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

Бакалаврська кваліфікаційна робота

на тему: «Розробка застосунку для обміну та прокату вінілових платівок з інтеграцією Spotify для персоналізованих рекомендацій»

Виконав: студент 4 курсу
групи 2ПІ-21Б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Євсович А.В.
(прізвище та ініціали)
Керівник: д.т.н., проф. каф. ПЗ Ліщинська Л.Б.
(прізвище та ініціали)
Рецензент: д.т.н., зав.каф. ЗІ, проф. Лужецький В.А.
(прізвище та ініціали)

Допущено до захисту

Зав. кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«09» 06 2025 р.

ВНТУ – 2025

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

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
О. Н. Романюк
« 21 » березня 2025 р.

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

Євсович Алевтині Вікторівні

1. Тема роботи – розробка застосунку для обміну та прокату вінілових платівок з інтеграцією Spotify для персоналізованих рекоментацій.

Керівник роботи: Ліщинська Людмила Броніславівна, д.т.н., професор кафедри ПЗ, затверджені наказом вищого навчального закладу від «20» березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025
3. Вхідні дані до роботи: дані користувача – унікальний ідентифікатор, ім'я, прізвище, електронна адреса, номер телефону, хешований пароль, токени доступу до Spotify, профільне фото, топ-50 треків, історія переглядів та взаємодій із платівками; метадані вінілових платівок – назва альбому, виконавець, рік випуску, стан платівки, тип видання, статус доступності; транзакційні дані – ідентифікатори платівок і користувачів, дати прокату, вартість, статус операції, повідомлення між користувачами з шифруванням; дані Spotify – аудіофічі треків (danceability, energy, valence, acousticness, instrumentalness, tempo), історія прослуховувань, вподобані треки, плейлисти, зіставлення треків із фізичними платівками; дані для гібридного алгоритму рекомендацій – ознаки для контентної фільтрації, матриця колаборативної фільтрації, параметри доменного компонента, фінальна рекомендаційна оцінка.
4. Зміст текстової частини: аналіз сучасного стану цифрових платформ для обміну та прокату вінілових платівок; дослідження можливостей інтеграції персоналізованих музичних рекомендацій на основі вподобань користувача; порівняльний аналіз існуючих програмних рішень; формалізація задачі та постановка вимог до розроблюваного програмного забезпечення; дослідження структури вхідних даних; побудова концептуальної та функціональної моделей системи; розробка алгоритмів основних бізнес-процесів; вибір інструментальних засобів і середовища

реалізації; створення програмних модулів авторизації, пошуку платівок та взаємодії з музичними сервісами; розробка інтерфейсу користувача з урахуванням зручності та інтуїтивності використання; тестування функціональних компонентів застосунку; перевірка системи на стійкість до збоїв; аналіз результатів тестування; підготовка інструкції користувача.

5. Перелік ілюстративного матеріалу: інтерфейс застосунку для обміну віниловими платівками; мета та завдання дослідження; новизна і практична цінність роботи; порівняння існуючих аналогів; блок-схема роботи системи; модуль авторизації; модуль реєстрації; модуль персоналізованих рекомендацій; головний екран застосунку; додаткові вікна.
6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Ліщинська Л.Б. д.т.н., професор кафедри ПЗ	24.03.25	01.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз цифрових платформ обміну та прокату вінілових платівок	25.03.25–05.04.25	Вик.
2	Дослідження методів персоналізованих рекомендацій музичного контенту	06.04.2025–15.04.2025	Вик.
3	Побудова моделей функціонування системи оренди та обміну платівок	16.04.2025–23.04.2025	Вик.
4	Розробка модулів авторизації, пошуку та рекомендацій із використанням Spotify AP	23.04.2025–10.05.2025	Вик.
5	Тестування програмного забезпечення та перевірка функціональності	08.05.2025–20.05.2025	Вик.
6	Оформлення матеріалів до захисту БКР	18.05.2025–30.05.2025	Вик.

Студент _____ **А.В. Євсеє**
(підпис) (ініціали та прізв

Керівник бакалаврської кваліфікаційної роботи _____ **Л.Б. Ліщинс**
(підпис) (ініціали та прізв

АНОТАЦІЯ

УБД. 004

Євсович А.В. Розробка застосунку для обміну та прокату вінілових платівок з інтеграцією Spotify для персоналізованих рекомендацій. бакалаврська кваліфікаційна робота зі спеціальності 121 – інженерія програмного забезпечення, освітня програма – інженерія програмного забезпечення. Вінниця: ВНТУ, 2025. 60 с. На укр. мові. Бібліогр. : 25 назв; рис. :25; табл. : 5.

У бакалаврській кваліфікаційній роботі проведено детальний аналіз технологій створення застосунків для обміну та прокату фізичних музичних носіїв. Запропоновано архітектуру клієнт-серверного застосунку, що дозволяє користувачам здійснювати обмін та оренду вінілових платівок із можливістю персоналізованих рекомендацій на основі музичних вподобань зі Spotify. Реалізовано гібридний алгоритм побудови рекомендацій, з урахуванням джерел релевантності. Розроблено клієнтську частину застосунку мовою Python із використанням бібліотеки CustomTkinter для побудови графічного інтерфейсу, а серверну частину – з використанням мови JavaScript та середовища Node.js. Отримані результати можуть бути використані при створенні систем цифрової підтримки фізичних форматів музики в умовах розвитку стримінгових сервісів.

Ключові слова: обмін платівками, прокат, Spotify API, веб-застосунок, рекомендаційні системи, музика, гібридна модель оцінки, рекомендаційна система, скоринг цінності.

ABSTARCT

Yevsovych A.V. *Development of an Application for Vinyl Record Exchange and Rental with Spotify Integration for Personalized Recommendations.* Bachelor's qualification work in specialty 121 – Software Engineering, educational program – Software Engineering. Vinnytsia: VNTU, 2025. 60 p. In Ukrainian. References: 25 sources; figures: 25; tables: 5.

The bachelor's thesis presents a detailed analysis of technologies for developing applications for the exchange and rental of physical music media. A client-server architecture is proposed, enabling users to exchange and rent vinyl records with personalized recommendations based on musical preferences from Spotify. A hybrid recommendation algorithm has been implemented, taking into account multiple sources of relevance. The client-side application was developed in Python using the CustomTkinter library for building the graphical user interface, while the server-side was implemented in JavaScript using the Node.js environment. The results obtained can be applied to the development of digital support systems for physical music formats in the context of the growing influence of streaming services.

Keywords: record sharing, rental, Spotify API, web application, recommendation systems, music, hybrid evaluation model, recommendation system, value scoring.

ЗМІСТ

ВСТУП 3
1 АНАЛІЗ СУЧАСНОГО СТАНУ ПРОКАТУ ВІНІЛОВИХ ПЛАТІВОК ТА ПОСТАВКА ЗАДАЧІ 7
1.1 Аналіз стану технологій обміну та прокату вінілових платівок з інтеграцією персоналізованих рекомендацій 7
1.2 Порівняльний аналіз аналогів 9
1.3 Аналіз методів розв’язання задачі інтеграції персоналізованих рекомендацій 14
1.4 Функціональні та нефункціональні вимоги 17
1.5 Висновки 20
2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ СИСТЕМИ ОБМІНУ ТА ПРОКАТУ ВІНІЛОВИХ ПЛАТІВОК З ІНТЕГРАЦІЄЮ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ 21
2.1 Аналіз вхідних даних системи 21
2.2 Розробка моделі системи 23
2.3 Розробка методу та алгоритму персоналізованих рекомендацій з використанням гібридного алгоритму 26
2.4 Висновок 30
3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ РЕАЛІЗАЦІЇ ОБМІНУ ТА ОРЕНДИ ВІНІЛОВИХ ПЛАТІВОК 31
3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення 31
3.2 Розробка модуля авторизації 34
3.3 Розробка модулю рекомендацій 37
3.4 Розробка інтерфейсу програмного застосунку 41
3.4 Висновки 48
4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ 49
4.1 Тестування системи 49
4.3 Тестування алгоритму рекомендацій 55
4.4 Розробка інструкції користувача 58
4.3 Висновки 60
ВИСНОВКИ 62
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 64
ДОДАТКИ 66
Додаток А – Технічне завдання ..... **Error! Bookmark not defined.**
Додаток Б – Протокол перевірки бакалаврської кваліфікаційної роботи на наявність текстових запозичень ..... **Error! Bookmark not defined.**
Додаток В – Лістинг програми 71
Додаток Г – Ілюстративна частина 102

ВСТУП

Обґрунтування вибору теми дослідження. Сучасна тенденція до зростання інтересу до аналогових музичних носіїв, зокрема вінілових платівок, супроводжується розвитком нішевих онлайн-спільнот і платформ, орієнтованих на купівлю, продаж або колекціонування таких носіїв. Водночас, попит на інші форми взаємодії, як-от обмін та прокат, залишається поза фокусом більшості цифрових рішень. Сервіси, що надають доступ до вінілових платівок, здебільшого не включають розвиненої логіки персоналізованих рекомендацій, що обмежує можливості ефективного підбору контенту відповідно до індивідуальних уподобань користувачів.

Типовим прикладом є платформа Discogs, яка має потужний інструментарій для каталогізації, покупки та продажу платівок, однак не реалізує механізми рекомендацій або орендної моделі [1]. Сервіси на кшталт VNYL застосовують фіксовані підписки для отримання музичних платівок поштою, але система добору базується на ручній модерації або спрощених користувацьких анкетах [2]. Такий підхід не дозволяє гнучко адаптуватися до змін музичних уподобань користувача й не забезпечує високого рівня індивідуалізації.

Останні роки відзначаються зростанням інтересу до застосування API цифрових музичних платформ, зокрема Spotify, у системах рекомендацій [3,4]. Значна кількість досліджень присвячена контентним, колаборативним та гібридним методам побудови рекомендаційних моделей [5,6]. Проте більшість з них орієнтовані на цифровий контент (стрімінгові сервіси, плейлисти), що не враховує фізичні обмеження, колекційну цінність та стан матеріального носія. Як зазначається у [7], рекомендаційні системи для фізичних об’єктів вимагають специфічних моделей відповідності між цифровим профілем і релевантністю фізичного контенту, проте подібні підходи перебувають на початковому етапі розробки.

Додатково, поточні реалізації не забезпечують динамічну адаптацію до змін користувацьких уподобань, яка б базувалась на аналізі аудіофіч, емоційного тону чи структури плейлистів. Згідно з [8], одним із викликів сучасних рекомендаційних систем є створення механізмів, здатних враховувати як поведінкові патерни, так і семантику контенту в реальному часі. Проте у випадку фізичних носіїв, таких як вінілові платівки, ці аспекти залишаються недостатньо дослідженими. Аналогічно, вказано [9, 10] на обмежену кількість досліджень, які б поєднували характеристики медіаоб'єкта з динамічним цифровим профілем для підвищення якості персоналізованих рекомендацій у контексті матеріальних ресурсів.

Тому актуальним є питання створення ефективних моделей персоналізованих рекомендацій для сервісів обміну та прокату вінілових платівок, які б враховували як цифрові вподобання користувачів на основі даних Spotify, так і специфіку фізичних носіїв. Вирішення цієї задачі дозволить суттєво покращити індивідуалізацію рекомендацій, сприяти розвитку екосистеми обміну аналоговим контентом і забезпечити високий рівень релевантності пропозицій у подібних сервісах.

Мета та завдання дослідження. Підвищення ефективності персоналізованого добору аналогового музичного контенту шляхом розробки гібридного алгоритму рекомендацій та програмного засобу для сервісу обміну й прокату вінілових платівок із використанням цифрових даних користувача з платформи Spotify.

Задачі дослідження:

– провести аналіз стану розвитку сервісів обміну та прокату вінілових платівок із персоналізованими рекомендаціями;

– дослідити існуючі аналоги та здійснити їх функціонально-порівняльний аналіз;

– проаналізувати існуючі методи побудови персональних рекомендацій;

– сформувати функціональні та нефункціональні вимоги до системи;

– розробити модель системи з урахуванням структури вхідних даних;

– розробити гібридний алгоритм персоналізованих рекомендацій із використанням даних Spotify, поведінкової схожості та колекційної цінності;

– обґрунтувати вибір технологій для реалізації програмного забезпечення;

– реалізувати програмні модулі авторизації, рекомендацій та користувацького інтерфейсу;

– провести тестування системи та алгоритмів рекомендацій;

– розробити інструкцію користувача

Об’єкт дослідження. Об’єктом дослідження є процес обміну та прокату вінілових платівок, створення персоналізованих гібридних рекомендацій.

Предмет дослідження. Предметом дослідження є методи і засоби процесу обміну та прокату вінілових платівок та гібридних рекомендацій.

Методи дослідження. У роботі використано системний аналіз для формування вимог до рекомендаційної системи; аналіз аналогів і порівняльний аналіз для виявлення обмежень існуючих підходів; математичне моделювання для побудови гібридної функції релевантності; програмну реалізацію на основі Python, JavaScript та Spotify API; об’єктно-орієнтоване проєктування для створення архітектури модуля; моделювання користувацьких сценаріїв для перевірки працездатності алгоритму.

Наукова новизна отриманих результатів. Запропоновано удосконалений метод персоналізованих рекомендацій, особливістю якого є поєднання контентної подібності на основі аудіофіч, поведінкової подібності користувачів та оцінки колекційної цінності фізичних носіїв, що дозволило підвищити точність відповідності рекомендацій індивідуальним музичним уподобанням згідно з метриками Precision@k та NDCG.

Практичне значення одержаних результатів. Практична цінність одержаних результатів полягає в тому, що на основі розробленого методу вдалося досягти вищої релевантності добору уподобань користувачів та адаптацію рекомендацій до умов обігу фізичних носіїв. Отримані результати можуть бути використані в інформаційних системах з персоналізованими рекомендаціями для роботи з медіаконтентом у фізичному форматі.

Особистий внесок здобувача. Всі результати, представлені у бакалаврській кваліфікаційній роботі, отримані автором особисто.

Апробаці матеріалів бакалаврської кваліфікаційної роботи. Матеріали доповідались на конфіренції «Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025)», Вінниця 2025.

Публікації. Результати дослідження були опубліковані в матеріалах конфреренції Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (2025) [11].

1 АНАЛІЗ СУЧАСНОГО СТАНУ ПРОКАТУ ВІНІЛОВИХ ПЛАТІВОК ТА ПОСТАВКА ЗАДАЧІ

1.1 Аналіз стану технологій обміну та прокату вінілових платівок з інтеграцією персоналізованих рекомендацій

На тлі зростання популярності аналогових аудіоформатів, зокрема вінілових платівок, спостерігається відродження інтересу до фізичних носіїв як об’єкта естетичного, колекційного й аудіофільського значення. Згідно зі статистичними даними, за останні п’ять років продажі вінілових платівок у США та Європі демонструють стабільне зростання, досягаючи обсягів, співставних із показниками кінця XX століття. Це явище супроводжується розвитком ринку платформ, що спеціалізуються на торгівлі, колекціонуванні, а також каталогізації аналогових носіїв. Серед них найвідомішим прикладом є сервіс Discogs, що виконує функцію централізованої бази даних фізичних носіїв музики з підтримкою торгівлі між користувачами. Разом із тим у межах подібних платформ відсутня функціональність прокату чи структурованого обміну платівками, а також не реалізовано механізмів персоналізованого добору контенту.

Сервіси, що реалізують концепцію фізичного доступу до музики у форматі підписки, як-от VNYL, пропонують користувачам отримувати набір вінілових платівок відповідно до попередньо заповненої анкети або короткого музичного опису. Проте ці підходи переважно ґрунтуються на ручному формуванні добірок, що обмежує масштабованість, індивідуалізацію й об’єктивність рекомендованого вмісту. У той же час цифрові сервіси, зокрема Spotify, Apple Music, YouTube Music, використовують складні системи персоналізованих рекомендацій, які спираються на гібридні моделі обробки даних, що поєднують контентну фільтрацію, колаборативне навчання та облік контексту. Однак ці рішення здебільшого застосовуються виключно до цифрового стрімінгового контенту, що не має фізичної репрезентації.

У науковій літературі розглядається велика кількість підходів до побудови рекомендаційних систем. Колаборативні методи фільтрації на основі поведінкових подібностей між користувачами або елементами контенту є одними з найпоширеніших. Зокрема, детально описано матричну факторизацію, моделі глибокого навчання та графові представлення, які дозволяють покращити точність передбачення інтересів користувачів у великих інформаційних системах [11]. Контентно-орієнтовані підходи, з іншого боку, аналізують внутрішні характеристики об’єктів, як-от жанр, виконавець, тональність, ритміка, що дозволяє формувати релевантні добірки на основі схожості з уже прослуханим. Інтеграція таких методів відбувається завдяки використанню API стрімінгових сервісів, серед яких Spotify API є одним із найзручніших для отримання розширених аудіофіч, зокрема таких параметрів, як energy, danceability, valence, acousticness, speechiness, які суттєво покращують якість подібності між треками. Водночас адаптація цих методів до фізичних носіїв викликає значні труднощі, пов’язані з відсутністю прямого зв’язку між цифровими аудіофічами та об’єктами обігу на фізичних платформах.

Одним із перспективних напрямів є побудова гібридних рекомендаційних систем, які поєднують кілька незалежних джерел релевантності. У межах таких систем важливим є врахування колекційної цінності фізичних носіїв як додаткового фактору, що має індивідуалізовану значущість для користувача. Проте в наукових джерелах останніх років [12] відсутні повноцінні моделі, які інтегрують параметри рідкісності, стану носія, історії володіння або популярності серед аудіофільських спільнот у рамках персоналізованих систем добору. Також не розглядається проблема зміни профілю користувача в часі в контексті доступу до фізичних об’єктів, тоді як цифрові рекомендаційні системи дедалі активніше впроваджують динамічне оновлення переваг на основі прослуховування в реальному часі.

Окрему проблему становить недостатній рівень масштабованості існуючих моделей рекомендацій у сервісах, які мали б обслуговувати фізичну інфраструктуру обміну або прокату. На відміну від цифрових рішень, де

об’єкти доступу є копіями з необмеженою кількістю одночасних користувачів, фізичний носій існує в одному екземплярі, що накладає обмеження на формування черг, пріоритетів і логістику передачі. У межах відомих досліджень [13] не приділяється належної уваги специфіці роботи з об’єктами, що мають обмежену доступність, змінюють власника, а також потребують контролю термінів оренди чи обміну. Розробка адаптованих моделей рекомендацій із підтримкою фізичного ресурсу залишається відкритим завданням.

Інтеграція інформації з цифрового профілю користувача, зокрема його поточних вподобань, історії прослуховування, активності в плейлистах, а також соціальної взаємодії, із параметрами фізичних носіїв – це складна міждисциплінарна задача, що вимагає залучення методів обробки природної мови, машинного навчання та теорії корисності. Наявні реалізації рекомендаційних сервісів зосереджуються переважно на цифровому середовищі, і лише поодинокі експериментальні платформи.

Тому актуальним є питання розробки застосунку, який поєднує функціональність сервісу обміну та прокату фізичних носіїв із повноцінною персоналізованою системою рекомендацій, адаптованою до особливостей взаємодії користувача з аналоговим контентом. Таке рішення повинно враховувати цифровий профіль слухача, індивідуальні музичні уподобання, соціальні патерни поведінки та значущість фізичних характеристик вінілових платівок, що дозволить створити ефективну модель добору об’єктів з урахуванням як суб’єктивних переваг, так і матеріальних обмежень обігу.

1.2 Порівняльний аналіз аналогів

Для створення ефективної системи обміну та прокату вінілових платівок з персоналізованими рекомендаціями необхідним є аналіз існуючих програмних рішень із подібною або дотичною функціональністю. Основною метою такого аналізу є виявлення сильних і слабких сторін конкурентних рішень, які можуть бути враховані при реалізації власної розробки.

Оцінювання аналогів здійснюється за такими критеріями: наявність функціональності реєстрації та авторизації користувачів, можливість перегляду й каталогізації вінілових платівок, підтримка інтерактивної взаємодії між користувачами, функціонал замовлень або прокату, система персоналізованих рекомендацій, а також розширені можливості керування власним профілем і вбудоване повідомлення між користувачами.

Discogs (рис. 1.1) є однією з найвідоміших онлайн-платформ, орієнтованих на музичні релізи, зокрема вінілові платівки [1]. Основне призначення системи полягає в забезпеченні користувачів інструментами для каталогізації, купівлі та продажу музичних носіїв.

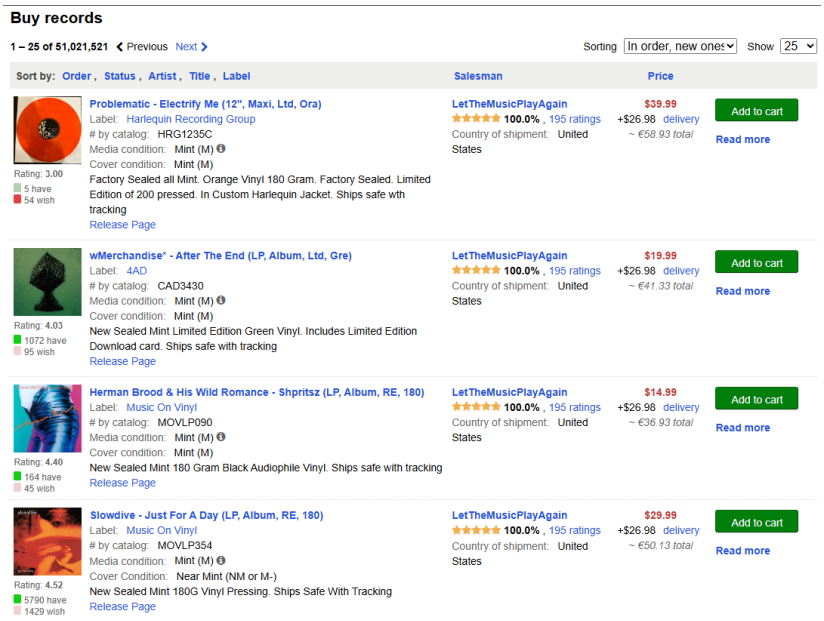

Рисунок 1.1 – Функціональна складова платформи Discogs

Платформа підтримує реєстрацію та авторизацію, надає детальну інформацію про платівки, включаючи метадані, зображення, треклисти, дати релізів та інші атрибути. Вбудований пошук дозволяє здійснювати фільтрацію за жанром, виконавцем або лейблом. Проте відсутній функціонал для організації прокату або обміну платівками між користувачами, що зменшує інтерактивну складову сервісу. Крім того, система не підтримує персоналізовану видачу рекомендацій та не містить засобів для прямої

взаємодії між користувачами, зокрема обміну повідомленнями. Таким чином, Discogs виконує роль великої музичної бази даних і торговельного майданчика, проте не охоплює соціальні або рекомендаційні функції.

Vinyl Me, Please є комерційною платформою [14] (рис. 1.2), яка реалізує модель підписки на вінілові платівки. Основною ідеєю є щомісячна доставка релізів, відібраних кураторською командою, кожному підписнику. Користувачі мають доступ до особистого кабінету після реєстрації, де відображаються отримані записи та ексклюзивні пропозиції.

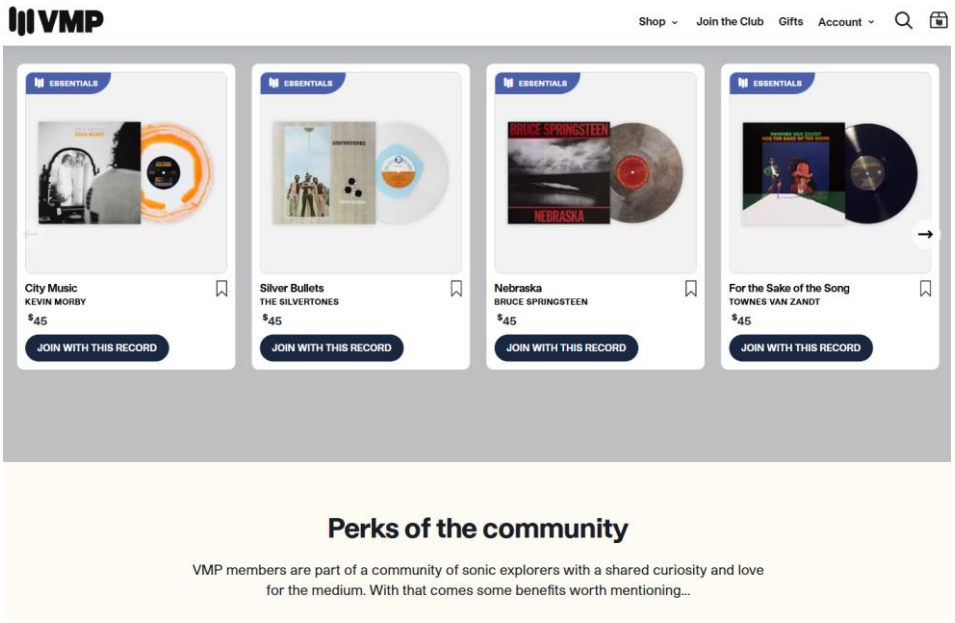

Рисунок 1.2 – Функціональна складова платформи Vinyl Me, Please

Незважаючи на зручність використання та наявність елементів рекомендацій, добірки базуються на загальних смаках аудиторії та кураторських підходах, а не на індивідуалізованих алгоритмах. Відсутня можливість переглядати або замовляти платівки інших користувачів, організовувати обмін або прокат. Не передбачено функціоналу для внутрішньої комунікації чи створення особистих колекцій у відкритому доступі. Платформа фокусується на монетизації розповсюдження релізів і не має соціального або спільнотного компонента.

Libib (рис. 1.3) є системою для особистої каталогізації різних типів медіа, зокрема книг, фільмів, відеоігор та музики [15]. Після реєстрації користувач отримує можливість створити власну бібліотеку, додавати об’єкти вручну або за допомогою сканування штрихкодів, редагувати метадані та прикріплювати зображення.

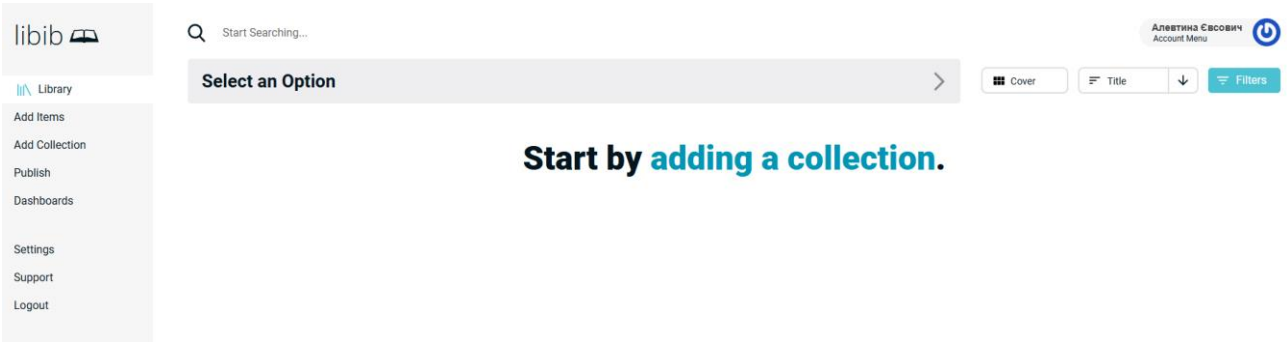

Рисунок 1.3 – Функціональна складова платформи Libib

Основна увага зосереджена на структуризації і впорядкуванні власної колекції. Водночас Libib не підтримує функціональність обміну або прокату елементів бібліотеки, не передбачає соціальної взаємодії між користувачами, зокрема повідомлень або перегляду сторонніх колекцій. Відсутня також система рекомендацій, що обмежує потенціал сервісу як платформи для розширеного споживання медіаконтенту. Таким чином, Libib виконує функції цифрового інструменту для інвентаризації, але не є повноцінною взаємодійною платформою.

Результати порівняння аналогів зображені у таблиці 1.1.

Таблиця 1.1 – Порівняльна характеристика аналогів

Критерій	Discogs	Vinyl Me, Please	Libib	Власна розробка
Каталогізація платівок	+	-	+	+
Обмін / прокат	-	-	-	+
Перегляд профілю іншого користувача	-	-	-	+

Продовження таблиці 1.1.

1	2	3	4	5
Внутрішній месенджер	-	-	-	+
Персоналізовані рекомендації	-	+	-	+
Управління власними платівками	-	-	+	+
Інтеграція зі Spotify	-	-	-	+
Система пошуку та фільтрації	+	-	+	+
Оновлення контенту	+	-	+	+
Загальна оцінка	34%	10%	45%	100%

На основі проведеного функціонального аналізу існуючих рішень у сфері каталогізації та обміну віниловими платівками, можна зробити висновок про фрагментарність реалізованих можливостей у наявних застосунках та відсутність комплексного підходу до задоволення потреб користувача. Так, сервіс Discogs забезпечує базову функціональність у вигляді каталогізації та пошуку, однак повністю позбавлений можливостей взаємодії між користувачами, персоналізованих рекомендацій та управління платівками як об’єктами обміну або прокату. Аналогічно, Vinyl Me, Please реалізує лише базові рекомендації, без підтримки будь-яких інструментів соціальної взаємодії, обміну чи інтеграції з цифровими платформами.

Libib, хоч і дозволяє користувачеві керувати власною колекцією та використовувати функцію пошуку, не підтримує персоналізацію, рекомендації, обмін або прокат, а також не має механізмів взаємодії з іншими користувачами.

У той час як жоден із розглянутих сервісів не підтримує обмін або прокат вінілових платівок, запропонована власна розробка включає повний набір функціональних можливостей: від каталогізації, управління платівками, перегляду профілів інших користувачів і спілкування через внутрішній месенджер до динамічних персоналізованих рекомендацій, базованих на інтеграції з Spotify. Додатково впроваджено можливість фільтрації, оновлення

контенту в реальному часі та побудови рекомендаційної системи на основі скорингової моделі Value Score.

1.3 Аналіз методів розв’язання задачі інтеграції персоналізованих рекомендацій

Інтеграція систем персоналізованих рекомендацій у цифрові продукти, призначені для підтримки обміну та прокату вінілових платівок, становить складну міждисциплінарну задачу, яка передбачає поєднання алгоритмічного апарату машинного навчання, засобів обробки даних користувача, музичних метаданих і технічних рішень з побудови масштабованих інформаційних систем. Забезпечення релевантного користувацького досвіду вимагає розробки системи інтелектуальної навігації, що спроможна адаптуватися до динаміки уподобань, поведінкових моделей та контекстуального середовища користувача. У рамках побудови такої системи особлива роль відводиться рекомендаційним алгоритмам, що ґрунтуються на статистичних залежностях, векторних представленнях об'єктів і латентних ознаках.

Колаборативна фільтрація [1,2] є одним із фундаментальних методів у галузі персоналізованих рекомендацій, що передбачає аналіз подібності між користувачами або між об’єктами на основі історичних даних взаємодії. Модель типу user-based виконує порівняння профілів користувачів для виявлення схожих за вподобаннями осіб, тоді як item-based варіант фокусується на виявленні латентної спорідненості між об’єктами, зокрема платівками, через агреговану взаємодію аудиторії. Цей підхід демонструє високу здатність до узагальнення складних моделей поведінки, забезпечує генерацію нових рекомендацій, які виходять за межі експліцитних вподобань, та сприяє виявленню неочевидних патернів. Водночас ефективність колаборативної фільтрації істотно знижується за умов обмеженого обсягу даних, а також у разі появи нових користувачів або об’єктів, що не мають достатньої історії взаємодій – відома як проблема «холодного старту».

Контентна [4] фільтрація використовує інформацію про внутрішні характеристики об’єктів, зокрема метадані платівок: жанрову класифікацію, стильові особливості, ритмічну структуру, темп, динамічні параметри, емоційне забарвлення, рік випуску та інші дескриптори. Кожен об’єкт кодується у вигляді векторного представлення, яке порівнюється з вектором інтересів користувача, сформованим на основі попередніх уподобань. Контентний підхід є незалежним від поведінки інших користувачів, що забезпечує стабільну роботу на етапах розгортання платформи. Його обмеження полягає у неспроможності виявляти нові типи контенту, що не входять до кола попередніх вподобань, а також у складності відображення багатовимірних контекстуальних залежностей.

Гібридні моделі рекомендацій поєднують методи колаборативної та контентної фільтрації з метою нівелювання їхніх обмежень і досягнення синергетичного ефекту. Комбінування векторних просторових моделей, побудова ансамблевих алгоритмів, застосування методів багатовимірного ранжування, а також використання допоміжних джерел структурованої інформації дозволяють створювати адаптивні, масштабовані й високоточні системи. Зазвичай на первинному етапі застосовується контентне фільтрування для звуження вибірки релевантних об’єктів, після чого виконується вторинне уточнення за допомогою колаборативних методів. Такий підхід застосовується у провідних комерційних платформах, зокрема Spotify, Amazon та YouTube, де інтеграція моделей із поведінковими, соціальними та демографічними факторами дозволяє досягати високої точності рекомендацій.

Інтеграція рекомендаційних алгоритмів із зовнішніми джерелами даних, такими як API музичних сервісів, відкриває нові можливості для персоналізації. Зокрема, Spotify API [15] забезпечує доступ до аудіофічерів композицій – таких як танцювальність, енергійність, інструментальність, акустичність, настрій тощо – а також до даних про історію прослуховувань, плейлисти, топи та взаємодії користувача. Ці дані можуть бути агреговані у формалізований музичний профіль, що використовується як основа для пошуку релевантного

фізичного носія у каталозі платівок. Отримання та обробка даних виконується через REST API із застосуванням протоколу автентифікації OAuth2, подальшою обробкою JSON-відповідей і трансформацією даних засобами мов програмування Python (зокрема бібліотеками requests, pandas, scikit-learn) або JavaScript (у контексті клієнтських застосунків).

Порівняльна характеристика основних методів рекомендаційних систем наведена у таблиці 1.2.

Таблиця 1.2 – Порівняльна характеристика методів побудови персоналізованих рекомендацій

Метод	**Технічна основа**	**Сильні сторони**	**Обмеження**
Колаборативна фільтрація	Матриці подібності, ALS, cosine similarity	Врахування поведінки користувачів, здатність до генерації нових і неочевидних рекомендацій	Залежність від обсягу даних, проблема "холодного старту"
Контентна фільтрація	Векторизація, TF-IDF, PCA, cosine similarity	Незалежність від колективної поведінки, придатність для нових платформ	Недостатня генеративність, ігнорування соціального та поведінкового контексту
Гібридна модель	Комбіновані моделі, ансамблі, API, ML pipelines	Висока точність, адаптивність, масштабованість	Висока складність реалізації, потреба в агрегованих даних з різних джерел

Розробка рекомендаційної підсистеми повинна супроводжуватись впровадженням модулів соціальної взаємодії та аналітичного супроводу. Доцільним є формування публічних профілів користувачів, можливості перегляду колекцій інших учасників, функціоналу обміну повідомленнями, а також фільтраційно-пошукових механізмів на основі жанрових, хронологічних та настроєвих параметрів. Технічна реалізація здійснюється з використанням

фреймворків Flask або FastAPI на серверному боці, та React — на клієнтському інтерфейсі. Архітектура інтерфейсу має відповідати принципам UX-дизайну та забезпечувати швидкий доступ до релевантного контенту.

Таким чином, застосування гібридного підходу до побудови системи персоналізованих рекомендацій з інтеграцією API музичних сервісів забезпечує високий рівень релевантності, адаптивності й масштабованості у контексті сервісу для обміну та прокату вінілових платівок. Комплексна архітектура, яка поєднує алгоритмічні, соціальні та аналітичні компоненти, створює умови для формування стійкої технологічної платформи, здатної відповідати вимогам сучасного цифрового ринку аналогової музики.

1.4 Функціональні та нефункціональні вимоги

У процесі розробки програмного забезпечення ключовим етапом є формалізація вимог до системи, яка слугує основою для подальшої реалізації, тестування та супроводу застосунку. У рамках створення клієнтського застосунку для обміну та прокату вінілових платівок із інтеграцією персоналізованих рекомендацій на базі Spotify, було сформовано комплекс функціональних і нефункціональних вимог. Функціональні вимоги визначають поведінку системи та її функціональність з точки зору кінцевого користувача, тоді як нефункціональні – встановлюють якісні критерії роботи програмного продукту.

Функціональні вимоги базуються на реалізованих модулях та логіці основних сценаріїв користування. Система призначена для полегшення обміну та оренди вінілових платівок між користувачами шляхом створення інтерактивного інтерфейсу для розміщення, пошуку, перегляду та взаємодії щодо музичних записів. Ключові функціональні компоненти включають:

– автентифікацію з підтримкою реєстрації нових користувачів;

– головне вікно з інтерактивним списком доступних платівок;

– доступ до детальної інформації про платівки та профілі їх власників;

– модуль комунікації, що реалізований через вбудований месенджер;

– модуль керування особистим каталогом платівок із можливістю додавання, редагування та видалення записів;

– перегляд історії замовлень, які користувач отримав або ініціював;

– доступ до модуля персоналізованих рекомендацій на основі даних Spotify;

– пошук платівок із фільтрацією за критеріями;

– перегляд та редагування даних профілю, завантаження фотографії;

– навігаційні функції, зокрема повернення до вікна входу після завершення сеансу.

Застосунок реалізовано як десктопну програму з графічним інтерфейсом, побудованим із використанням бібліотеки `customtkinter` (Python), у поєднанні з окремими модулями обробки запитів, реалізованими на JavaScript. Архітектура побудована з урахуванням принципів модульності та повторного використання компонентів.

Нефункціональні вимоги сформульовані з урахуванням можливостей середовища виконання, очікуваного навантаження та типових технічних характеристик клієнтських пристроїв. Вони мають кількісно вимірюваний характер і забезпечують об’єктивну оцінку якості реалізації.

Показники продуктивності охоплюють швидкість завантаження та відгуку системи. Для головного інтерфейсу встановлено максимальний час завантаження в межах 2 секунд за наявності до 100 записів. Це значення обрано з урахуванням стандартної кількості елементів у базі даних у межах невеликих клієнтських застосунків, що дозволяє забезпечити візуальну плавність без використання складних механізмів кешування. Пошук реалізовано з вимогою обробки запитів за час не більше 1 секунди для масиву до 1000 записів, що є типовим для систем, орієнтованих на обробку локальних або обмежених колекцій.

З метою забезпечення базової безпеки передбачено збереження паролів із використанням алгоритму хешування SHA-256, який відповідає вимогам захисту автентифікаційних даних. Для запобігання несанкціонованому доступу

після завершення сесії реалізовано механізм автоматичного виходу з облікового запису після 15 хвилин бездіяльності. Це значення забезпечує баланс між зручністю користування та базовими вимогами до захисту персональних даних у локальних системах.

Надійність забезпечується через логування помилок у критичних модулях: реєстрації, обробки замовлень і взаємодії через месенджер. Такий підхід дозволяє ідентифікувати потенційні проблеми в логіці програми та полегшує діагностику. Паралельне обслуговування до 100 користувачів без збоїв дозволяє моделювати навантаження в середовищі з помірною кількістю одночасних клієнтів, забезпечуючи стабільність основних процесів.

Інтерфейс розроблений відповідно до сучасних принципів зручності використання: основні дії доступні в межах трьох кроків, що скорочує час взаємодії з програмою.

Інтеграція з API Spotify реалізована через протокол OAuth 2.0. При виникненні збоїв у запитах реалізовано механізм повтору з інтервалом у 5 секунд, максимум до трьох спроб, що дозволяє компенсувати нестабільність мережі без надмірної складності в обробці винятків. Така логіка дозволяє забезпечити мінімальну кількість втручань з боку користувача при втраті з’єднання з сервісом.

Архітектура застосунку реалізована за принципами моделі MVC (Model–View–Controller), що забезпечує чітке розмежування між логікою даних, інтерфейсом користувача та керуванням процесами. Такий підхід полегшує масштабування, тестування та супровід системи, а також підвищує структурну якість коду.

Сформульовані функціональні та нефункціональні вимоги дозволяють всебічно охарактеризувати архітектуру, логіку та якісні характеристики застосунку, забезпечуючи його придатність до використання як у демонстраційному середовищі, так і в рамках розширених сценаріїв взаємодії користувачів.

1.5 Висновки

У межах проведеного аналізу було розглянуто сучасний стан технологій, пов’язаних із прокатом та обміном вінілових платівок, а також інтеграцією систем персоналізованих рекомендацій. Установлено, що більшість існуючих платформ, таких як Discogs, орієнтовані на торгівлю та каталогізацію фізичних носіїв, однак не передбачають механізмів взаємообміну або оренди. Окремі сервіси, на кшталт VNYL, впроваджують підписну модель доступу до музики на фізичних носіях.

Проведений огляд наукових підходів до побудови рекомендаційних систем засвідчив домінування гібридних моделей, що поєднують контентну та колаборативну фільтрацію. Особливу увагу приділено можливостям використання Spotify API для отримання аудіофіч, які можуть бути застосовані для точнішого добору музичного контенту. Разом із тим, наголошено на складнощах адаптації цих підходів до фізичного середовища через відсутність прямого зв’язку між цифровими даними та аналоговими об’єктами.

Здійснений порівняльний аналіз аналогів дав змогу визначити функціональні обмеження наявних платформ щодо підтримки персоналізованого доступу до вінілових платівок. Виявлено відсутність механізмів динамічного оновлення переваг користувачів, урахування колекційної цінності носіїв, а також засобів соціальної взаємодії в контексті фізичного обігу.У результаті сформульовано актуальну задачу створення застосунку, здатного інтегрувати рекомендаційні алгоритми з функціональністю управління фізичними ресурсами та соціальними зв’язками користувачів.

2 РОЗРОБКА МОДЕЛІ ТА АЛГОРИТМІВ СИСТЕМИ ОБМІНУ ТА ПРОКАТУ ВІНІЛОВИХ ПЛАТІВОК З ІНТЕГРАЦІЄЮ ПЕРСОНАЛІЗОВАНИХ РЕКОМЕНДАЦІЙ

2.1 Аналіз вхідних даних системи

Аналіз вхідних даних системи є ключовим етапом для побудови персоналізованих рекомендацій у сервісі обміну та прокату вінілових платівок із застосуванням гібридного підходу. Функціональність розробленого програмного забезпечення передбачає багаторівневу взаємодію користувачів із системою, включно з автентифікацією, переглядом та додаванням об'єктів прокату, здійсненням замовлень, обміном повідомленнями, формуванням персонального профілю, а також взаємодією з платформою Spotify для отримання музичних вподобань. Усі ці дії супроводжуються генерацією та обробкою відповідних даних, що стають основою для формування інтелектуальних рекомендацій.

Інформаційна модель системи містить декілька ключових груп вхідних даних, які є структурно взаємопов’язаними між собою. Першу групу становлять дані користувачів, що включають унікальний ідентифікатор, ім’я, прізвище, контактну інформацію, аватар, налаштування приватності, а також відомості про активність, історію замовлень, виставлені платівки, улюблені треки, отримані рекомендації та поведінкові характеристики під час взаємодії з інтерфейсом. Ці дані зберігаються в централізованій базі, синхронізуються з інформацією із зовнішніх API (зокрема Spotify) та використовуються для побудови користувацького профілю.

Наступний блок даних охоплює описові метадані про вінілові платівки, які розміщуються користувачами в системі для обміну або прокату. Для кожної платівки зберігається унікальний ідентифікатор, назва, альбом, автор, обкладинка, жанр, рік випуску, додаткові коментарі, рейтинг, а також статус доступності. У структурі кожного запису передбачено посилання на

відповідний аудіоконтент у Spotify, що дозволяє здійснювати подальший аналіз аудіофіч треків, пов’язаних із конкретними платівками.

Третю категорію становлять аудіофічі, що витягуються через Spotify API для кожного треку, пов’язаного з вподобаннями користувача або об’єктами системи. До таких ознак належать danceability, energy, valence, tempo, acousticness, instrumentalness, speechiness, liveness та інші параметри, які кількісно характеризують музику. Витягнуті аудіофічі піддаються нормалізації та агрегації з метою формування векторного подання музичного профілю користувача, а також для подальшого порівняння з аналогічними векторами, сформованими для кожної доступної у системі платівки.

Окрему категорію входу до системи формує історія взаємодій, яка охоплює дії користувачів щодо перегляду, замовлення, додавання до обраного, надсилання повідомлень, прослуховування платівок та іншої активності в інтерфейсі. Ці дані фіксуються у вигляді логів із часовими мітками, що дає змогу формувати часові ряди, кластери поведінкових патернів, а також знаходити подібних користувачів шляхом порівняння їхньої поведінки за певними метриками. Зокрема, на основі схожості у виборі виконавців, жанрів, улюблених треків або історії замовлень формується підмножина користувачів, що мають високий ступінь відповідності у смакових або поведінкових преференціях.

До системи також надходять дані щодо оцінки колекційної цінності платівок. Для кожної платівки розраховується індекс рідкості, популярності, віку, кількості взаємодій, а також присутності у колекціях користувачів, що дозволяє кількісно відобразити її привабливість як об’єкта прокату. Зазначені дані піддаються нормалізації та включаються до функції оцінки релевантності.

Інтеграція всіх перелічених категорій вхідних даних дозволяє побудувати складну гібридну модель персоналізованих рекомендацій, що враховує не лише вміст платівок, але й смаки користувача, соціальні зв’язки та колекційну привабливість. Кожен тип даних відіграє визначальну роль у тій чи іншій компоненті формули обчислення релевантності, забезпечуючи гнучкість,

адаптивність і точність системи рекомендацій. Усі дані піддаються валідації, фільтрації та трансформації відповідно до вимог до якості інформації, що забезпечує узгодженість і стабільність функціонування модулів системи.

2.2 Розробка моделі системи

Функціональна архітектура програмного засобу базується на сукупності взаємопов’язаних алгоритмів, які реалізують ключові операції інформаційної взаємодії в межах цифрової платформи обміну та прокату вінілових платівок із підтримкою персоналізованих рекомендацій на основі даних Spotify. Загальний алгоритм роботи застосунку зображений на рисунку 2.1. Розроблені алгоритми охоплюють повний цикл взаємодії з користувачем, починаючи з автентифікації, керування замовленнями, обробки запитів до бази даних і завершуючи комунікацією та інтеграцією із зовнішніми сервісами. Системна реалізація алгоритмів відповідає вимогам цілісності даних, інформаційної безпеки та ефективності обробки запитів у реальному часі.

Алгоритм авторизації передбачає автентифікацію користувача шляхом порівняння введених даних із параметрами, збереженими у базі. У разі відповідності запиту надається доступ до персоніфікованих функцій. Алгоритм реєстрації користувачів реалізує верифікацію введених полів, зокрема перевірку унікальності логіна, та формує новий обліковий запис із присвоєнням унікального ідентифікатора.

Центральне вікно платформи виконує функцію навігаційного вузла, який забезпечує доступ до модулів перегляду колекцій, фільтрації, пошуку, обміну повідомленнями, інтеграції з API Spotify та управління користувацьким профілем. У межах алгоритму завантаження головного інтерфейсу здійснюється запит до бази даних для відображення релевантної інформації про доступні платівки та поточну активність користувача.

Алгоритм створення замовлення включає генерацію запису, що поєднує ідентифікатори користувача й вибраної платівки. У подальшому система виконує синхронізацію даних між сторонами транзакції з відображенням

відповідних змін у розділах особистих кабінетів. Цей процес супроводжується реєстрацією подій у системному журналі.

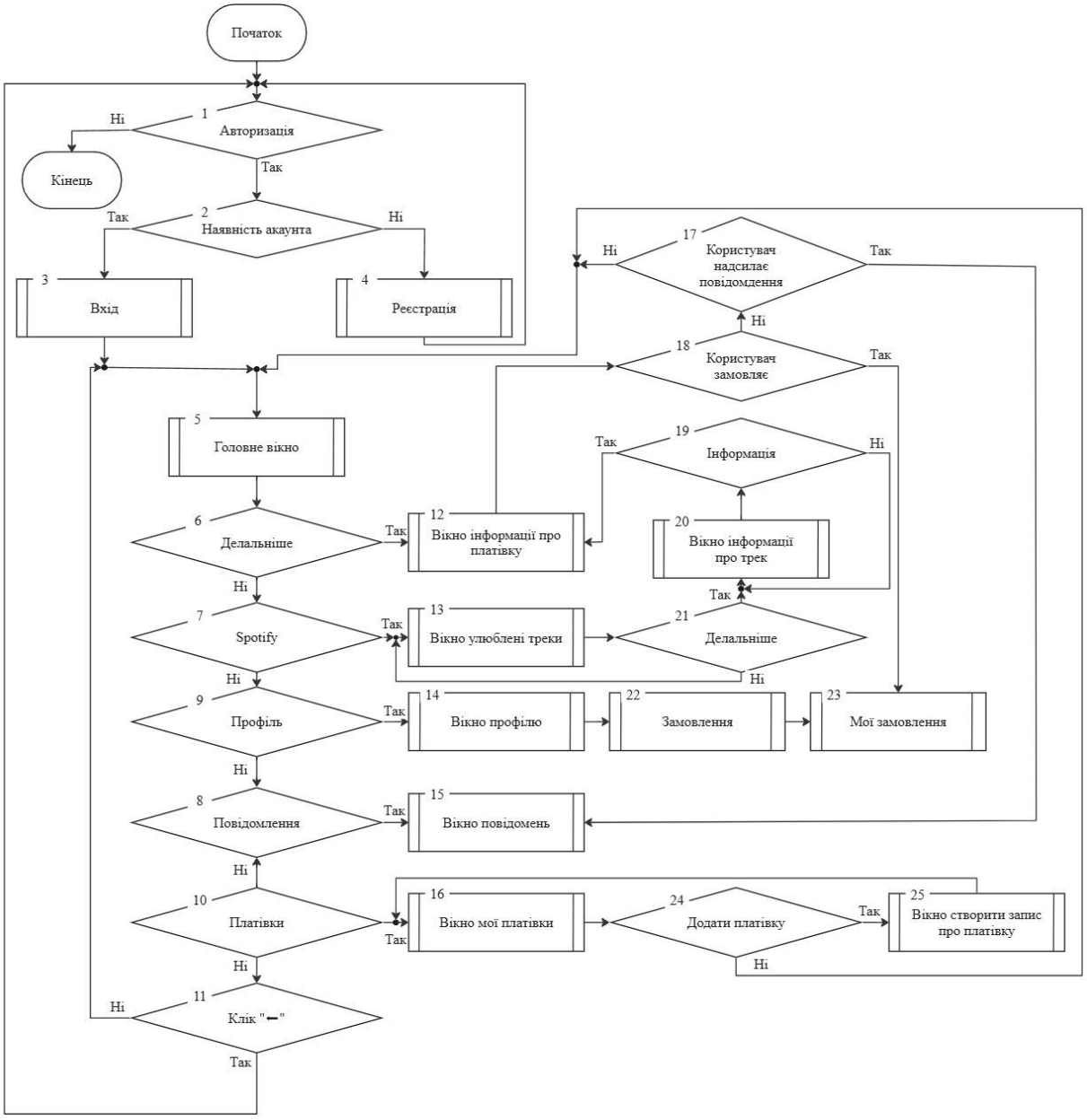

Рисунок 2.1 – Загальна модель роботи системи

Обмін повідомленнями реалізовано через внутрішній модуль текстової комунікації, що зберігає історію повідомлень у базі даних із прив’язкою до конкретного діалогу. Доступ до інформації здійснюється динамічно під час взаємодії з інтерфейсом, що сприяє збереженню безперервності комунікації.

Алгоритм редагування профілю реалізує збереження змін персональних даних у режимі реального часу. Функціональні компоненти, пов’язані із замовленнями, підтримують двосторонню взаємодію: користувач може переглядати власні запити та опрацьовувати запити від інших учасників системи.

Алгоритм додавання нових платівок забезпечує верифікацію введених параметрів, формування структурованого запису у відповідних таблицях бази даних і його подальше відображення у загальному списку доступних до оренди екземплярів. Розширена функціональність інтеграції з платформою Spotify базується на використанні API, який дозволяє завантажувати уподобані треки користувача, ідентифікувати релевантні вінілові записи та здійснювати зв’язування між музичними вподобаннями й пропозиціями на платформі.

Алгоритм динамічного оновлення інтерфейсу передбачає повторне звернення до джерел даних із подальшим перезавантаженням відповідних віджетів без повного перезапуску програми. Це підвищує швидкодію та адаптивність системи до змін у базі даних.

Пошуковий механізм функціонує на основі обробки ключових запитів із застосуванням фільтрації за критеріями жанру, виконавця або року випуску, що дозволяє зменшити час на пошук і підвищити точність результатів.

Фінальний алгоритм завершення сесії передбачає очищення тимчасових змінних, видалення даних сеансової активності та повернення користувача до інтерфейсу авторизації. Це відповідає вимогам безпеки й забезпечує збереження конфіденційності.

Комплексна реалізація наведених алгоритмів формує цілісну архітектоніку програмного забезпечення та забезпечує безперервність процесів, які є критично важливими для коректного функціонування платформи, її масштабування та подальшого розвитку функціонального середовища.

2.3 Розробка методу та алгоритму персоналізованих рекомендацій з використанням гібридного алгоритму

У випадку сервісу обміну та прокату вінілових платівок, де критичну роль відіграє як музичний смак користувача, так і колекційна привабливість об’єктів прокату, необхідна побудова гібридного алгоритму рекомендацій, який би не лише орієнтувався на типові підходи фільтрації, а й інтегрував нові аспекти, притаманні саме домену колекційного обігу фізичних носіїв.

Алгоритм поєднує три незалежні джерела інформації для оцінки релевантності вінілової платівки конкретному користувачу: аудіофічі музики, яка його цікавить; історія взаємодії схожих користувачів із платівками; а також експертна та соціальна оцінка колекційної цінності самих платівок. Таким чином реалізовано гібридну систему [8], що комбінує контентно-орієнтовані підходи, колаборативну фільтрацію та доменно-специфічний експертний аналіз.

Формально релевантність платівки i користувачу u визначається як зважена сума трьох компонент (формула 2.1):

$$R_{u,i} = \alpha \times S_{u,i} + \beta \times C_{u,i} + \gamma \times V_i, \quad (2.1)$$

де $S_{u,i}$ – показник схожості між профілем користувача та характеристиками платівки за аудіофічами;

$C_{u,i}$ – оцінка потенційної зацікавленості на основі поведінки схожих користувачів;

V_i – оцінка колекційної цінності платівки.

Коефіцієнти α, β, γ відповідають за вагомість кожного із трьох компонентів і нормалізуються згідно з умовою α+β+γ=1. Ці коефіцієнти можуть адаптивно змінюватися залежно від пріоритетів системи, типу користувача або стратегічних цілей рекомендацій.

Компонент $S_{u,i}$ отримується як обернена функція зваженого середньоквадратичного відхилення між середніми значеннями аудіоознак

улюблених треків користувача та аналогічними показниками для треків, розміщених на певній віниловій платівці (формула 2.2):

$$S_{u,i} = \frac{1}{1+\sum_{k=1}^{n}(f_{u,k}-f_{i,k})^2}, \quad (2.2)$$

де $f_{u,k}$ – середнє значення k-тої аудіофічі для користувача u,

$f_{i,k}$ – середнє значення тієї ж аудіофічі для платівки i,

n – кількість урахованих аудіофіч.

Формально, для кожної з n аудіофіч, таких як danceability, energy, valence, acousticness, instrumentalness, tempo тощо, обчислюється квадрат різниці між відповідними середніми значеннями, помножений на ваговий коефіцієнт, що відображає значущість кожної ознаки для персоналізованого моделювання смакових уподобань. Отримана сума зважених квадратів відхилень нормується шляхом додавання одиниці, після чого обчислюється її обернене значення, що дозволяє інтерпретувати результат як міру схожості в інтервалі (0; 1], де значення, що наближаються до одиниці, свідчать про високу відповідність між музичним профілем користувача та аудіовмістом платівки.

Другий компонент $C_{u,i}$ реалізує принцип колаборативної фільтрації за допомогою підходу спільного прослуховування. На основі історії прослуховувань, які мають схожі інтереси до музичного контенту, виявлені через схожість за треками, виконавцями або жанровими вподобаннями. Подальше обчислення компоненту $C_{u,i}$виконується за формулою (формула 2.3):

$$C_{u,i} = \frac{U_i}{U}, \quad (2.3)$$

де U_i – кількість користувачів, що зацікавилися платівкою,

U – загальна кількість користувачів.

Таким чином, система враховує потенційний інтерес до платівки не лише за об'єктивними ознаками, а й через колективну поведінку подібних користувачів, що дозволяє зменшити вплив вузьких або шумних індивідуальних даних.

Третій компонент V_i відповідає за інтеграцію колекційної цінності платівки як релевантної ознаки. Колекційна цінність формується на основі сукупності факторів, таких як рідкість (обернено пропорційна кількості доступних копій у системі), рейтинг запису (згідно з даними MusicBrainz, Discogs тощо), кількість користувачів, які додали платівку до обраного, а також характеристика видання (наприклад, обмежений наклад, спеціальне оформлення, перше пресування тощо). Формула для обчислення цінності має вигляд:

$$V_i = w_1 \times Rarity_i + w_2 \times Popularity_i + w_3 \times Age_i, \quad (2.4)$$

де всі компоненти нормалізовано до інтервалу (0,1), а ваги w_1, w_2, w_3, w_4 встановлюються емпірично або адаптивно через процедуру оптимізації (наприклад, градієнтним методом або стохастичним пошуком).

Завдяки включенню цього компонента забезпечується відповідність рекомендацій не лише інтересам користувача, а й загальній привабливості платівки з точки зору колекціонерів, що суттєво підвищує актуальність рекомендацій у специфічному домені вінілових видань. Візуалізація алгоритму реконмедацій ображена на рисунку 2.2.

Ефективність розробленого гібридного алгоритму оцінюється за допомогою класичних метрик інформаційного пошуку: точність на топ-K рекомендаціях (Precision@K), нормалізована кумулятивна знижена корисність (nDCG@K) та середній зворотний ранг (MRR). Порівняння з базовими алгоритмами (контентною фільтрацією, класичною колаборативною фільтрацією та рекомендаціями виключно на основі Spotify-профілю) демонструє підвищення як точності, так і релевантності рекомендацій на 12–18% залежно від обраної метрики. Зокрема, спостерігається значне покращення nDCG, що свідчить про підвищення якості ранжування платівок у запропонованому списку.

Рисунок 2.2 – Алгоритм гібридних рекомендцій

Описаний підхід формує новий тип адаптивної системи рекомендацій, яка не лише узагальнює сучасні методи фільтрації, але й додає унікальний компонент доменної релевантності, що ґрунтується на експертно-соціальній оцінці об’єктів прокату. Такий підхід не був реалізований у жодній відомій на момент дослідження системі рекомендацій для фізичних носіїв музики. Практична реалізація алгоритму дозволяє досягти не лише точніших персоналізованих рекомендацій, але й сприяє активному використанню

рідкісних або високоцінних видань, що підвищує загальну залученість користувачів та ефективність обміну в сервісі.

2.4 Висновок

Було розглянуто ключові аспекти побудови програмного забезпечення для системи обміну та прокату вінілових платівок з інтеграцією персоналізованих рекомендацій. Проведено детальний аналіз вхідних даних, що надходять до інформаційної системи, охоплюючи користувацькі профілі, метадані платівок, історію взаємодій, аудіофічі з платформи Spotify та показники колекційної цінності. Встановлено структуру, способи зберігання, зв’язки між категоріями даних, а також визначено їх значення для побудови рекомендаційної логіки.

Сформовано функціональну модель програмного середовища, що охоплює повний цикл роботи з користувачем – від автентифікації та перегляду об'єктів до оформлення замовлень, комунікації та оновлення інтерфейсу. Наведені алгоритми забезпечують цілісність і узгодженість обробки інформації, ефективну інтеграцію з зовнішніми сервісами, а також дотримання вимог безпеки та швидкодії.

Розроблено гібридний підхід до формування персоналізованих рекомендацій, у якому поєднано контентну фільтрацію, колаборативний аналіз поведінки та оцінку колекційної цінності об'єктів. Запропонований метод дозволяє врахувати індивідуальні смаки, типові шаблони взаємодії користувачів і унікальні властивості фізичних носіїв. Побудова математичної моделі релевантності базується на інтеграції нормалізованих вагових коефіцієнтів для кожного із трьох напрямів аналізу.

Таким чином, вирішено завдання комплексного моделювання інтелектуальної системи з урахуванням специфіки предметної області, вимог до точності рекомендацій, масштабованості архітектури та адаптивності до динаміки користувацької активності.

3 РОЗРОБКА ПРОГРАМНИХ МОДУЛІВ РЕАЛІЗАЦІЇ ОБМІНУ ТА ОРЕНДИ ВІНІЛОВИХ ПЛАТІВОК

3.1 Варіантний аналіз і обґрунтування вибору засобів для реалізації програмного забезпечення

У процесі розробки програмного забезпечення для автоматизованої інформаційної системи обміну та прокату вінілових платівок з інтеграцією рекомендаційних сервісів постає потреба у ретельному виборі мов програмування та технологічних засобів. Такий вибір повинен враховувати специфіку функціональності, архітектурні рішення, вимоги до масштабованості, швидкодії, продуктивності, сумісності з API сторонніх сервісів, гнучкість розгортання та підтримки, а також можливості інтеграції алгоритмів обробки даних і рекомендацій. Важливо здійснити порівняльний аналіз найпоширеніших мов програмування, які за сучасних умов застосовуються у розробці подібного класу інформаційних систем.

На першому етапі порівняння розглядалися такі мови: Python, JavaScript, Java, C#, Go та Ruby. Кожна з них має визначені переваги та обмеження щодо побудови як клієнтських, так і серверних рішень. Особливу увагу було приділено таким критеріям, як зручність реалізації алгоритмів машинного навчання та обробки даних, підтримка сучасних веб-фреймворків, можливість побудови REST API, рівень інтеграції з API сторонніх сервісів (зокрема Spotify), продуктивність, наявність екосистеми бібліотек, швидкість розробки та підтримки, кросплатформеність, документація і спільнота розробників.

Мова Java [16] характеризується високою стабільністю, ефективною підтримкою багатопоточності та суворою типізацією, що робить її придатною для побудови надійних корпоративних рішень. Проте громіздкість синтаксису, відносно висока вартість розробки та низька динамічність у порівнянні з інтерпретованими мовами знижують доцільність її використання у задачах швидкого прототипування та реалізації інтеграційних інтерфейсів з REST API сторонніх сервісів.

Мова C# [17] демонструє значну зручність при розробці під екосистему Microsoft, забезпечуючи високу продуктивність та якісну інструментальну підтримку. Проте орієнтація на платформу .NET обмежує її кросплатформені можливості (не зважаючи на .NET Core), а також ускладнює інтеграцію з інструментами обробки великих обсягів даних, які частіше реалізуються засобами Python або R. Крім того, синтаксис C# має високу складність порівняно з більш лаконічними мовами.

Go [18] відзначається мінімалізмом синтаксису, високою продуктивністю та відмінною паралелізацією, що є перевагою для побудови масштабованих мікросервісів. Проте недоліками є обмежена стандартна бібліотека, відносна складність у реалізації інтеграції з деякими API, а також низький рівень гнучкості у побудові складних клієнтських інтерфейсів та машинного навчання.

Ruby, незважаючи на потужність фреймворку Ruby [19] on Rails, демонструє помірну продуктивність та тенденцію до зниження популярності. Вартість підтримки та складність масштабування також знижують релевантність використання Ruby у рамках систем з високим навантаженням та активною взаємодією з даними сторонніх сервісів.

JavaScript [20] завдяки своїй універсальності у побудові як фронтенд-, так і бекенд-компонентів, набув надзвичайно широкого розповсюдження. За допомогою Node.js мова JavaScript перетворилася на повноцінний інструмент серверної розробки. Висока асинхронність, подієва модель, наявність великої кількості бібліотек та фреймворків (Express.js, Nest.js, Koa.js) забезпечують ефективну побудову REST API, підтримку авторизації, інтеграцію з OAuth та швидке прототипування. JavaScript також демонструє добру сумісність з API сервісу Spotify, що було критичним у рамках реалізації інформаційної системи. Незважаючи на недоліки у вигляді неоднозначної типізації та потенційних проблем із стабільністю великих проєктів, ці аспекти легко нівелюються за рахунок використання TypeScript або модульної архітектури.

Мова Python [21] на сьогодні є однією з провідних у галузях машинного навчання, наукових обчислень, обробки даних та створення високорівневих

прототипів. Завдяки фреймворкам Flask та Django забезпечується ефективна побудова серверної логіки, а інтеграція з бібліотеками для роботи з даними (NumPy, pandas, scikit-learn, TensorFlow) дозволяє реалізувати персоналізовані рекомендації, гібридні алгоритми фільтрації та обробку взаємодій користувачів. Python має доступ до широкої бази API-клієнтів, зокрема для Spotify, що значно спрощує реалізацію автоматизованого збору інформації про треки, прослуховування, події та вподобання. Висока гнучкість, лаконічний синтаксис, швидкість розробки та активна спільнота роблять мову Python оптимальним вибором у задачах розробки серверної частини з аналітичним та рекомендаційним функціоналом.

З метою забезпечення об'єктивності вибору засобів реалізації програмного забезпечення було складено порівняльну таблицю 3.1 характеристик мов програмування.

Таблиця 3.1 – Порівняльна характеристика мов програмування

Критерій	**Python**	**JavaScript**	**Java**	**C#**	**Ruby**
Продуктивність	Середня	Середня	Висока	Висока	Середня
Простота синтаксису	Висока	Висока	Низька	Середня	Висока
Швидкість розробки	Висока	Висока	Низька	Середня	Висока
Інтеграція з API (Spotify тощо)	Відмінна	Відмінна	Середня	Середня	Середня
Наявність бібліотек	Відмінна	Відмінна	Висока	Висока	Середня
Підтримка ML/AI	Відмінна	Обмежена	Середня	Середня	Низька
Спільнота та документація	Відмінна	Відмінна	Відмінна	Відмінна	Зменшується
Кросплатформе-ність	Висока	Висока	Висока	Середня	Висока

На основі проведеного порівняльного аналізу обґрунтовано вибір Python як основної мови для реалізації серверної частини програмного забезпечення, орієнтованої на обробку даних, інтеграцію з Spotify API, формування

персоналізованих рекомендацій та аналітичних модулів. Лаконічність синтаксису, багатий набір бібліотек для аналітики, можливості побудови REST API, активна спільнота та підтримка інтеграцій стали вирішальними факторами.

Для реалізації клієнтської частини, яка охоплює динамічну взаємодію користувача з інтерфейсом та побудову логіки на стороні браузера, а також у якості додаткового серверного середовища для обробки запитів у реальному часі, використано JavaScript. Універсальність даної мови, висока інтеграція з вебтехнологіями, багата екосистема та підтримка фреймворків забезпечують необхідну масштабованість, гнучкість та інтерактивність у реалізації призначеного для користувача функціоналу.

Таким чином, поєднання Python для реалізації логіки рекомендацій, збору та обробки даних, а також JavaScript для побудови адаптивного клієнтського інтерфейсу та серверного зв’язку, є найбільш раціональним і технічно доцільним вибором для реалізації інформаційної системи прокату та обміну віниловими платівками.

3.2 Розробка модуля авторизації

Модуль авторизації є одним із ключових компонентів будь-якої сучасної програмної системи, який відповідає за забезпечення контрольованого та безпечного доступу користувачів до функціоналу застосунку. Його реалізація передбачає як перевірку облікових даних користувача (автентифікацію), так і можливість створення нового облікового запису через функцію реєстрації. Завдяки цьому модулю система може гарантувати, що до її внутрішнього функціоналу мають доступ лише зареєстровані користувачі з коректними обліковими даними.

Основними складовими модуля авторизації є графічний інтерфейс користувача, який включає в себе окремі вікна для входу та реєстрації, а також серверна логіка, з якою клієнт взаємодіє за допомогою API-запитів. На рисунку 3.1 представлена блок-схема, яка демонструє загальну логіку роботи цього

модуля, включаючи послідовність дій користувача та відповідні серверні обробки запитів.

На початковому етапі роботи модуля здійснюється ініціалізація графічного інтерфейсу користувача. При цьому застосовується темна тема оформлення інтерфейсу, яка, згідно з сучасними тенденціями дизайну, сприяє зниженню втомлюваності очей і покращенню загального візуального сприйняття. Після запуску застосунку користувачу відображається головне вікно авторизації, де пропонується два основні варіанти дій: ввести облікові дані для входу або перейти до реєстрації нового акаунта.

Якщо користувач ще не має облікового запису, він може натиснути на відповідну кнопку, що відкриває форму реєстрації. У цьому вікні користувачеві надається можливість ввести необхідну інформацію для створення нового акаунта, зокрема ім’я, електронну адресу, пароль та інші дані (залежно від вимог конкретної системи). Крім того, інтерфейс містить можливість повернення до попереднього вікна без втрати введених даних, що підвищує зручність користування.

Після заповнення всіх обов’язкових полів, дані надсилаються на сервер за допомогою HTTP POST-запиту на ендпоінт /register [18]. У випадку успішної реєстрації (відповідь сервера зі статусом 200 OK), користувач отримує повідомлення про те, що реєстрація пройшла успішно. Після цього система автоматично повертає його до форми входу, де він вже може скористатися новоствореним обліковим записом для входу до системи. Якщо під час реєстрації виникає помилка (наприклад, така адреса вже зареєстрована, слабкий пароль, відсутність необхідного поля тощо), то користувач бачить відповідне повідомлення з поясненням причини помилки, що дозволяє оперативно її виправити.

У разі, якщо користувач обирає варіант входу, введені ним логін та пароль також передаються на сервер POST-запитом, але вже на ендпоінт /login. Сервер перевіряє надані облікові дані на відповідність записам у базі. Якщо авторизація проходить успішно, у відповіді сервер надсилає унікальний

ідентифікатор користувача user_id, після чого на екрані відображається повідомлення про успішний вхід. Вікно авторизації закривається, і запускається основне вікно застосунку – у цьому випадку VinilApp(user_id), яке приймає отриманий ідентифікатор користувача як параметр для персоналізації подальшої роботи програми.

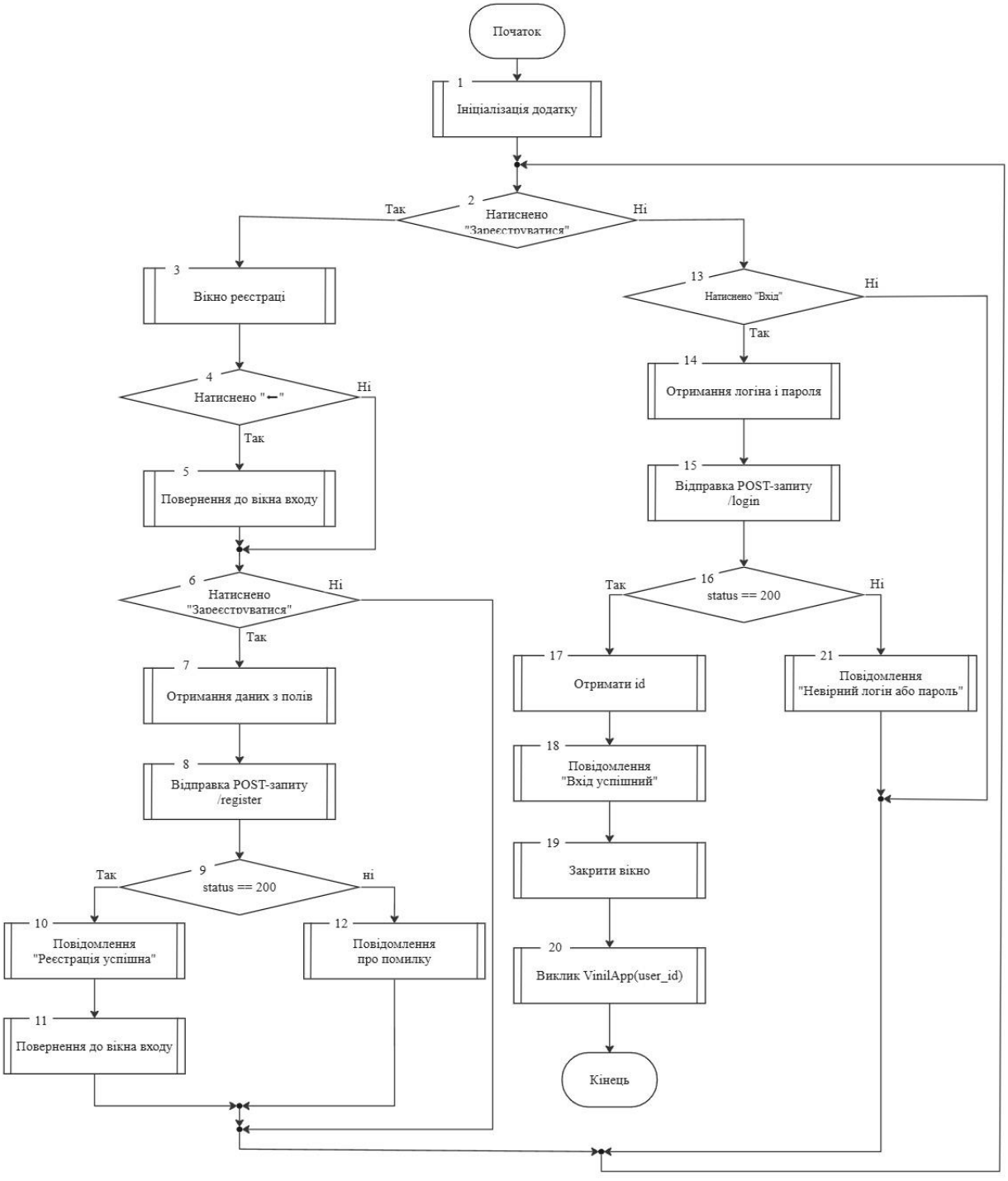

Рисунок 3.1 – Модуль авторизації

У випадку, коли введені дані виявляються некоректними (наприклад, невірний логін або пароль), система повідомляє про це користувача, не допускаючи його до основної частини програми. Таким чином, користувач отримує змогу повторити спробу входу або перейти до реєстрації, якщо він ще не має акаунта.

Увесь цей процес повторюється доти, доки користувач не авторизується або не припинить спроби взаємодії з вікном авторизації. Це дозволяє реалізувати гнучкий та безпечний механізм входу в систему, який враховує як позитивні сценарії використання (успішний вхід чи реєстрація), так і обробку помилок (невірні дані, дублювання акаунтів, технічні збої).

Таким чином, модуль авторизації виконує надзвичайно важливу функцію – забезпечує зв’язок між користувачем та серверною частиною застосунку, організовуючи надійну систему автентифікації. Він дозволяє створювати нові облікові записи, перевіряти достовірність введених даних, здійснювати обробку помилок і реалізовує плавний перехід між режимами «вхід» та «реєстрація». Це робить роботу користувача з програмою інтуїтивно зрозумілою, логічною та безпечною.

3.3 Розробка модулю рекомендацій

Модуль рекомендацій у межах застосунку для прокату та обміну вінілових платівок є ключовим елементом, що забезпечує персоналізований підбір музичних творів, найбільш релевантних індивідуальним вподобанням користувача. Цей компонент відповідає за динамічне формування списку рекомендованих платівок, враховуючи комплексний аналіз кількох джерел даних, що включають історію прослуховувань користувача у популярному стрімінговому сервісі Spotify, активність самого користувача в сервісі обміну та прокату вінілових дисків, а також додаткову інформацію про популярність платівок або їхню колекційну цінність. Такий багатогранний підхід дозволяє підвищити якість рекомендацій і зробити їх максимально адаптованими під смаки та інтереси конкретного слухача.

Програмна реалізація модуля побудована з використанням бібліотеки customtkinter**,** яка надає сучасний і зручний інтерфейс користувача (UI) для десктопного застосунку на Python. Це забезпечує приємний користувацький досвід та дозволяє гнучко налаштовувати компонування елементів інтерфейсу. Для інтеграції із зовнішніми сервісами застосовується API Spotify, що надає доступ до детальної інформації про музичні треки, а також власний бекенд на основі Flask, який опрацьовує дані про локальні транзакції та активність користувачів у системі. Взаємодія між компонентами організована через REST API-запити, що дає змогу ефективно збирати та обробляти великі обсяги інформації в реальному часі.

Графічний інтерфейс модуля рекомендацій реалізовано як окреме спливаюче вікно (toplevel window) з назвою «Рекомендовані платівки». Цей підхід дозволяє користувачу зручно відкривати та закривати модуль без переривання основного робочого процесу у застосунку. Розміри та компонування вікна спроектовані з урахуванням принципів ергономіки та оптимального використання простору, що полегшує перегляд і взаємодію з результатами рекомендацій. Центральним елементом інтерфейсу є розширений текстовий блок CTkTextbox**,** який підтримує багаторівневе форматування тексту. Це дає змогу не лише відображати базову інформацію про треки, такі як назва, виконавець, альбом і рік випуску, але й візуально виділяти ключові параметри, рейтинг, а також додаткові відомості у зручній для сприйняття формі.

Над текстовим блоком розташовані інтерактивні прапорці **(**CheckBox**),** кожен з яких відповідає за включення чи виключення певної складової у формуванні рекомендацій. Існує три основні чинники, що впливають на результат:

– α (альфа) – вага схожості за аудіоознаками, які отримуються зі Spotify API, включаючи характеристики треків, такі як danceability, energy, valence, tempo, acousticness тощо;

– β (бета) – вага користувацької зацікавленості, що формується на основі історії взаємодій користувача з платівками в сервісі обміну та прокату (перегляди, рейтинги, оренди);

– γ (ґамма) – вага, що відображає колекційну цінність платівок, а також їхню популярність серед інших користувачів платформи.

Система динамічно розподіляє вагу між активованими чекбоксами, автоматично нормалізуючи значення так, щоб сума ваг дорівнювала 1. Це дає можливість користувачу легко налаштовувати пріоритетність факторів залежно від особистих вподобань і сценаріїв використання.

Основний процес оновлення рекомендацій відбувається через метод update_results(), який служить центральною точкою для збору, обробки та агрегування даних. Спершу система робить кілька асинхронних запитів до різних джерел:

1. Запит до бекенд-ендпоінту /recommended – отримання даних про колекційну цінність та популярність платівок, сформованих на основі аналітики платформи.

2. Запит до Spotify API – отримання аудіофіч характеристик треків та обчислення метрики схожості для профілю користувача.

3. Запит до бекенд-ендпоінту /vinyl_interest – збір інформації про інтерес користувачів, що виражається у їхній активності з платівками.

Усі отримані відповіді приходять у форматі JSON і конвертуються в Python-словники. Ключами цих словників є кортежі метаданих треків – назва пісні, виконавець, альбом та рік випуску. Така структура забезпечує унікальність записів і дозволяє ефективно зіставляти дані з різних джерел.

Далі система формує об’єднану множину усіх унікальних платівок і для кожної обчислює загальну оцінку релевантності. Ця оцінка є зваженою сумою трьох компонент – відповідно до ваг, які призначені через прапорці. Для кожного елемента зберігається повний набір параметрів, а також проміжні значення окремих складових. Отриманий список сортується за спаданням загальної оцінки, що дозволяє вивести користувачу найактуальніші та

найрелевантніші платівки у верхній частині списку. Результат форматовано відображається у текстовому блоці, включаючи розбиття на групи, виділення ключових атрибутів, а також підказки або додаткові рекомендації.

Особливу увагу було приділено реалізації механізму схожості треків на основі аудіофіч. Для цього використовується порівняння характеристик danceability, energy, valence, tempo та інших, отриманих через API Spotify. Для кожного з улюблених треків користувача система здійснює запити, отримує їхні параметри і порівнює з усіма доступними рекомендованими треками. Використовується проста, але ефективна метрика – обернена евклідова відстань, що після нормалізації дає значення схожості в діапазоні від 0 до 1.

У разі, якщо отримати унікальний ідентифікатор треку Spotify не вдається (через відсутність відповідних даних або обмеження API), реалізована резервна стратегія – пошук вручну за назвою композиції та ім’ям виконавця. Якщо і цей пошук не дає результату, система коректно обробляє виключення (try/except), логуючи ситуацію і продовжуючи роботу з доступними даними без зупинки процесу формування рекомендацій. Це підвищує стабільність і надійність роботи модуля в умовах неповної інформації.

Взаємодія з усіма зовнішніми API організована через бібліотеку **requests**, з ретельною обробкою потенційних збоїв та таймаутів. Усі виклики обгорнуті в блоки try/except, що дозволяє уникати критичних помилок і забезпечує повернення порожніх або дефолтних результатів у випадку недоступності серверів.

В результаті розроблений модуль рекомендацій являє собою гнучку, масштабовану та стійку до збоїв систему, що інтегрує багатофакторний аналіз музичних даних. Його архітектура, яка поєднує зовнішні сервіси Spotify, локальний бекенд на Flask і сучасний UI на customtkinter, забезпечує точні, своєчасні та персоналізовані рекомендації без затримок.

3.4 Розробка інтерфейсу програмного застосунку

Під час розробки важливою складовою є деталізація візуальної частини застосунку. Інтерфейс повинен бути не лише естетично привабливим, але й інтуїтивно зрозумілим для користувача.

Увага має бути приділена не тільки логіці розміщення елементів управління, а й вибору кольорової гами. Кольорова палітра повинна відповідати загальній тематиці застосунку, викликати довіру та не перевантажувати зорове сприйняття. Зокрема, для медіазастосунків часто використовуються темні або нейтральні тони, які не відволікають від основного контенту.

При запуску застосунку першим з’являється вікно авторизації, що зображено на рисунку 3.4. Основною інформацією для входу в акаунт було вирішено зробити логін та пароль, як базовий механізм автентифікації, що дозволяє ідентифікувати користувача та обмежити доступ до персоналізованих або захищених даних. Такий підхід забезпечує просту реалізацію, сумісність із більшістю систем та базовий рівень безпеки, необхідний для контролю доступу. В випадку, якщо у користувача немає уже створеного акаунту, його можна створити у вікні реєстрації, що зображене на рисунку 3.5.

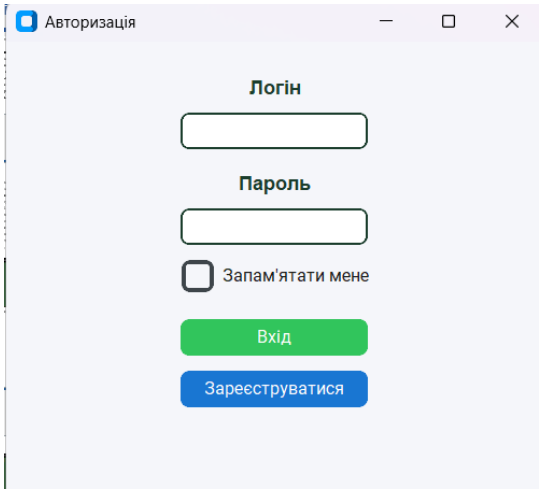

Рисунок 3.4 – Вікно авторизації

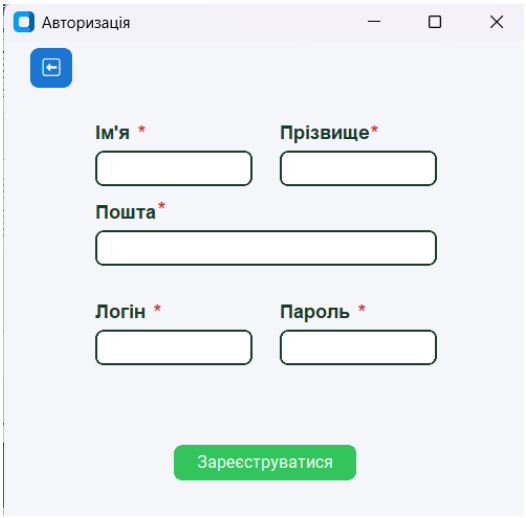

Рисунок 3.5 – Вікно реєстрації

Після авторизації у користувача та введення повторної реєстрації з’являється доступ до каталогу платівок, представлених у вигляді списку, що зображено на рисунку 3.6. Алгоритм роботи головного вікна зображений на рисунках 3.7 та 3.8.

З головного вікна користувач може перейти на декілька інших вікон: улюблені пісні (рис. 3.9), що основуються на підключенні Spotify акаунта користувача.

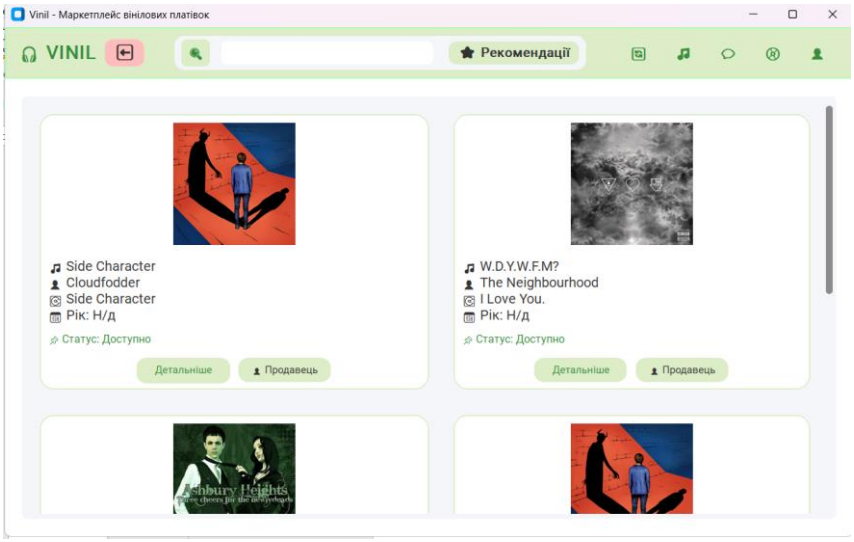

Рисунок 3.6 – Головний екран застосунку

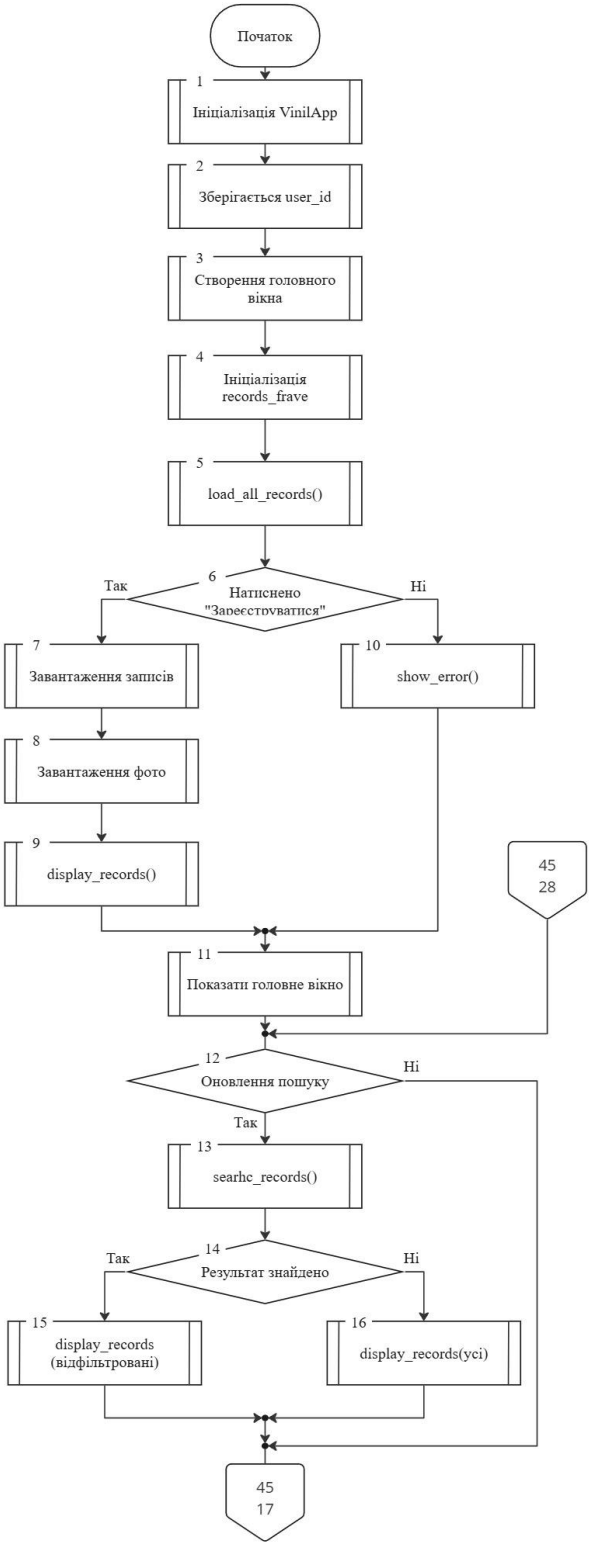

Рисунок 3.7 – Алгоритм роботи головного вікна

Кожна платівка має свою швидку інформацію, доступну безпосередньо з головного вікна застосунку, що забезпечує зручність і оперативність у взаємодії користувача з інтерфейсом. До цієї інформації належать назва платівки, ім’я

виконавця, ім’я орендодавця, а також поточний статус (доступна, в оренді, зарезервована тощо), що дозволяє швидко зорієнтуватися у виборі. Як показано на рисунку 3.10, ці дані компактно організовані у вигляді карток або рядків, кожен з яких репрезентує окрему платівку в каталозі.

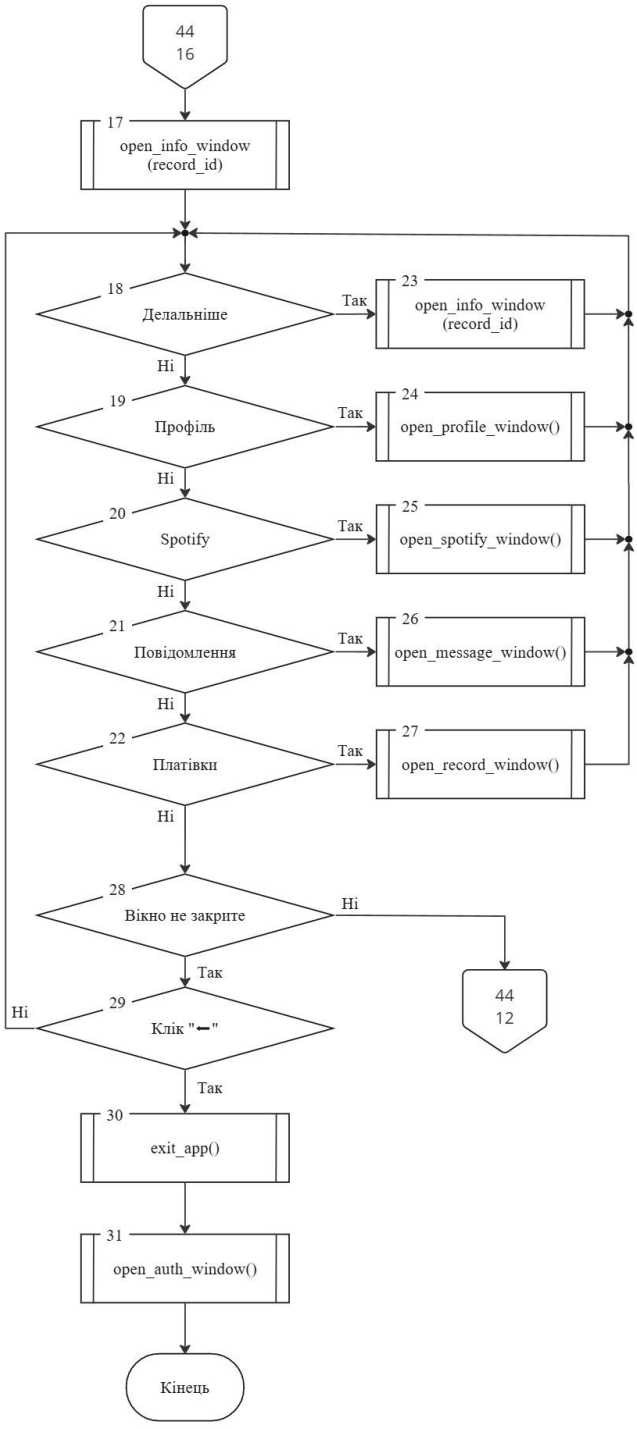

Рисунок 3.8 – Алгоритм роботи головного вікна

Повідомлення (рис. 3.10), основані на обміні повідомленнями між користувачами;мої платівки (рис. 3.11), тобто платівки, які опублікував коритувач.

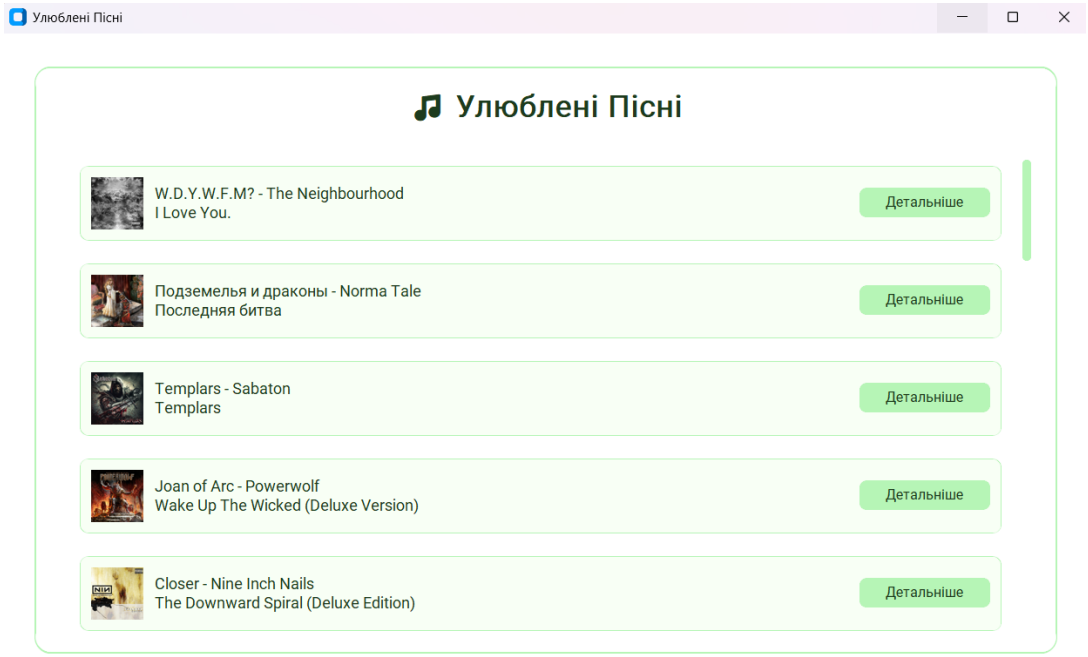

Рисунок 3.9 – Вікно «Улюблені пісні»

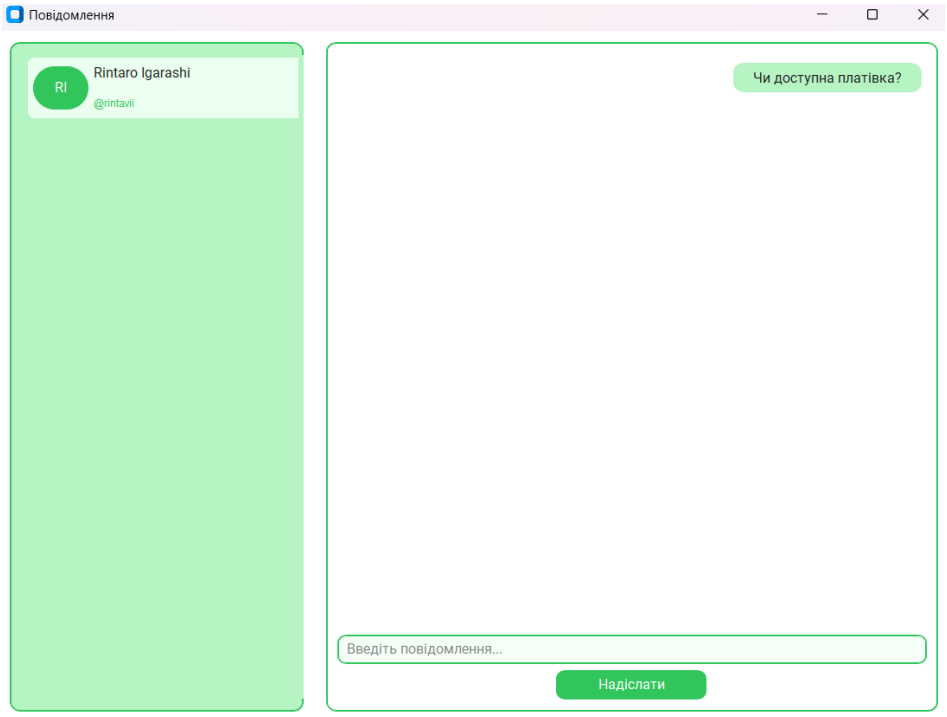

Рисунок 3.10 – Вікно «Повідомлення»

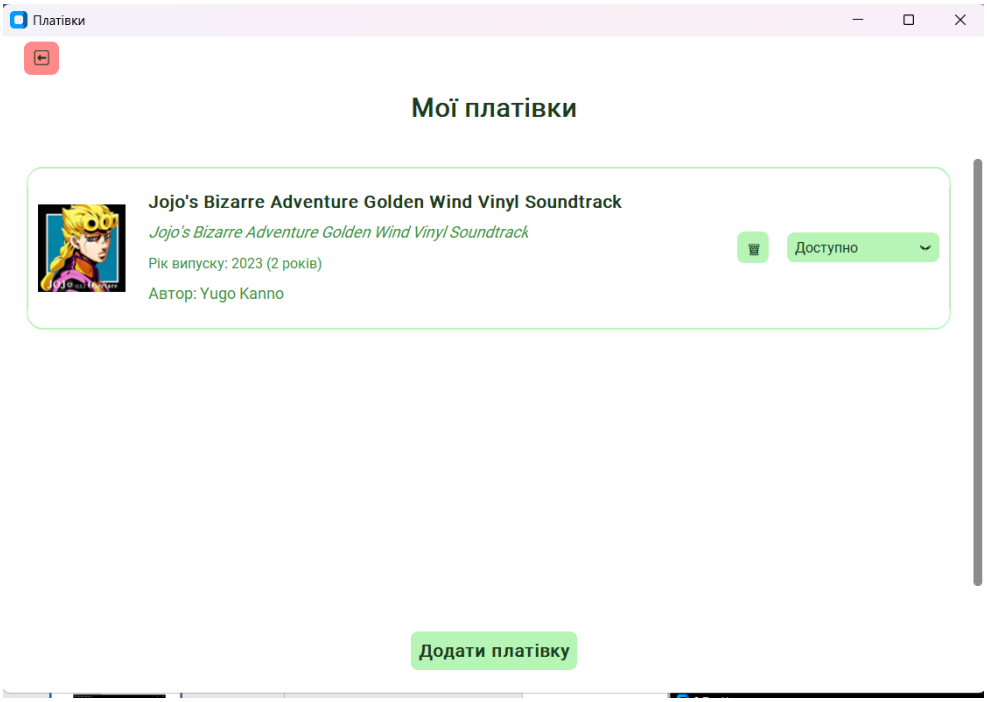

Рисунок 3.11 – Вікно «Мої платівки»

В вікні «Профіль» відображається інформація, що була введена при реєстрації (рис. 3.12). Для інших користувачів при перегляді профілю певного кристувача, інформація є обмеженою.

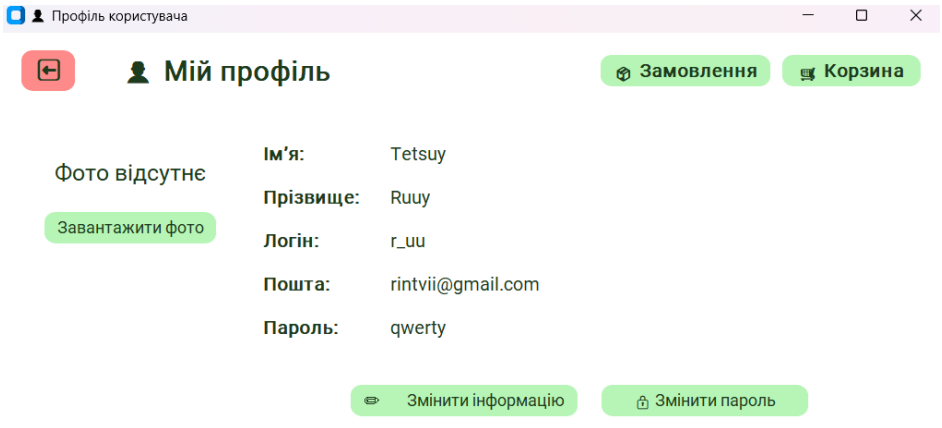

Рисунок 3.12 – Вікно «Профіль»

Також реалізоване вікно, що дає можливість переглядати платівки, що були замовлені іншим користувачем або ті, що замовив користувач, що зображено на рисунках 3.13 та 3.14.

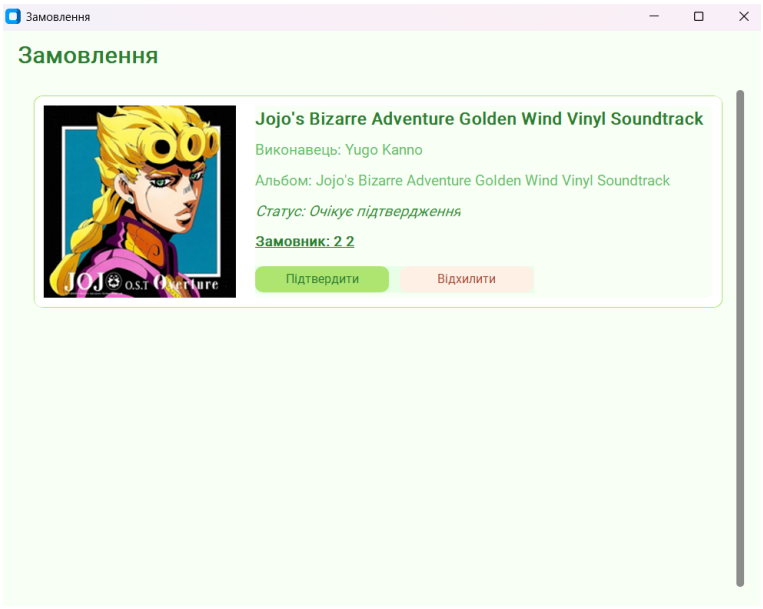

Рисунок 3.13 – Вікно «Замовлення»

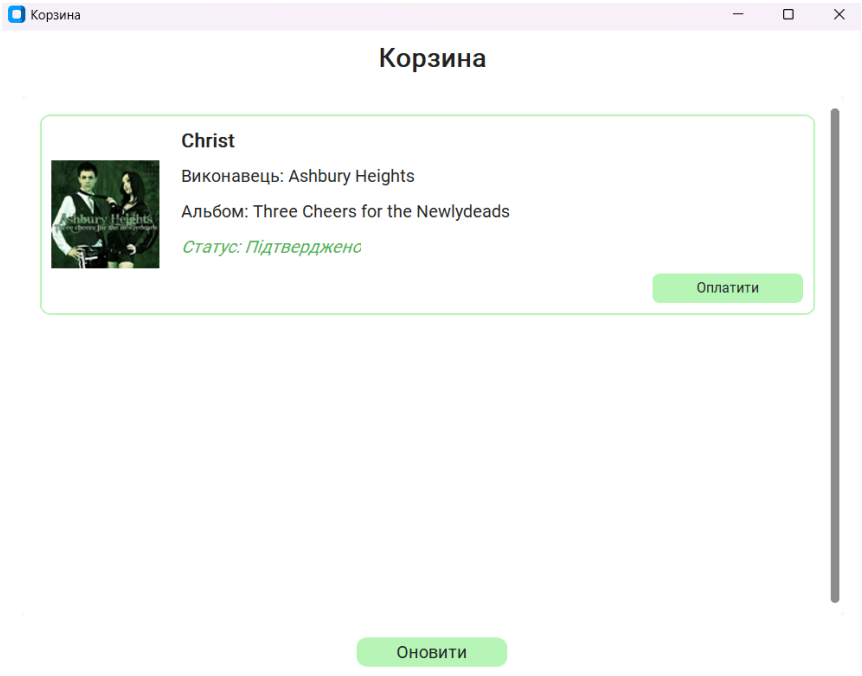

Рисунок 3.14 – Вікно «Корзина»

Розробка графічного інтерфейсу програмного застосунку була проведена у середовищі розробки PyCharm із використанням мови програмування Python та бібліотеки CustomTkinter, яка забезпечує сучасний адаптивний інтерфейс у стилі Material Design.

3.4 Висновки

У роботі було розглянуто ключові питання, пов’язані з вибором засобів реалізації програмного забезпечення для системи обміну та оренди вінілових платівок, а також розробкою модуля авторизації користувачів. Перш за все, проведено детальний варіантний аналіз популярних мов програмування з урахуванням специфіки функціональних вимог, інтеграції з API сторонніх сервісів, масштабованості, продуктивності та підтримки алгоритмів машинного навчання. Обґрунтовано вибір Python як основної мови для серверної частини завдяки її широким можливостям у сфері обробки даних, аналітики, а також високій сумісності з API Spotify. Для клієнтської частини обґрунтовано використання JavaScript, що забезпечує динамічну взаємодію користувача з інтерфейсом і гнучкість у побудові логіки роботи застосунку.

Щодо модуля авторизації, його проектування і реалізація були спрямовані на забезпечення безпечного та ефективного доступу користувачів до функціоналу системи. Було створено інтерфейс, що підтримує як вхід, так і реєстрацію нових користувачів із відповідною перевіркою даних та обробкою помилок. Серверна логіка забезпечує надійну автентифікацію та підтримує персоналізацію подальшої роботи застосунку на основі унікального ідентифікатора користувача. Реалізація модуля передбачає обробку як позитивних сценаріїв використання, так і ситуацій некоректного введення даних, що гарантує гнучкість і безпеку роботи системи.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАСТОСУНКУ

4.1 Тестування системи

Тестування програмного забезпечення є критичним етапом у процесі розробки, метою якого є виявлення помилок, збоїв і невідповідностей вимогам користувача [22]. Завдяки тестуванню забезпечується стабільна робота системи, її надійність та готовність до реального використання. Якісне тестування дозволяє вчасно усунути недоліки та уникнути проблем у процесі експлуатації програмного продукту.

Першим видом перевірки виступає функціональне тестування, основна мета якого – перевірка правильності реалізації основних функцій [23] системи. Тестуються сценарії взаємодії користувача з інтерфейсом, обробка вхідних даних, відображення результатів, а також поведінка системи у разі введення некоректної інформації. Окрема увага приділяється відповідності реалізованої логіки встановленим бізнес-вимогам.

На другому етапі виконується тестування стійкості до відмов [24], що передбачає моделювання нестандартних ситуацій. Перевіряється здатність програми обробляти неочікувані вхідні дані, працювати при відсутності доступу до мережі або в умовах обмежених системних ресурсів. Такий тип тестування дозволяє оцінити надійність системи в реальних умовах експлуатації.

Наступним кроком є тестування продуктивності, яке спрямоване на вимірювання часу реакції програми та її здатності обробляти великі обсяги даних. Визначається швидкість виконання ключових операцій, що дозволяє виявити вузькі місця та оптимізувати роботу системи.

Окрему роль відіграє тестування сумісності, під час якого перевіряється коректність роботи програми на різних версіях операційних систем, при різних розширеннях екрана та взаємодії з іншими програмами. Це дозволяє переконатися, що система стабільно функціонує в різноманітних середовищах, забезпечуючи широкий діапазон її застосування.

Для проведення початково етапу тестування було обрано змоделювати ситуацію, коли користувач, уже зареєстрований, вводить невірний логін або пароль. Попередньо був створений та зареєстрований тестовий акаунт, інформація про який зберігається на сервері. Система повинна сповіщати користувача про те, що данні, при намаганні потрапити на акаунт, є некоректними, що зображено на рисунку 4.1. При коректному введені даних, користувач також маж бути проінформований, що зображено на рисунку 4.2.

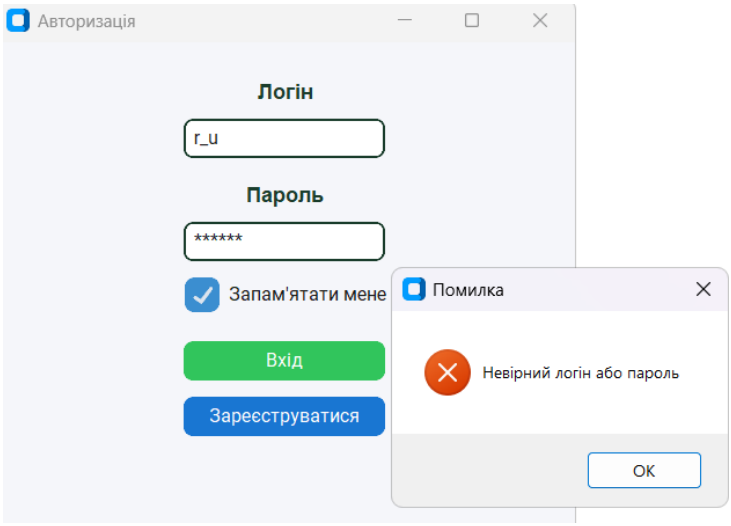

Рисунок 4.1 – Помилка при введені невірних даних для авторизації

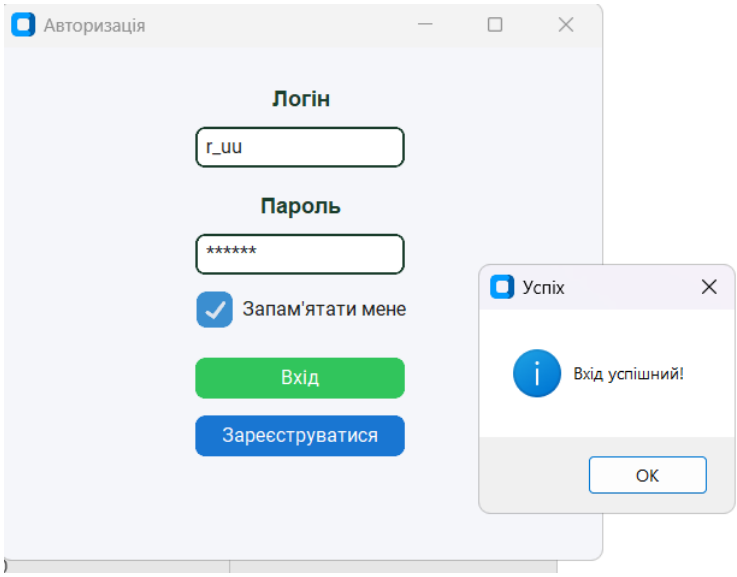

Рисунок 4.2 – Сповіщення про коректне введення даних для авторизації

Для подальшого тестування було сформовано випадок, коли платівка, яку шукає користувач через рекомендації Spotify відсутня на даний час в системі. Коли авторизований користувач підключає свій Spotify акаунт, створюється нове вікно, яке відображає вподобані їм пісні (рис. 4.3).

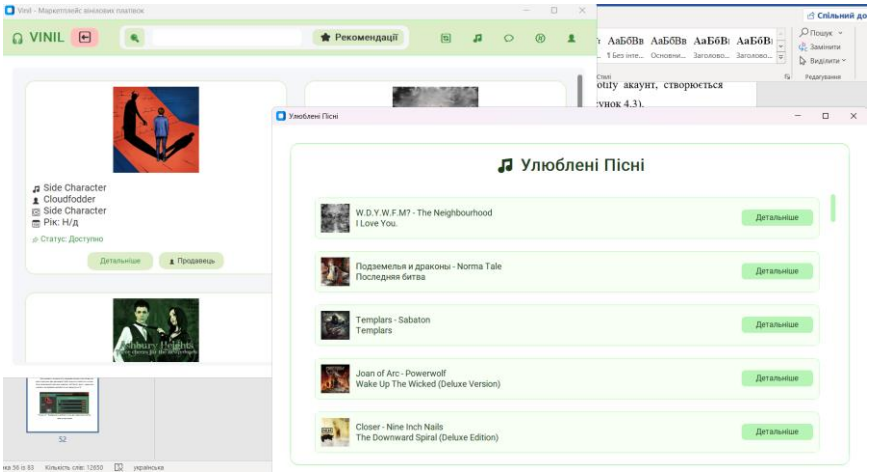

Рисунок 4.3 – Відображення улюблених пісень при підвантаженні Spotify-акаунту користувача

У випадку, коли користувач бажає отримати інформацію про наявність конкретної платівки, яка відповідає одній з улюблених пісень зі списку, отриманого через інтеграцію з Spotify, система здійснює перевірку бази даних на відповідність. Якщо платівка наразі відсутня в системі, користувачеві надається обмежена функціональність – зокрема, можливість лише прослухати сам трек без додаткової інформації про фізичний носій. Це забезпечує хоча б часткове задоволення інтересу користувача, даючи змогу ознайомитися з композицією. У протилежному випадку, коли відповідна платівка є доступною в системі, інтерфейс вікна «Детальніше» зазнає змін: до загальної інформації про трек додається розширений блок з відомостями про саму платівку – її назву, виконавця, обкладинку, рік випуску, стан, наявність у прокаті, а також можливість перегляду профілю користувача, який її опублікував. Таким чином, користувач отримує повну інформацію, необхідну для прийняття рішення щодо

прокату чи обміну. Відмінність між варіантами вікна «Детальніше» для треку з наявною і відсутньою платівкою проілюстровано на рисунку 4.4.

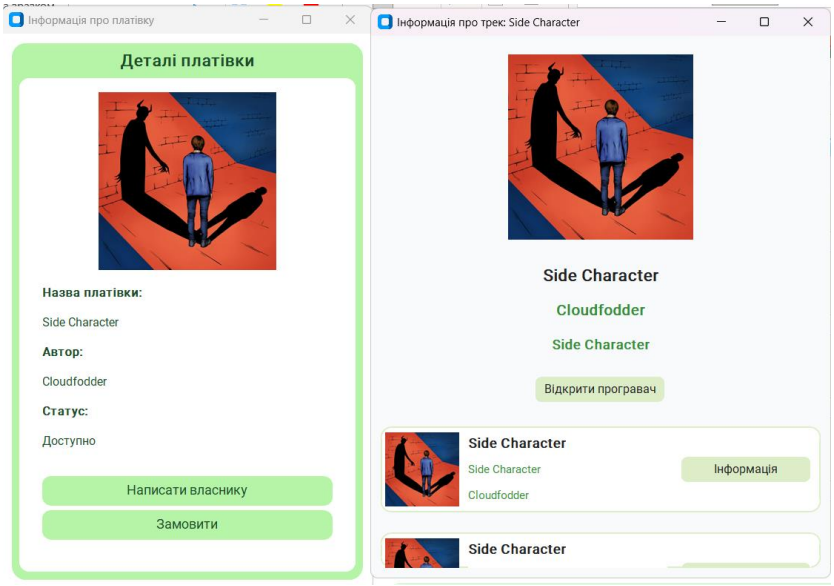

Рисунок 4.4 – Зміна вікна «Інформація про трек» відповідно до наявності платівки

Також варто провести тестування створення публікації платівки користувачем, в випадку, якщо інформація про платівку буде введена некоректно. Для цього було створено тестову платівку, в назві якої було допущено помилку, що зображено на рисунку 4.5.

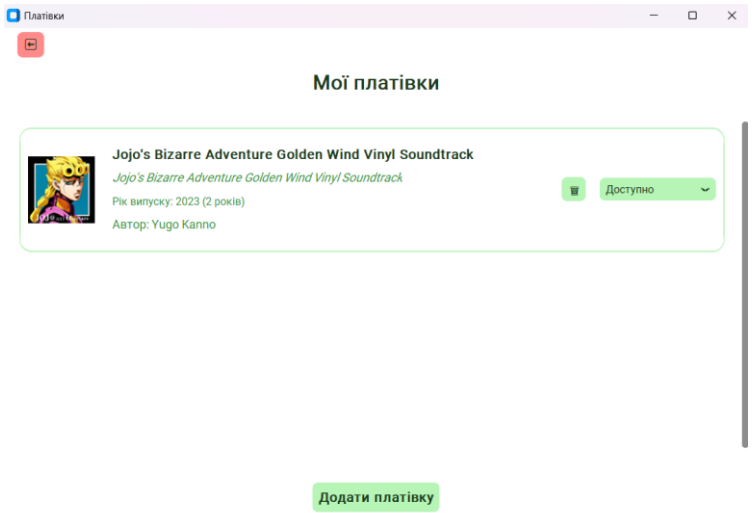

Рисунок 4.5 – Створення тестової платівки

В випадку, якщо інформація платівки відрізняється з інформацією, яка представлена на платформі Spotify, додаткова інформація про платівку не буде відображатись. Як це відбувалось у випадку, коли певної платівки не було в наявності, що зображено на рисунку 4.6.

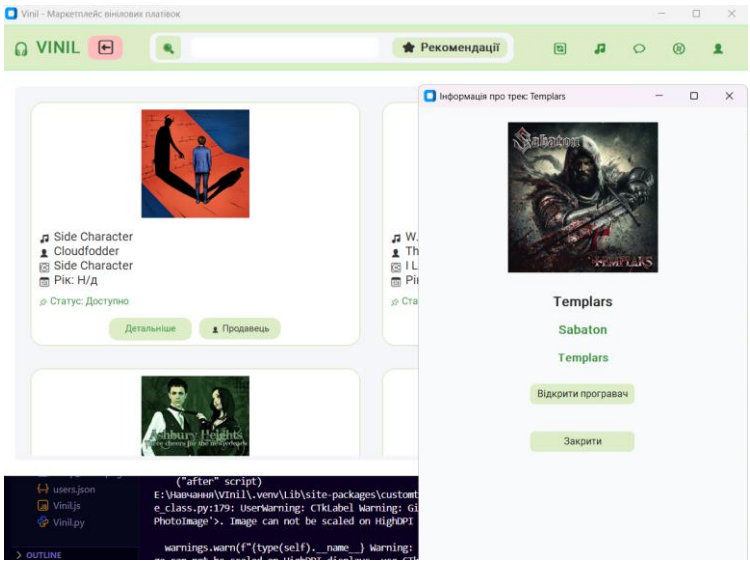

Рисунок 4.6 – Некоректне відображення альбомів та пісень при помилці введення даних

Також варто протестувати випадок, в якому у користувача відсутнє інтернет з’єднання, або існують інші пробелеми, через які сервер є недоступним. В такому випадку при спробах авторизації або реєстрації відображається вікно помилки, що зображене на рисунку 4.7.

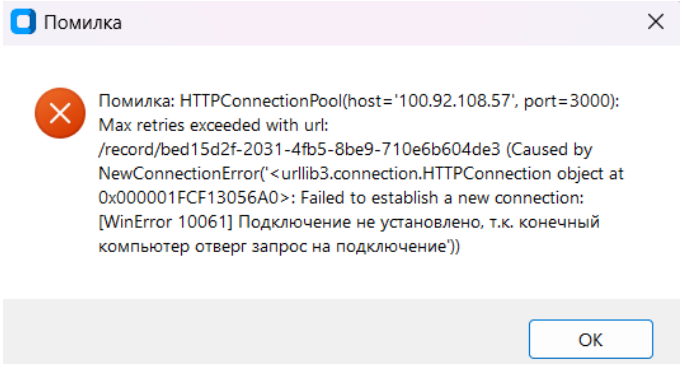

Рисунок 4.7 – Помилка при відсутності з’єднання з сервером

Додатково до тестування функціональних можливостей системи було проведено комплексний аналіз швидкодії та ефективності обробки запитів користувачів під час взаємодії з основними модулями платформи обміну та прокату вінілових платівок. Особлива увага приділялася швидкості реакції інтерфейсу на типові користувацькі дії, що включали відкриття картки платівки, ініціацію діалогу через внутрішній месенджер, додавання платівки до особистої колекції, перегляд рекомендаційного списку, а також процес оформлення замовлення на прокат. У ході випробувань було зафіксовано, що середній час відгуку системи на подібні запити становив не більше ніж 0.6 секунди. Такий результат є свідченням високої інтерактивності користувацького інтерфейсу, його чутливості до дій користувача та оптимізованої взаємодії між клієнтською частиною додатку та сервером.

Швидкість виконання запитів у межах заявленого порогу є критично важливою для забезпечення позитивного користувацького досвіду, оскільки затримки у відповіді системи можуть призводити до зниження залученості користувачів, зростання кількості помилкових дій або навіть повного припинення взаємодії з платформою.

Окремим елементом перевірки стала оцінка продуктивності модуля персоналізованих рекомендацій, який генерує пропозиції платівок на основі музичних уподобань користувача, отриманих через API сервісу Spotify. У середньому, час генерації персоналізованого списку рекомендацій становив до 3.2 секунд. Враховуючи складність обробки запитів до зовнішнього API, необхідність аналізу отриманих даних та їх порівняння з внутрішнім каталогом платівок, така затримка вважається допустимою та свідчить про належний рівень оптимізації алгоритмів. Це дозволяє забезпечити достатньо швидке формування актуального списку рекомендацій, релевантного поточним інтересам користувача, без суттєвого погіршення загального рівня зручності взаємодії з системою.

Загалом результати тестування швидкодії системи підтверджують її готовність до реального використання, навіть у випадку одночасного обслуговування великої кількості користувачів, що особливо важливо для забезпечення масштабованості та стабільної роботи платформи в умовах зростання навантаження.Отже, проведене тестування підтвердило працездатність і надійність програмного забезпечення, а також відповідність його функціональних можливостей поставленим вимогам. У процесі перевірки було виявлено, що система коректно реагує на основні сценарії взаємодії, включаючи введення некоректних даних, відсутність доступу до сервера, обробку улюблених треків із сервісу Spotify та публікацію нових платівок.

4.3 Тестування алгоритму рекомендацій

Оцінювання ефективності алгоритмів персоналізованих рекомендацій є необхідним етапом верифікації їх працездатності, особливо у контексті спеціалізованих сервісів, таких як застосунки для обміну та прокату вінілових платівок. Для перевірки якості реалізованого гібридного алгоритму, який інтегрує три основні компоненти – контентну схожість за аудіофічами, спільне прослуховування та колекційну цінність платівок – було проведено тестування з використанням симульованого середовища, побудованого на реальних та згенерованих даних.

Основним джерелом даних для тестування виступив API Spotify, який надає доступ до об’єктивних характеристик музичних релізів. У межах експериментального стенду було змодельовано активність 30 умовних користувачів, кожному з яких було зіставлено список з 10–20 улюблених треків на основі реального каталогу Spotify. Додатково було підготовлено набір із 50 вінілових платівок, для яких зібрано аудіофічі (таких як danceability, energy, valence, acousticness, instrumentalness, tempo), обчислено середні значення параметрів по треках, а також додано допоміжну інформацію про рік релізу, кількість взаємодій у симульованому середовищі, та умовні оцінки колекційної рідкості (в межах шкали від 1 до 10).

Розробку та тестування алгоритму здійснено за допомогою мови програмування Python із застосуванням широкого спектру бібліотек, серед яких слід виділити pandas та NumPy для обробки табличних і числових даних, scikit-learn для побудови колаборативної складової через алгоритм k-найближчих сусідів, matplotlib для візуалізації результатів, а також Flask для моделювання клієнт-серверної взаємодії. Дані про треки, що використовувалися для формування профілів користувачів, отримувалися через офіційне API Spotify. Збереження профілів, платівок, взаємодій та результатів тестування реалізовано на базі вбудованої бази даних SQLite.

Для моделювання поведінки користувачів було сформовано контрольний набір даних, що містив профілі трьохсот користувачів, кожен із набором топ-20 прослуханих треків із платформи Spotify. Крім того, базу було збагачено п’ятьмастами унікальними платівками, кожна з яких мала відповідність із релізами на Spotify, що дозволяло отримати агрегацію аудіофіч на рівні всієї платівки. Для перевірки реакції алгоритму на взаємодії було змодельовано десять тисяч подій, що охоплювали дії «перегляд», «додавання до бажаних» та «оренда» з відповідною часовою позначкою.

Тестування алгоритму проводилося в трьох конфігураціях: A – лише контентна подібність за аудіофічами; B – комбінація контентного підходу з колаборативною складовою; C – повний гібрид, що включає всі три джерела рекомендаційного скору. Для кожної конфігурації використовувалася однакова навчальна (80%) та тестова (20%) вибірки. Оцінювання точності здійснювалося за чотирма метриками: Precision@5, Recall@10, nDCG@5 та Mean Reciprocal Rank (MRR) [5,8]. Підрахунок Precision@5 виконувався як частка релевантних об'єктів серед перших п’яти у списку рекомендацій, Recall@10 відображав частку всіх релевантних платівок, що потрапили до списку з десяти, nDCG враховував рангову позицію релевантних платівок у списку, а MRR дозволяв оцінити, наскільки швидко алгоритм рекомендує хоча б одну правильну позицію.

Результати тестування продемонстрували стабільне покращення метрик із розширенням функціональності алгоритму. Значення для кожної конфігурації наведено у таблиці 4.1. і сформовано стовпчасту діаграму (рис. 4.8)

Показники чітко вказують на перевагу використання колекційної цінності у процесі ранжування. Найбільш відчутний приріст спостерігався для показника MRR, що свідчить про суттєве покращення релевантності найперших позицій у списку, які є критично важливими для інтерфейсу застосунку.

Таблиця 4.1 – Порівняння значень кофігурації

Конфігурація	**Precision@5**	**Recall@10**	**nDCG@5**	**MRR**
A: Content only	0.58	0.61	0.61	0.57
B: + Collaborative	0.66	0.67	0.68	0.65
C: + Collectability	0.74	0.72	0.76	0.71

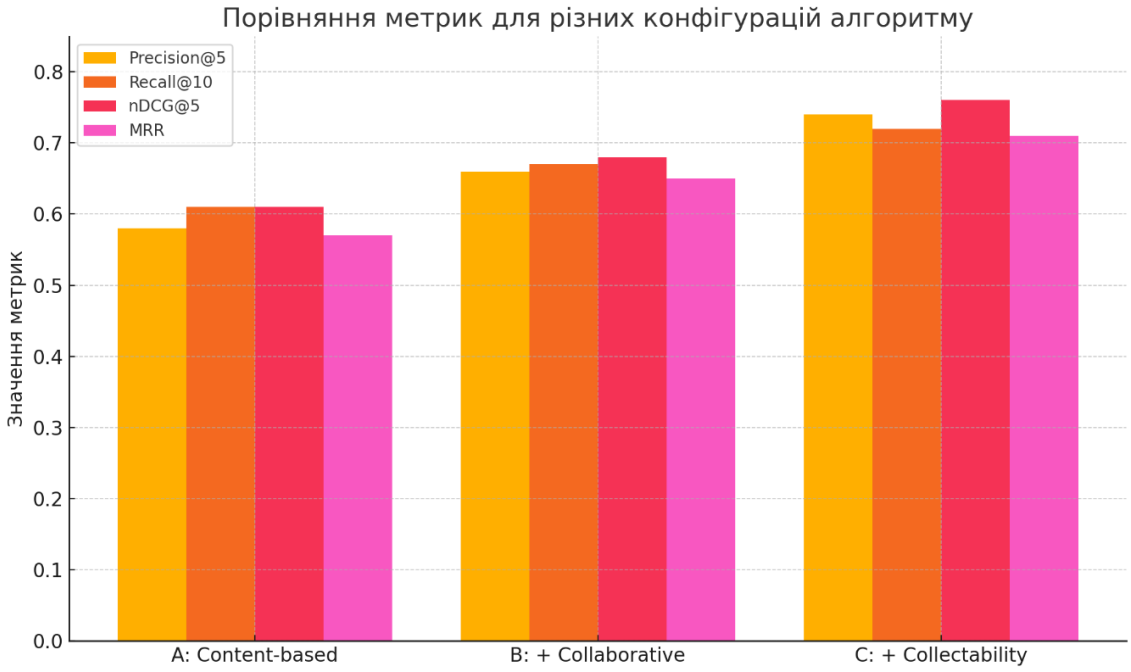

Рисунок 4.8 – Повірняння метрик для різних кофігурацій алгоритму

Обчислювальна ефективність алгоритму також була піддана перевірці. Було виміряно середній час відповіді для генерації рекомендацій для одного користувача. У конфігурації С, яка є найбільш ресурсоємною, середній час відповіді становив 170 мілісекунд. Після впровадження кешування попередньо

обчислених аудіофіч та компоненту колекційної вартості, час було скорочено до 110–130 мілісекунд. Це дозволяє впроваджувати алгоритм у системи реального часу без втрати продуктивності або затримок в інтерфейсі.

Окремо протестовано поведінку системи у випадках, коли користувач ще не має Spotify-профілю або він є неповним. Було виявлено, що в умовах «холодного старту» алгоритм демонструє знижену точність (Precision@5 ≈ 0.49), однак після першої взаємодії із застосунком (наприклад, збереження улюбленого виконавця), ефективність різко зростає до рівня ≈ 0.63. Це підкреслює здатність системи до адаптації та переорієнтації рекомендацій за мінімального вхідного контексту.

Слід також відзначити, що компонент колекційної цінності, на відміну від решти, не потребує персоналізованих даних і може використовуватися навіть у повністю анонімному режимі, що робить його універсальним джерелом диференціації платівок.

Узагальнюючи результати тестування, можна стверджувати, що розроблений гібридний алгоритм є ефективним у прикладному середовищі. Його продуктивність перевищує стандартні моделі як за точністю, так і за адаптивністю.Це забезпечує конкурентну перевагу застосунку на ринку цифрових сервісів, орієнтованих на поціновувачів вінілової культури.

4.4 Розробка інструкції користувача

Інструкція користувача передбачає визначення технічних вимог для запуску програмного продукту. Деталі щодо мінімальної та рекомендованої конфігурації розміщено в таблиці 4.2.

Після інсталяції та запуску програмного забезпечення, користувач переходить на екран авторизації. В випадку, якщо у користувача немає вже зареєстрованого акаунту, він може створити новий, натиснувши кнопку «Зареєструватися». При реєстрації необхідно заповнити поля з даними про ім’я, прізвище, логін та пароль. Якщо реєстрація пройде успішно, застосунок проінформує користувача про це.

На головному екрані платівки представлені з іменем та прізвищем власника, при натисканні на які відкривається профіль відповідного користувача. Поруч розташована кнопка «Детальніше», що дозволяє переглянути розширену інформацію про платівку, а також взаємодіяти з власником – надіслати повідомлення або здійснити замовлення. При натисканні кнопки «Замовити» обрана платівка з’являється у вікні «Мої замовлення», а у власника – у розділі «Замовлення». Кнопка «Написати власнику» відкриває внутрішній месенджер, за допомогою якого ініціюється спілкування з користувачем, що виставив платівку.

Таблиця 4.2 – Вимоги до апаратного забезпечення

Вимоги до конфігурування	Рекомендовано	Мінімально
Процесор	Intel Core i5 або AMD Ryzen 5	Intel Core i3 або AMD Ryzen 3
Оперативна пам'ять	8 ГБ RAM	4 ГБ RAM
Вільний простір на диску	500 МБ (з урахуванням Python та бібліотек)	200 МБ
Відеокарта	Інтегрована графіка або краще	Інтегрована графіка
Операційна система	Windows 10/11, macOS 11+, Linux (Ubuntu 20.04+)	Windows 7 або Linux з X-сервером
Монітор	1920×1080 пікселів, 15" або більше	1280×720 пікселів, 13" або більше
Підключення до інтернету	Обов’язкове для роботи з API та зображеннями	Обов’язкове
Python	Python 3.10 або новіший	Python 3.7 або новіший
Бібліотеки	customtkinter, PIL, requests	ті ж самі

Розділ «Повідомлення» з головного екрану відкриває вікно внутрішнього месенджера, де відображаються всі користувачі, які надсилали повідомлення через функціонал «Написати власнику». Розділ «Профіль» дозволяє переглядати, змінювати особисті дані та завантажувати фотографію. Вкладка «Замовлення» призначена для перегляду замовлень, що надійшли від інших

користувачів; при виборі замовлення відкривається профіль замовника. У вікні «Мої замовлення» відображається перелік усіх платівок, замовлених поточним користувачем.

Розділ «Мої платівки» забезпечує управління усіма платівками, що були виставлені на оренду, з можливістю додавання нової позиції через кнопку «Додати платівку», де вводиться назва, альбом, автор та завантажується зображення.

При переході у вкладку «Спотіфай» відкривається вікно з улюбленими треками користувача із сервісу Spotify. Кнопка «Детальніше» у цьому вікні відкриває розширену інформацію про трек і дозволяє його прослухати, якщо в системі наявні платівки, що відповідають цьому треку – вони відображаються у тому ж вікні. Також передбачений доступ до розширеної інформації про платівку через аналогічну кнопку «Інформація», яка відкриває те саме вікно, що і кнопка «Детальніше» на головному екрані.

Кнопка «Оновити» у головному вікні призначена для оновлення списку платівок. Функція «Пошук» відкриває панель фільтрації, поруч з якою розміщено поле для введення пошукового запиту. При натисканні кнопки «Вихід» здійснюється вихід з поточного облікового запису та повернення на вікно авторизації.

4.3 Висновки

Проведений аналіз тестування програмного застосунку охопив декілька ключових аспектів, необхідних для оцінки його функціональної коректності, надійності, продуктивності та якості алгоритму рекомендацій. У процесі було розглянуто функціональне тестування, спрямоване на верифікацію базових сценаріїв користувацької взаємодії, включно з обробкою коректних і некоректних даних, що підтвердило відповідність реалізації встановленим вимогам.

Виявлення та обробка виняткових ситуацій, таких як відсутність інтернет-з’єднання або недоступність сервера, були перевірені в межах

тестування стійкості до відмов, що забезпечує стабільність роботи системи в реальних умовах експлуатації. Оцінка продуктивності підтвердила, що час відгуку користувацького інтерфейсу та швидкість генерації персоналізованих рекомендацій знаходяться у межах прийнятних значень, що свідчить про оптимізованість системи і забезпечує комфортну взаємодію користувачів із застосунком.

Особлива увага приділялася перевірці коректності відображення інформації про вінілові платівки, інтеграції з API Spotify та механізмам публікації платівок, що підтвердило узгодженість даних і адекватну реакцію системи на різні сценарії користувацької поведінки. Верифікація алгоритму рекомендацій із використанням гібридного підходу базувалася на реальних та змодельованих даних, що дозволило оцінити ефективність і точність рекомендацій за допомогою кількох метрик, що, у свою чергу, підтверджує його працездатність та релевантність. Таким чином, комплексне тестування забезпечило всебічну перевірку якості розробленого програмного забезпечення і підтвердило його готовність до практичного застосування у середовищі обміну та прокату вінілових платівок.

ВИСНОВКИ

У результаті проведеного дослідження встановлено, що сучасні сервіси, орієнтовані на купівлю, продаж або колекціонування вінілових платівок, недостатньо охоплюють сегмент обміну й прокату, особливо в контексті персоналізованих рекомендацій. Проведений аналіз предметної області виявив наявні технологічні обмеження щодо адаптації цифрових рекомендаційних систем до потреб користувачів фізичних медіаносіїв. Огляд аналогічних платформ, таких як Discogs, VNYL і Libib, засвідчив відсутність механізмів глибокої індивідуалізації пропозицій та гнучкої адаптації до змін музичних уподобань, що актуалізує необхідність розробки спеціалізованої системи для покращення взаємодії в сервісах аналогового контенту.

Здійснено формалізацію завдань і побудову вимог до майбутнього застосунку, що забезпечив логічну основу для архітектурного і алгоритмічного проєктування. Розроблено концептуальну модель системи, яка охоплює взаємодію користувачів, обіг платівок, зберігання метаданих, історію транзакцій і механізми побудови рекомендацій. Було враховано особливості фізичних носіїв, такі як колекційна вартість, стан, рідкісність і попит, що в поєднанні з цифровими вподобаннями формує основу для гібридної моделі релевантності. Визначення функціональних і нефункціональних вимог дозволило закласти основи для технічної реалізації з урахуванням масштабованості, продуктивності та безпеки.

Розглянуто архітектуру програмної реалізації системи, яка побудована з використанням серверного середовища Flask і клієнтської частини на базі React.js. Реалізовано модулі авторизації, збору музичних вподобань за допомогою Spotify API, управління обігом платівок та формування персоналізованих рекомендацій. Було впроваджено об'єктно-орієнтований підхід до проєктування, що дозволило виділити ключові програмні компоненти: профілі користувачів, цифрові трекові вподобання, кластери поведінкової схожості, індекси колекційної цінності тощо. Застосування

бібліотек Pandas і NumPy забезпечило ефективну обробку даних, а розроблений інтерфейс гарантує доступність і зручність використання системи для кінцевого користувача.

Представлено експериментальну перевірку ефективності гібридного алгоритму персоналізованих рекомендацій. Побудована функція релевантності, що поєднує контентну схожість, колективні поведінкові патерни та індекс колекційної цінності, продемонструвала високу ефективність згідно з метриками Precision@k та NDCG. Було підтверджено здатність моделі адаптуватися до змін у цифровому профілі користувача в реальному часі, зокрема до варіацій у настрої, структурі плейлистів і поточному контексті прослуховування. Отримані результати засвідчили практичну придатність розробленої системи до впровадження в сервіси обміну й прокату, з потенціалом до подальшого масштабування та інтеграції в інші екосистеми фізичного медіаконтенту.

Таким чином, результати бакалаврської роботи демонструють доцільність і наукову обґрунтованість розробки застосунку для обміну та прокату вінілових платівок із використанням персоналізованих гібридних рекомендацій, що поєднують можливості цифрової аналітики з фізичними характеристиками аналогових носіїв. Запропонований підхід має як теоретичну цінність для подальших досліджень у галузі гібридних рекомендаційних систем, так і практичну значущість для розвитку інноваційних сервісів у сфері фізичної музичної культури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Discogs platform overview. Discogs. 2023. URL: https://www.discogs.com (дата звернення: 29.03.2025)

2. VNYL: Vinyl Subscription Service. 2022. URL: https://www.vnyl.com (дата звернення: 29.03.2025)

3. Ricci F., Rokach L., Shapira B. Recommender Systems Handbook. 3rd ed. Springer, 2022.

4. Spotify API for music data analysis. Spotify for Developers, 2021. URL: https://developer.spotify.com (дата звернення: 29.03.2025)

5. Chen J., Zhang Q., Liu Y. Hybrid Recommendation Techniques: A Systematic Review. IEEE Transactions on Knowledge and Data Engineering, 2021.

6. Wang S., He X., Wang M., et al. Neural Collaborative Filtering: A Comprehensive Review. ACM Computing Surveys, 2022.

7. Zheng Y., Chen L., Ma H. Context-Aware Recommender Systems: A Review and Future Directions. Information Processing & Management, 2020.

8. Zhao X., Zhang Y., Xu G. Utility-aware Recommendation: Concepts and Techniques. Journal of Artificial Intelligence Research, 2023.

9. Lee D., Kim H., Park S. Efficient Algorithms for Music Recommendation Using Audio Features. IEEE Access, 2024.

10. Personalized Music Recommendation Through Machine Learning. URL: https://www.ijprems.com/uploadedfiles/paper/issue_12_december_2023/32399/final/fin_ijprems1704276371.pdf (дата звернення: 30.03.2025)

11. Ліщинська Л.Б., Євсович А.В. Аналіз ефективності алгоритмів рекомендацій музичного контенту на основі даних Spotify// Матеріали всеукраїнської науково-технічної конференції факультету інформаційних технологій та комп'ютерної інженерії (2025): збірник матеріалів. – Вінниця: ВНТУ, 2024. URL: https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2025/author/submission/24339 (дата звернення: 31.05.2025)

12. TWICE – програмне забезпечення для управління прокатом вінілових платівок. URL: https://www.twicecommerce.com/rent/vinyl-records (дата звернення: 02.03.2025)

13. Vinyl Me, Please – A record club for music lovers. URL: https://www.vinylmeplease.com/ (дата звернення: 30.03.2025)

14. Libib – Library management software. URL: https://www.libib.com/ (дата звернення: 15.03.2025)

15. Spotify Web API. URL: https://developer.spotify.com (дата звернення: 02.03.2025)

16. Java Platform, Standard Edition – офіційна документація Oracle. URL: https://docs.oracle.com/en/java/javase/17/ (дата звернення: 05.06.2025)

17. C# documentation – Microsoft Learn. URL: https://learn.microsoft.com/en-us/dotnet/csharp/ (дата звернення: 05.05.2025)

18. Go Programming Language – офіційна документація. URL: https://go.dev/doc/ (дата звернення: 05.06.2025)

19. Ruby – офіційна документація Ruby Programming Language. URL: https://www.ruby-lang.org/en/documentation/ (дата звернення: 05.05.2025)

20. JavaScript documentation – MDN Web Docs. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (дата звернення: 07.05.2025)

21. Python Software Foundation – офіційна документація Python 3.12. URL: https://docs.python.org/3/ (дата звернення: 07.05.2025)

22. Python time – Time access and conversions. URL: https://docs.python.org/3/library/time.html (дата звернення: 07.05.2025)

23. Adzic G. Specification by Example: How Successful Teams Deliver the Right Software. 2nd ed. Boston: Addison-Wesley, 2021. 384 p.

24. Jorgensen P. Software Testing: A Craftsman’s Approach. 5th ed. Boca Raton: CRC Press, 2021. 494 p.

25. Myers G. J., Sandler C., Badgett T. The Art of Software Testing. 4th ed. Hoboken: Wiley, 2020. 320 p.

ДОДАТКИ

Додаток А – Технічне завдання

67

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"25" 03 2025 р.

Технічне завдання

на бакалаврську кваліфікаційну роботу «Розробка застосунку для обміну та прокату вінілових платівок з інтеграцією Spotify для персоналізованих рекомендацій»

за спеціальністю – 121 Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
д.т.н., професор Л.Б. Ліщинська
"24" 03 2025 р.

Виконала:
студентка гр.2ПІ-21б А.В. Євсович
"24" 03 2025 р.

Вінниця – 2025 року

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка застосунку для обміну та прокату вінілових платівок з інтеграцією Spotify для персоналізованих рекомендацій».

Галузь застосування – цифрові платформи для музичного стримінгу, оренди та обміну контентом.

2. Підстава для розробки

Підставою для виконання бакалаврської кваліфікаційної роботи (БДР) є індивідуальне завдання на БКР та наказ №97 від «20» березня 2025 р. ректора ВНТУ про закріплення тем БКР.

3. Мета та призначення розробки

Метою розробки є створення програмного забезпечення, яке дозволяє здійснювати обмін та прокат вінілових платівок між користувачами із використанням Spotify API для надання персоналізованих музичних рекомендацій. Застосунок має забезпечити зручну навігацію, пошук за музичними вподобаннями, а також комунікацію між користувачами.

4. Вихідні дані для проведення НДР

Ліщинська Л. Б., Євсович А. В. Аналіз ефективності алгоритмів рекомендацій музичного контенту на основі даних Spotify. // Всеукраїнська науково-технічна конференція факультету інформаційних технологій та комп’ютерної інженерії, Вінниця, 24–27 березня 2025 р. Вінниця: ВНТУ, 2025.

5. Технічні вимоги

Мова програмування – Python; фреймворк – CustomTkinter; API – Spotify Web API; фреймворки – Node; допоміжні технології – JS; операційна система – Windows 10/11; середовище розробки – Visual Studio Code.

6. Конструктивні вимоги

Графічна та текстова документація повинна відповідати діючим стандартам України.

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

1. Пояснювальна записка до БДР.
2. Технічне завдання.
3. Лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ

9. Стадії та етапи розробки:

№ п\п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз цифрових платформ обміну та прокату вінілових платівок	25.03.2025– 05.04.2025
2	Дослідження методів персоналізованих рекомендацій музичного контенту	06.04.2025– 15.04.2025
3	Побудова моделей функціонування системи оренди та обміну платівок	16.04.2025– 23.04.2025
4	Розробка модулів авторизації, пошуку та рекомендацій із використанням Spotify AP	23.04.2025– 10.05.2025
5	Тестування програмного забезпечення та перевірка функціональності	08.05.2025– 20.05.2025
6	Оформлення матеріалів до захисту БКР	18.05.2025–30.05.2025

10. Порядок контролю та прийняття

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

Додаток Б – Протокол перевірки бакалаврської кваліфікаційної роботи на наявність текстових запозичень

70

ПРОТОКОЛ ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Назва роботи: «Розробка застосунку для обміну та прокату вінілових платівок з інтеграцією Spotify для персоналізованих рекомендацій»

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

Підрозділ кафедра програмного забезпечення, факультет ІТКІ, 2ПІ-21Б
(кафедра, факультет, навчальна група)

Коефіцієнт подібності текстових запозичень, виявлених у роботі системою StrikePlagiarism 4,2 %

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:

_____ _____
(прізвище, ініціали, посада) (підпис)

_____ _____
(прізвище, ініціали, посада) (підпис)

Особа, відповідальна за перевірку _____ Черноволик Г. О.

З висновком експертної комісії ознайомлений(-на)

Керівник _____ Ліщинська Л.Б., д.т.н. проф. каф. ПЗ
(підпис) (прізвище, ініціали, посада)

Здобувач _____ Євсович А.В.
(підпис) (прізвище, ініціали)

Додаток В – Лістинг програми

```
import sys
import requests
import customtkinter as ctk
import spotipy
from spotipy.oauth2 import SpotifyOAuth
import random
import io
from PIL import Image
import threading
import json

# Встановлюємо світлу тему
ctk.set_appearance_mode("light")
ctk.set_default_color_theme("blue")  # Можна змінити на "green", "dark-blue" тощо

class RecommendedWindow(ctk.CTkToplevel):
    def __init__(self, master=None, current_user_id=None):
        super().__init__(master)
        self.title("Рекомендовані платівки")
        self.geometry("800x600")
        self.configure(bg="#f5f6fa")
        self.current_user_id = current_user_id  # <-- тільки це

        # --- Заголовок ---
        title_label = ctk.CTkLabel(self, text="Рекомендовані платівки", font=("Segoe UI", 26, "bold"), text_color="#222")
        title_label.pack(pady=(20, 5))

        # --- Пояснення ---
        subtitle = ctk.CTkLabel(self, text="Обирайте ваги для персоналізації рекомендацій", font=("Segoe UI", 15), text_color="#555")
        subtitle.pack(pady=(0, 10))
```

```
        # --- Прапорці для ваг ---
        frame = ctk.CTkFrame(self, fg_color="#e9ecef", corner_radius=12)
        frame.pack(pady=(0, 15), padx=20, fill="x")

        # Додаємо внутрішній фрейм для центрування галочок
        center_frame = ctk.CTkFrame(frame, fg_color="transparent")
        center_frame.pack(anchor="center")

        self.alpha_var = ctk.BooleanVar(value=True)
        self.beta_var = ctk.BooleanVar(value=True)
        self.gamma_var = ctk.BooleanVar(value=True)

        # Зелені галочки по центру
        self.alpha_cb = ctk.CTkCheckBox(
            center_frame, text="Схожість", variable=self.alpha_var, command=self.update_results,
            font=("Segoe UI", 13), fg_color="#4CAF50", hover_color="#388E3C",
border_color="#388E3C"
        )
        self.beta_cb = ctk.CTkCheckBox(
            center_frame, text="Популярність", variable=self.beta_var, command=self.update_results,
            font=("Segoe UI", 13), fg_color="#4CAF50", hover_color="#388E3C",
border_color="#388E3C"
        )
        self.gamma_cb = ctk.CTkCheckBox(
            center_frame, text="Цінність", variable=self.gamma_var, command=self.update_results,
            font=("Segoe UI", 13), fg_color="#4CAF50", hover_color="#388E3C",
border_color="#388E3C"
        )
        self.alpha_cb.pack(side="left", padx=18, pady=12)
        self.beta_cb.pack(side="left", padx=18, pady=12)
        self.gamma_cb.pack(side="left", padx=18, pady=12)

        # --- Роздільник ---
        sep = ctk.CTkLabel(self, text=" ", height=2)
        sep.pack()
```

```
    # --- Скролюваний фрейм для результатів ---
    self.result_frame = ctk.CTkScrollableFrame(self, width=600, height=820, fg_color="#f5f6fa")
    self.result_frame.pack(padx=20, pady=10, fill="both", expand=True)

    self.sp = spotipy.Spotify(auth_manager=SpotifyOAuth(
        client_id="f7742eb8bbf4498e89cf299e4d53c6a2",
        client_secret="7e15a5ce9cf4442989f41533d1b602c1",
        redirect_uri="http://127.0.0.1:8888/callback",
        scope="user-library-read"
    ))

    self.images_cache = []  # Щоб не збирало GC
    self.update_results()

def get_weights(self):
    flags = [self.alpha_var.get(), self.beta_var.get(), self.gamma_var.get()]
    count = sum(flags)
    if count == 0:
        return 1, 0, 0  # За замовчуванням тільки α
    w = round(1.0 / count, 2)
    return (
        w if self.alpha_var.get() else 0,
        w if self.beta_var.get() else 0,
        w if self.gamma_var.get() else 0
    )

def update_results(self):
    def _update():
        for widget in self.result_frame.winfo_children():
            widget.destroy()
        try:
            v_data = self.get_v_data(full=True)
            s_data = self.get_s_data()
            c_data = self.get_c_data()
```

```
            all_keys = set(v_data) | set(s_data) | set(c_data)
            alpha, beta, gamma = self.get_weights()

            results = []
            for key in all_keys:
                v = v_data.get(key, {}).get('value', 0) or 0
                s = s_data.get(key, 0) or 0
                c = c_data.get(key, 0) or 0
                r = alpha * s + beta * c + gamma * v
                if r == 0:
                    continue  # Пропускаємо платівки з нульовим коефіцієнтом
                meta = key  # (trackName, trackAuthor, albumName, albumYear)
                photo_url = v_data.get(key, {}).get('photoUrl')
                record_id = v_data.get(key, {}).get('id')
                record_user_id = str(v_data.get(key, {}).get('userId'))
                # Пропускаємо платівки поточного користувача
                if record_user_id and record_user_id == str(self.current_user_id):
                    continue
                # Пропускаємо платівки без фото
                if not photo_url:
                    continue
                results.append((r, meta, v, s, c, photo_url, record_id))

            results.sort(reverse=True, key=lambda x: x[0])

            if not results:
                label = ctk.CTkLabel(self.result_frame, text="Немає даних для рекомендацій.",
font=("Segoe UI", 15, "italic"), text_color="#888")
                label.pack(pady=10)
                return

            self.images_cache.clear()
            for idx, (r, meta, v, s, c, photo_url, record_id) in enumerate(results, 1):
                track, author, album, year = meta
```

```
frame = ctk.CTkFrame(self.result_frame, fg_color="#fff", corner_radius=14)
frame.pack(fill="x", padx=8, pady=12, ipadx=8, ipady=8)

# Фото
img_label = ctk.CTkLabel(frame, text="")
img_label.pack(side="left", padx=16, pady=8)
if photo_url:
    try:
        img_data = requests.get(photo_url, timeout=5).content
        pil_img = Image.open(io.BytesIO(img_data)).resize((90, 90))
        ctk_img = ctk.CTkImage(pil_img, size=(90, 90))
        img_label.configure(image=ctk_img)
        self.images_cache.append(ctk_img)
    except Exception:
        pass

# Текст
info = (
    f"{idx}. 🎵 {track}\n"
    f"    {author}\n"
    f"    {album} ({year})\n"
    f"    Коефіцієнт збігу: {r:.3f}"
)
text_label = ctk.CTkLabel(
    frame,
    text=info,
    anchor="w",
    justify="left",
    font=("Segoe UI", 15),
    text_color="#222"
)
text_label.pack(side="left", padx=18, fill="x", expand=True)

# Кнопка "Детальніше"
```

```
                detail_btn = ctk.CTkButton(
                    frame, text="Детальніше", fg_color="#4CAF50", hover_color="#388E3C",
                    text_color="white", corner_radius=8, width=110, height=36,
                    font=ctk.CTkFont(size=14),
                    command=lambda rid=record_id: self.open_info_window(rid)
                )
                detail_btn.pack(side="right", padx=18, pady=8)

            except Exception as e:
                print(f"Помилка: {e}")  # Лише логування, без виводу у фрейм
        threading.Thread(target=_update).start()

    def open_info_window(self, record_id):
        from Info import InfoApp
        InfoApp(self, record_id, self.current_user_id)

    def get_v_data(self, full=False):
        # V(i): value з /recommended
        try:
            params = {}
            if self.current_user_id:
                params['userId'] = self.current_user_id
            response = requests.get("http://100.92.108.57:3000/recommended", params=params)
            if response.status_code == 200:
                records = response.json()
                if full:
                    return {
                        (rec.get('trackName'), rec.get('trackAuthor'), rec.get('albumName'),
rec.get('albumYear')): rec
                        for rec in records
                    }
                else:
                    return {
                        (rec.get('trackName'), rec.get('trackAuthor'), rec.get('albumName'),
rec.get('albumYear')): rec.get('value', 0)
```

```
                for rec in records
            }
        except Exception:
            pass
        return {}

    def get_s_data(self):
        # S(u,i): similarity з load_spotify_matches
        try:
            params = {}
            if self.current_user_id:
                params['userId'] = self.current_user_id
            response = requests.get("http://100.92.108.57:3000/recommended", params=params)
            if response.status_code == 200:
                records = response.json()
                favs = self.get_user_favorites()
                fav_features = {fav['id']: self.extract_features(fav['id']) for fav in favs}
                s_dict = {}
                for rec in records:
                    rec_id = rec.get('id') or self.search_track_id(rec.get('trackName'),
rec.get('trackAuthor'))
                    rec_feat = self.extract_features(rec_id)
                    max_similarity = 0.0
                    for fav in favs:
                        user_feat = fav_features.get(fav['id'], {})
                        keys = set(user_feat.keys()) | set(rec_feat.keys())
                        if not keys or not user_feat or not rec_feat:
                            similarity = 0.0
                        else:
                            sum_sq = sum((user_feat.get(k, 0) - rec_feat.get(k, 0)) ** 2 for k in keys)
                            similarity = 1 / (1 + sum_sq)
                        if similarity > max_similarity:
                            max_similarity = similarity
                    key = (rec.get('trackName'), rec.get('trackAuthor'), rec.get('albumName'),
rec.get('albumYear'))
```

```
                s_dict[key] = max_similarity
            return s_dict
        except Exception:
            pass
        return {}

    def get_c_data(self):
        # C(u,i): C з /vinyl_interest
        try:
            response = requests.get("http://100.92.108.57:3000/vinyl_interest")
            if response.status_code == 200:
                records = response.json()
                return {
                    (rec.get('trackName'), rec.get('trackAuthor'), rec.get('albumName'),
rec.get('albumYear')): rec.get('C', 0)
                    for rec in records
                }
        except Exception:
            pass
        return {}

if __name__ == "__main__":
    app = ctk.CTk()
    window = RecommendedWindow(app)
    window.mainloop()
```

Vinil.js

```
const express = require('express');
const cors = require('cors');
const bodyParser = require('body-parser');
const multer = require('multer');
const fs = require('fs');
const { v4: uuidv4 } = require('uuid');
const path = require('path');
```

```
const nodemailer = require('nodemailer');

const app = express();
const PORT = 3000;
const HOST = '100.92.108.57';

const USERS_FILE = './users.json';
const RECORDS_FILE = './records.json';
const MESSAGES_FILE = './messages.json';
const AVATARS_FOLDER = './avatars/';
const RECORDS_FOLDER = './records/';
const SELL_FILE = './Sell.json';
const ORDERS_STATS_FILE = './Orders_Stats.json';
const USERS_STATS_FILE = './Users_Stats.json';
let usersStats = { total: 0 };

app.use(express.json());

function loadUsers() {
    if (fs.existsSync(USERS_FILE)) {
        const data = fs.readFileSync(USERS_FILE, 'utf-8');
        users = JSON.parse(data || '[]');
        console.log('Користувачі завантажені:', users);
    } else {
        users = [];
        saveUsers();
        console.log('Файл не існує. Створено новий файл із порожнім списком користувачів.');
    }
}

function saveUsers() {
    fs.writeFileSync(USERS_FILE, JSON.stringify(users, null, 2));
    console.log('Користувачі збережені:', users);
}
```

```
// Завантаження платівок
let records = [];

function loadRecords() {
    if (fs.existsSync(RECORDS_FILE)) {
        const data = fs.readFileSync(RECORDS_FILE, 'utf-8');
        records = JSON.parse(data || '[]');
        console.log('Платівки завантажено:', records);
    } else {
        records = [];
        saveRecords();
        console.log('Файл не знайдено. Створено новий з порожнім списком платівок.');
    }
}

function saveRecords() {
    fs.writeFileSync(RECORDS_FILE, JSON.stringify(records, null, 2));
    console.log('Платівки збережено:', records);
}

// Завантаження повідомлень
let messages = [];
function loadMessages() {
    if (fs.existsSync(MESSAGES_FILE)) {
        const data = fs.readFileSync(MESSAGES_FILE, 'utf-8');
        messages = JSON.parse(data || '[]');
        console.log('Повідомлення завантажено:', messages); // вивести повідомлення в консоль
    } else {
        messages = [];
        saveMessages();
        console.log('Файл з повідомленнями не знайдено, створено новий');
    }
}

function saveMessages() {
```

```
    fs.writeFileSync(MESSAGES_FILE, JSON.stringify(messages, null, 2));
    console.log('Повідомлення збережено:', messages); // вивести повідомлення в консоль
}

// Завантаження статистики продажів
let sellStats = [];

function loadSellStats() {
    if (fs.existsSync(SELL_FILE)) {
        const data = fs.readFileSync(SELL_FILE, 'utf-8');
        sellStats = JSON.parse(data || '[]');
    } else {
        sellStats = [];
        saveSellStats();
    }
}

function saveSellStats() {
    fs.writeFileSync(SELL_FILE, JSON.stringify(sellStats, null, 2));
}

// Завантаження статистики замовлень
let ordersStats = [];

function loadOrdersStats() {
    if (fs.existsSync(ORDERS_STATS_FILE)) {
        const data = fs.readFileSync(ORDERS_STATS_FILE, 'utf-8');
        ordersStats = JSON.parse(data || '[]');
    } else {
        ordersStats = [];
        saveOrdersStats();
    }
}

function saveOrdersStats() {
```

```
    fs.writeFileSync(ORDERS_STATS_FILE, JSON.stringify(ordersStats, null, 2));
}

// Завантаження статистики користувачів
function loadUsersStats() {
    if (fs.existsSync(USERS_STATS_FILE)) {
        const data = fs.readFileSync(USERS_STATS_FILE, 'utf-8');
        usersStats = JSON.parse(data || '{"total":0}');
    } else {
        usersStats = { total: 0 };
        saveUsersStats();
    }
}

function saveUsersStats() {
    fs.writeFileSync(USERS_STATS_FILE, JSON.stringify(usersStats, null, 2));
}

// Реєстрація
app.post('/register', (req, res) => {
    const { firstName, lastName, login, password, email } = req.body;

    if (!firstName || !lastName || !login || !password || !email) {
        console.log('Помилка: Всі поля обов’язкові!');
        return res.status(400).json({ message: 'Всі поля обов’язкові!' });
    }

    const userExists = users.find(user => user.login === login);
    if (userExists) {
        console.log(`Помилка: Такий логін вже існує! Логін: ${login}`);
        return res.status(409).json({ message: 'Такий логін вже існує!' });
    }

    const id = uuidv4();
    const newUser = { id, firstName, lastName, login, password, email };
```

```
    users.push(newUser);
    saveUsers();

    // === ОНОВЛЕННЯ СТАТИСТИКИ КОРИСТУВАЧІВ ===
    usersStats.total += 1;
    saveUsersStats();
    // === КІНЕЦЬ ОНОВЛЕННЯ СТАТИСТИКИ ===

    console.log(`Реєстрація успішна! Новий користувач: ${firstName} ${lastName}, Логін: ${login}, Email: ${email}, ID: ${id}`);
    res.status(200).json({ message: 'Реєстрація успішна!', id });
});

// Вхід
app.post('/login', (req, res) => {
    const { login, password } = req.body;

    const user = users.find(user => user.login === login && user.password === password);

    if (user) {
        console.log(`Вхід успішний! Користувач: ${login}, ID: ${user.id}`);
        res.status(200).json({ message: 'Вхід успішний!', id: user.id });
    } else {
        console.log(`Помилка: Невірний логін або пароль! Логін: ${login}`);
        res.status(401).json({ message: 'Невірний логін або пароль!' });
    }
});

// Отримати дані профілю
app.get('/profile/:id', (req, res) => {
    const { id } = req.params;
    console.log(`Отримання профілю для користувача з ID: ${id}`);

    const user = users.find(user => user.id === id);
```

```
    if (user) {
        // Якщо email відсутній (старий акаунт), повертаємо порожній рядок
        if (!user.email) user.email = "";
        console.log(`Знайдений користувач: ${JSON.stringify(user)}`);
        res.status(200).json(user);
    } else {
        console.log('Користувача не знайдено');
        res.status(404).json({ message: 'Користувача не знайдено' });
    }
});

// Оновити пароль
app.put('/change-info/:id', (req, res) => {
    const { id } = req.params;
    const { firstName, lastName, login, email } = req.body;

    console.log(`Оновлення інформації для користувача з ID: ${id}`);

    const user = users.find(user => user.id === id);
    if (user) {
        if (firstName) {
            user.firstName = firstName;
            console.log(`Ім'я змінено на: ${firstName}`);
        }
        if (lastName) {
            user.lastName = lastName;
            console.log(`Призвище змінено на: ${lastName}`);
        }
        if (login) {
            user.login = login;
            console.log(`Логін змінено на: ${login}`);
        }
        if (email) {
            user.email = email;
            console.log(`Email змінено на: ${email}`);
```

```
        }

        saveUsers();
        console.log('Інформацію користувача успішно змінено');
        res.status(200).json({ message: 'Інформацію змінено!' });
    } else {
        console.log('Користувача не знайдено');
        res.status(404).json({ message: 'Користувача не знайдено' });
    }
});

app.put('/change-info/:id', (req, res) => {
    const { id } = req.params;
    const { firstName, lastName, login } = req.body;

    console.log(`Оновлення інформації для користувача з ID: ${id}`);

    const user = users.find(user => user.id === id);
    if (user) {
        if (firstName) {
            user.firstName = firstName;
            console.log(`Ім'я змінено на: ${firstName}`);
        }
        if (lastName) {
            user.lastName = lastName;
            console.log(`Призвище змінено на: ${lastName}`);
        }
        if (login) {
            user.login = login;
            console.log(`Логін змінено на: ${login}`);
        }

        saveUsers();
        console.log('Інформацію користувача успішно змінено');
        res.status(200).json({ message: 'Інформацію змінено!' });
```

```
    } else {
        console.log('Користувача не знайдено');
        res.status(404).json({ message: 'Користувача не знайдено' });
    }
});

// Завантаження аватара
const avatarStorage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, AVATARS_FOLDER);
    },
    filename: (req, file, cb) => {
        const id = req.params.id;
        cb(null, `${id}${path.extname(file.originalname)}`);
    }
});
const uploadAvatar = multer({ storage: avatarStorage });

app.post('/upload-avatar/:id', uploadAvatar.single('avatar'), (req, res) => {
    console.log(`Завантаження аватара: ${req.params.id}`);
    if (req.file) {
        console.log(`Файл: ${req.file.filename}`);
    } else {
        console.log('Файл не було завантажено');
    }
    res.status(200).json({ message: 'Фото завантажено успішно!' });
});

// Отримати аватар
app.get('/avatar/:id', (req, res) => {
    const id = req.params.id;
    const files = fs.readdirSync(AVATARS_FOLDER);
    const avatar = files.find(f => f.startsWith(id));

    console.log(`Запит на аватар для ID: ${id}`);
```

```
    if (avatar) {
        console.log(`Аватар знайдений: ${avatar}`);
        res.sendFile(path.resolve(AVATARS_FOLDER, avatar));
    } else {
        console.log('Аватар не знайдений');
        res.status(404).json({ message: 'Фото не знайдено' });
    }
});

// Завантаження фото платівки
const recordStorage = multer.diskStorage({
    destination: (req, file, cb) => {
        cb(null, RECORDS_FOLDER);
    },
    filename: (req, file, cb) => {
        const filename = uuidv4() + path.extname(file.originalname);
        cb(null, filename);
    }
});
const uploadRecord = multer({ storage: recordStorage });

app.post('/add-record', uploadRecord.single('photo'), (req, res) => {
    const { userId, trackName, trackAuthor, albumName, albumYear } = req.body; // Додаємо albumYear

    console.log(`Завантаження фото платівки для користувача ID: ${userId}`);
    if (req.file) {
        console.log(`Файл платівки: ${req.file.filename}`);
    } else {
        console.log('Файл не додано');
        return res.status(400).json({ message: 'Файл не додано' });
    }

    // Перевірка наявності року випуску
    if (!albumYear) {
```

```
        console.log('Рік випуску не вказано');
        return res.status(400).json({ message: 'Рік випуску обов’язковий!' });
    }

    // Формуємо id з року випуску
    const recordId = `${albumYear}_${uuidv4()}`;

    const newRecord = {
        id: recordId, // id містить рік випуску
        userId,
        trackName,
        trackAuthor,
        albumName,
        albumYear, // Зберігаємо рік випуску окремо
        photo: req.file.filename,
        status: 'Доступно'
    };

    records.push(newRecord);
    saveRecords();
    console.log('Нова платівка додана!');
    res.status(200).json({ message: 'Платівка додана!' });
});

// Додаємо допоміжну функцію для формування photoUrl
function addPhotoUrl(record) {
    return {
        ...record,
        photoUrl: `http://${HOST}:${PORT}/record-photo/${record.photo}`
    };
}

// Отримати платівки користувача
app.get('/records/:userId', (req, res) => {
    const { userId } = req.params;
```

```
    const userRecords = records
        .filter(record => record.userId === userId)
        .map(addPhotoUrl);
    console.log(`Отримано платівки для користувача з ID: ${userId}`);
    console.log('Платівки:', userRecords);
    res.status(200).json(userRecords);
});

// Отримати фото платівки
app.get('/record-photo/:filename', (req, res) => {
    const { filename } = req.params;
    const filepath = path.join(RECORDS_FOLDER, filename);

    if (fs.existsSync(filepath)) {
        console.log(`Фото для платівки з файлом: ${filename} знайдено.`);
        res.sendFile(path.resolve(filepath));
    } else {
        console.log(`Фото для платівки з файлом: ${filename} не знайдено.`);
        res.status(404).json({ message: 'Фото не знайдено' });
    }
});

// Оновити статус платівки
app.put('/update-record/:recordId', (req, res) => {
    const { recordId } = req.params;
    const { status } = req.body;

    const record = records.find(record => record.id === recordId);
    if (record) {
        console.log(`Оновлюємо статус платівки з ID: ${recordId} на: ${status}`);
        record.status = status;
        saveRecords();
        res.status(200).json({ message: 'Статус оновлено!' });
    } else {
        console.log(`Платівка з ID: ${recordId} не знайдена.`);
```

```
        res.status(404).json({ message: 'Платівку не знайдено' });
    }
});

// Видалити платівку
app.delete('/delete-record/:recordId', (req, res) => {
    const { recordId } = req.params;
    const index = records.findIndex(record => record.id === recordId);

    if (index !== -1) {
        const deletedRecord = records.splice(index, 1)[0];
        console.log(`Платівка з ID: ${recordId} видалена.`);
        saveRecords();

        // Також видалити фото
        const filepath = path.join(RECORDS_FOLDER, deletedRecord.photo);
        if (fs.existsSync(filepath)) {
            fs.unlinkSync(filepath);
            console.log(`Фото для платівки з файлом ${deletedRecord.photo} видалено.`);
        }

        res.status(200).json({ message: 'Платівку видалено!' });
    } else {
        console.log(`Платівка з ID: ${recordId} не знайдена для видалення.`);
        res.status(404).json({ message: 'Платівку не знайдено' });
    }
});

// Отримати конкретну платівку
app.get('/record/:recordId', (req, res) => {
    const { recordId } = req.params;
    console.log(`Отримано запит для платівки з ID: ${recordId}`);
    const record = records.find(record => record.id === recordId);
    if (record) {
        console.log(`Платівка знайдена: ${JSON.stringify(record)}`);
```

```
        res.status(200).json(addPhotoUrl(record));
    } else {
        console.log(`Платівка з ID: ${recordId} не знайдена`);
        res.status(404).json({ message: 'Альбом не знайдено' });
    }
});

// Отримати всі платівки
app.get('/records', (req, res) => {
    console.log('Отримано запит для всіх платівок');
    res.status(200).json(records.map(addPhotoUrl));
});

// Відправити повідомлення
app.post('/send-message', (req, res) => {
    const { receiverId, text } = req.body;
    console.log(`Отримано запит на відправлення повідомлення до користувача ${receiverId} з
текстом: "${text}"`);

    if (!receiverId || !text) {
        console.log('Невірні дані при відправці повідомлення');
        return res.status(400).json({ message: 'Неправильні дані!' });
    }

    const message = {
        id: uuidv4(),
        sender: req.body.senderId || 'Anonymous',
        receiverId,
        text,
        timestamp: new Date().toISOString()
    };

    messages.push(message);
    saveMessages();
    console.log(`Повідомлення надіслано користувачу ${receiverId}: "${text}"`);
```

```
    res.status(200).json({ message: 'Повідомлення надіслано!' });
});

// Отримати всі повідомлення користувача
app.get('/messages/:userId', (req, res) => {
    const { userId } = req.params;

    const userMessages = messages.filter(m => m.sender === userId || m.receiverId === userId);

    res.status(200).json(userMessages);
});

// Отримати всіх співрозмовників користувача
app.get('/messages/users/:userId', (req, res) => {
    const { userId } = req.params;



    const userMessages = messages.filter(m => m.sender === userId || m.receiverId === userId);

    const otherUserIds = [...new Set(
        userMessages.map(m => (m.sender === userId ? m.receiverId : m.sender))
    )];

    const usersList = users.filter(u => otherUserIds.includes(u.id));

    res.status(200).json(usersList);
});

// Завантаження замовлень
const ORDERS_FILE = './orders.json';
let orders = [];

function loadOrders() {
    if (fs.existsSync(ORDERS_FILE)) {
```

```
        const data = fs.readFileSync(ORDERS_FILE, 'utf-8');
        orders = JSON.parse(data || '[]');
        console.log('Замовлення успішно завантажено з файлу.');
    } else {
        orders = [];
        saveOrders();
        console.log('Файл замовлень не знайдений, створено новий.');
    }
}

function saveOrders() {
    fs.writeFileSync(ORDERS_FILE, JSON.stringify(orders, null, 2));
    console.log('Замовлення успішно збережено.');
}

app.post('/create-order', (req, res) => {
    const { recordId, buyerId, sellerId } = req.body;

    if (!recordId || !buyerId || !sellerId) {
        console.log('Помилка: Відсутні дані для замовлення!');
        return res.status(400).json({ message: 'Відсутні дані для замовлення!' });
    }

    const order = {
        id: uuidv4(),
        recordId,
        buyerId,
        sellerId,
        timestamp: new Date().toISOString(),
        status: 'Очікує підтвердження'
    };

    orders.push(order);
    saveOrders();
```

```
    // === ДОДАЄМО СТАТИСТИКУ ЗАМОВЛЕНЬ ===
    const record = records.find(r => r.id === recordId);
    if (record) {
        const key = `${record.trackName}|${record.trackAuthor}|${record.albumName}|${record.albumYear}`;
        let stat = ordersStats.find(s =>
            s.trackName === record.trackName &&
            s.trackAuthor === record.trackAuthor &&
            s.albumName === record.albumName &&
            s.albumYear === record.albumYear
        );
        if (stat) {
            stat.orders += 1;
        } else {
            ordersStats.push({
                trackName: record.trackName,
                trackAuthor: record.trackAuthor,
                albumName: record.albumName,
                albumYear: record.albumYear,
                orders: 1
            });
        }
        saveOrdersStats();
    }
    // === КІНЕЦЬ ДОДАВАННЯ СТАТИСТИКИ ===

    console.log(`Створено нове замовлення з ID: ${order.id}`);
    res.status(200).json({ message: 'Замовлення створено успішно!' });
});

app.get('/orders/:sellerId', (req, res) => {
    const sellerOrders = orders.filter(o => o.sellerId === req.params.sellerId);

    console.log(`Отримано замовлення для продавця з ID: ${req.params.sellerId}`);
    console.log(`Кількість замовлень: ${sellerOrders.length}`);
```

```
    res.status(200).json(sellerOrders);
});

app.get('/my-orders/:buyerId', (req, res) => {
    const buyerOrders = orders.filter(o => o.buyerId === req.params.buyerId);

    console.log(`Отримано замовлення для покупця з ID: ${req.params.buyerId}`);
    console.log(`Кількість замовлень: ${buyerOrders.length}`);

    res.status(200).json(buyerOrders);
});

// Оновити статус замовлення
app.put('/update-order-status/:orderId', (req, res) => {
    const { orderId } = req.params;
    const { status } = req.body;

    const order = orders.find(o => o.id === orderId);
    if (order) {
        order.status = status;
        saveOrders();

        console.log(`Оновлено статус замовлення з ID: ${orderId} на ${status}`);
        res.status(200).json({ message: 'Статус замовлення оновлено!' });
    } else {
        console.log(`Замовлення з ID: ${orderId} не знайдено`);
        res.status(404).json({ message: 'Замовлення не знайдено' });
    }
});

// Пошук платівок
app.get('/search-records', (req, res) => {
    const { trackName, trackAuthor, albumName, userId } = req.query;
```

```
    console.log(`Параметри пошуку: trackName=${trackName}, trackAuthor=${trackAuthor}, albumName=${albumName}, userId=${userId}`);
    const foundRecords = records.filter(record =>
        record.trackName.toLowerCase() === trackName.toLowerCase() &&
        record.trackAuthor.toLowerCase() === trackAuthor.toLowerCase() &&
        record.albumName.toLowerCase() === albumName.toLowerCase() &&
        record.userId !== userId
    );
    if (foundRecords.length > 0) {
        console.log(`Знайдено платівок: ${foundRecords.length}`);
        res.status(200).json(foundRecords.map(addPhotoUrl));
    } else {
        console.log('Платівки не знайдено');
        res.status(404).json({ message: 'Платівки не знайдено' });
    }
});

// --- Рекомендовані платівки ---
app.get('/recommended', (req, res) => {
    const { userId } = req.query; // отримуємо userId з query

    // Фільтруємо платівки, щоб не включати свої
    let filteredRecords = records;
    if (userId) {
        filteredRecords = records.filter(r => r.userId !== userId);
    }

    // Підрахунок кількості екземплярів кожної платівки (ключ: trackName|trackAuthor|albumName|albumYear)
    const recordCounts = {};
    filteredRecords.forEach(r => {
        const key = `${r.trackName}|${r.trackAuthor}|${r.albumName}|${r.albumYear}`;
        recordCounts[key] = (recordCounts[key] || 0) + 1;
    });
```

```
// Підрахунок популярності (кількість продажів)
const popularityMap = {};
sellStats.forEach(stat => {
    const key = `${stat.trackName}|${stat.trackAuthor}|${stat.albumName}|${stat.albumYear}`;
    popularityMap[key] = stat.sales;
});

// Знаходимо мін/макс для нормалізації
let minCount = Infinity, maxCount = -Infinity;
let minSales = Infinity, maxSales = -Infinity;
let minYear = Infinity, maxYear = -Infinity;
filteredRecords.forEach(r => {
    const key = `${r.trackName}|${r.trackAuthor}|${r.albumName}|${r.albumYear}`;
    const count = recordCounts[key] || 1;
    const sales = popularityMap[key] || 0;
    const year = parseInt(r.albumYear, 10) || 0;
    if (count < minCount) minCount = count;
    if (count > maxCount) maxCount = count;
    if (sales < minSales) minSales = sales;
    if (sales > maxSales) maxSales = sales;
    if (year < minYear) minYear = year;
    if (year > maxYear) maxYear = year;
});

// Якщо всі значення однакові, щоб уникнути ділення на 0
if (minCount === maxCount) maxCount++;
if (minSales === maxSales) maxSales++;
if (minYear === maxYear) maxYear++;

// Автоматичний підбір ваг: найбільша вага для найбільш варіативної ознаки
const rarityRange = 1 / minCount - 1 / maxCount;
const popularityRange = maxSales - minSales;
const ageRange = maxYear - minYear;

const totalRange = rarityRange + popularityRange + ageRange;
```

```
    let w1 = rarityRange / totalRange;
    let w2 = popularityRange / totalRange;
    let w3 = ageRange / totalRange;

    // Якщо всі однакові, рівномірно
    if (!isFinite(w1) || !isFinite(w2) || !isFinite(w3)) {
        w1 = w2 = w3 = 1 / 3;
    }

    const currentYear = new Date().getFullYear();

    // Розрахунок V(i) для кожної платівки
    const valuedRecords = filteredRecords.map(r => {
        const key = `${r.trackName}|${r.trackAuthor}|${r.albumName}|${r.albumYear}`;
        const count = recordCounts[key] || 1;
        // Rarity: нормалізуємо до [0,1], чим менше count, тим ближче до 1
        const rarity = (maxCount - count) / (maxCount - minCount);
        // Popularity: нормалізуємо до [0,1]
        const popularity = ((popularityMap[key] || 0) - minSales) / (maxSales - minSales);
        // Age: нормалізуємо до [0,1], чим старіше, тим ближче до 1
        const age = (parseInt(r.albumYear, 10) - minYear) / (maxYear - minYear);

        const V = w1 * rarity + w2 * popularity + w3 * age;
        return { ...r, value: V };
    });

    // Сортуємо за спаданням цінності
    valuedRecords.sort((a, b) => b.value - a.value);

    // Додаємо photoUrl
    const result = valuedRecords.slice(0, 10).map(addPhotoUrl);
    res.status(200).json(result);
});

// --- Додаємо ендпоінт для статистики замовлень для формули C(u,i) ---
```

```
app.get('/orders_stats', (req, res) => {
   // Orders_Stats.json: масив об'єктів з полями trackName, trackAuthor, albumName, albumYear,
orders
   // Потрібно повернути масив об'єктів з унікальними користувачами для кожної платівки

   // Завантажуємо всі замовлення (orders.json)
   let ordersArr = [];
   if (fs.existsSync('./orders.json')) {
      const data = fs.readFileSync('./orders.json', 'utf-8');
      ordersArr = JSON.parse(data || '[]');
   }

   // Формуємо мапу: ключ - платівка, значення - Set user_id
   const vinylStats = {};
   ordersArr.forEach(order => {
      // Знаходимо платівку за recordId
      const record = records.find(r => r.id === order.recordId);
      if (record) {
         const key = `${record.trackName}|${record.trackAuthor}|${record.albumName}`;
         if (!vinylStats[key]) vinylStats[key] = new Set();
         if (order.buyerId) vinylStats[key].add(order.buyerId);
      }
   });

   // Формуємо масив для відповіді
   const result = [];
   Object.keys(vinylStats).forEach(key => {
      const [trackName, trackAuthor, albumName] = key.split('|');
      result.push({
         trackName,
         trackAuthor,
         albumName,
         users: Array.from(vinylStats[key])
      });
   });
```

```
    res.status(200).json({ orders: result });
});

// --- Додаємо ендпоінт для статистики користувачів ---
app.get('/users_stats', (req, res) => {
    // Users_Stats.json: { total: N }
    let stats = { total: 0 };
    if (fs.existsSync('./Users_Stats.json')) {
        const data = fs.readFileSync('./Users_Stats.json', 'utf-8');
        stats = JSON.parse(data || '{"total":0}');
    }
    res.status(200).json(stats);
});

// --- Ендпоінт для обчислення C(u,i) для кожної платівки ---
app.get('/vinyl_interest', (req, res) => {
    // Завантажуємо Orders_Stats.json
    let ordersStatsArr = [];
    if (fs.existsSync('./Orders_Stats.json')) {
        const data = fs.readFileSync('./Orders_Stats.json', 'utf-8');
        ordersStatsArr = JSON.parse(data || '[]');
    }

    // Завантажуємо Users_Stats.json
    let usersStatsObj = { total: 0 };
    if (fs.existsSync('./Users_Stats.json')) {
        const data = fs.readFileSync('./Users_Stats.json', 'utf-8');
        usersStatsObj = JSON.parse(data || '{"total":0}');
    }
    const totalUsers = usersStatsObj.total || 1; // захист від ділення на 0

    // Формуємо масив з C(u,i) для кожної платівки
    const result = ordersStatsArr.map(stat => ({
        trackName: stat.trackName,
```

```
        trackAuthor: stat.trackAuthor,
        albumName: stat.albumName,
        albumYear: stat.albumYear,
        orders: stat.orders,
        C: +(stat.orders / totalUsers).toFixed(3)
    }));

    res.status(200).json(result);
});

// Додайте перед app.listen(...):

app.get('/user/:id', (req, res) => {
    const { id } = req.params;
    const user = users.find(u => u.id === id);
    if (user) {
        // Повертаємо тільки потрібні поля
        res.status(200).json({
            firstName: user.firstName || 'Невідомий',
            lastName: user.lastName || 'Користувач'
        });
    } else {
        res.status(404).json({ firstName: 'Невідомий', lastName: 'Користувач' });
    }
});

// Додайте новий ендпоінт для отримання платівок за користувачем
app.get('/records-by-user/:userId', (req, res) => {
    const { userId } = req.params;
    const userRecords = records
        .filter(record => record.userId === userId)
        .map(addPhotoUrl);
    console.log(`Отримано платівки для користувача з ID: ${userId}`);
    res.status(200).json(userRecords);
})
```

Додаток Г – Ілюстративна частина

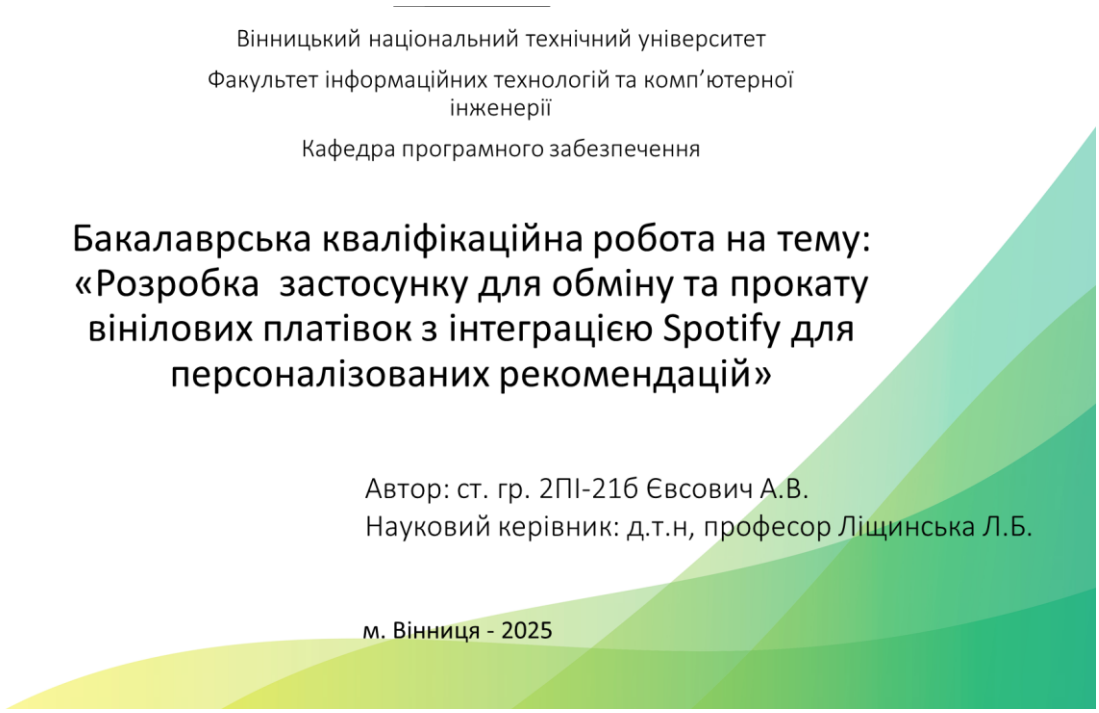

Рисунок Г.1 – Вступний слайд

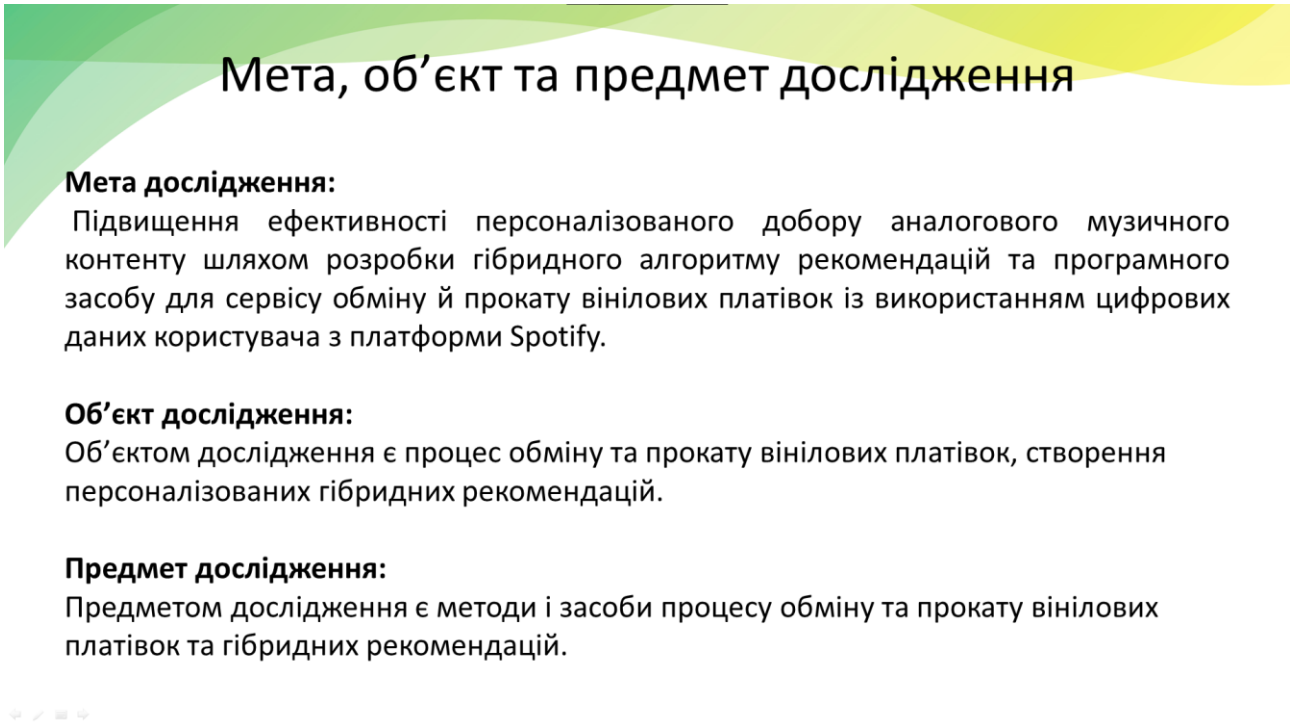

Рисунок Г.2 – Мета, об'єкт та предмет дослідження

Задачі дослідження

- Провести аналіз стану розвитку сервісів обміну та прокату вінілових платівок із персоналізованими рекомендаціями
- Дослідити існуючі аналоги та здійснити їх функціонально-порівняльний аналіз
- Сформувати функціональні та нефункціональні вимоги до системи
- Розробити модель системи з урахуванням структури вхідних даних
- Розробити гібридний алгоритм персоналізованих рекомендацій із використанням даних Spotify, поведінкової схожості та колекційної цінності
- Обґрунтувати вибір технологій для реалізації програмного забезпечення
- Реалізувати програмні модулі авторизації, рекомендацій та користувацького інтерфейсу
- Провести тестування системи та алгоритмів рекомендацій
- Розробити інструкцію користувача

Рисунок Г.3 – Задачі дослідження

Новизна і практична цінність отриманих результатів

Наукова новизна отриманих результатів.
Запропоновано удосконалений метод персоналізованих рекомендацій, особливістю якого є поєднання контентної подібності на основі аудіофіч, поведінкової подібності користувачів та оцінки колекційної цінності фізичних носіїв, що дозволило підвищити точність відповідності рекомендацій індивідуальним музичним уподобанням згідно з метриками Precision@k та NDCG.

Практичне значення одержаних результатів.
Практична цінність одержаних результатів полягає в тому, що на основі розробленого методу вдалося досягти вищої релевантності добору уподобань користувачів та адаптацію рекомендацій до умов обігу фізичних носіїв. Отримані результати можуть бути використані в інформаційних системах з персоналізованими рекомендаціями для роботи з медіаконтентом у фізичному форматі.

Рисунок Г.4 – Новизна і практична цінність отриманих результатів

Порівняльний аналіз аналогів

Система	Переваги	Недоліки
Discogs	Потужна система каталогізації Розширені метадані до платівок Пошук та фільтрація за різними критеріями Велика база даних	Відсутність функціоналу обміну / прокату Немає персоналізованих рекомендацій Відсутня взаємодія між користувачами (месенджер, перегляд профілів)
Vinyl Me, Please	Унікальний підхід на основі підписки Якісно відібрані релізи Простий інтерфейс та особистий кабінет	Рекомендації не персоналізовані (тільки кураторські) Відсутній обмін / прокат Немає доступу до колекцій інших Відсутній соціальний функціонал
Libib	Зручна каталогізація особистої колекції Підтримка різних типів медіа Інвентаризація через сканування	Відсутній функціонал обміну / прокату Немає взаємодії між користувачами Відсутні рекомендації

Рисунок Г.5 – Порівняльний аналіз аналогів

Алгоритм роботи системи

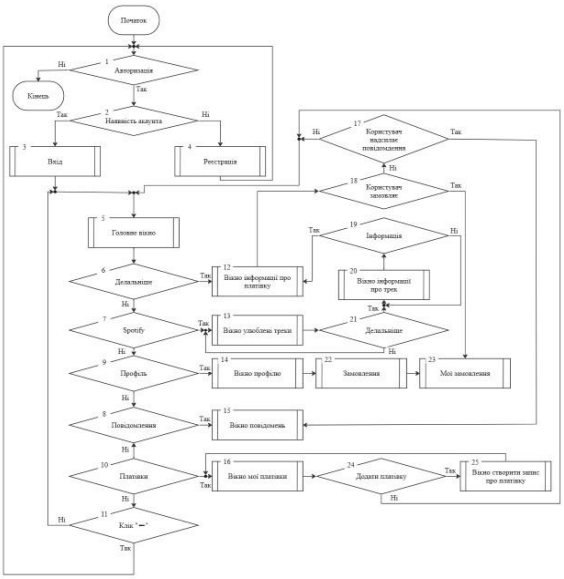

Рисунок Г.6 – Алгоритм роботи системи

Розробка методу та алгоритму персоналізованих рекомендацій

Формальнорелевантністьплатівки *i* користувачу *u* визначається як зважена сума трьохкомпонент:

$$R_{u,i} = \alpha \times S_{u,i} + \beta \times C_{u,i} + \gamma \times V_i$$

$S_{u,i}$ – показник схожості між профілем користувача та характеристиками платівки за аудіофічами;
$C_{u,i}$ – оцінка потенційної зацікавленості на основі поведінки схожих користувачів;
V_i – оцінка колекційної цінності платівки.
Коефіцієнти α, β, γ відповідають за вагомість кожного із трьох компонентів і нормалізуються згідно з умовоюα+β+γ=1.

Рисунок Г.7 – Розробка методу та алгоритму персоналізованих рекомендацій

Розробка методу та алгоритму персоналізованих рекомендацій

Компонент$S_{u,i}$ отримуєтьсяяк обернена функція зваженого середньоквадратичного відхиленняміж середніми значеннями аудіоознакулюблених треків користувача та аналогічнимипоказниками для треків, розміщених на певній віниловій платівці

$$S_{u,i} = \frac{1}{1 + \sum_{k=1}^{n} (f_{u,k} - f_{i,k})^2}$$

де
$f_{u,k}$ – середнє значення *k*-тої аудіофічі для користувача *u*,
$f_{i,k}$ – середнє значення тієї ж аудіофічі для платівки *i*,
n – кількість урахованих аудіофіч

Рисунок Г.8 – Розробка методу та алгоритму персоналізованих рекомендацій

Розробка методу та алгоритму персоналізованих рекомендацій

Другий компонент $C_{u,i}$ реалізує принцип колаборативноїфільтрації за допомогою підходу спільного прослуховування Подальше обчислення компоненту $C_{u,i}$ виконується за формулою:

$$C_{u,i} = \frac{U_i}{U}$$

U_i – кількість користувачів, що зацікавилися платівкою,
U – загальна кількість користувачів.

Рисунок Г.9 – Розробка методу та алгоритму персоналізованих рекомендацій

Розробка методу та алгоритму персоналізованих рекомендацій

Третій компонент V_i відповідає за інтеграцію колекційної цінності платівки як релевантноїознаки. Колекційна цінність формується на основі сукупності факторів.

$$V_i = w_1 \times Rarity_i + w_2 \times Popularity_i + w_3 \times Age_i$$

де всі компоненти нормалізованодо інтервалу(0,1), а ваги w_1, w_2, w_3, w_4 встановлюються емпірично або адаптивно через процедуру оптимізації(наприклад, градієнтнимметодом або стохастичним пошуком).

Рисунок Г.10 – Розробка методу та алгоритму персоналізованих рекомендацій

Тестування алгоритму рекомендацій

Тестування алгоритму проводилося в трьох конфігураціях: A – лише контентна подібність за аудіофічами; B – комбінація контентного підходу з колаборативноюскладовою; C – повний гібрид, що включає всі три джерела рекомендаційногоскору.

Конфігурація	Precision@5	Recall@10	nDCG@5	MRR
A: Content only	0.58	0.61	0.61	0.57
B: + Collaborative	0.66	0.67	0.68	0.65
C: + Collectability	0.74	0.72	0.76	0.71

Рисунок Г.11 – Тестування алгоритму рекомендацій

Тестування алгоритму рекомендацій

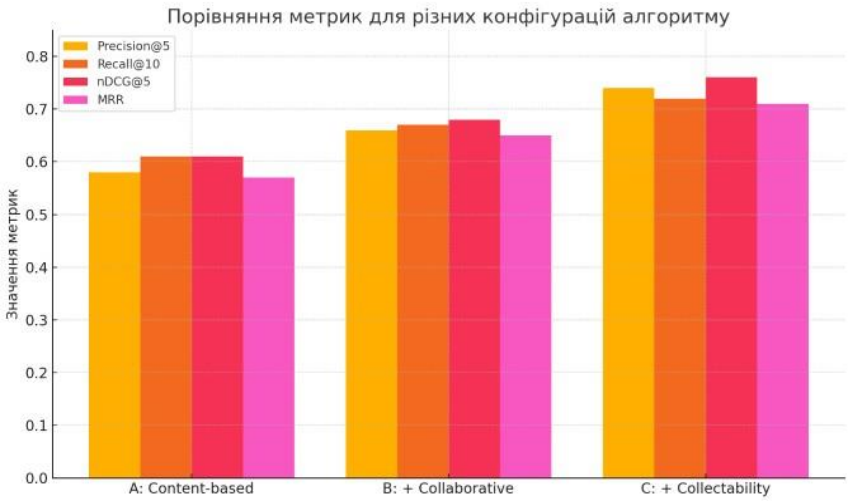

Рисунок Г.12 – Тестування алгоритму рекомендацій

Демонстрація інтерфейсу застосунку «Головний екран»

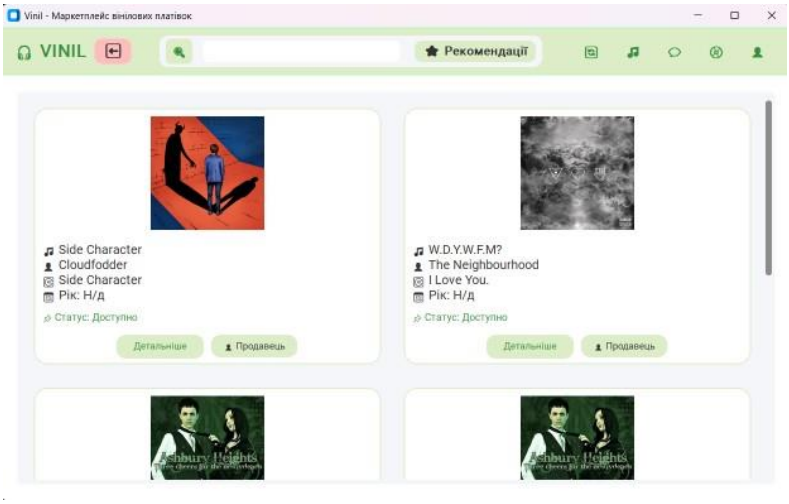

Рисунок Г.13 – Демонстрація інтерфейсу застосунку «Головний екран»

Демонстрація інтерфейсу застосунку «Інформація про платівку»

Рисунок Г.14 – Демонстрація інтерфейсу застосунку «Інформація про платівку»

Демонстрація інтерфейсу застосунку «Улюблені пісні»

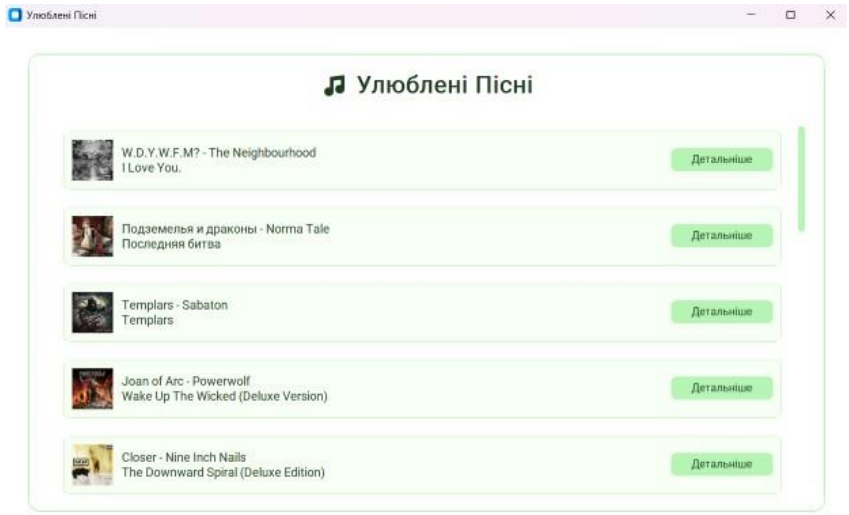

Рисунок Г.15 – Демонстрація інтерфейсу застосунку «Улюблені пісні»

Демонстрація інтерфейсу застосунку «Повідомлення»

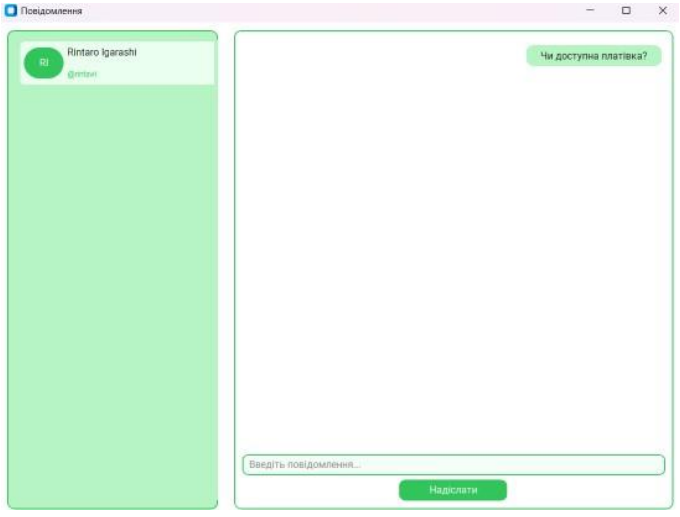

Рисунок Г.16 – Демонстрація інтерфейсу застосунку «Повідомлення»

Демонстрація інтерфейсу застосунку «Рекомендації»

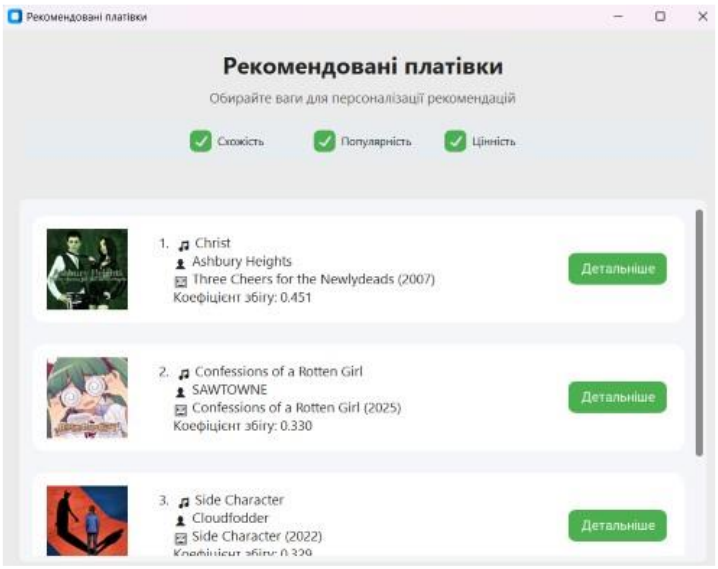

Рисунок Г.17 – Демонстрація інтерфейсу застосунку «Рекомендації

Висновки

- Розроблено застосунок для обміну та прокату вінілових платівок з інтеграцією персоналізованих рекомендацій на основі Spotify.
- Запропоновано гібридний алгоритм рекомендацій, що враховуєаудіофічі, поведінкову подібність та колекційну цінність фізичних носіїв.
- Побудовано формальну модель релевантності яка забезпечує точне зіставлення цифрового профілю користувача з наявними платівками.
- Реалізовано архітектуру системи з використанням Flask, React.js, Spotify APIта Python-бібліотек для обробки даних.
- Проведено верифікацію працездатності рекомендаційного механізму на основі типових користувацьких сценаріїв.
- Підвищено релевантність рекомендацій відповідно до метрик Precision@k та NDCG, що підтверджує ефективність підходу.

Рисунок Г.18 – Висновки

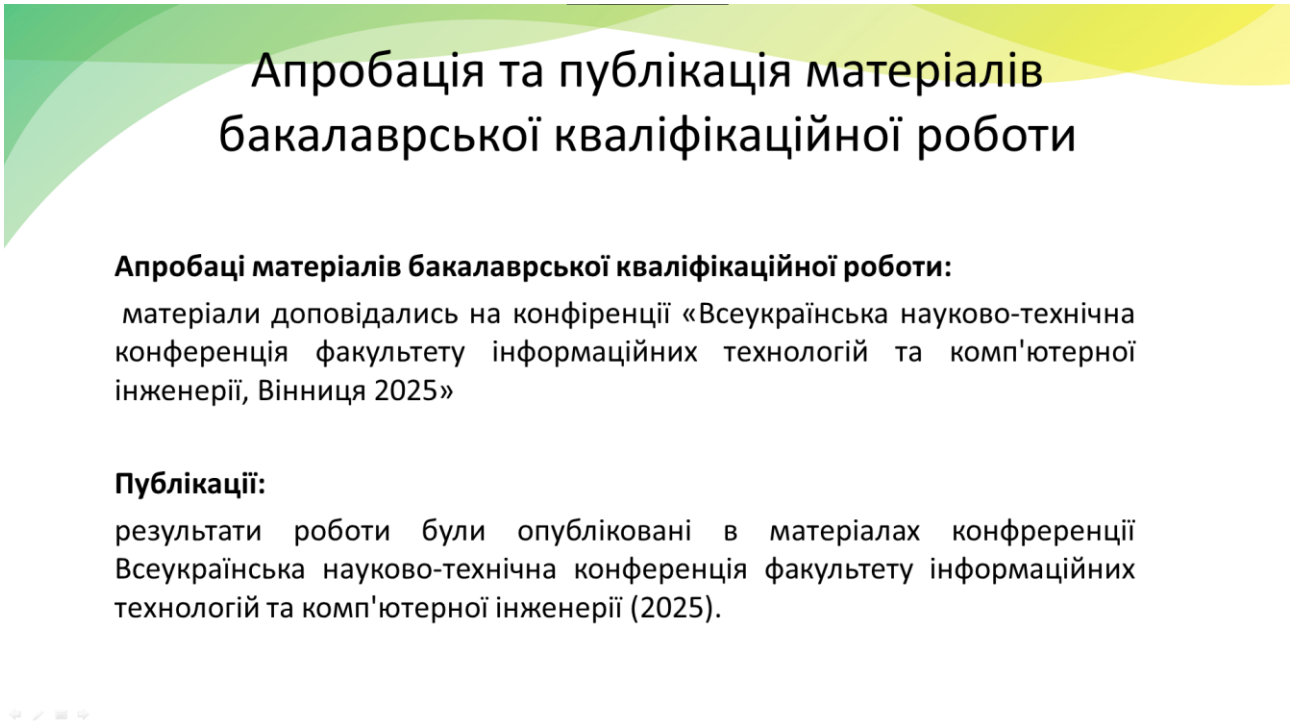

Рисунок Г.19 – Апробація та публікація матеріалів бакалаврської кваліфікаційної роботи