Вони складаються з комбінованого списку зі знаками порівняння (>, <) та поля введення числового значення («Spinbox»), де можна встановити бажаний поріг пошуку.

```
self.duration_sign = ttk.Combobox(duration_frame, values=[">", "<"],
                                  width=5, style='TCombobox')
self.duration_spin = tk.Spinbox(duration_frame, from_=30, to=300...)
...
self.year_sign = ttk.Combobox(year_frame, values=[">", "<"],
                              width=5, style='TCombobox')
self.year_sign.pack(side='left')
self.year_spin = tk.Spinbox(year_frame, from_=1980, to=2025...).
```

Для завершення інтерфейсу додано кнопку «Отримати рекомендації», яка викликає метод get_recommendations і починає генерувати рекомендації. Для

покращення користувацького досвіду кнопка змінює колір при наведенні на неї курсору миші.

```
    submit_button = tk.Button(main_frame, text="Get Recommendations",
                              command=self.get_recommendations ...)
submit_button.pack(pady=20)
submit_button.bind('<Enter>', lambda e: submit_button.config(bg='#7F39FB'))
submit_button.bind('<Leave>', lambda e: submit_button.config(bg='#6200EE')).
```

Функція get_recommendations збирає всі фільтри та параметри, введені користувачем в інтерфейсі, створює на їх основі об'єкт self.preferences і надсилає його рекомендору для отримання відповідного списку фільмів. Спочатку перевіряються три поля вибору жанрів, і якщо користувач обирає хоча б один, ці жанри додаються до списку вподобань. Далі визначається, чи користувач обирає контент з віковими обмеженнями, чи без, обробляється обрана мова, країна виробництва, тривалість і рік випуску фільму зі знаком порівняння (< або >). Тривалість і рік конвертуються в цілі числа, і якщо виникає помилка, відображається повідомлення про помилку, а операція завершується. Якщо вся інформація правильна, вона передається до модуля рекомендацій (self.recommender.recommend), а результат записується в журнал за допомогою методу self.log_recommendations і виводиться на екран за допомогою методу self.show_results_page. Таким чином, ця функціональність забезпечує весь цикл обробки від взаємодії з користувацьким інтерфейсом до відображення персональних результатів.

```
def get_recommendations(self):
    genres = []
    for combo in [self.genre_combo1, self.genre_combo2, self.genre_combo3]:
        genre = combo.get()
        if genre and genre != "":
            genres.append(genre)
    self.preferences.genres = genres

    self.preferences.adult_only = False if self.age_limit.get() == "No" else True
    selected_lang = self.lang_combo.get()
    self.preferences.language = selected_lang.split("(")[-1].replace(")",
                                                                     "") if "(" in
```

```
selected_lang else selected_lang
    self.preferences.country = self.country_combo.get()

    try:
        self.preferences.duration_value = int(self.duration_spin.get())
        self.preferences.duration_sign = self.duration_sign.get()
        self.preferences.release_year = int(self.year_spin.get())
        self.preferences.release_sign = self.year_sign.get()
    except ValueError:
        messagebox.showerror("Error", "Invalid duration or year entered")
        return

    results = self.recommender.recommend(self.preferences)

    self.log_recommendations(genres, self.preferences, results)

    self.show_results_page(results).
```

Функція log_recommendations відіграє важливу роль, записуючи історію запитів користувачів до системи рекомендацій, що дозволяє в подальшому аналізувати вподобання або відстежувати ефективність рекомендацій. Спочатку функція записує поточну дату і час за допомогою модуля datetime. Далі створюється словник input_data, що містить усю інформацію з обраних користувачем параметрів: жанри, вікові обмеження, мову, країну, бажану тривалість фільму та рік випуску, а також список рекомендованих фільмів, який конвертується в рядок. Якщо значення відсутнє, функція замінює його на «None», щоб уникнути помилок пам'яті. Отримані дані записуються в об'єкт DataFrame, який потім додається до CSV-файлу з назвою recommendation_log.csv. Заголовки зберігаються тільки в тому випадку, якщо файл більше не існує. Якщо під час збереження виникають помилки, функція виводить повідомлення про помилку і відповідний текст, щоб надати користувачеві практичну інформацію. Таким чином, ця функція реалізує зручний і гнучкий механізм логування взаємодії з системою, підвищуючи її функціональність і надійність

```
def log_recommendations(self, genres, preferences, movies):
    timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    input_data = {
        'timestamp': timestamp,
```

```
        'genres': ','.join(genres) if genres else 'None',
        'adult_only': preferences.adult_only,
        'language': preferences.language if preferences.language else 'None',
        'country': preferences.country if preferences.country else 'None',
        'duration': f"{preferences.duration_sign}{preferences.duration_value}" if
preferences.duration_value else 'None',
        'release_year': f"{preferences.release_sign}{preferences.release_year}" if
preferences.release_year else 'None',
        'recommended_movies': ','.join(movies['title'].tolist()) if not
movies.empty else 'None'
    }

    log_df = pd.DataFrame([input_data])

    try:
        log_df.to_csv('recommendation_log.csv', mode='a', index=False,
                      header=not
pd.io.common.file_exists('recommendation_log.csv'))
    except Exception as e:
        messagebox.showerror("Error", f"Failed to save recommendation log:
{str(e)}").
```

При виконанні функції def show_results_page на початку створюється нове дочірнє вікно (Toplevel) з темною темою, заголовком і фіксованим розміром. Всередині нього додається полотно для позиціонування контенту з вертикальною прокруткою, смуга прокрутки та scroll_frame, що містить усі віджети.

```
results_window = tk.Toplevel(bg='#2A2A2E')
results_window.title("Recommended Movies")
results_window.geometry("1000x700")

canvas = tk.Canvas(results_window, bg='#2A2A2E', highlightthickness=0)
scrollbar = tk.Scrollbar(results_window, orient="vertical", command=canvas.yview)
scroll_frame = tk.Frame(canvas, bg='#2A2A2E')

scroll_frame.bind("<Configure>", lambda e:
canvas.configure(scrollregion=canvas.bbox("all")))
canvas.create_window((0, 0), window=scroll_frame, anchor='nw')
canvas.configure(yscrollcommand=scrollbar.set)

canvas.pack(side="left", fill="both", expand=True)
scrollbar.pack(side="right", fill="y").
```

Далі відбувається перевірка: якщо Movie DataFrame порожній (тобто немає фільмів, що відповідають фільтрам), виводиться повідомлення «Не знайдено жодного фільму, що відповідає критеріям». В іншому випадку функція переглядає кожен рядок DataFrame, тобто кожен рекомендований фільм, і створює для нього окремий блок.

```
if movies.empty:
    tk.Label(scroll_frame, text="No movies found matching your criteria",
             bg='#2A2A2E', fg='#FFFFFF', font=('Roboto', 14,
'bold')).pack(pady=20)
else:
    for _, movie in movies.iterrows():
        frame = tk.Frame(scroll_frame, bg='#3A3A3E', bd=2, relief='flat')
        frame.pack(fill='x', pady=10, padx=20)

        img_url = f"https://image.tmdb.org/t/p/w200{movie['poster_path']}"
        try:
            response = requests.get(img_url)
            img = Image.open(BytesIO(response.content))
            img = img.resize((150, 225), Image.Resampling.LANCZOS)
            photo = ImageTk.PhotoImage(img)
            label_img = tk.Label(frame, image=photo, bg='#3A3A3E')
            label_img.image = photo
            label_img.pack(side='left', padx=10, pady=10)
        except:
            pass.
```

Для кожного фільму створюється блок з темним фоном, в який завантажується зображення постера зліва з сайту TMDB. Зображення масштабується до 150×225 пікселів і відображається в інтерфейсі; якщо не вдається завантажити зображення (наприклад, у випадку, якщо зовсім немає постера), блок просто ігнорується. Праворуч від зображення створюється новий блок, що містить текстову інформацію про фільм: назва, рік випуску, рейтинг, тривалість, мова, країна, жанр і короткий опис, обмежений по ширині для зручності читання.

```
info_frame = tk.Frame(frame, bg='#3A3A3E')
info_frame.pack(side='left', padx=10, pady=10)

tk.Label(info_frame, text=f"Title: {movie['title']}",
         font=('Roboto', 14, 'bold'), bg='#3A3A3E', fg='#FFFFFF').pack(anchor='w')
...
```

```
tk.Label(info_frame, text=f"Overview: {movie['overview']}",
         wraplength=600, justify='left', bg='#3A3A3E', fg='#B0B0B0',
         font=('Roboto', 11)).pack(anchor='w', pady=5).
```

Після списку фільмів на сторінці з'являється текстове запитання: «Чи підходять вам ці рекомендації?» і дві кнопки під ним. Кнопка Yes завершує програму (self.root.destroy), а кнопка No викликає метод ask_continue, який дозволяє змінити фільтри або згенерувати новий список. Цей блок додає інтерактивність та зворотний зв’язок, що дозволяє користувачеві продовжити пошук або завершити роботу з застосунком.

```
tk.Label(scroll_frame, text="Are these recommendations suitable?",
         font=('Roboto', 14, 'bold'), bg='#2A2A2E', fg='#FFFFFF', pady=20).pack()

buttons_frame = tk.Frame(scroll_frame, bg='#2A2A2E')
buttons_frame.pack(pady=20)

tk.Button(buttons_frame, text="Yes", command=self.root.destroy,
          bg='#00C853', fg='#FFFFFF', font=('Roboto', 12, 'bold'),
          relief='flat', padx=20, pady=10).pack(side='left', padx=10)
tk.Button(buttons_frame, text="No", command=self.ask_continue,
          bg='#DD2C00', fg='#FFFFFF', font=('Roboto', 12, 'bold'),
          relief='flat', padx=20, pady=10).pack(side='left', padx=10).
```

Функція ask_continue(self) відкриває діалогове вікно, в якому користувач може обрати, чи бажає він продовжити пошук. Якщо вибрано «Так», інтерфейс скидається викликом reset_interface і користувач може ввести нові налаштування. Якщо вибрано «Ні», головне вікно знищується, а програма закривається.

```
def ask_continue(self):
    response = messagebox.askyesno("Continue?", "Would you like to continue
searching?")
    if response:
        self.reset_interface()
    else:
        self.root.destroy().
```

Функція reset_interface(self) скидає користувацький інтерфейс до початкового вигляду: видаляє всі поля для вибору жанру, віку, мови, країни, періоду та року і створює новий об'єкт UserPreferences. Це дозволяє розпочати новий пошук без необхідності перезапуску програми.

```
def reset_interface(self):
    self.preferences = UserPreferences()
    self.genre_combo1.set("")
    self.genre_combo2.set("")
    self.genre_combo3.set("")
    self.age_limit.set("")
    self.lang_combo.set("")
    self.country_combo.set("")
    self.duration_sign.set("")
    self.duration_spin.delete(0, tk.END)
    self.duration_spin.insert(0, "90")
    self.year_sign.set("")
    self.year_spin.delete(0, tk.END)
    self.year_spin.insert(0, "2000").
```

Функція run(self) запускає основний цикл обробки подій Tkinter, який забезпечує графічний інтерфейс користувача і взаємодію програми з користувачем. Без виклику функції mainloop() вікно не відображається і взаємодія з елементами графічного інтерфейсу неможлива.

```
def run(self):
    self.root.mainloop().
```

Розглянемо детальніше всі функції, які виконує модуль MovieDataset.

Конструктор __init__(self, csv_path) класу MovieDataset, завантажує набір даних з CSV-файлу за вказаним шляхом. Після завантаження одразу викликається допоміжна функція _preprocess, яка готує дані для подальшого використання, очищаючи та конвертуючи деякі поля.

```
def __init__(self, csv_path):
    self.df = pd.read_csv(csv_path)
    self._preprocess().
```

Функція _preprocess(self) готує набір даних. Вона видаляє рядки з відсутніми значеннями в ключових стовпчиках, таких як стать, зріст, мова, країна-виробник і шлях до маніфесту. Вона також перетворює стать і країну з рядків у списки, розділені комами, що полегшує фільтрацію даних у подальших операціях.

```
def _preprocess(self):
    self.df.dropna(subset=['genres', 'runtime', 'original_language',
'production_countries', 'poster_path'], inplace=True)
    self.df['genres'] = self.df['genres'].str.split(', ')
```

```
    self.df['production_countries'] = self.df['production_countries'].str.split(',
').
```

Функція get_unique_genres(self) отримує всі унікальні жанри зі стовпця Genres, де кожне значення є списком жанрів у фільмі. Вона об'єднує всі ці списки в один набір, щоб видалити дублікати, і повертає їх відсортованими.

```
def get_unique_genres(self):
    genres = set()
    for genre_list in self.df['genres']:
        genres.update(genre_list)
    return sorted(list(genres)).
```

Ця функція get_unique_languages(self) просто повертає список унікальних мов оригіналу, присутніх у даних. Для простоти результат відсортовано за алфавітом.

```
def get_unique_languages(self):
    return sorted(self.df['original_language'].unique()).
```

Подібно до жанрів, ця функція get_unique_countries(self) повертає набір унікальних країн-виробників для всіх фільмів. Вона перевіряє кожен список країн, об'єднує їх у набір, видаляє дублікати і повертає впорядкований список.

```
def get_unique_countries(self):
    countries = set()
    for country_list in self.df['production_countries']:
        countries.update(country_list)
    return sorted(list(countries)).
```

Функція get_filtered_movies(self, preferences) виконує базову логіку для фільтрації фільмів за вподобаннями користувача: жанр, вікове обмеження, мова, країна, тривалість та індикатор посилання. Після покрокового застосування кожного фільтра результати сортуються за середнім рейтингом у порядку спадання і повертаються 5 найкращих фільмів.

```
def get_filtered_movies(self, preferences):
    df_filtered = self.df.copy()

    if preferences.genres:
        df_filtered = df_filtered[df_filtered['genres'].apply(lambda x: any(genre
in x for genre in preferences.genres))]

    if preferences.adult_only is not None:
        df_filtered = df_filtered[df_filtered['adult'] == preferences.adult_only]
```

```
    if preferences.language and preferences.language != 'Не важливо':
        df_filtered = df_filtered[df_filtered['original_language'] ==
preferences.language]

    if preferences.country and preferences.country != 'Не важливо':
        df_filtered = df_filtered[df_filtered['production_countries'].apply(lambda
x: preferences.country in x)]

    if preferences.duration_sign:
        if preferences.duration_sign == '<':
            df_filtered = df_filtered[df_filtered['runtime'] <
preferences.duration_value]
        elif preferences.duration_sign == '>':
            df_filtered = df_filtered[df_filtered['runtime'] >
preferences.duration_value]

    return df_filtered.sort_values(by='vote_average', ascending=False).head(5).
```

Розглянемо детальніше всі функції, які виконує модуль UserPreferences.

Функція __init__ модуля UserPreferences є конструктором, який ініціалізує об'єкт порожнім об'єктом або початковими значеннями всіх налаштувань користувача, що використовуються в системі налаштувань. Він створює шість атрибутів: genres – порожній список жанрів, обраних для зберігання, adult_only – логічне значення (або None), що вказує, чи дозволено фільми тільки з віковими обмеженнями, language and country – мова і країна, що цікавлять користувача (спочатку None), a duration_sign і duration_value – оператор порівняння (<, > або None) і значення тривалості потрібного фільму та року його виходу. Таким чином, цей конструктор надає початкову структуру для зберігання вибраних користувачем параметрів у графічному інтерфейсі.

```
def __init__(self):
    self.genres = []
    self.adult_only = None
    self.language = None
    self.country = None
    self.duration_sign = None
    self.duration_value = None.
```

Розглянемо детальніше всі функції, які виконує модуль Recommender.

Функція __init__ модуля Recommender є конструктором класу, який приймає сутність набору даних (типу MovieDataset) і зберігає її в атрибуті self.dataset для подальшого використання. Таким чином встановлюється зв'язок між модулем рекомендацій та джерелом MovieDataset.

```
def __init__(self, dataset):
    self.dataset = dataset.
```

Функція recommend виконує основну логіку отримання рекомендацій: вона приймає об'єкт preferences (типу UserPreferences), що містить усі параметри вибору користувача, і викликає метод get_filtered_movies модуля MovieDataset. Результатом є відфільтрований список фільмів, які найкраще відповідають вподобанням користувача. Таким чином, recommend виступає, як інтерфейс між вподобаннями користувача та системою фільтрації.

```
    def recommend(self, preferences):
    return self.dataset.get_filtered_movies(preferences).
```

3.3 Тестування роботи програмного модуля підбору кінематографічного твору на основі уподобань користувача

У процесі тестування програмного модуля підбору кінематографічного твору на основі вподобань користувача важливо переконатися в коректності його роботи на всіх етапах взаємодії з користувачем. Зокрема, необхідно перевірити працездатність механізмів введення користувацьких вподобань, відповідність наданих рекомендацій заданим значенням, коректність роботи системи у випадках, коли не вдається підібрати відповідні фільми, а також правильність збереження інформації про сесії підбору у файл та їх відображення користувачу. Система має коректно опрацьовувати вхідні дані, пов’язані з жанрами, тривалістю, роком випуску, мовою та країною виробництва, а також надавати користувачу зручний і зрозумілий інтерфейс для ознайомлення з результатами.

Тестування проводилося в середовищі розробки PyCharm з використанням тестових даних, які охоплюють різні варіанти користувацьких запитів. Окремо перевірялася робота функціоналу повторного підбору у випадку, якщо користувача

не влаштовують надані рекомендації. Також здійснювалася валідація збереження результатів підбору у форматі CSV для подальшого аналізу історії рекомендацій. За допомогою тестування програмного модулю буде забезпечено його правильну роботу та буде можливість виявити та виправити можливі недоліки чи проблеми у функціональності.

Спершу перевіримо коректність роботи механізму авторизації, що включає реєстрацію нового користувача та подальший вхід у систему. На рисунку 3.1 наведено зовнішній вигляд вікна входц та реєстрації, яке з’являється після запуску програми.

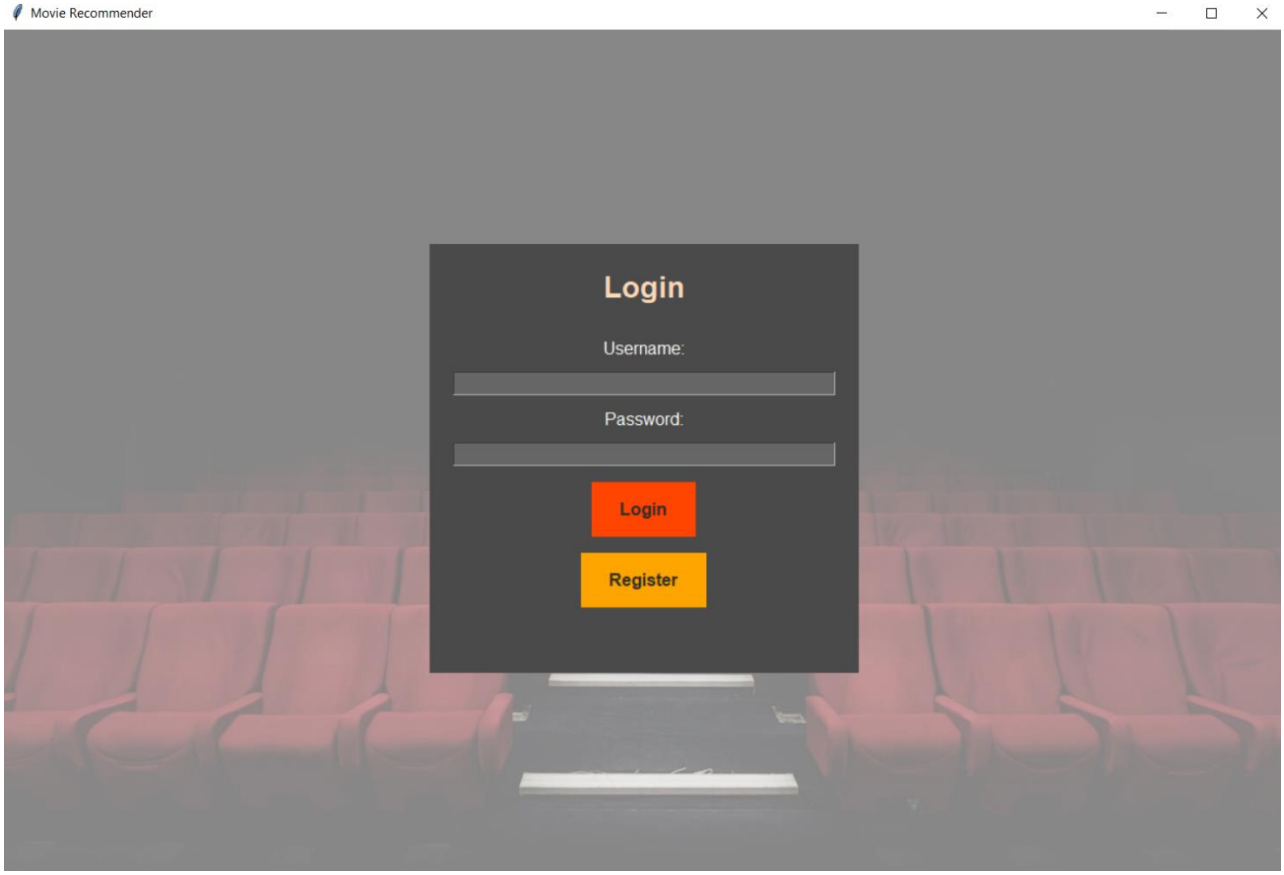

Рисунок 3.1 – Загальний вигляд вікна форми авторизації в систему

Для перевірки функціоналу реєстрації створимо обліковий запис користувача з ім’ям test. У результаті успішної реєстрації система має зберегти дані користувача у відповідному файлі та відобразити повідомлення про успішну реєстрацію, що показано на рисунку 3.2.

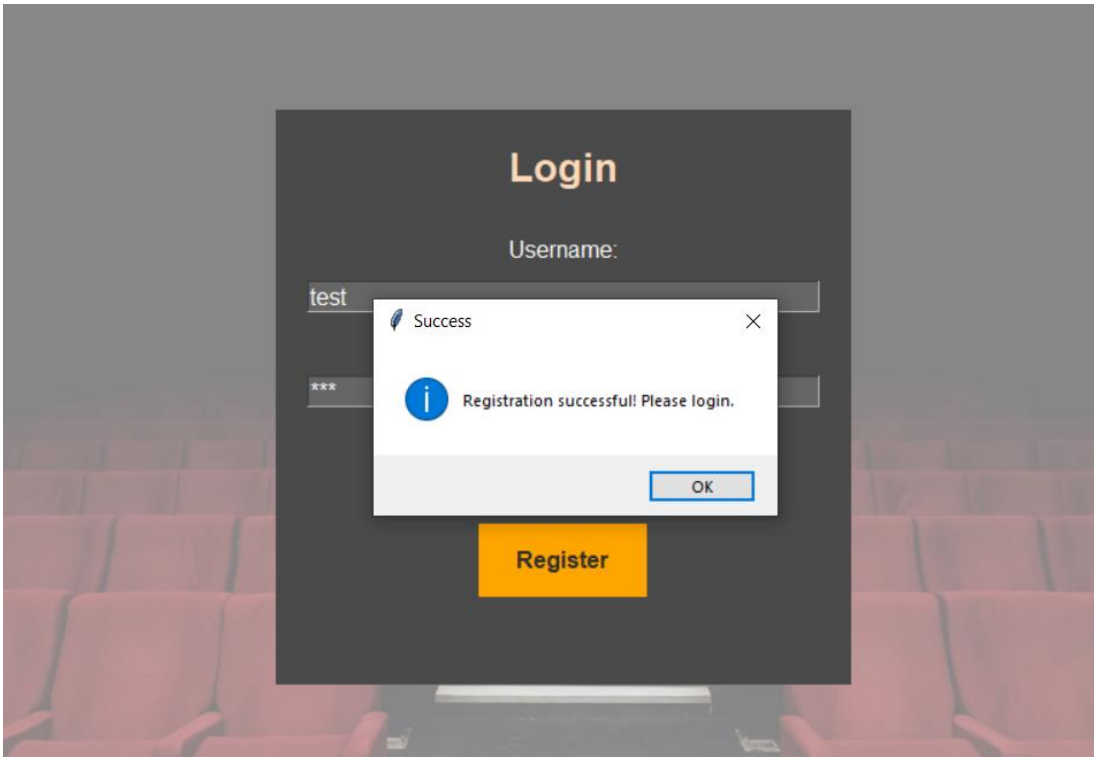

Рисунок 3.2 – Повідомлення про успішну реєстрацію користувача в системі

Далі перевіримо процес авторизації. На цьому етапі спочатку виконаємо тестування негативного сценарію: введемо неправильні логін або пароль. Система має вивести повідомлення про помилку, як показано на рисунку 3.3.

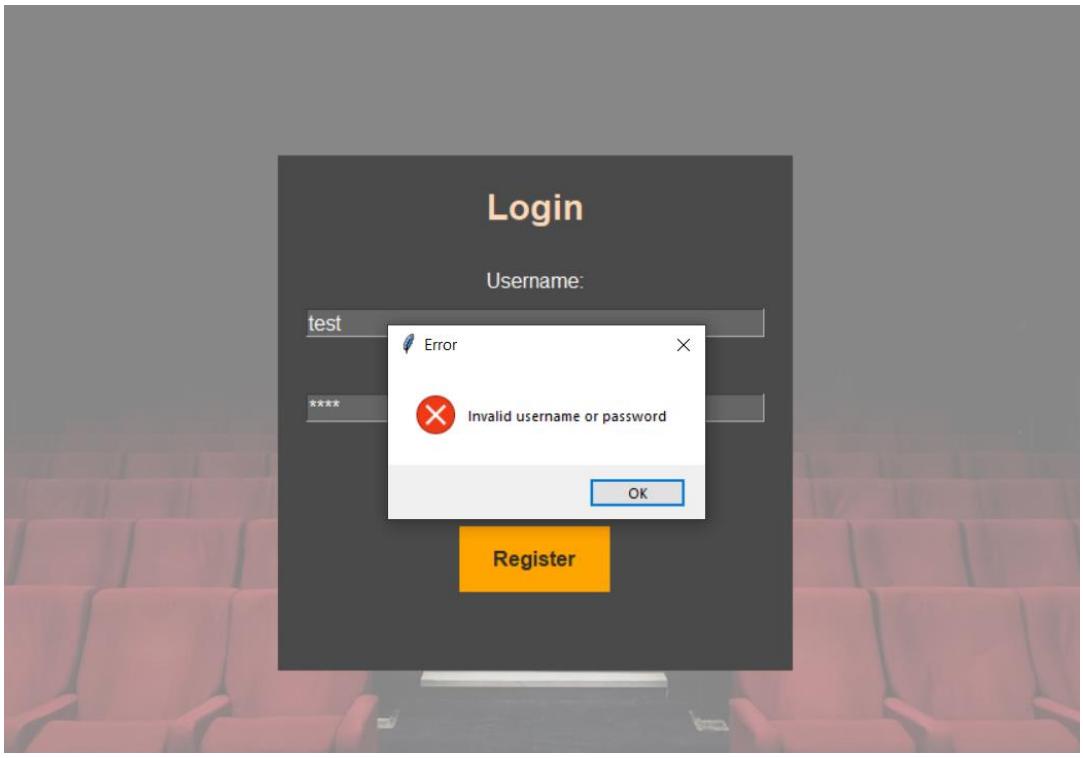

Рисунок 3.3 – Повідомлення про некоректні вхідні дані при спробі входу

Після цього введемо правильні дані для входу. У разі успішної авторизації має відкритись основне вікно з інтерфейсом вибору параметрів для формування рекомендацій, що представлено на рисунку 3.4.

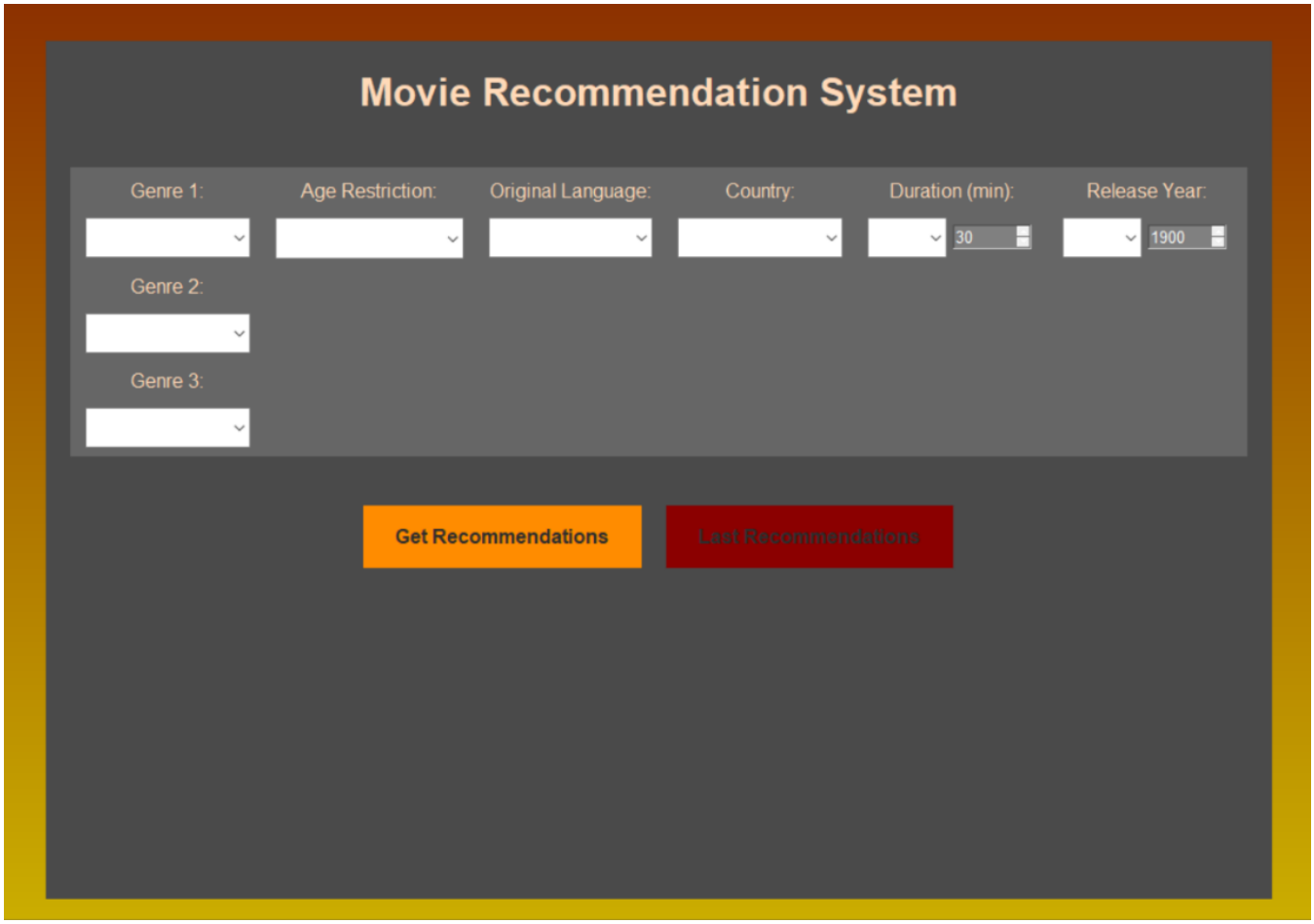

Рисунок 3.4 – Загальний вигляд вікна вибору параметрів для формування рекомендацій

Далі протестуємо роботу системи рекомендацій. Виберемо коректні значення для всіх доступних фільтрів, зокрема жанри, обмеження за віком, мову, країну, тривалість та рік виходу, а саме жанри – Action, Adventure, Comedy. В полі Age restriction оберемо опцію No. Мова оригіналу – English, країна – United States of America. Значення тривалості в хвилинах зазначимо більше 60, а рік виходу більше 2010. Приклад заповнення форми наведено на рисунку 3.5.

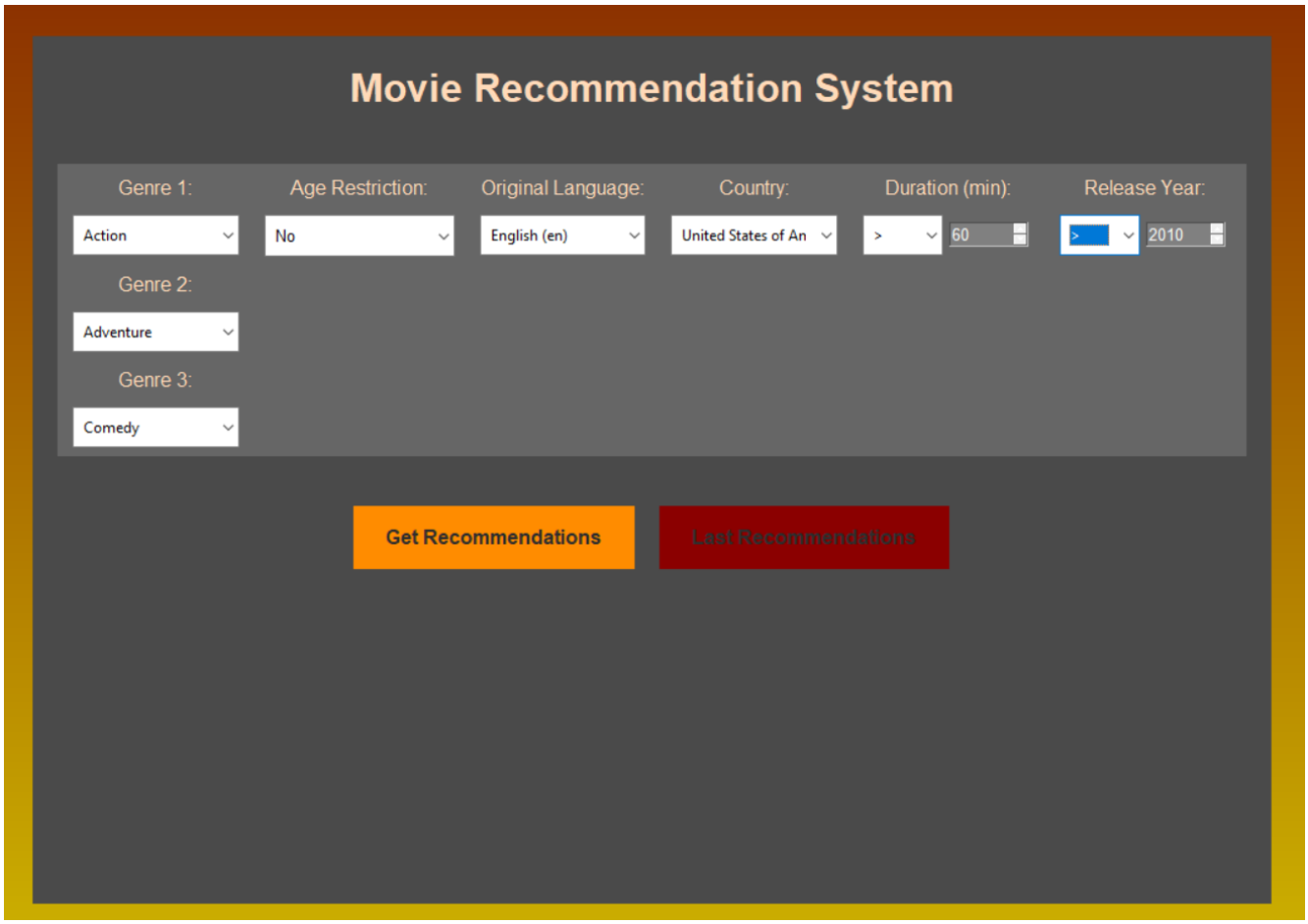

Рисунок 3.5 – Приклад заповнення даних для надання рекомендацій

Після натискання кнопки «Отримати рекомендації» система має проаналізувати введені дані та сформувати перелік кінематографічних творів, що відповідатимуть обраним характеристикам. На рисунку 3.6 можемо побачити, що розроблений модуль надав 5 різних кінематографічних творів і кожен із рекомендованих фільмів відповідає зазначеним фільтрам – за жанрами, мовою, країною виробництва, тривалістю та роком виходу, що свідчить про правильну роботу логіки відбору. Відповідно до кожного кінематографічного твору відображаються правильні вихідні дані, а саме постер, назва, рік виходу, оцінка, тривалість, мова, країна, жанри та опис твору.

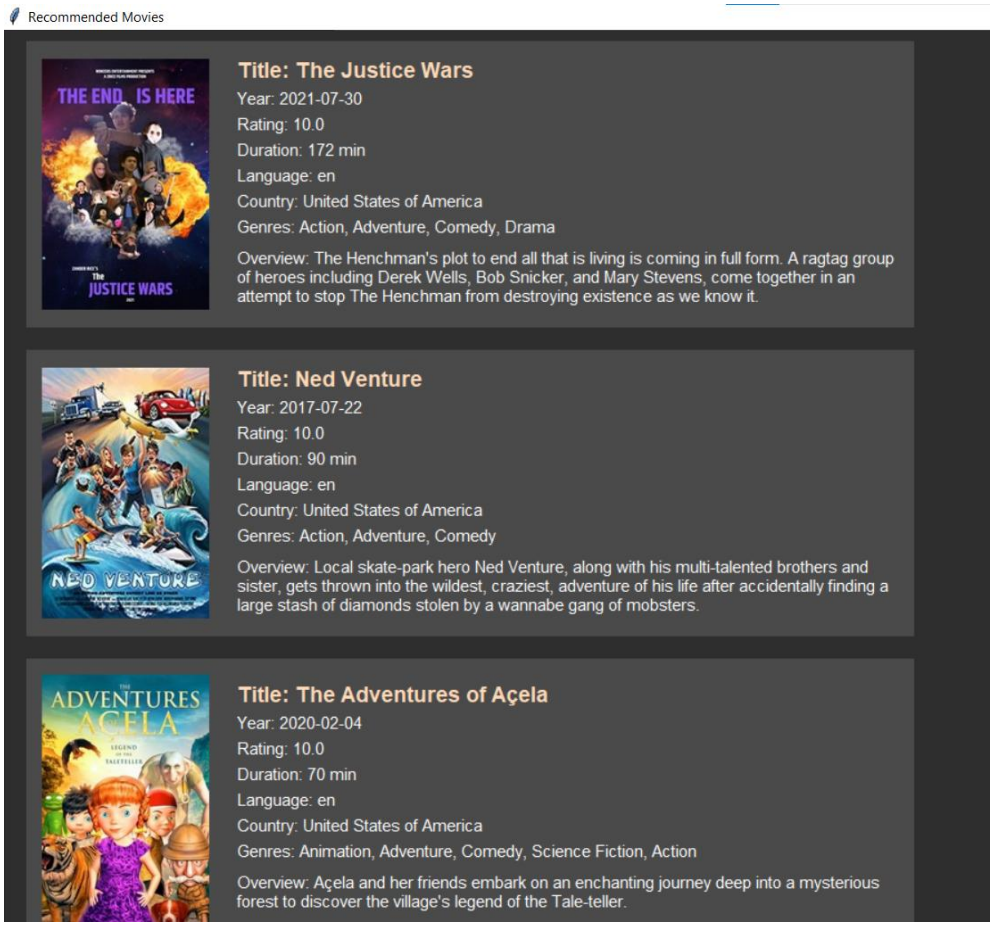

Рисунок 3.6 – Загальний вигляд вікна результатів після надання рекомендацій

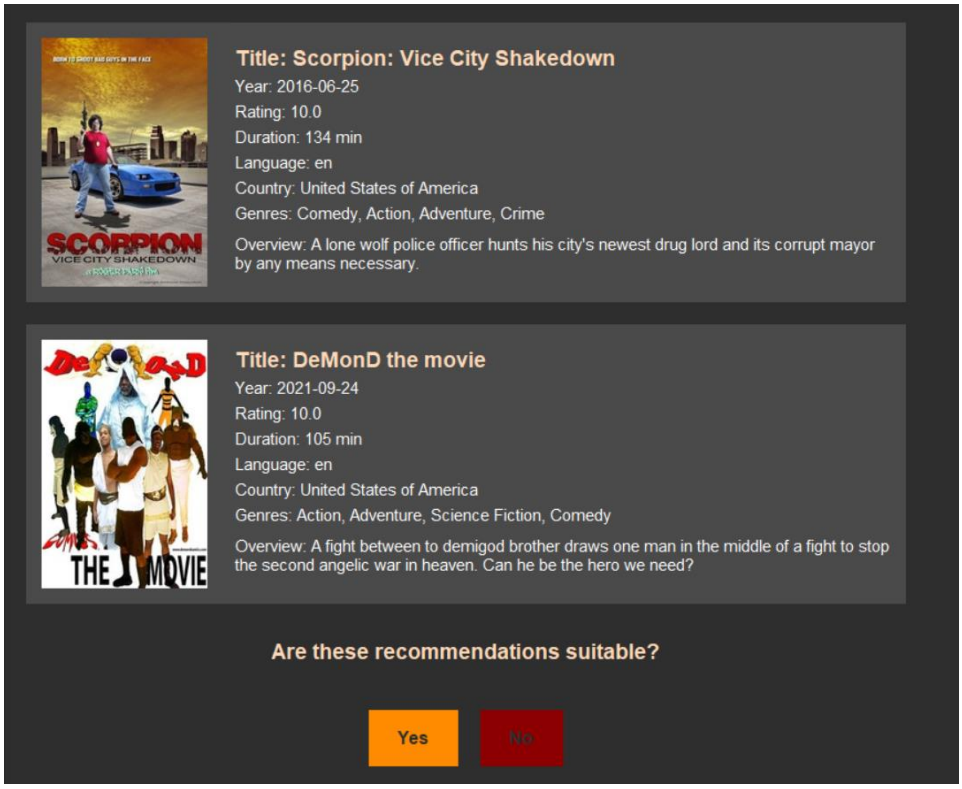

Рисунок 3.6 – продовження

Окремо перевіримо механізм уточнення, чи задовольняють користувача запропоновані фільми. У випадку натискання кнопки «Так» програмний модуль повинен завершити свою роботу. Якщо ж обрати «Ні», відкриється діалогове вікно з уточненням, чи бажає користувач продовжити пошук, як зображено на рисунку 3.7. Якщо відповісти «Так», програма знову відкриває головне вікно для введення параметрів. Якщо ж відповісти «Ні» – система завершить роботу.

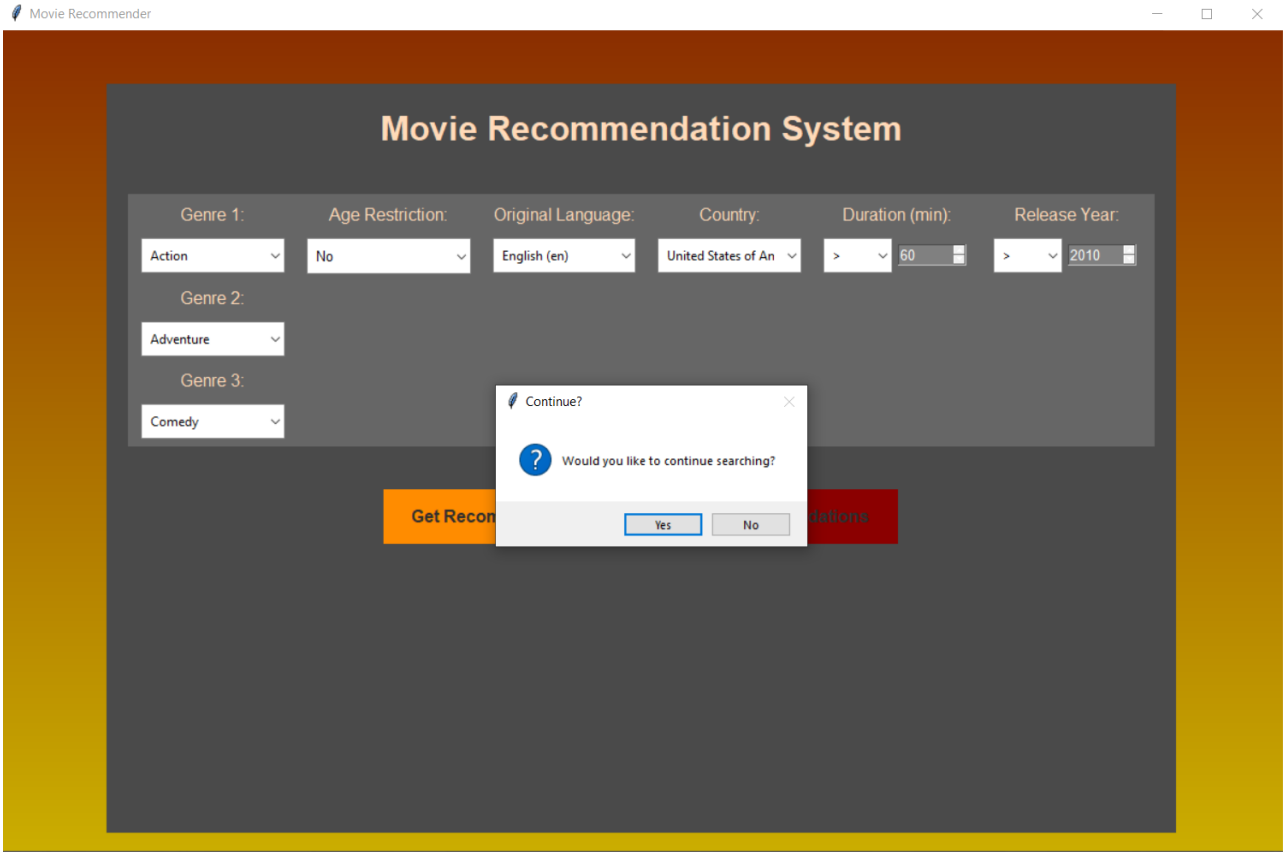

Рисунок 3.7 – Загальний вигляд діалогового вікна уточнення потреби продовження підбору кінематографічних творів

Далі протестуємо функціонал перегляду історії попередніх рекомендацій. Натиснемо кнопку «Останні рекомендації» — система має зчитати відповідні дані з файлу та відобразити їх у новому вікні. Результат відображення історії продемонстровано на рисунку 3.8.

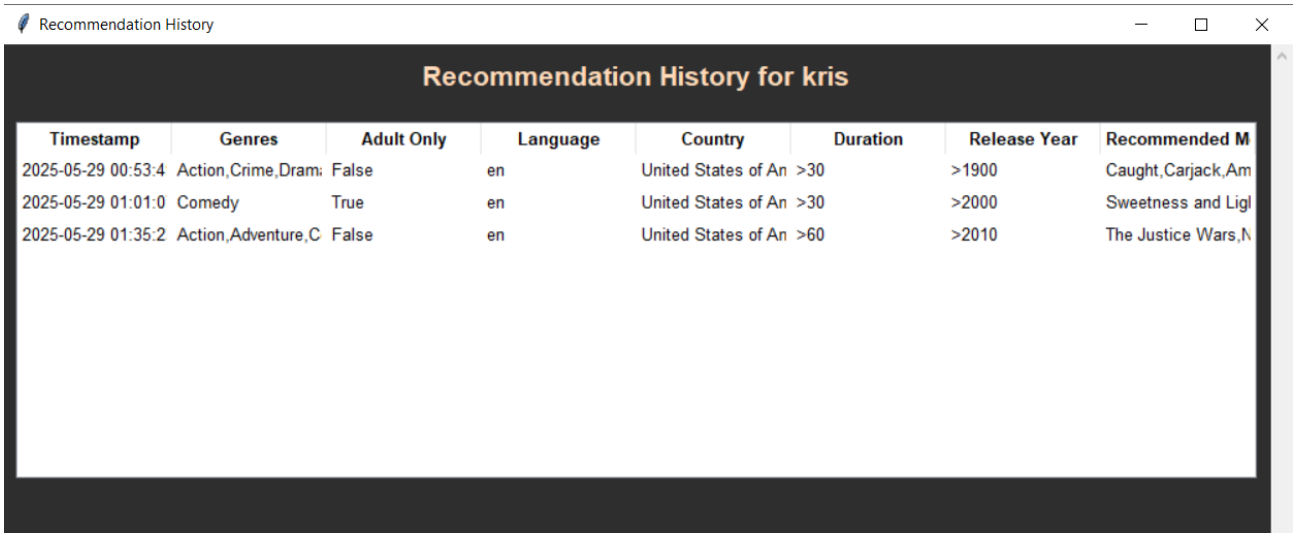

Рисунок 3.8 – Загальний вигляд вікна історії наданих рекомендацій певному користувачу

Також перевіримо ситуацію, коли в користувача ще немає жодних записів про рекомендації. У цьому випадку програма повинна повідомити про відсутність збережених даних, що показано на рисунку 3.9.

Рисунок 3.9 – Загальний вигляд вікна історії наданих рекомендацій користувачу, в якого немає записів про надані раніше рекомендації

Для визначення ефективності та точності надання рекомендацій персоналізованою системою проведемо тестування двох варіантів підбору кінематографічних творів, а саме для навчального набору даних, використовуючи який рекомендації формувалися на основі узагальнених значень, та тестового набору даних, де в значеннях вказані індивідуальні вподобання користувача.

У рамках тестування кожному з трьох користувачів було запропоновано по 5 фільмів. Результати тестування наведемо в таблиці 3.2:

Таблиця 3.2 – Порівняльний аналіз результатів тестування системи на навчальному та тестовому наборах даних

Користувач №	Навчальний набір даних	Точність	Тестовий набір даних	Точність
1	1	20%	4	80%
2	1	20%	3	60%
3	2	40%	4	80%
Середнє значення	1.3	26.7%	3.7	73.3%

Таким чином, розроблена система забезпечила значне підвищення точності рекомендацій – із 26.7% до 73.3%, тобто на 46.6%, що свідчить про ефективність використання індивідуальних уподобань користувача під час формування рекомендацій.

У результаті проведеного тестування було встановлено, що програмний модуль функціонує стабільно та правильно обробляє всі можливі сценарії взаємодії з користувачем. Система забезпечує коректну роботу механізмів реєстрації та входу до облікового запису, дозволяє обирати параметри пошуку фільмів, успішно формує персоналізовані рекомендації відповідно до зазначених фільтрів, обробляє як позитивні, так і негативні відповіді користувача на отримані результати, а також веде журнал попередніх рекомендацій.

3.4 Висновок до розділу 3

У третьому розділі було проведено важливий етап розробки – реалізацію програмного модуля системи рекомендацій кінематографічних творів. У першій частині розділу було обґрунтовано вибір мови програмування, технологій та інструментів розробки. Мовою реалізації було обрано мову Python через її широкий спектр бібліотек для обробки даних та зручний синтаксис. Python також дозволяє ефективно поєднувати обчислювальну логіку та графічну побудову

інтерфейсу користувача, що є важливою частиною створення інтуїтивно зрозумілої системи рекомендацій.

Наступним кроком була реалізація основних функціональних частин розроблюваної системи. Зокрема, були створені окремі модулі для обробки бази даних фільмів, збору вподобань користувачів, фільтрації фільмів за введеними параметрами та модуль для генерації персоналізованих рекомендацій. Особливу увагу було приділено логіці відбору фільмів за критеріями.

На завершення поетапно були протестовані всі основні сценарії взаємодії з системою. Були протестовані позитивні та негативні кейси: авторизація з правильними та неправильними даними, запити з повними та неповними фільтрами, поведінка системи без рекомендацій та збережених запитів. Результати тестування показали, що програмний модуль працює стабільно, не дає збоїв і виконує всі необхідні функції. Інтерфейс системи є зручним і зрозумілим, а результати рекомендацій відповідали вхідним критеріям.

Отже, в рамках третьої частини проекту було повністю реалізовано програмну частину проекту, перевірено її працездатність та якість функціонування.

ВИСНОВКИ

Завдання поставлені перед бакалаврською кваліфікаційною роботою було виконано повністю, а саме:

1. Проведено аналіз актуальності розробки рекомендаційних систем в сфері кінематографу та розглянуто основні проблеми в цій сфері.
2. Розроблено загальний алгоритм програмного модуля підбору кінематографічних творів на основі вподобань користувача.
3. Виконано програмну реалізацію модуля підбору кінематографічних творів на основі вподобань користувача.
4. Проведено тестування програмного модулю підбору кінематографічних творів на основі вподобань користувача.
5. Розроблено інструкцію користувача програмного модуля підбору кінематографічних творів на основі вподобань користувача.

У першому розділі бакалаврської кваліфікаційної роботи проаналізовано актуальність розробки зважаючи на важливість створення систем рекомендацій фільмів з огляду на стрімке зростання мультимедійного контенту та необхідність персоналізованого підходу до вибору фільмів. Було досліджено основні виклики при розробці таких систем, такі як складність збору даних про вподобання користувачів, забезпечення релевантності результатів та обробка великих обсягів даних. Розглянуто основні підходи до побудови рекомендаційних систем, зокрема контент-орієнтованих, колаборативних та гібридних систем. На основі порівняльного аналізу було визначено оптимальний підхід – фільтрація на основі контенту, який було реалізовано в цій роботі. Окремо проаналізовано існуючі рішення в цій галузі, що дозволило виявити їх сильні сторони та існуючі обмеження.

У другому розділі було розроблено архітектуру програмного забезпечення, що реалізує функціонал рекомендаційної системи. Були визначені основні компоненти системи та їх взаємодія. Створено UML-діаграму класів для опису логічної структури проєкту, об'єктів та взаємозв'язків між ними. На основі сформованих вимог був розроблений загальний алгоритм роботи рекомендаційного модуля, який враховує

параметри, що вводяться користувачем, і дозволяє генерувати персоналізовані результати.

У третьому розділі обґрунтовано вибір мови програмування та середовища розробки для реалізації системи. Основною мовою програмування було обрано Python, яка пропонує високий ступінь гнучкості в обробці даних і дозволяє легко реалізувати графічний інтерфейс користувача. Реалізовано основні модулі системи: модуль побудови інтерфейсу користувача, модуль заванатаження та обробки даних, модуль формування переваг користувача та рекомендаційний модуль. Всі функції застосунку були протестовані комплексно, від реєстрації користувачів до перевірки точності результатів пошуку та обробки виключень. Результати показали, що система працює стабільно та надійно.

Підсумовуючи, можемо сказати, що у ході виконання кваліфікаційної роботи було повністю досягнуто поставленої мети починаючи від теоретичного аналізу проблем та методів побудови рекомендаційних систем до практичної реалізації повноцінного програмного модуля, здатного адаптуватися до вподобань користувача та надавати відповідні рекомендації щодо фільмів. Підвищено точність індивідуального підбору кінематографічних творів на 46.6%. Отримана система демонструє ефективність обраного підходу і може слугувати основою для подальшого розвитку та масштабування.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. The Importance of Recommendation Systems Nowadays, [Електронний ресурс]. Режим доступу: https://qymatix.de/en/recommendation-systems-nowadays/.

2. Build a Movie Recommendation System Using Machine Learning, [Електронний ресурс]. Режим доступу: https://www.appliedaicourse.com/blog/movie-recommendation-system-using-machine-learning/.

3. Кульбіда Х. О. «Рекомендаційний метод підбору музичних композицій» в Матеріалах конференції «LIV Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2025)», Вінниця, 2025. [Електронний ресурс] – Режим доступу: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2025/paper/view/23946

4. Movie Recommender Systems: Concepts, Methods, Challenges, and Future Directions, [Електронний ресурс]. Режим доступу: https://pmc.ncbi.nlm.nih.gov/articles/PMC9269752/.

5. Movie Recommendation Systems: A Business Guide, [Електронний ресурс]. Режим доступу: https://stratoflow.com/movie-recommendation-system/.

6. User preference modeling for movie recommendations based on deep learning, [Електронний ресурс]. Режим доступу: https://www.nature.com/articles/s41598-025-00030-5?utm_source=chatgpt.com.

7. Goyani M., Chaurasiya N. A Review of Movie Recommendation System: Limitations, Survey and Challenges // ELCVIA: Electronic Letters on Computer Vision and Image Analysis. – 2020. – Т. 19, № 3. – С. 18–37

8. Movie Recommendation Algorithm Using Social Network Analysis to Alleviate Cold-Start Problem, [Електронний ресурс]. Режим доступу: https://xml.jips-k.org/full-text/view?doi=10.3745%2FJIPS.04.0121&.

9. Content Based Filtering in Machine Learning [Електронний ресурс]. Режим доступу: https://www.scaler.com/topics/machine-learning/content-based-filtering/.

10. ML – Content Based Recommender System, [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/ml-content-based-recommender-system/.

11. User-User Collaborative Filtering For Jokes Recommendation, [Електронний ресурс]. Режим доступу: https://towardsdatascience.com/user-user-collaborative-filtering-for-jokes-recommendation-b6b1e4ec8642/.

12. Collaborative Filtering in Machine Learning, [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/collaborative-filtering-ml/.

13. A deep learning based hybrid recommendation model for internet users, [Електронний ресурс]. Режим доступу: https://www.nature.com/articles/s41598-024-79011-z.

14. Knowledge-based recommender systems: overview and research directions, [Електронний ресурс]. Режим доступу: https://www.frontiersin.org/journals/big-data/articles/10.3389/fdata.2024.1304439/full.

15. Bakumenko N., Tolstoluzka O., Yasinskyi Y. Analysis of clustering algorithms for product recommendations // Bulletin of V.N. Karazin Kharkiv National University, series «Mathematical modeling. Information technology. Automated control systems». 2024. (61). C. 6-13.

16. Як працює система рекомендацій Netflix, [Електронний ресурс]. Режим доступу: https://help.netflix.com/uk/node/100639.

17. Personalized Recommendations: How Netflix and Amazon Use Deep Learning to Enhance User Experience, [Електронний ресурс]. Режим доступу: https://medium.com/%40zhonghong9998/personalized-recommendations-how-netflix-and-amazon-use-deep-learning-to-enhance-user-experience-e7bd6fcd18ff.

18. Architectural Design - Software Engineering, [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/software-engineering-architectural-design/?utm_source=chatgpt.com.

19. Interaction Models and the Key to Creating Optimum Flow, [Електронний ресурс]. Режим доступу: https://builtin.com/articles/interaction-model.

20. The Importance of Class Diagrams in Software Development, [Електронний ресурс]. Режим доступу: https://www.geeksforgeeks.org/unified-modeling-language-uml-class-diagrams/.

21. Flowchart and Algorithm are Used for?, [Електронний ресурс]. Режим доступу: https://unacademy.com/content/question-answer/gk/flowchart-and-algorithm-are-used-for/.

22. Костюченко А. О. Основи програмування мовою Python: навчальний посібник. Ч.: ФОП Баликіна С.М., 2020. 180 с.

23. Java Introduction – [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3schools.com/java/java_intro.asp

24. А. А. Яровий, О. В. Сілагін, І. В. Богач. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 122 «Комп'ютерні науки» Вінниця : ВНТУ, 2023. 54 с.

ДОДАТОК А
ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Програмний модуль підбору кінематографічного твору на основі уподобань користувача

Тип роботи: бакалаврська кваліфікаційна робота
(бакалаврська кваліфікаційна робота / магістерська кваліфікаційна робота)

Підрозділ кафедра комп'ютерних наук
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 0,72 %

Висновок щодо перевірки кваліфікаційної роботи (відмітити потрібне)

✓ Запозичення, виявлені у роботі, є законними і не містять ознак плагіату, фабрикації, фальсифікації. Роботу прийняти до захисту

☐ У роботі не виявлено ознак плагіату, фабрикації, фальсифікації, але надмірна кількість текстових запозичень та/або наявність типових розрахунків не дозволяють прийняти рішення про оригінальність та самостійність її виконання. Роботу направити на доопрацювання.

☐ У роботі виявлено ознаки плагіату та/або текстових маніпуляцій як спроб укриття плагіату, фабрикації, фальсифікації, що суперечить вимогам законодавства та нормам академічної доброчесності. Робота до захисту не приймається.

Експертна комісія:

Андрій ЯРОВИЙ, зав. каф. КН _____ (підпис)
(прізвище, ініціали, посада)

Олег КОЛЕСНИЦЬКИЙ, проф. каф КН _____ (підпис)
(прізвище, ініціали, посада)

Особа, відповідальна за перевірку _____ Володимир ОЗЕРАНСЬКИЙ
(підпис) (прізвище, ініціали)

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Ярослав ІВАНЧУК
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Христина КУЛЬБІДА
(підпис) (прізвище, ініціали)

ДОДАТОК Б

Інструкція користувача

1. Відкрийте файл main.py у середовищі розробки (наприклад, PyCharm) або у будь-якому інтерпретаторі мови Python, який підтримує графічні інтерфейси, та запустіть його.

2. У вікні авторизації введіть бажаний логін та пароль у відповідні поля й натисніть кнопку Register. Якщо реєстрація пройшла успішно, на екрані з’явиться відповідне повідомлення.

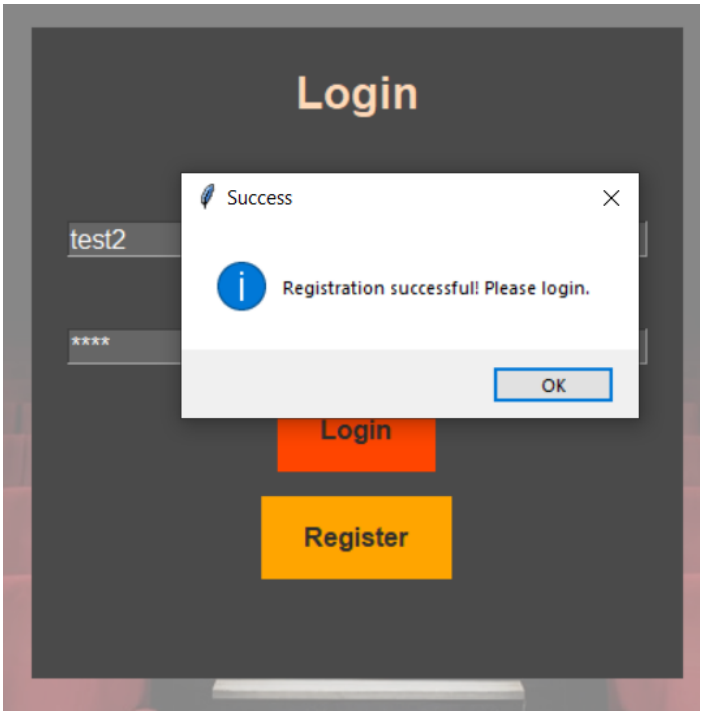

Рисунок Б.1 – Зовнішній вигляд повідомлення про успішну реєстрацію користувача в системі

3. Після реєстрації введіть ті ж логін і пароль у вікні входу та натисніть кнопку Login. У разі успішної авторизації відкриється головне вікно для налаштування параметрів рекомендацій.

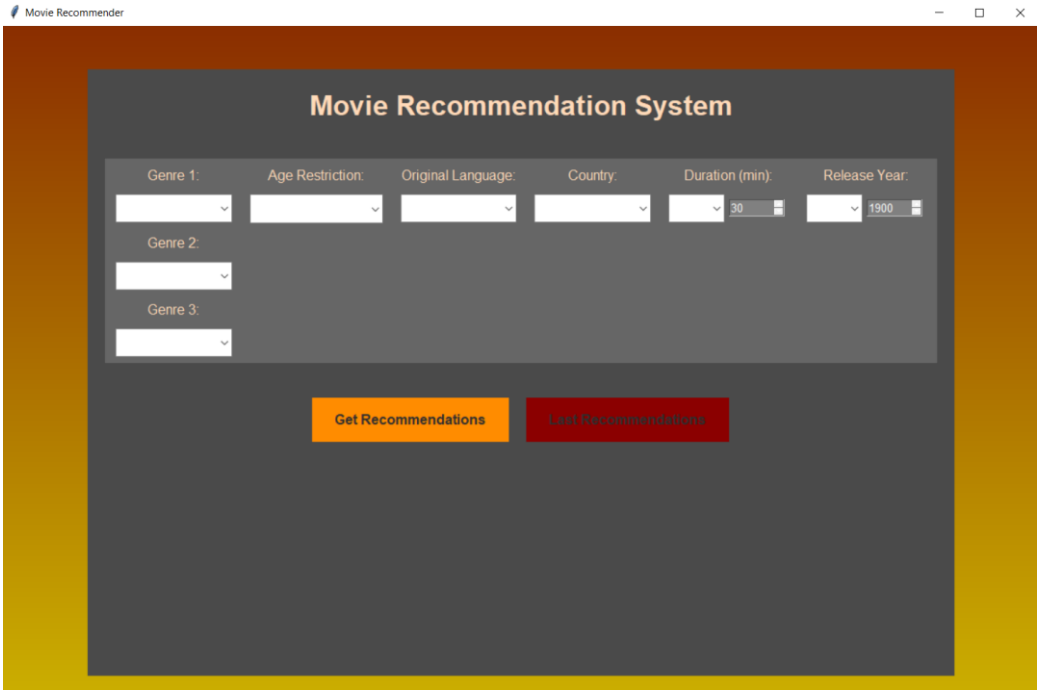

Рисунок Б.2 – Зовнішній вигляд вікна вибору параметрів для формування рекомендацій

4. Заповніть поля, що відповідають за жанри, вікові обмеження, мову, країну виробництва, тривалість, та рік виходу фільму. Зверніть увагу, що поля Duration та Release year мають логіку «більше»/«менше» – система буде шукати фільми, що задовольняють умові, обраній у відповідному полі.

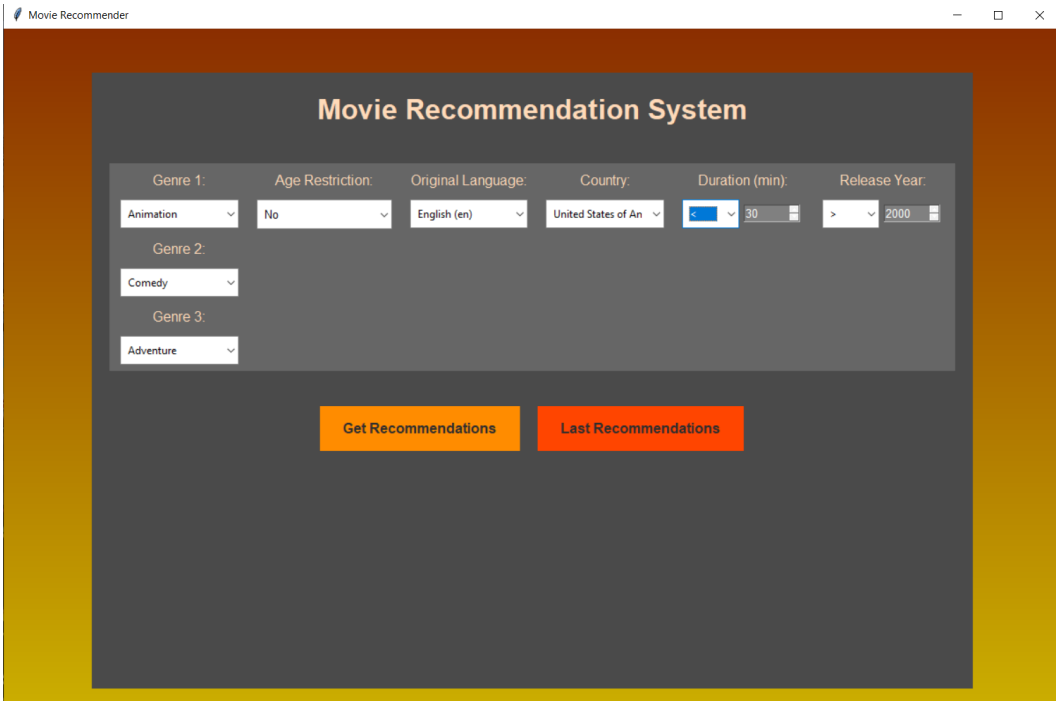

Рисунок Б.3 – Приклад заповнення даних для надання рекомендацій

5. Натисніть кнопку Get Recommendations. В результаті відкриється нове вікно зі списком фільмів, що відповідають заданим критеріям.

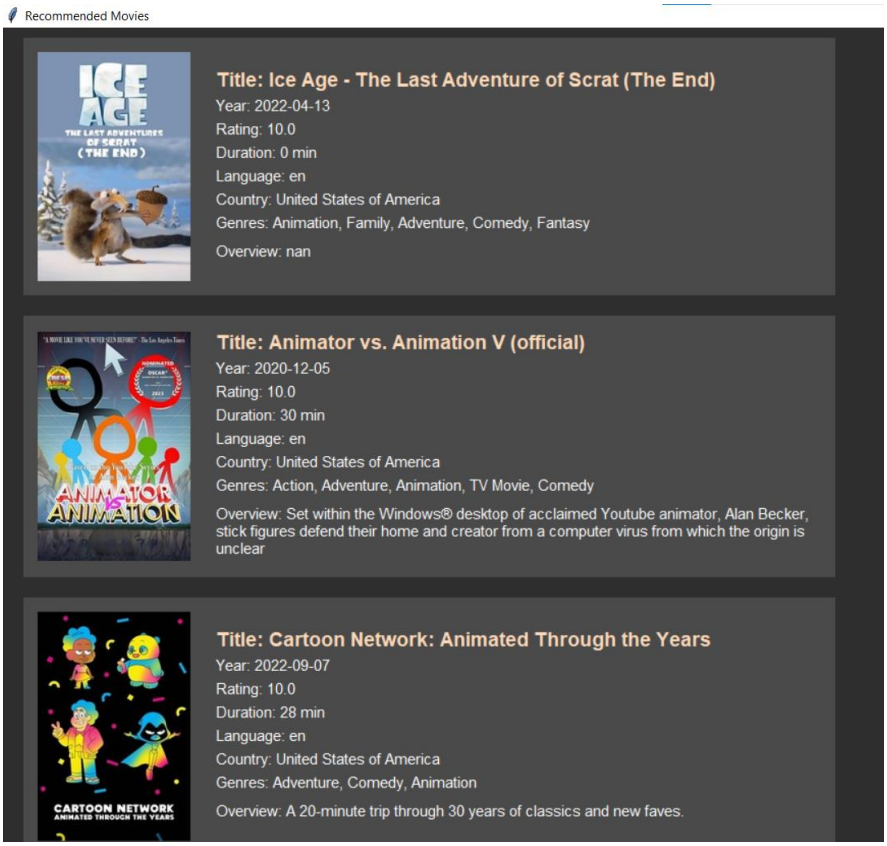

Рисунок Б.4 – Зовнішній вигляд вікна результатів після надання рекомендацій

6. У нижній частині вікна результатів з’явиться питання: "Are these recommendations suitable?"

– Якщо рекомендації вас влаштовують, натисніть Yes – після цього програма завершить свою роботу.

– Якщо рекомендації не підходять, натисніть No – відкриється уточнююче вікно.

Рисунок Б.5 – Зовнішній вигляд секції у вікні результатів із запитанням про підходящість рекомендацій

7. У діалоговому вікні буде поставлено питання: "Would you like to continue searching?"

– Якщо натиснути Yes, ви будете перенаправлені назад до вікна вибору параметрів для нового пошуку.

– Якщо натиснути No, програма завершить свою роботу.

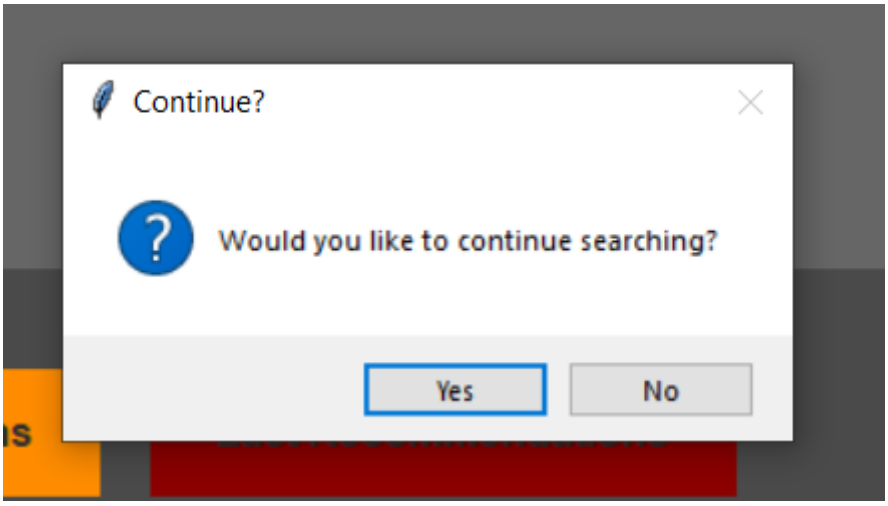

Рисунок Б.6 – Загальний вигляд діалогового вікна уточнення потреби продовження підбору кінематографічних творів

8. Для перегляду раніше отриманих результатів натисніть кнопку Останні рекомендації. Якщо рекомендації вже були отримані, вони відобразяться у новому вікні. Якщо історія ще відсутня — система повідомить про це.

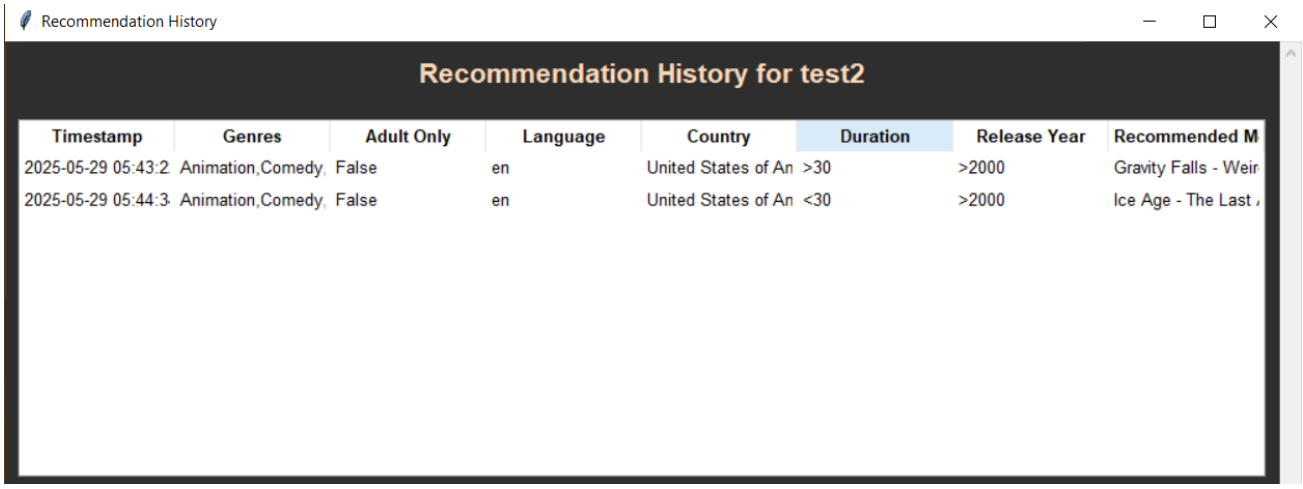

Рисунок Б.7 – Загальний вигляд вікна історії наданих рекомендацій певному користувачу

ДОДАТОК В

Лістинг програмного коду

Модуль побудови інтерфейсу користувача

```
import tkinter as tk
from tkinter import ttk, messagebox
from PIL import Image, ImageTk
import requests
from io import BytesIO
from data_loader import MovieDataset
from user_preferences import UserPreferences
from recommender import Recommender
import pandas as pd
from datetime import datetime
import os
import csv
LANGUAGE_MAP = {
    'en': 'English', 'fr': 'French', 'de': 'German', 'es': 'Spanish', 'it':
'Italian',
    'ja': 'Japanese', 'zh': 'Chinese', 'ko': 'Korean',
    'pt': 'Portuguese', 'hi': 'Hindi', 'ar': 'Arabic', 'tr': 'Turkish', 'nl':
'Dutch',
    'sv': 'Swedish', 'pl': 'Polish', 'fi': 'Finnish', 'no': 'Norwegian', 'da':
'Danish',
    'cs': 'Czech', 'el': 'Greek', 'he': 'Hebrew', 'ro': 'Romanian', 'th':
'Thai',
    'vi': 'Vietnamese', 'id': 'Indonesian', 'uk': 'Ukrainian', 'hu':
'Hungarian',
    'bg': 'Bulgarian', 'sk': 'Slovak', 'sr': 'Serbian', 'hr': 'Croatian',
    'ms': 'Malay', 'fa': 'Persian', 'ta': 'Tamil', 'bn': 'Bengali', 'ml':
'Malayalam',
    'af': 'Afrikaans', 'am': 'Amharic', 'as': 'Assamese', 'az': 'Azerbaijani',
    'be': 'Belarusian', 'bo': 'Tibetan', 'bs': 'Bosnian', 'ca': 'Catalan',
    'cy': 'Welsh', 'dz': 'Dzongkha', 'et': 'Estonian', 'eu': 'Basque',
    'ff': 'Fulah', 'ga': 'Irish', 'gl': 'Galician', 'gu': 'Gujarati',
    'ha': 'Hausa', 'is': 'Icelandic', 'iu': 'Inuktitut', 'jv': 'Javanese',
    'ka': 'Georgian', 'kk': 'Kazakh', 'km': 'Khmer', 'kn': 'Kannada',
    'ky': 'Kyrgyz', 'la': 'Latin', 'lb': 'Luxembourgish', 'lo': 'Lao',
    'lt': 'Lithuanian', 'lv': 'Latvian', 'mg': 'Malagasy', 'mi': 'Maori',
    'mk': 'Macedonian', 'mn': 'Mongolian', 'mr': 'Marathi', 'mt': 'Maltese',
    'my': 'Burmese', 'nb': 'Norwegian Bokmål', 'ne': 'Nepali', 'or': 'Odia',
    'pa': 'Punjabi', 'ps': 'Pashto', 'qu': 'Quechua', 'rw': 'Kinyarwanda',
    'sd': 'Sindhi', 'si': 'Sinhala', 'sl': 'Slovenian', 'sm': 'Samoan',
    'sn': 'Shona', 'so': 'Somali', 'sq': 'Albanian', 'sw': 'Swahili',
    'te': 'Telugu', 'tg': 'Tajik', 'tl': 'Tagalog', 'to': 'Tongan',
    'tt': 'Tatar', 'ur': 'Urdu', 'uz': 'Uzbek', 'wo': 'Wolof',
    'xh': 'Xhosa', 'yi': 'Yiddish', 'yo': 'Yoruba', 'zu': 'Zulu'
}
class MovieAppGUI:
    def __init__(self):
        self.root = tk.Tk()
        self.root.title("Movie Recommender")
        self.root.geometry("1200x800")
        self.username = None  # Store logged-in username

        self.style = ttk.Style()
        self.style.configure('TCombobox', padding=8, font=('Roboto', 10))
        self.style.configure('TButton', padding=8, font=('Roboto', 10,
'bold'))
        self.style.configure('Treeview', font=('Roboto', 10), rowheight=25)
        self.style.configure('Treeview.Heading', font=('Roboto', 10, 'bold'))
```

```
        self.bg_image = None
        try:
            response = requests.get(
                "https://images.unsplash.com/photo-1489599849927-
2ee91cede3ba?auto=format&fit=crop&w=1200")
            img = Image.open(BytesIO(response.content))
            img = img.resize((1200, 800), Image.Resampling.LANCZOS)
            img = img.convert('RGBA')
            alpha = Image.new('RGBA', img.size, (255, 255, 255, 100))  #
Softer opacity
            img = Image.blend(img, alpha, 0.4)
            self.bg_image = ImageTk.PhotoImage(img)
        except:
            pass

        self.show_login_screen()

    def show_login_screen(self):
        self.login_frame = tk.Frame(self.root, bg='#2E2E2E')
        self.login_frame.place(relwidth=1, relheight=1)

        canvas = tk.Canvas(self.login_frame, highlightthickness=0)
        canvas.place(relwidth=1, relheight=1)
        if self.bg_image:
            canvas.create_image(0, 0, image=self.bg_image, anchor='nw')

        login_box = tk.Frame(self.login_frame, bg='#4A4A4A', bd=2,
relief='flat')
        login_box.place(relx=0.5, rely=0.5, anchor='center', width=400,
height=400)

        tk.Label(login_box, text="Login", font=('Roboto', 20, 'bold'),
bg='#4A4A4A', fg='#FFDAB9').pack(pady=20)

        tk.Label(login_box, text="Username:", bg='#4A4A4A', fg='#FFFFFF',
font=('Roboto', 12)).pack(pady=5)
        self.username_entry = tk.Entry(login_box, font=('Roboto', 12),
bg='#666666', fg='#FFFFFF', insertbackground='#FFFFFF')
        self.username_entry.pack(pady=5, padx=20, fill='x')

        tk.Label(login_box, text="Password:", bg='#4A4A4A', fg='#FFFFFF',
font=('Roboto', 12)).pack(pady=5)
        self.password_entry = tk.Entry(login_box, show='*', font=('Roboto',
12), bg='#666666', fg='#FFFFFF', insertbackground='#FFFFFF')
        self.password_entry.pack(pady=5, padx=20, fill='x')

        tk.Button(login_box, text="Login", command=self.login, bg='#FF4500',
fg='#2E2E2E',
                  font=('Roboto', 12, 'bold'), relief='flat', padx=20,
pady=10).pack(pady=10)
        tk.Button(login_box, text="Register", command=self.register,
bg='#FFA500', fg='#2E2E2E',
                  font=('Roboto', 12, 'bold'), relief='flat', padx=20,
pady=10).pack(pady=5)

    def login(self):
        username = self.username_entry.get().strip()
        password = self.password_entry.get().strip()

        if not username or not password:
            messagebox.showerror("Error", "Please enter both username and
password")
            return
```

```
        try:
            users_df = pd.read_csv('users.csv') if os.path.exists('users.csv')
else pd.DataFrame(columns=['username', 'password'])
            user_match = users_df[(users_df['username'] == username) &
(users_df['password'] == password)]
            if not user_match.empty:
                self.username = username
                self.login_frame.destroy()
                self.initialize_app()
            else:
                messagebox.showerror("Error", "Invalid username or password")
        except Exception as e:
            messagebox.showerror("Error", f"Login failed: {str(e)}")

    def register(self):
        username = self.username_entry.get().strip()
        password = self.password_entry.get().strip()

        if not username or not password:
            messagebox.showerror("Error", "Please enter both username and
password")
            return

        try:
            users_df = pd.read_csv('users.csv') if os.path.exists('users.csv')
else pd.DataFrame(columns=['username', 'password'])
            if username in users_df['username'].values:
                messagebox.showerror("Error", "Username already exists")
                return
            new_user = pd.DataFrame([[username, password]],
columns=['username', 'password'])
            users_df = pd.concat([users_df, new_user], ignore_index=True)
            users_df.to_csv('users.csv', index=False)
            messagebox.showinfo("Success", "Registration successful! Please
login.")
        except Exception as e:
            messagebox.showerror("Error", f"Registration failed: {str(e)}")

    def initialize_app(self):
        self.dataset = MovieDataset("movies.csv")
        self.preferences = UserPreferences()
        self.recommender = Recommender(self.dataset)
        self.build_interface()

    def build_interface(self):
        bg_canvas = tk.Canvas(self.root, highlightthickness=0, bg='#2E2E2E')
        bg_canvas.pack(fill='both', expand=True)
        if self.bg_image:
            bg_canvas.create_image(0, 0, image=self.bg_image, anchor='nw')

        # Create a subtle gradient using shades of gray and orange
        for i in range(256):
            g = 46 + (i // 2)  # Gray from #2E2E2E to #5E5E5E
            r = 139 + (i // 4)  # Red from #8B0000 to #FF4500 range
            color = f'#{r:02x}{g:02x}00'  # Red-gray gradient
            bg_canvas.create_line(0, i * 3, 1200, i * 3, fill=color, width=3)

        main_frame = tk.Frame(bg_canvas, bg='#4A4A4A')
        main_frame.place(relx=0.5, rely=0.5, anchor='center', width=1000,
height=700)

        title = tk.Label(main_frame, text="Movie Recommendation System",
                         font=('Roboto', 24, 'bold'), bg='#4A4A4A',
```

```
fg='#FFDAB9')
        title.pack(pady=20)

        self.filters_frame = tk.Frame(main_frame, bg='#666666', bd=2,
relief='flat')
        self.filters_frame.pack(pady=20, padx=20, fill='x')

        self.filters_frame.columnconfigure((0, 1, 2, 3, 4, 5), weight=1)

        tk.Label(self.filters_frame, text="Genre 1:", bg='#666666',
fg='#FFDAB9',
                 font=('Roboto', 12)).grid(row=0, column=0, pady=5, padx=5)
        self.genre_combo1 = ttk.Combobox(self.filters_frame, values=[""] +
list(self.dataset.get_unique_genres()),
                                         style='TCombobox')
        self.genre_combo1.grid(row=1, column=0, padx=10, pady=5, sticky='ew')

        tk.Label(self.filters_frame, text="Genre 2:", bg='#666666',
fg='#FFDAB9',
                 font=('Roboto', 12)).grid(row=2, column=0, pady=5, padx=5)
        self.genre_combo2 = ttk.Combobox(self.filters_frame, values=[""] +
list(self.dataset.get_unique_genres()),
                                         style='TCombobox')
        self.genre_combo2.grid(row=3, column=0, padx=10, pady=5, sticky='ew')

        tk.Label(self.filters_frame, text="Genre 3:", bg='#666666',
fg='#FFDAB9',
                 font=('Roboto', 12)).grid(row=4, column=0, pady=5, padx=5)
        self.genre_combo3 = ttk.Combobox(self.filters_frame, values=[""] +
list(self.dataset.get_unique_genres()),
                                         style='TCombobox')
        self.genre_combo3.grid(row=5, column=0, padx=10, pady=5, sticky='ew')

        tk.Label(self.filters_frame, text="Age Restriction:", bg='#666666',
fg='#FFDAB9',
                 font=('Roboto', 12)).grid(row=0, column=1, pady=5, padx=5)
        self.age_limit = ttk.Combobox(self.filters_frame, values=["Yes",
"No"],
                                      style='TCombobox', font=('Roboto', 10))
        self.age_limit.grid(row=1, column=1, padx=10, pady=5, sticky='ew')

        tk.Label(self.filters_frame, text="Original Language:", bg='#666666',
fg='#FFDAB9',
                 font=('Roboto', 12)).grid(row=0, column=2, pady=5, padx=5)
        self.lang_combo = ttk.Combobox(self.filters_frame,
                                       values=[f"{LANGUAGE_MAP.get(code,
code)} ({code})"
                                               for code in
self.dataset.get_unique_languages()],
                                       style='TCombobox')
        self.lang_combo.grid(row=1, column=2, padx=10, pady=5, sticky='ew')

        tk.Label(self.filters_frame, text="Country:", bg='#666666',
fg='#FFDAB9',
                 font=('Roboto', 12)).grid(row=0, column=3, pady=5, padx=5)
        self.country_combo = ttk.Combobox(self.filters_frame,

values=self.dataset.get_unique_countries(),
                                          style='TCombobox')
        self.country_combo.grid(row=1, column=3, padx=10, pady=5, sticky='ew')

        tk.Label(self.filters_frame, text="Duration (min):", bg='#666666',
fg='#FFDAB9',
                 font=('Roboto', 12)).grid(row=0, column=4, pady=5, padx=5)
```

```
        duration_frame = tk.Frame(self.filters_frame, bg='#666666')
        duration_frame.grid(row=1, column=4, padx=10, pady=5, sticky='ew')
        self.duration_sign = ttk.Combobox(duration_frame, values=[">", "<"],
                                          width=5, state='readonly',
style='TCombobox')
        self.duration_sign.pack(side='left')
        self.duration_spin = tk.Spinbox(duration_frame, from_=30, to=300,
                                        width=10, bg='#808080', fg='#FFFFFF',
                                        font=('Roboto', 10),
insertbackground='#FFFFFF')
        self.duration_spin.pack(side='left', padx=5)

        tk.Label(self.filters_frame, text="Release Year:", bg='#666666',
fg='#FFDAB9',
                 font=('Roboto', 12)).grid(row=0, column=5, pady=5, padx=5)
        year_frame = tk.Frame(self.filters_frame, bg='#666666')
        year_frame.grid(row=1, column=5, padx=10, pady=5, sticky='ew')
        self.year_sign = ttk.Combobox(year_frame, values=[">", "<"],
                                      width=5, state='readonly',
style='TCombobox')
        self.year_sign.pack(side='left')
        self.year_spin = tk.Spinbox(year_frame, from_=1900, to=2025,
                                    width=10, bg='#808080', fg='#FFFFFF',
                                    font=('Roboto', 10),
insertbackground='#FFFFFF')
        self.year_spin.pack(side='left', padx=5)

        button_frame = tk.Frame(main_frame, bg='#4A4A4A')
        button_frame.pack(pady=20)
        submit_button = tk.Button(button_frame, text="Get Recommendations",
                                  command=self.get_recommendations,
                                  bg='#FF8C00', fg='#2E2E2E',
                                  font=('Roboto', 12, 'bold'),
                                  relief='flat', padx=20, pady=10)
        submit_button.pack(side='left', padx=10)
        submit_button.bind('<Enter>', lambda e:
submit_button.config(bg='#FFA500'))
        submit_button.bind('<Leave>', lambda e:
submit_button.config(bg='#FF8C00'))

        history_button = tk.Button(button_frame, text="Last Recommendations",
                                   command=self.show_recommendation_history,
                                   bg='#8B0000', fg='#2E2E2E',
                                   font=('Roboto', 12, 'bold'),
                                   relief='flat', padx=20, pady=10)
        history_button.pack(side='left', padx=10)
        history_button.bind('<Enter>', lambda e:
history_button.config(bg='#FF4500'))
        history_button.bind('<Leave>', lambda e:
history_button.config(bg='#8B0000'))

    def get_recommendations(self):
        genres = []
        for combo in [self.genre_combo1, self.genre_combo2,
self.genre_combo3]:
            genre = combo.get()
            if genre and genre != "":
                genres.append(genre)
        self.preferences.genres = genres

        self.preferences.adult_only = False if self.age_limit.get() == "No"
else True
        selected_lang = self.lang_combo.get()
        self.preferences.language = selected_lang.split("(")[-1].replace(")",
```

```
"") if "(" in selected_lang else selected_lang
        self.preferences.country = self.country_combo.get()

        try:
            duration_value = self.duration_spin.get()
            if duration_value.strip():
                self.preferences.duration_value = int(duration_value)
                self.preferences.duration_sign = self.duration_sign.get() if
self.duration_sign.get() else None
            else:
                self.preferences.duration_value = None
                self.preferences.duration_sign = None

            release_year = self.year_spin.get()
            if release_year.strip():
                self.preferences.release_year = int(release_year)
                self.preferences.release_sign = self.year_sign.get() if
self.year_sign.get() else None
            else:
                self.preferences.release_year = None
                self.preferences.release_sign = None
        except ValueError:
            messagebox.showerror("Error", "Invalid duration or year entered")
            return

        if (self.preferences.release_year is not None and
                self.preferences.release_sign not in [">", "<"]):
            messagebox.showerror("Error", "Please select a valid year
comparison operator (> or <)")
            return

        if (self.preferences.duration_value is not None and
                self.preferences.duration_sign not in [">", "<"]):
            messagebox.showerror("Error", "Please select a valid duration
comparison operator (> or <)")
            return

        results = self.recommender.recommend(self.preferences)
        self.log_recommendations(genres, self.preferences, results)
        self.show_results_page(results)

    def log_recommendations(self, genres, preferences, movies):
        """Log recommendation details to a CSV file with proper quoting."""
        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        input_data = {
            'username': self.username,
            'timestamp': timestamp,
            'genres': ','.join(genres) if genres else 'None',
            'adult_only': preferences.adult_only,
            'language': preferences.language if preferences.language else
'None',
            'country': preferences.country if preferences.country else 'None',
            'duration':
f"{preferences.duration_sign}{preferences.duration_value}" if
preferences.duration_value and preferences.duration_sign else 'None',
            'release_year':
f"{preferences.release_sign}{preferences.release_year}" if
preferences.release_year and preferences.release_sign else 'None',
            'recommended_movies': ','.join(movies['title'].tolist()) if not
movies.empty else 'None'
        }

        try:
            with open('recommendation_log.csv', mode='a', newline='',
```

```
encoding='utf-8') as f:
                writer = csv.DictWriter(f, fieldnames=input_data.keys(),
quoting=csv.QUOTE_ALL)
                if not os.path.exists('recommendation_log.csv'):
                    writer.writeheader()
                writer.writerow(input_data)
        except Exception as e:
            messagebox.showerror("Error", f"Failed to save recommendation log:
{str(e)}")

    def show_results_page(self, movies):
        results_window = tk.Toplevel(bg='#2E2E2E')
        results_window.title("Recommended Movies")
        results_window.geometry("1000x700")

        canvas = tk.Canvas(results_window, bg='#2E2E2E', highlightthickness=0)
        scrollbar = tk.Scrollbar(results_window, orient="vertical",
command=canvas.yview)
        scroll_frame = tk.Frame(canvas, bg='#2E2E2E')

        scroll_frame.bind("<Configure>", lambda e:
canvas.configure(scrollregion=canvas.bbox("all")))
        canvas.create_window((0, 0), window=scroll_frame, anchor='nw')
        canvas.configure(yscrollcommand=scrollbar.set)

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

        if movies.empty:
            tk.Label(scroll_frame, text="No movies found matching your
criteria",
                     bg='#2E2E2E', fg='#FFDAB9', font=('Roboto', 14,
'bold')).pack(pady=20)
        else:
            for _, movie in movies.iterrows():
                frame = tk.Frame(scroll_frame, bg='#4A4A4A', bd=2,
relief='flat')
                frame.pack(fill='x', pady=10, padx=20)

                img_url =
f"https://image.tmdb.org/t/p/w200{movie['poster_path']}"
                try:
                    response = requests.get(img_url)
                    img = Image.open(BytesIO(response.content))
                    img = img.resize((150, 225), Image.Resampling.LANCZOS)
                    photo = ImageTk.PhotoImage(img)
                    label_img = tk.Label(frame, image=photo, bg='#4A4A4A')
                    label_img.image = photo
                    label_img.pack(side='left', padx=10, pady=10)
                except:
                    pass

                info_frame = tk.Frame(frame, bg='#4A4A4A')
                info_frame.pack(side='left', padx=10, pady=10)

                tk.Label(info_frame, text=f"Title: {movie['title']}",
                         font=('Roboto', 14, 'bold'), bg='#4A4A4A',
fg='#FFDAB9').pack(anchor='w')
                tk.Label(info_frame, text=f"Year: {movie['release_date']}",
                         bg='#4A4A4A', fg='#FFFFFF', font=('Roboto',
11)).pack(anchor='w')
                tk.Label(info_frame, text=f"Rating: {movie['vote_average']}",
                         bg='#4A4A4A', fg='#FFFFFF', font=('Roboto',
11)).pack(anchor='w')
```

```
                tk.Label(info_frame, text=f"Duration: {movie['runtime']} min",
                         bg='#4A4A4A', fg='#FFFFFF', font=('Roboto',
11)).pack(anchor='w')
                tk.Label(info_frame, text=f"Language:
{movie['original_language']}",
                         bg='#4A4A4A', fg='#FFFFFF', font=('Roboto',
11)).pack(anchor='w')
                tk.Label(info_frame, text=f"Country: {',
'.join(movie['production_countries'])}",
                         bg='#4A4A4A', fg='#FFFFFF', font=('Roboto',
11)).pack(anchor='w')
                tk.Label(info_frame, text=f"Genres: {',
'.join(movie['genres'])}",
                         bg='#4A4A4A', fg='#FFFFFF', font=('Roboto',
11)).pack(anchor='w')
                tk.Label(info_frame, text=f"Overview: {movie['overview']}",
                         wraplength=600, justify='left', bg='#4A4A4A',
fg='#FFFFFF',
                         font=('Roboto', 11)).pack(anchor='w', pady=5)

        tk.Label(scroll_frame, text="Are these recommendations suitable?",
                 font=('Roboto', 14, 'bold'), bg='#2E2E2E', fg='#FFDAB9',
pady=20).pack()

        buttons_frame = tk.Frame(scroll_frame, bg='#2E2E2E')
        buttons_frame.pack(pady=20)

        tk.Button(buttons_frame, text="Yes", command=self.root.destroy,
                  bg='#FF8C00', fg='#2E2E2E', font=('Roboto', 12, 'bold'),
                  relief='flat', padx=20, pady=10).pack(side='left', padx=10)
        tk.Button(buttons_frame, text="No", command=self.ask_continue,
                  bg='#8B0000', fg='#2E2E2E', font=('Roboto', 12, 'bold'),
                  relief='flat', padx=20, pady=10).pack(side='left', padx=10)

    def show_recommendation_history(self):
        history_window = tk.Toplevel(bg='#2E2E2E')
        history_window.title("Recommendation History")
        history_window.geometry("1000x600")

        canvas = tk.Canvas(history_window, bg='#2E2E2E', highlightthickness=0)
        scrollbar = tk.Scrollbar(history_window, orient="vertical",
command=canvas.yview)
        scroll_frame = tk.Frame(canvas, bg='#2E2E2E')

        scroll_frame.bind("<Configure>", lambda e:
canvas.configure(scrollregion=canvas.bbox("all")))
        canvas.create_window((0, 0), window=scroll_frame, anchor='nw')
        canvas.configure(yscrollcommand=scrollbar.set)

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

        tk.Label(scroll_frame, text=f"Recommendation History for
{self.username}",
                 font=('Roboto', 16, 'bold'), bg='#2E2E2E',
fg='#FFDAB9').pack(pady=10)

        try:
            log_df = pd.read_csv('recommendation_log.csv',
quoting=csv.QUOTE_ALL,
                                 on_bad_lines='skip') if
os.path.exists('recommendation_log.csv') else pd.DataFrame()
            if not log_df.empty:
                print(f"Loaded DataFrame columns: {log_df.columns.tolist()}")
```

```
# Debug print
                print(f"Raw data:\n{log_df}")  # Debug print
                user_logs = log_df[log_df['username'] == self.username]
                if user_logs.empty:
                    tk.Label(scroll_frame, text="No recommendations found for
this user",
                             bg='#2E2E2E', fg='#FFFFFF', font=('Roboto',
12)).pack(pady=20)
                else:
                    columns = ['timestamp', 'genres', 'adult_only',
'language', 'country', 'duration', 'release_year',
                               'recommended_movies']
                    tree = ttk.Treeview(scroll_frame, columns=columns,
show='headings', style='Treeview')
                    for col in columns:
                        tree.heading(col, text=col.replace('_', ' ').title())
                        tree.column(col, width=120, anchor='w')
                    tree.pack(fill='both', expand=True, padx=10, pady=10)

                    for _, row in user_logs.iterrows():
                        values = [row.get(col, 'N/A') for col in columns]  #
Handle missing columns
                        tree.insert('', 'end', values=values)
            else:
                tk.Label(scroll_frame, text="No recommendation history
available",
                         bg='#2E2E2E', fg='#FFFFFF', font=('Roboto',
12)).pack(pady=20)
        except Exception as e:
            tk.Label(scroll_frame, text=f"Error loading history: {str(e)}",
                     bg='#2E2E2E', fg='#FF4500', font=('Roboto',
12)).pack(pady=20)

    def ask_continue(self):
        response = messagebox.askyesno("Continue?", "Would you like to
continue searching?")
        if response:
            self.reset_interface()
        else:
            self.root.destroy()

    def reset_interface(self):
        self.preferences = UserPreferences()
        self.genre_combo1.set("")
        self.genre_combo2.set("")
        self.genre_combo3.set("")
        self.age_limit.set("")
        self.lang_combo.set("")
        self.country_combo.set("")
        self.duration_sign.set("")
        self.duration_spin.delete(0, tk.END)
        self.duration_spin.insert(0, "90")
        self.year_sign.set("")
        self.year_spin.delete(0, tk.END)
        self.year_spin.insert(0, "2000")

    def run(self):
        self.root.mainloop()

if __name__ == "__main__":
    app = MovieAppGUI()
    app.run()
```

Модуль завантаження та обробки даних

```
import pandas as pd
class MovieDataset:
    def __init__(self, csv_path):
        self.df = pd.read_csv(csv_path)
        self._preprocess()

    def _preprocess(self):
        self.df.dropna(subset=['genres', 'runtime', 'original_language',
'production_countries', 'poster_path'], inplace=True)
        self.df['genres'] = self.df['genres'].str.split(', ')
        self.df['production_countries'] =
self.df['production_countries'].str.split(', ')

    def get_unique_genres(self):
        genres = set()
        for genre_list in self.df['genres']:
            genres.update(genre_list)
        return sorted(list(genres))

    def get_unique_languages(self):
        return sorted(self.df['original_language'].unique())

    def get_unique_countries(self):
        countries = set()
        for country_list in self.df['production_countries']:
            countries.update(country_list)
        return sorted(list(countries))

    def get_filtered_movies(self, preferences):
        df_filtered = self.df.copy()

        if preferences.genres:
            df_filtered = df_filtered[df_filtered['genres'].apply(lambda x:
all(genre in x for genre in preferences.genres))]

        if preferences.adult_only is not None:
            df_filtered = df_filtered[df_filtered['adult'] ==
preferences.adult_only]

        if preferences.language and preferences.language != 'Не важливо':
            df_filtered = df_filtered[df_filtered['original_language'] ==
preferences.language]

        if preferences.country and preferences.country != 'Не важливо':
            df_filtered =
df_filtered[df_filtered['production_countries'].apply(lambda x:
preferences.country in x)]

        if preferences.duration_sign and preferences.duration_value is not
None:
            if preferences.duration_sign == '<':
                df_filtered = df_filtered[df_filtered['runtime'] <=
preferences.duration_value]
            elif preferences.duration_sign == '>':
                df_filtered = df_filtered[df_filtered['runtime'] >=
preferences.duration_value]

        if preferences.release_sign and preferences.release_year is not None:
            df_filtered['movie_year'] =
pd.to_datetime(df_filtered['release_date'], errors='coerce').dt.year
            df_filtered = df_filtered.dropna(subset=['movie_year'])  # Drop
invalid dates
```

```
            df_filtered['movie_year'] = df_filtered['movie_year'].astype(int)
            if preferences.release_sign == '<':
                df_filtered = df_filtered[df_filtered['movie_year'] <=
preferences.release_year]
            elif preferences.release_sign == '>':
                df_filtered = df_filtered[df_filtered['movie_year'] >=
preferences.release_year]
        return df_filtered.sort_values(by='vote_average',
ascending=False).head(5)
```

Модуль переваг користувача

```
class UserPreferences:
    def __init__(self):
        self.genres = []
        self.adult_only = None
        self.language = None
        self.country = None
        self.duration_sign = None
        self.duration_value = None
        self.release_year = None
        self.release_sign = None
```

Модуль рекомендацій

```
class Recommender:
    def __init__(self, dataset):
        self.dataset = dataset

    def recommend(self, preferences):
        return self.dataset.get_filtered_movies(preferences)
```

ДОДАТОК Г

ІЛЮСТРАТИВНА ЧАСТИНА

«Програмний модуль підбору кінематографічного твору на основі уподобань користувача»

Виконала: студентка 4 курсу, групи 2КН-21б
Спеціальності 122 – Комп'ютерні науки
(шифр і назва напрямку підготовки, спеціальності)
Христина КУЛЬБІДА
(прізвище та ініціали)
Керівник: д.т.н. проф. каф. КН
Ярослав ІВАНЧУК
(прізвище та ініціали)

«03» червня 2025 р.

Вінниця ВНТУ – 2025 рік

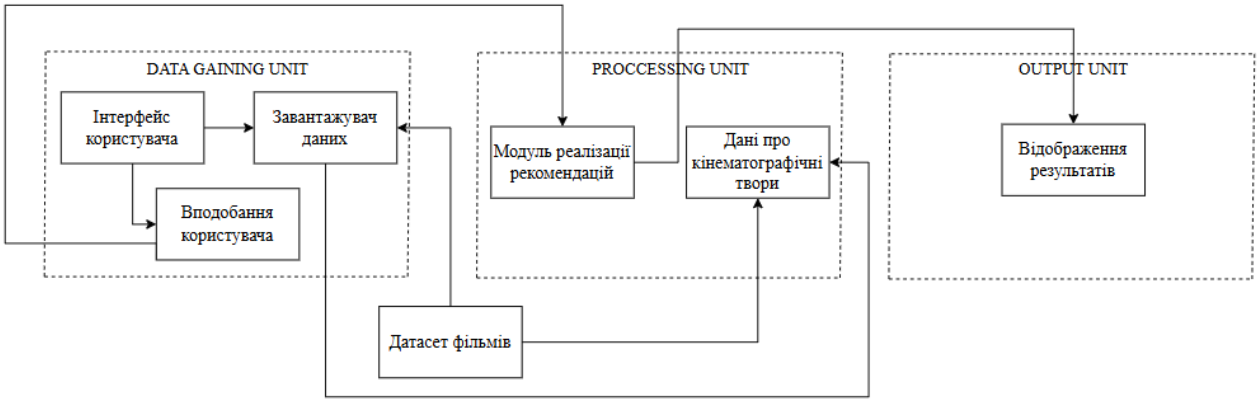

Рисунок Г.1 – Архітектура програмного модуля підбору кінематографічного твору на основі вподобань користувача

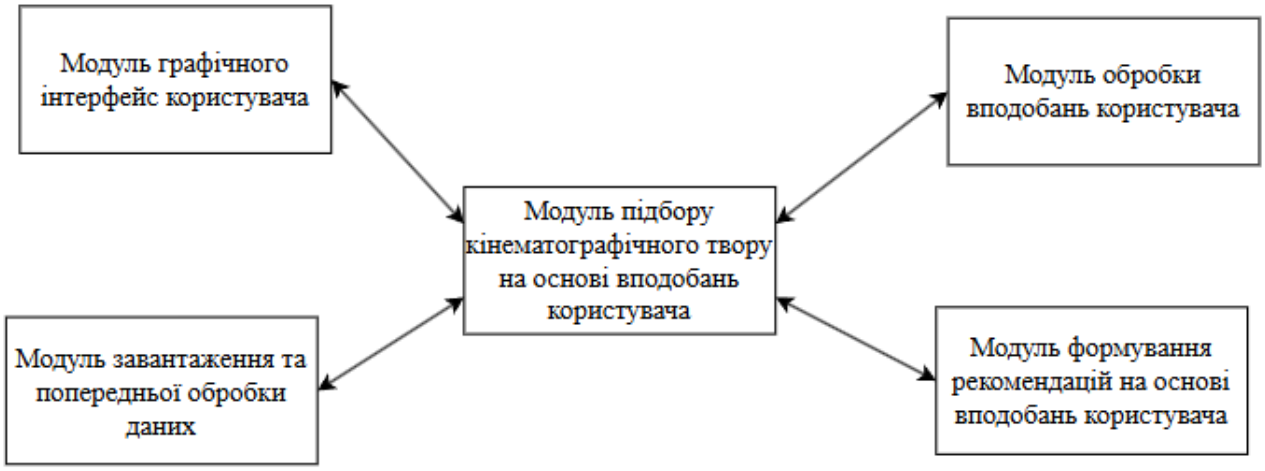

Рисунок Г.2 – Структурна схема програмного модуля підбору кінематографічного твору на основі вподобань користувача

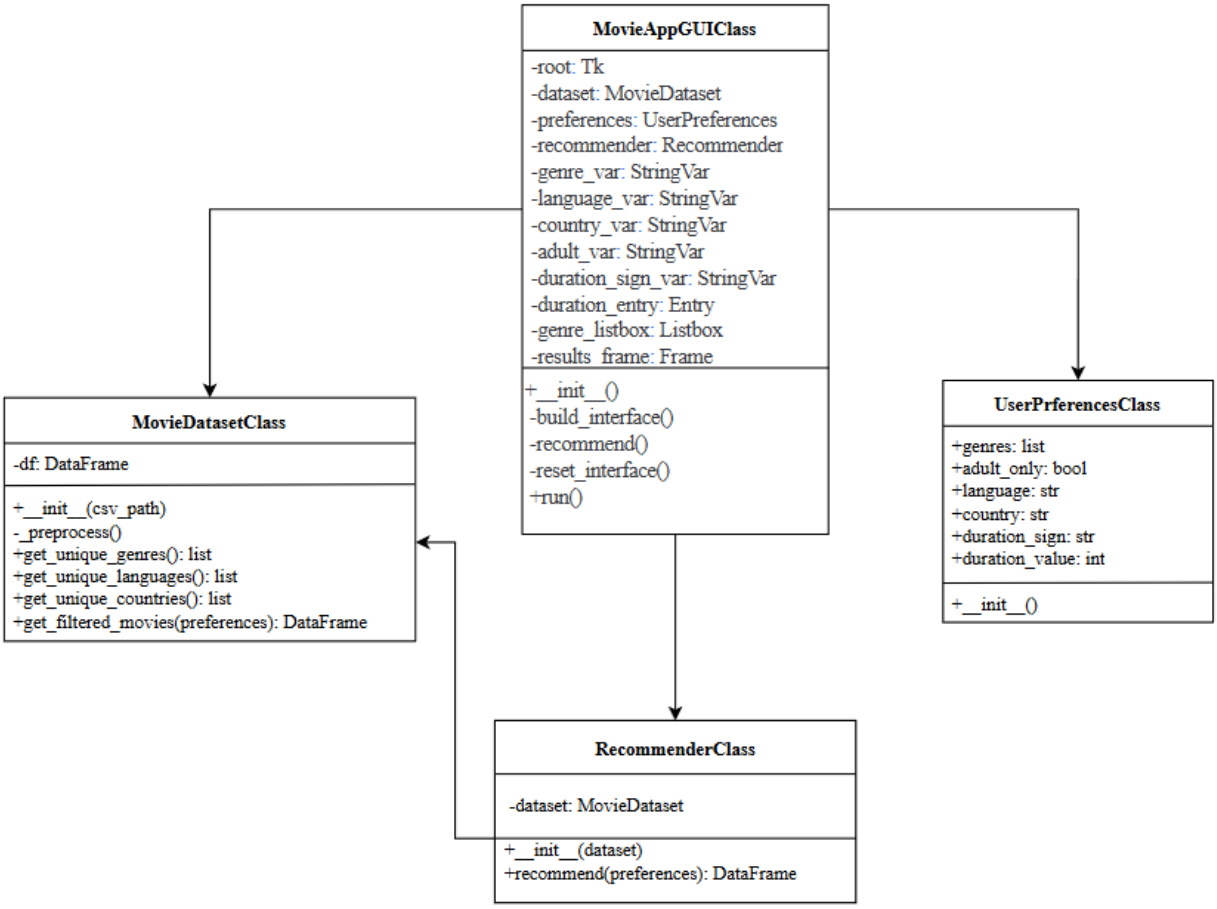

Рисунок Г.3 – UML-діаграма класів програмного модулю підбору кінематографічного твору на онові вподобань користувача

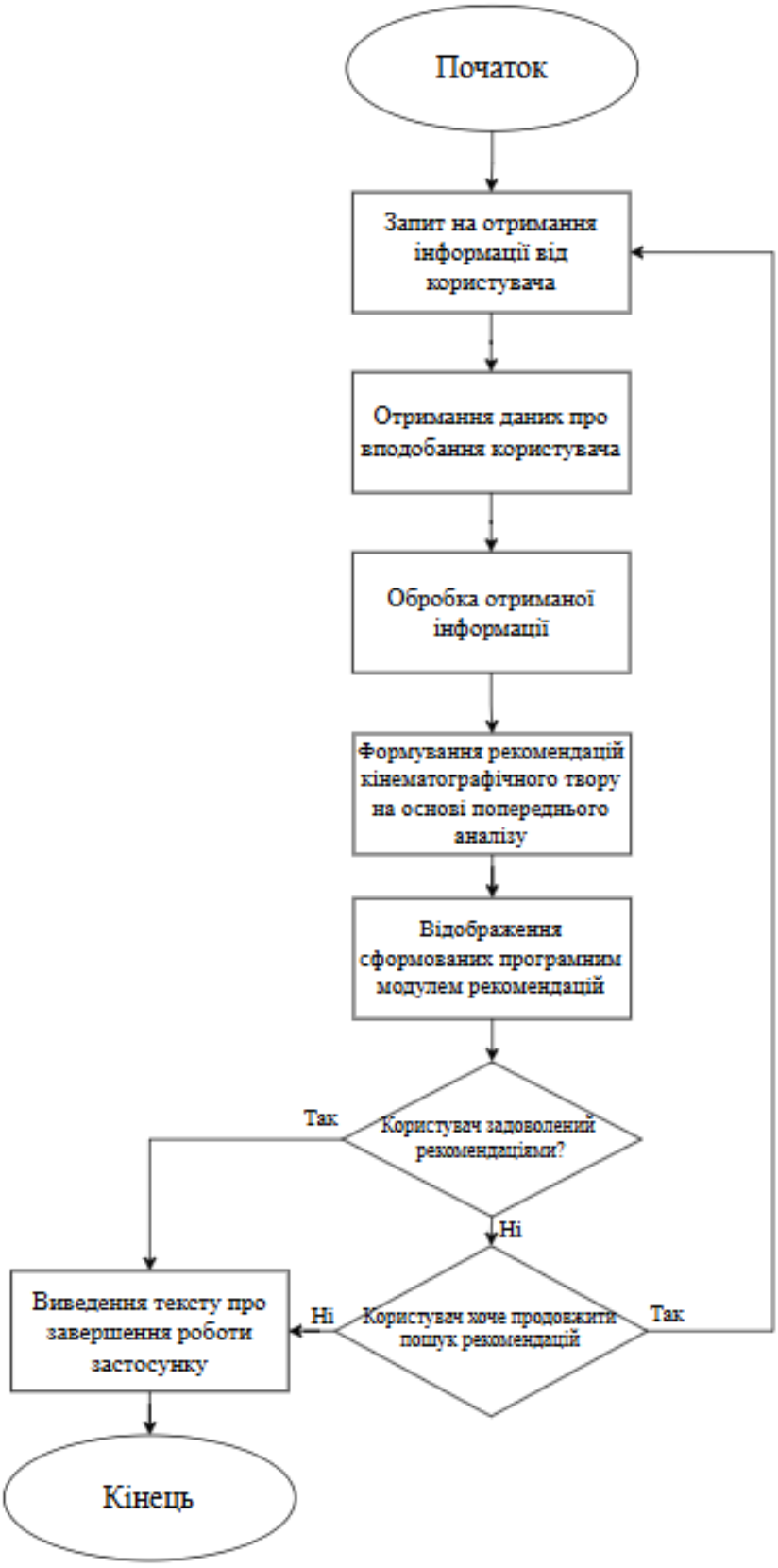

Рисунок Г.4 – Схема загального алгоритму роботи програмного модулю підбору кінематографічного твору на основі вподобань користувача

Рисунок Г.5 – Схема алгоритму роботи MovieAppGUI модуля

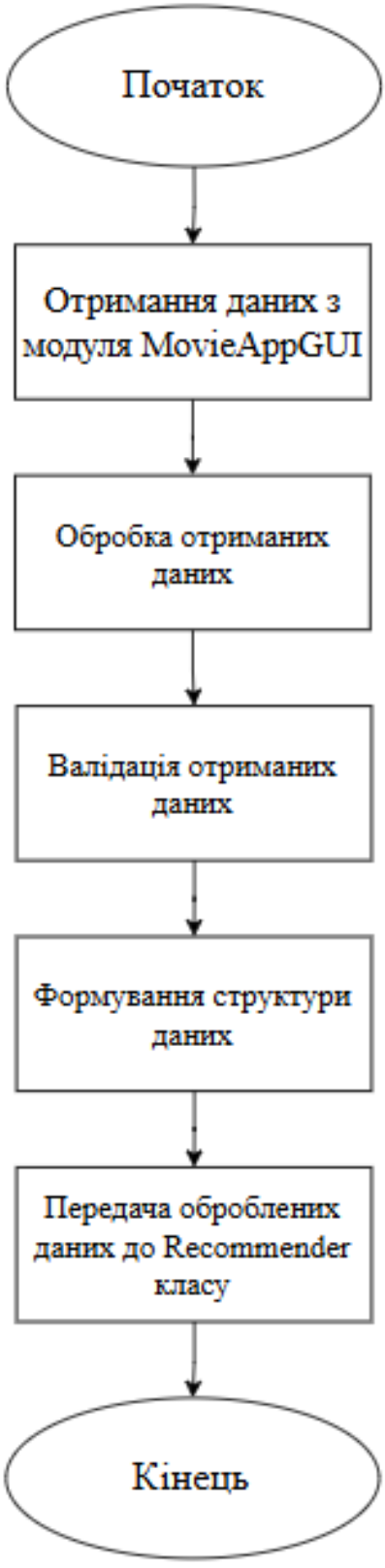

Рисунок Г.6 – Схема алгоритму роботи UserPreferences модуля

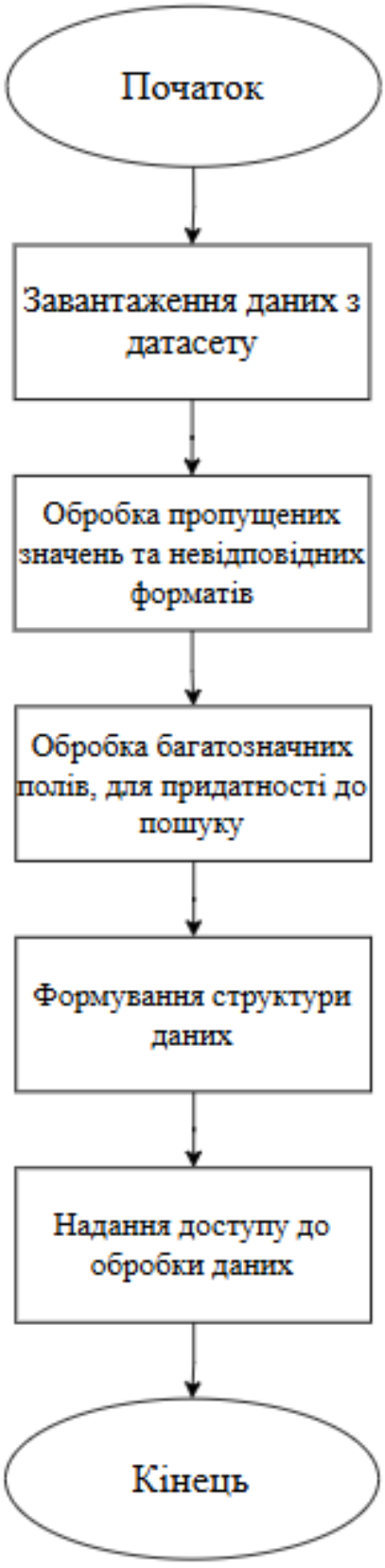

Рисунок Г.7 – Схема алгоритму роботи MovieDataset модуля

Рисунок Г.8 – Схема алгоритму роботи MovieAppGUI модуля

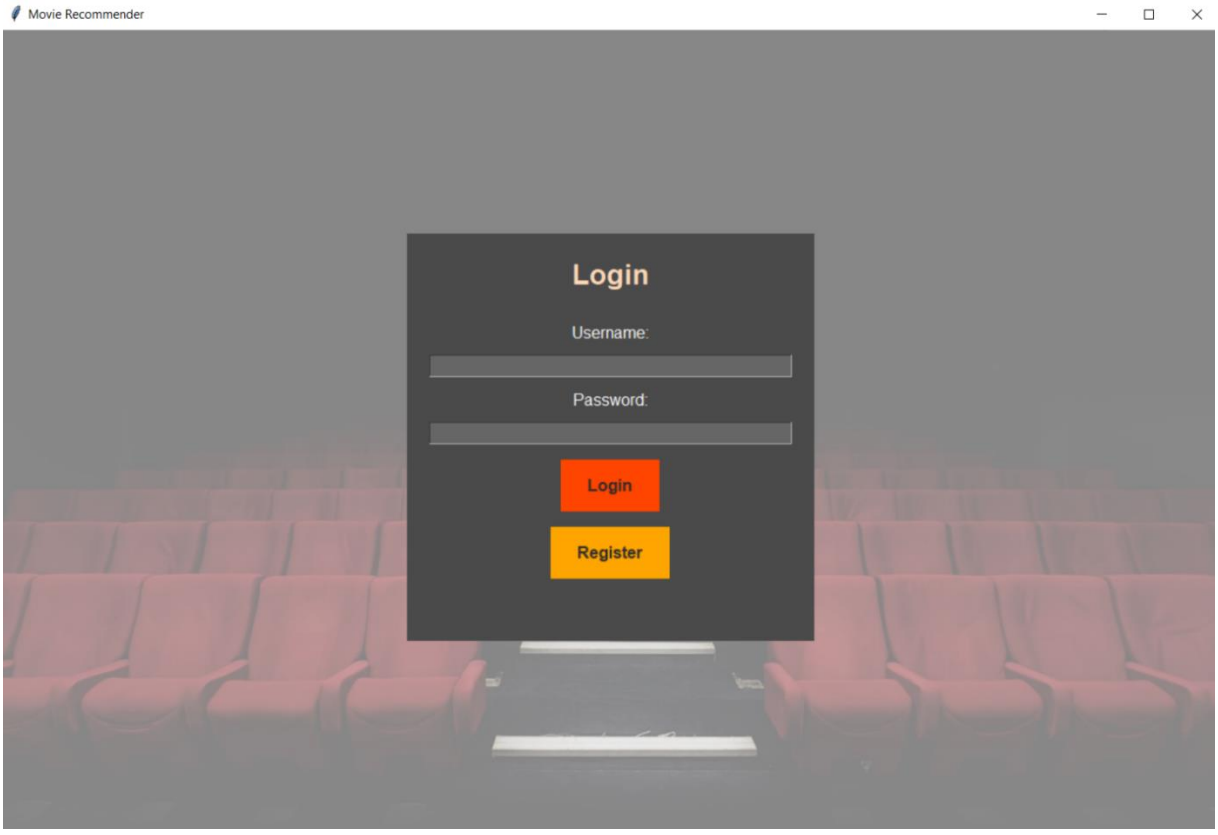

Рисунок Г.9 – Загальний вигляд вікна форми авторизації в систему

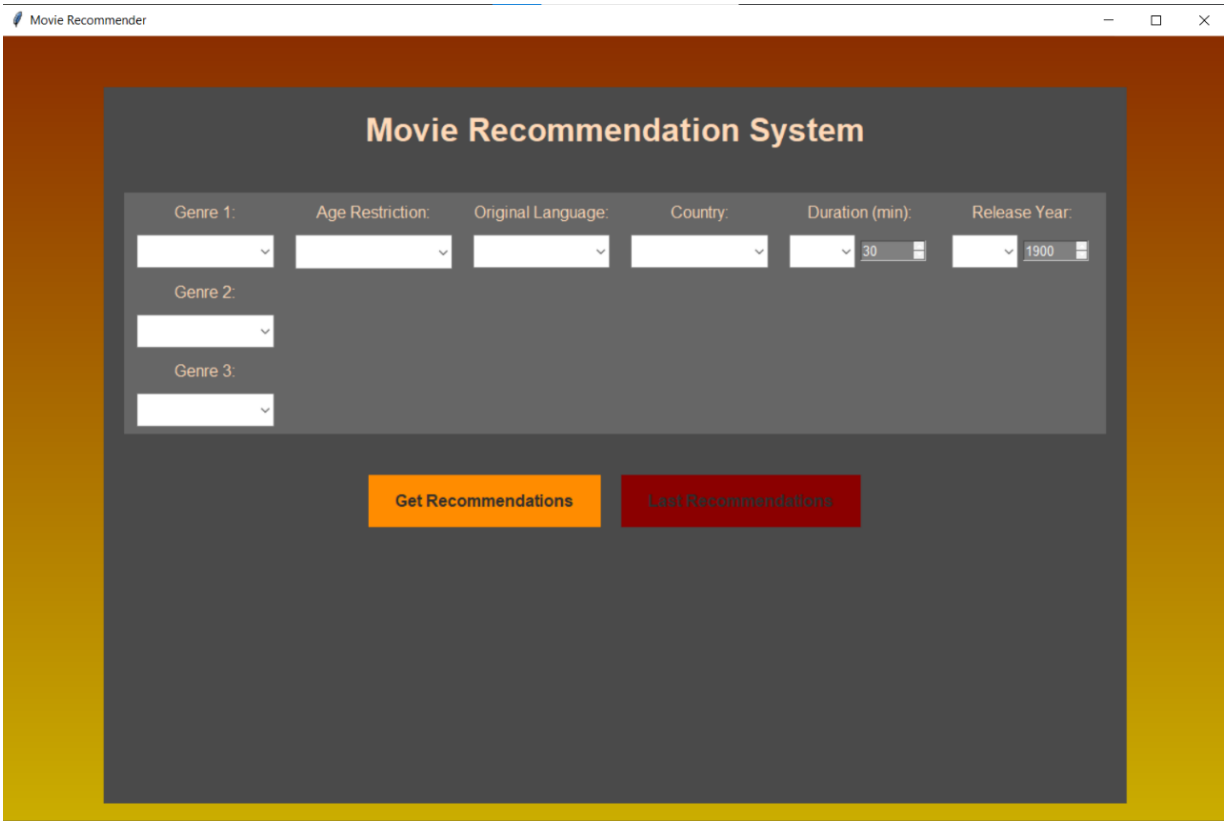

Рисунок Г.10 – Загальний вигляд вікна вибору параметрів для формування рекомендацій

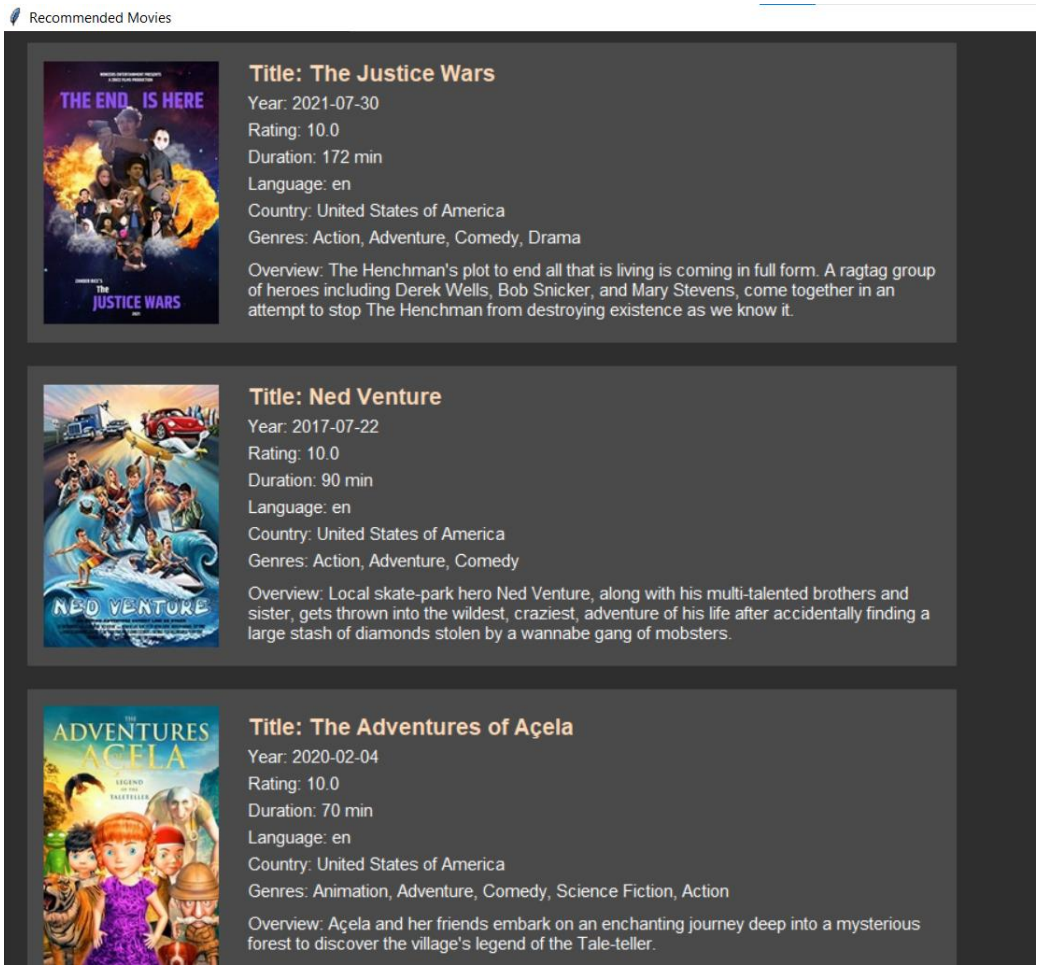

Рисунок Г.11 – Загальний вигляд вікна результатів після надання рекомендацій

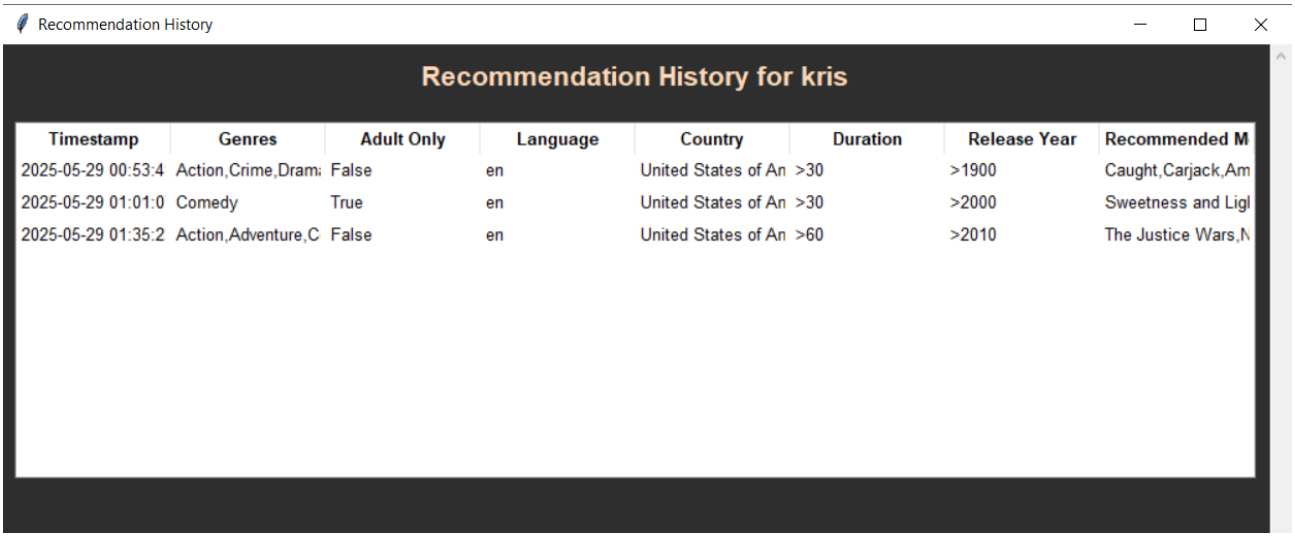

Рисунок Г.12 – Загальний вигляд вікна історії наданих рекомендацій певному користувачу