МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

ЗБОРОВСЬКА Анастасія Романівна

Мобільний застосунок аналітики для рекомендаційної системи читання книг/Analytics mobile application for book reading recommendation system

Спеціальність 122 – Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

Кваліфікаційна робота

Виконала студентка групи КН-41
А. Р. Зборовська

Науковий керівник:
к.т.н., старший викладач кафедри
ІОСУ М. З. Домбровський

Кваліфікаційну роботу допущено
до захисту
«___»_____ 2025 р.

В.о. завідувача кафедри
_____Н.М. Васильків

Тернопіль – 2025

Факультет комп’ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
спеціальність 122 – Комп’ютерні науки
освітньо-професійна програма – Комп’ютерні науки

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ Н.М. Васильків
«____» _____2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
ЗБОРОВСЬКА Анастасія Романівна
(прізвище, ім’я, по батькові)

1. Тема кваліфікаційної роботи: Мобільний застосунок аналітики для рекомендаційної системи читання книг/Analytics mobile application for book reading recommendation system
керівник роботи М. З. Домбровський, к.т.н., старший викладач кафедри ІОСУ
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 20 грудня 2024 року №928
2. Строк подання студентом закінченої кваліфікаційної роботи 25 травня 2025 р.
3. Вихідні дані до кваліфікаційної роботи: наукові статті, технічна література.
4. Основні питання, які потрібно розробити:
- проаналізувати існуючі рішення мобільних застосунків аналітики для читання книг з рекомендаційною системою;
- сформувати концепцію побудови мобільного застосунку аналітики для рекомендаційної системи читання книг на основі проаналізованих існуючих рішень, враховуючи їхні переваги та недоліки;
- розробити архітектуру мобільного застосунку; створити простий і логічно зрозумілий дизайн додатку;
- реалізувати дизайн і програмний код мобільного застосунку аналітики для рекомендаційної системи читання книг;
- провести тестування додатку, зокрема робочість усіх функцій.
5. Перелік графічного матеріалу в роботі:
- зображення аналогів;
- діаграми архітектури;
- таблиці функцій;

6. Консультанти розділів кваліфікаційної роботи

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

7. Дата видачі завдання 20 грудня 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів кваліфікаційної роботи	Примітка
1	Затвердження теми кваліфікаційної роботи, ознайомлення з літературними джерелами та складання плану роботи	до 01.01. 2025 р.	
2	Написання 1 розділу кваліфікаційної роботи	до 01.02. 2025 р.	
3	Написання 2 розділу кваліфікаційної роботи	до 01.04.2025 р.	
4	Написання 3 розділу кваліфікаційної роботи	до 01.05. 2025 р.	
5	Представлення попереднього варіанту кваліфікаційної роботи, перевірка та внесення змін керівником	до 15.05.2025 р.	
6	Опрацювання зауважень та представлення завершеного варіанту кваліфікаційної роботи. Підготовка супроводжуючих документів	до 25.05.2025 р.	
7	Перевірка кваліфікаційної роботи на оригінальність тексту	до 30.05.2025 р.	
8	Оформлення кваліфікаційної роботи та отримання допуску до захисту	до 10.06.2025 р.	
9	Подання кваліфікаційної роботи до захисту на засіданні атестаційної комісії	до 10.06.2025 р.	

Студент_____ Зборовська А. Р.
(підпис) (прізвище та ініціали)

Керівник кваліфікаційної роботи_____ Домбровський М. З
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Мобільний застосунок аналітики для рекомендаційної системи читання книг» на здобуття освітнього ступеня «бакалавр» зі спеціальності 122 «Комп'ютерні науки» освітньої програми «Комп'ютерні науки» написана обсягом 84 сторінки і містить 37 ілюстрацій, 3 таблиці, 2 додатки та 36 використаних джерел.

Метою роботи є створення мобільного застосунку, який забезпечить зручне середовище для збору та аналізу статистичних даних прогресу читання і формування на основі цих даних персональних рекомендацій.

Методами розроблення обрано: метод аналізу (для вивчення існуючих підходів до реалізації аналітичних модулів та рекомендаційних систем у мобільних застосунках), метод синтезу (для поєднання переваг наявних рішень та формування оптимальної структури функціональних можливостей розроблюваного додатку), методи моделювання (для побудови UML-діаграм, що відображають логіку функціонування системи, взаємозв'язки між її компонентами; для забезпечення візуалізації процесів взаємодії користувача з додатком), метод порівняльного аналізу (для оцінювання ефективності підходів до збереження даних на стороні клієнта).

Внаслідок виконання роботи обґрунтовано доцільність створення мобільного застосунку для аналітики прогресу читання та формування рекомендацій, а також реалізовано програмний застунок, що забезпечує зручне середовище для збору, збереження, обробки та аналізу даних про читання книг.

Результати дослідження можуть бути використані в освітніх закладах, дослідницьких установах та компаніях, що розробляють мобільні застосунки, аналітичні й рекомендаційні системи.

Ключові слова: МОБІЛЬНИЙ ЗАСТОСУНОК, РЕКОМЕНДАЦІЙНА СИСТЕМА, АНАЛІТИКА ДЛЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ, ТРЕКЕР ДЛЯ ЧИТАННЯ, SHAREDPREFERENCES.

ANNOTATION

Qualification work on the topic " Analytics mobile application for book reading recommendation system" for Bachelor’s degree on speciality 122 «Computer Science» educational and professional program «Computer Science» is written on 84 pages and it contains 37 figures, 3 tables, 2 annexes and 36 sources.

The purpose of the work is to create a mobile application that will provide a convenient environment for collecting and analyzing statistical data on reading progress and forming personalized recommendations based on this data.

Research methods are analysis (to study existing approaches to implementing analytical modules and recommendation systems in mobile applications), synthesis (to combine the advantages of existing solutions and form the optimal structure of the functionality of the application being developed), modeling (to build UML diagrams that reflect the logic of the system's functioning and the relationships between its components; to provide visualization of the processes of user interaction with the application), comparison (to evaluate the effectiveness of client-side data retention approaches).

As a result of the work the feasibility of creating a mobile application for reading progress analytics and recommendations was substantiated, and a software tool was implemented that provides a convenient environment for collecting, storing, processing, and analyzing data on book reading.

The results of the research can be used in educational institutions, research institutions, and companies that develop mobile applications, analytical and recommender systems.

Keywords: MOBILE APPLICATION, RECOMMENDATION SYSTEM, ANALYTICS FOR RECOMMENDATION SYSTEM, READING TRACKER, SHAREDPREFERENCES.

ЗМІСТ

ВСТУП

Актуальність кваліфікаційної роботи зумовлена тим, що у сучасному інформаційному суспільстві щоденно зростає потреба у швидкому доступі до персоналізованої інформації. Це стосується й галузі читання, де користувачі дедалі частіше шукають ефективні інструменти для організації та аналізу власного читацького досвіду. Традиційно для цього використовували читацькі щоденники, у яких фіксували назви прочитаних книг, враження та динаміку читання. Проте з розвитком цифрових технологій виникає потреба в автоматизованих засобах ведення таких записів. У зв'язку з цим зростає популярність мобільних застосунків для відстеження читацької активності, відомих як трекери читання. Такі інструменти дають змогу значно спростити процес моніторингу читацької активності, зробити його зручнішим, доступнішим і більш інтерактивним.

Доказом поширеності обраної теми в Україні, служить для прикладу кількість завантажень українського трекера для читання книг «Rork», що склала дотепер понад 100 тисяч. Отже, це свідчить на користь припущення, що для читачів, які користуються електронними книгами буде зручно мати у своєму мобільному телефоні електронну бібліотеку книг. А також не менш зручно буде отримувати відразу після прочитання книги персоналізовані рекомендації, які будуть заохочувати українців ще більше читати.

За вказаних обставин, рекомендаційна система, програмний модуль що слугує виконанню її функції – це інструмент для фільтрації онлайн-інформації. Саме тому тематика проєктування і розроблення мобільного застосунку аналітики для рекомендаційної системи читання книг є актуальною.

Мобільний застосунок аналітики для рекомендаційної системи забезпечує додавання книги (назва, автор, жанр і т.п.), відстежування кількості прочитаних сторінок, виконання звітів за роки та місяці і, на основі прочитаних книг, формувати персональні рекомендації.

Метою дипломної роботи є створення мобільного застосунку, який забезпечить зручне середовище для збору та аналітики статистичних даних прогресу читання і формування на основі цих даних персональних рекомендацій.

Для досягнення поставленої мети необхідно виконати наступні завдання:

1. Проаналізувати існуючі рішення мобільних застосунків аналітики для читання книг з рекомендаційною системою.

2. Сформувати концепцію розробки мобільного застосунку аналітики для рекомендаційної системи читання книг на основі проаналізованих рішень, враховуючи їхні переваги та недоліки.

3. Розробити алгоритмічне та інформаційне забезпечення застосунку аналітики, що включає механізми збору, попередньої обробки та аналізу кількісних даних.

4. Реалізувати дизайн і програмний код мобільного застосунку аналітики для рекомендаційної системи читання книг.

5. Провести тестування застосунку, зокрема роботу функцій інтерфейсу користувача.

Об’єкт дослідження – процеси збору, представлення, обробки, зберігання, передачі та доступу до інформації в комп’ютерних системах.

Предмет дослідження – сучасні моделі, методи, алгоритми, технології процесів аналізу даних рекомендаційної системи читання книг.

У роботі використано методи аналізу наукової літератури та існуючих рішень щодо предметної області, інтегроване середовище розробки Android Studio, програмне забезпечення виконане на мові програмування Java, для інтерфейсу використовувалась розширювана мова розмітки XML і середовище розроблення дизайну Figma.

Апробація результатів дослідження. Результати виконання кваліфікаційної роботи пройшли апробацію й опубліковані у матеріалах доповідей студентської науково-практичної конференції «Інтелектуальні інформаційні технології в прикладних дослідженнях» (IITAR-2025), (Тернопіль, 27–29 травня 2025 р.). – Тернопіль: ЗУНУ, 2025. 372 с. (додаток Б).

1 АНАЛІЗ ПРОБЛЕМИ АНАЛІТИКИ ДЛЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ЧИТАННЯ КНИГ

1.1 Опис предметної області мобільного застосунку для рекомендаційної системи читання книг

Трекер для читання - це цифровий інструмент, який виконує функції читацького щоденника, забезпечуючи користувачам можливість фіксувати, аналізувати та відстежувати свій читацький прогрес за допомогою інформаційно-комунікаційних технологій. Такі застосунки поєднують елементи баз даних, користувацького інтерфейсу та аналітичних модулів, що дозволяє автоматизувати процес зберігання й обробки даних про прочитані книги та читацьку поведінку користувача [1-3].

Слід зазначити, що згідно з аналітичними даними, наведеними у статті «Книги в Україні: як змінилася поведінка читачів під час повномасштабного вторгнення», опублікованій на ресурсі Forbes.ua, частка громадян України, які читають книги щодня, зросла більш ніж удвічі [4]. Така тенденція свідчить про зростання загального інтересу до читання, зокрема в цифровому форматі (рисунок 1.1).

Рисунок 1.1 – Читання різних видів книжок, 2023-2020 рр.

Як видно з рисунка 1.1, попит на електронні книжки також демонструє позитивну динаміку, що безпосередньо впливає на актуальність розробки та вдосконалення цифрових інструментів для підтримки читацької активності. У контексті комп'ютерних наук це створює передумови для активного застосування інформаційних технологій, зокрема мобільних застосунків, хмарного зберігання даних та інтелектуального аналізу користувацької поведінки, з метою створення персоналізованих рішень для цифрового читання [5-8].

Мобільний застосунок аналітики для рекомендаційної системи читання книг входить до класу інформаційних систем, призначених для персонального користування з метою збору статистичних даних про читання книг і отримання індивідуальних рекомендацій.

У рамках предметної області можна виділити такі основні функції: додавання книг, відстеження часу читання, формування аналітичних звітів та персональних рекомендацій. При додаванні книг обов'язково потрібно ввести назву книги, автора, жанр та кількість сторінок. Ці дані будуть використані під час аналітики. Після прочитання можна оцінити книгу від одного до п'яти.

Відстежування часу читання необхідне для формування місячних та річних аналітичних звітів. За допомогою цього можна відстежувати, які жанри та автори найбільш популярні для користувача. Персональні рекомендації формуються на основі збору та аналізу статистичних даних при додаванні книги та її оцінювання.

1.2 Огляд і аналіз існуючих рішень

У галузі комп'ютерних наук значну увагу приділяють розробці мобільних застосунків, які підтримують персоналізоване читання та надають інструменти для глибокої аналітики користувацької поведінки. Такі застосунки, зокрема трекери читання, поєднують інтерфейси користувача, механізми збору даних, рекомендаційні алгоритми та візуалізацію інформації, що вимагає комплексного підходу до їх розробки та реалізації [2, 3, 9-12]. Для визначення основних функціональних вимог і перспектив розвитку було проведено огляд існуючих

мобільних рішень, зосередившись на застосунках, доступних в Україні, та світових лідерах галузі [13–17].

Пошук аналогів у Google Play Маркет показав, що трекери для читання книг поки що недостатньо поширені на українському ринку. Водночас серед локальних продуктів варто відзначити український застосунок Rork, який поєднує функції аналітики та рекомендацій, і був обраний для детального дослідження. Для порівняння також було проаналізовано три англомовних застосунки: Goodreads, StoryGraph та Basmo, що відрізняються своїми підходами до відстеження читання та рекомендацій [6–9, 16, 18].

Для систематизації і порівняння функціоналу використано метод таблиці конкурентів, який дозволяє візуалізувати рівень реалізації ключових функцій у кожному додатку (таблиця 1.1) [10, 18, 19].

Таблиця 1.1 – Таблиця конкурентів

Існуючі рішення	Goodreads		StoreGraph		Basmo		Rork	
Додавання книг	0	-	0	-	2	+	3	+
Відстеження часу читання	0	-	0	-	1	+	3	+
Оцінка книги	3	+	3	+	1	+	3	+
Аналітичні звіти	0	-	3	+	3	+	3	+
Рекомендації	3	+	2	+	0	-	1	+
Середній бал	1.2		1.6		1.4		2.6	

Оцінки були виставленні наступним чином:

- 0 балів – немає реалізації;
- 1 бал – погана реалізація;
- 2 бали – нормальна реалізація;
- 3 бали – ідеальна реалізація.

Функція додавання книг є базовою для трекерів читання, оскільки від її якості залежить точність подальшої аналітики і рекомендацій. У додатку Basmo доступні такі параметри, як назва, автор, ISBN і обкладинка (рисунок 1.2).

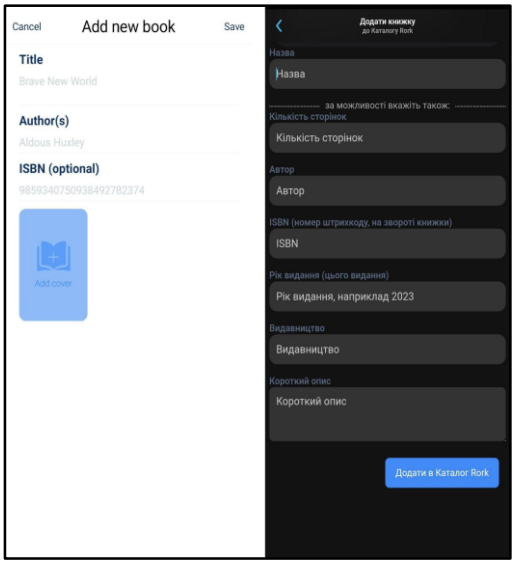


Рисунок 1.2 – Екран «Додавання книги» додатків Basmo та Rork

Автоматичне розпізнавання книги за ISBN - це стандартна практика, що значно полегшує взаємодію користувача із системою та забезпечує коректне наповнення бази даних [11]. Однак, цей додаток орієнтований переважно на паперові книги, і не дозволяє вводити жанр чи кількість сторінок, що ускладнює аналіз електронних версій та точність рекомендацій.

Rork розширює функціонал, дозволяючи вводити додаткові атрибути - кількість сторінок, рік видання, видавництво та опис книги, що покращує можливості алгоритмів рекомендацій [6, 13, 16].

Відстеження часу читання є важливим інструментом для аналізу поведінки користувача. У Basmo ця функція реалізована, проте має низьку якість UX, зокрема

через порушення принципу балансу в дизайні інтерфейсу, що негативно впливає на зручність використання (рисунок 1.3) [12, 17].

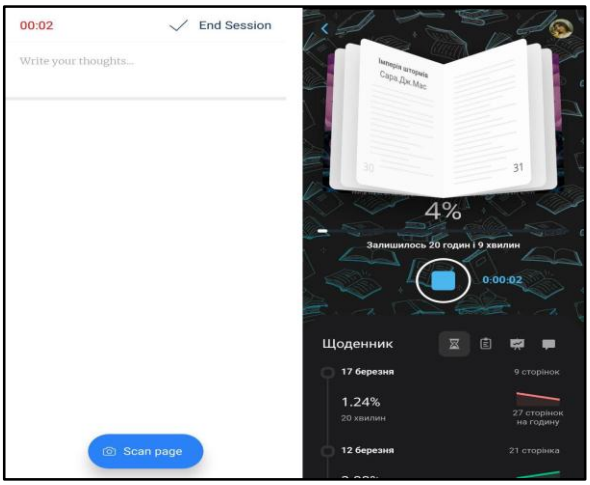


Рисунок 1.3 – Екрани «Відстеження часу читання» у додатках Basmo і Rork

У додатку Rork відстеження часу читання інтегроване більш гармонійно, що дозволяє користувачам ефективно контролювати свій прогрес і підвищує інформативність аналітики.

Оцінювання книг є ключовою функцією для побудови рекомендаційних систем. У Goodreads, StoryGraph та Rork ця функція реалізована на високому рівні: користувач може оцінити книгу одразу після її прочитання, що сприяє отриманню об’єктивних та своєчасних відгуків (рисунок 1.4).

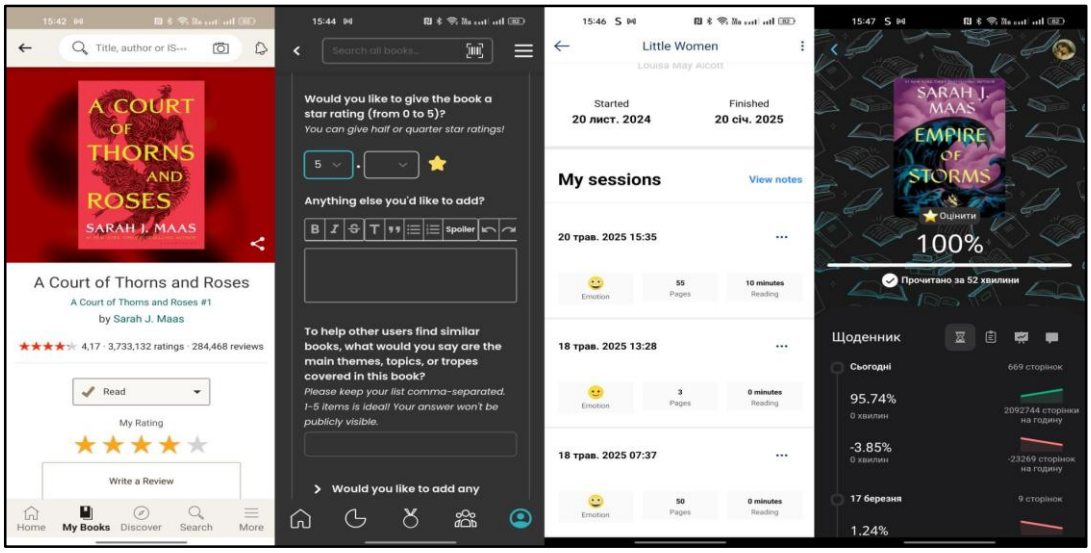


Рисунок 1.4 – Екран «Оцінка книги» у додатках goodreads, StoreGraph, Basmo та Rork

У Basmo оцінювання обмежується вибором «емоджі» за читальну сесію, що менш ефективно для рекомендаційних алгоритмів, але може бути цікавим для емоційного фідбеку.

Аналітика у трекерах забезпечує візуалізацію даних про прогрес читання. У StoryGraph і Basmo це реалізовано через графіки, які дозволяють користувачам відслідковувати свої досягнення (рисунок 1.5).

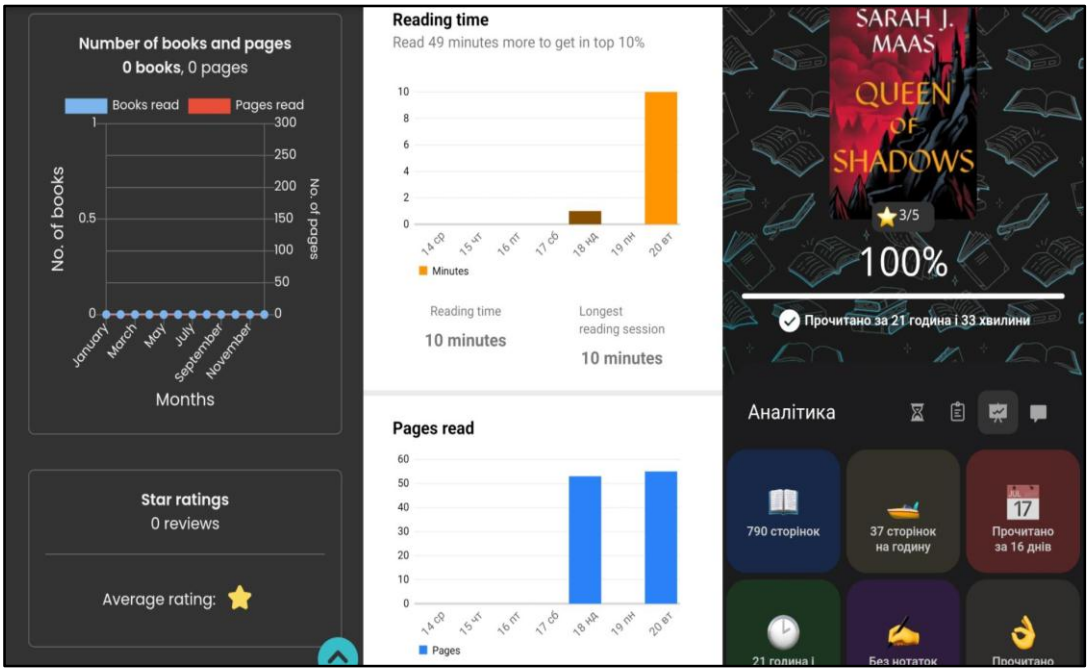


Рисунок 1.5 – Екран «Аналітичні звіти» у додатках StoreGraph, Basmo та Rork

Rork також надає розширені аналітичні функції, включаючи прогнозування читацьких показників на рік у платній версії, що відповідає сучасним тенденціям розвитку мобільної аналітики [2, 14-16].

Рекомендаційні системи у розглянутих додатках базуються на різних підходах. Goodreads пропонує рекомендації на основі оцінок користувачів, однак не дозволяє переглянути опис книги без переходу до зовнішніх джерел (рисунок 1.6). StoryGraph забезпечує різні види рекомендацій, але не всі вони персоналізовані. У Rork система рекомендацій сформована переважно на основі рекомендацій інших користувачів, без урахування метаданих та оцінок, що знижує точність та індивідуалізацію рекомендацій [3, 15, 16].

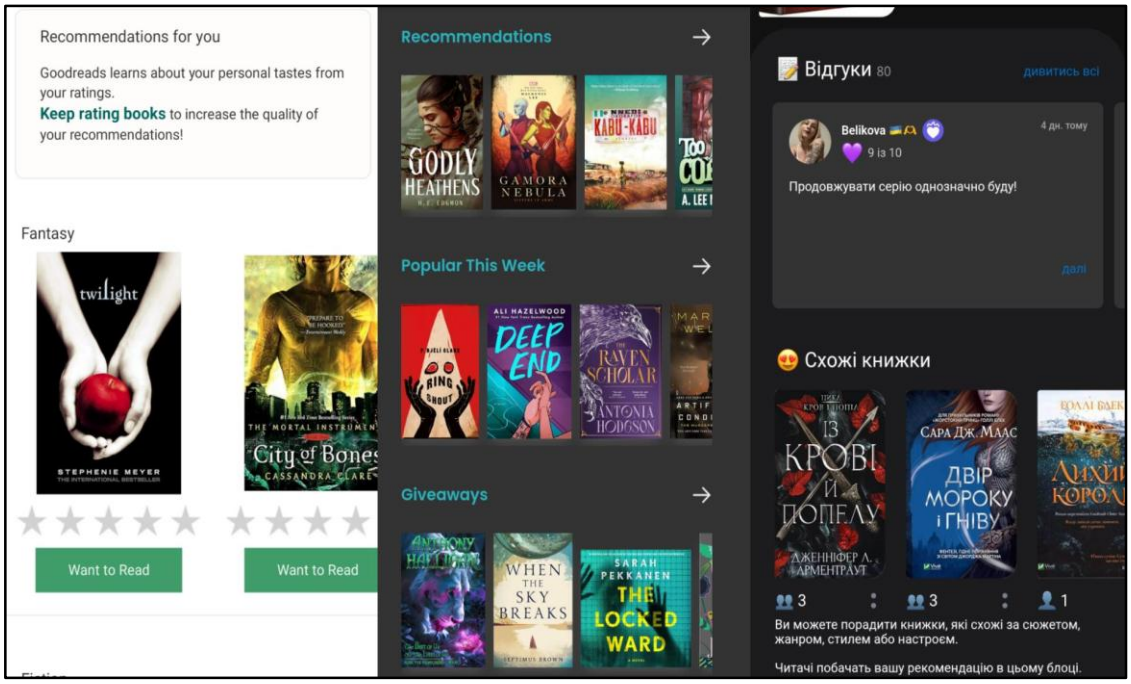


Рисунок 1.6 – Екран «Рекомендації» у додатках goodreads, StoreGraph і Rork

Проведений аналіз дозволив окреслити ключові функціональні недоліки та ефективні практики в існуючих мобільних трекерах для читання. Це створює підґрунтя для розробки нового застосунку, орієнтованого на покращену аналітику читацької активності для більш релевантного та персоналізованого добору книжкових рекомендацій.

1.3 Постановка задачі розроблення мобільного застосунку аналітики для рекомендаційної системи читання книг

На основі попереднього аналізу предметної області, структури існуючих рішень і функціональних характеристик мобільних трекерів для читання, у цьому підрозділі сформульовано загальні положення та завдання, що лягають в основу проєктування програмного рішення. Постановка задачі включає визначення функціональних вимог до застосунку, його логічної структури, основних модулів, а також принципів забезпечення надійності та цілісності даних [6, 14, 18-20].

Для підвищення ефективності збору, обробки та аналізу читацьких даних у модулі, що проєктується, заплановано реалізувати низку інженерних рішень, зокрема побудову діаграми дерева функцій, діаграми варіантів використання та

діаграми діяльності. Ці інструменти дозволили формалізувати функціональну структуру системи, визначити взаємозв’язки між її модулями та описати сценарії взаємодії з користувачем. Реалізацію програмного продукту (мобільного застосунку) здійснено у середовищі Android Studio, з використанням мови Kotlin, що дозволяє забезпечити сумісність з переважною більшістю сучасних мобільних пристроїв [6, 15, 19-22].

Діаграма дерева функцій є одним із ключових інструментів, що дозволяє структурувати функціональність системи. Вона демонструє ієрархічний розподіл функцій мобільного застосунку за категоріями, зокрема: управління книгами, трекінг прогресу читання, аналітична обробка даних та рекомендаційна система [8, 17]. Для мобільного застосунку аналітики для рекомендаційної системи було побудовано діаграму дерева функцій, яка зображена на рисунку 1.7.

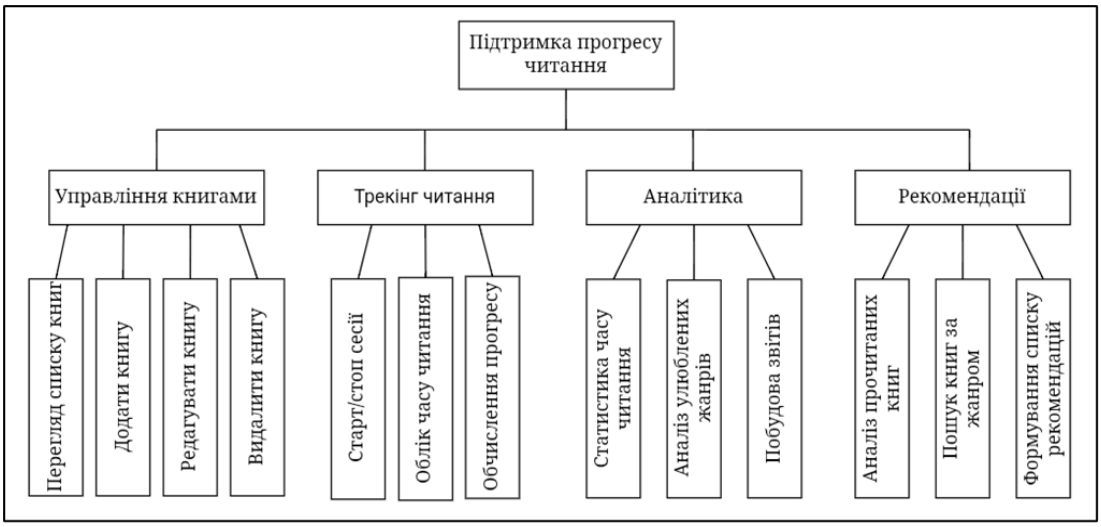


Рисунок 1.7 – Діаграма дерева рішень для підтримки прогресу читання

Транзакційна складова реалізована як логічна одиниця програмної архітектури, що забезпечує збереження цілісності даних у процесі виконання завершених дій користувача. Основними транзакціями в межах мобільного застосунку є: додавання книги до бібліотеки, відстеження сесій читання та виставлення оцінки книзі [7, 16]. Усі транзакції реалізуються за принципом «усе або нічого», що гарантує достовірність інформації для подальшої аналітики й уникнення хибних рекомендацій. У випадку збою системи незавершені транзакції не зберігаються, що запобігає накопиченню некоректних даних.

Аналітична підсистема реалізована як модуль збору, агрегування та візуалізації читацьких даних. Вона забезпечує аналіз часу читання, обчислення кількості прочитаних сторінок, ідентифікацію найчастіше обираних жанрів і відстеження прогресу в динаміці (за день, місяць, рік) [11, 18, 19]. На основі цих даних формуються аналітичні звіти, що інтегруються у механізм персоналізованих рекомендацій. Основою рекомендаційного алгоритму є статистична обробка оцінених книг, аналіз жанрових уподобань користувача та історії його останніх прочитаних видань [16, 20].

Оскільки метою кваліфікаційної роботи визначено створення мобільного застосунку, який забезпечить зручне та ефективне середовище для збору, зберігання й аналітики даних читацького прогресу, а також формування персоналізованих книжкових рекомендацій на їх основі [13, 14], застосунок, що розробляється, орієнтований на користувачів, які прагнуть структурувати власний читацький досвід та отримувати релевантні пропозиції щодо нових книг.

Особливістю архітектури системи є обрано локальну реалізацію рекомендаційної логіки, що не потребує з’єднання з зовнішніми серверами. Алгоритм рекомендацій побудований на внутрішньому порівнянні параметрів книг, зокрема: жанру, оцінки, обсягу й частоти читання [15, 17]. Інтерфейс користувача розроблений відповідно до принципів UX/UI-дизайну, з урахуванням адаптивності до різних розмірів екранів та з інтуїтивною логікою навігації [12, 21].

Аналітичні та рекомендаційні модулі активуються на основі введених користувачем даних при додаванні книг до бібліотеки, зокрема: назви, автора, жанру, кількості сторінок, обкладинки та за потреби опису [9, 13, 14]. Після завершення читання користувач має можливість оцінити книгу, що слугує ключовим індикатором для формування релевантних рекомендацій у майбутньому.

2 АЛГОРИТМІЧНЕ ТА ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ ЗАСТОСУНКУ

2.1 Загальна структура проєктованого застосунку

Для розробки мобільного застосунку аналітики для рекомендаційної системи спроєктовано архітектуру на основі наступних таблиць та діаграм: перелік функцій та їх характеристики; інформаційні зв’язки між елементами системи; діаграма дерева функцій; діаграма варіантів використання; діаграма діяльності; блок-схема логіки; компонентна діаграма; діаграма розгортання та архітектурна схема.

Перелік основних функцій мобільного застосунку подано у таблиці 2.1.

Таблиця 2.1– Перелік функцій та їх характеристики

№	Назва	Характеристика
1	Додавання книги	Дозволяє користувачу ввести нову книгу, вказавши її назву, автора, жанр, кількість сторінок, опис і обкладинку
2	Редагування/видалення книги	Дозволяє оновлювати дані книги або видаляти її з бібліотеки
3	Відстежування часу читання	Фіксує скільки часу користувач витратив на читання книги
4	Побудова аналітики	Формує аналітичні звіти та календарі
5	Формування рекомендацій	На основі жанру та оцінки рекомендує схожі книги з бази додатку

У програмуванні розрізняють два види алгоритмів: функції та процедури.

Функція – це незалежна іменована частина програми, яку можна викликати для виконання певних дій. Вона дозволяє знайти лише одне рішення і повернути його в точку виклику. Ім'я функції може бути частиною виразу (входити як операнд) [22].

Процедура має відмінність від функції в тому, що вона не може бути частиною виразу (операндом). Після виконання процедури завжди можна використати кілька її результатів на відміну від функції.

Діаграма варіантів використання або діаграма прецедентів – це діаграма, за допомогою якої ілюструють контекст системи, яку потрібно розробити, а також її функціонал [23, 24].

Актори (або учасники) – це ролі, які знаходяться поза системою і логічно пов’язані зі сутностями або прецедентами. Акторами можуть бути як люди, так і інша система, які знаходяться поза межами сутності. Акторів прийнято позначати «людьми».

Прецеденти (або варіанти використання) – це опис типових сценаріїв взаємодії актора і прецедента. За допомогою діаграми прецедентів відображають те, як система повинна реагувати на запити користувача або іншої системи і, у разі потреби, удосконалити її.

Для мобільного застосунку аналітики для рекомендаційної системи було побудовано діаграму варіантів використання, яка зображена на рисунку 2.1.

На діаграмі показано основні прецеденти та основні зв’язки (непреривна лінія). Також на діаграмі зображені такі зв’язки як <<include>> та <<extend>>.

Зв’язок <<include>> показує зв’язки без яких дія не відбудеться: вони взаємопов'язані. Наприклад, прецедент «Отримання рекомендацій» пов’язаний цим зв’язком з прецедентом “Оцінювання книг”. Це означає, що без оцінювання книг не відбудеться отримання рекомендацій, так як рекомендації будуються на основі оцінки книги користувачем після прочитання.

Зв’язок <<extend>> – це такий зв’язок, який показує можливий варіант використання, але він не є обов’язковим. Наприклад, прецедент «Отримання рекомендацій» пов’язаний з прецедентом «Перегляд списку книг». Це означає, що для формування рекомендацій не обов’язково буде використовуватись список доданих книг. Цей список буде використовуватись для рекомендаційної системи лише до того часу, поки користувач не оцінить першу книгу. Після того, рекомендаційна система буде працювати за рахунок оцінювання користувачем прочитаних книг.

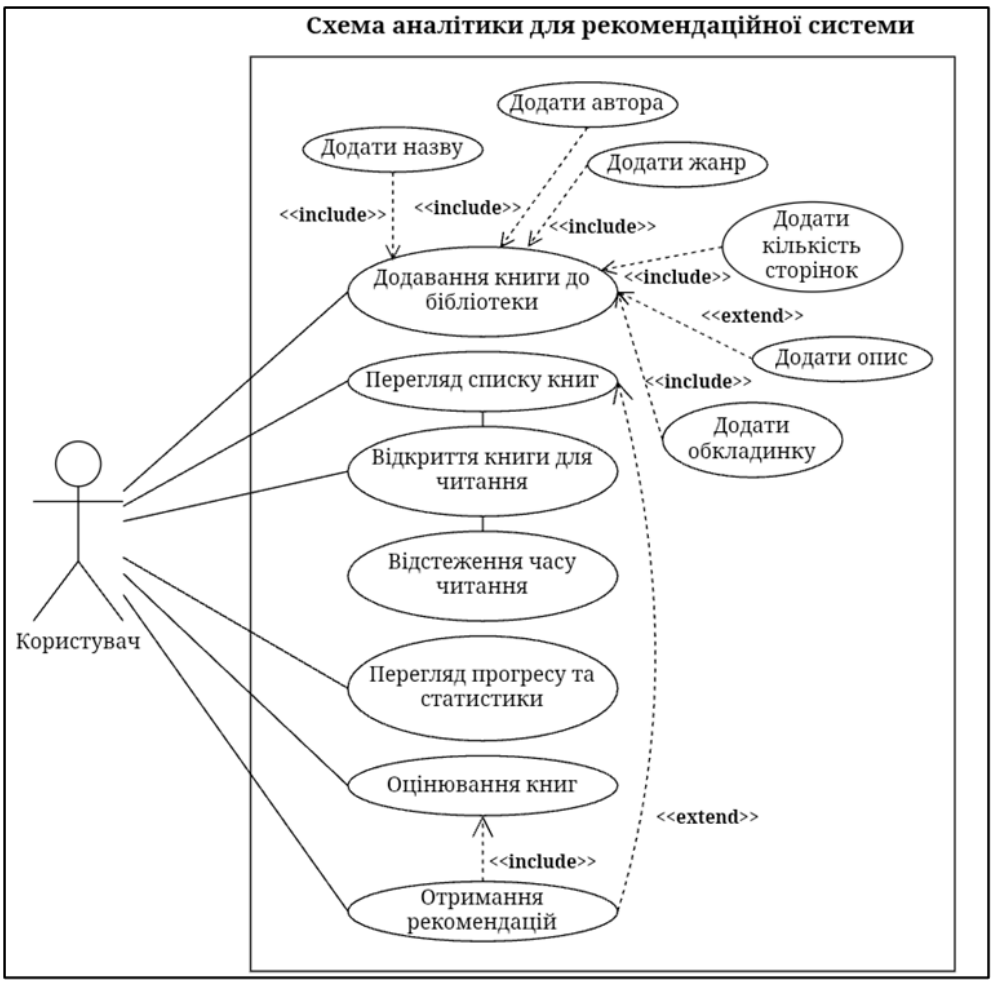


Рисунок 2.1 – Діаграма прецедентів аналітики для рекомендаційної системи

Діаграма компонентів – це діаграма UML для систем великих за обсягом, за допомогою якої ці системи діляться на менші підсистеми і реалізують зв’язки між ними [15, 23].

Компонент – це змінна та виконувана частина системи, деталі якої є прихованими. Діаграма компонентів для мобільного застосунку аналітики для рекомендаційної системи читання книг зображена на рисунку 2.2.

На діаграмі є п’ять компонентів, а саме: інтерфейс користувача, обробка даних, аналітика, рекомендації та локальне сховище.

Інтерфейс користувача відповідає за правильне графічне виведення інформації та взаємодію з користувачем. Тут знаходяться такі екрани, як додати книгу, відстежити час читання, аналітика та рекомендації.

Обробка даних відповідає за прийняття та оброблення даних, які поступають від користувача. Після того цей компонент з'єднується з двома наступними: аналітика та рекомендації.

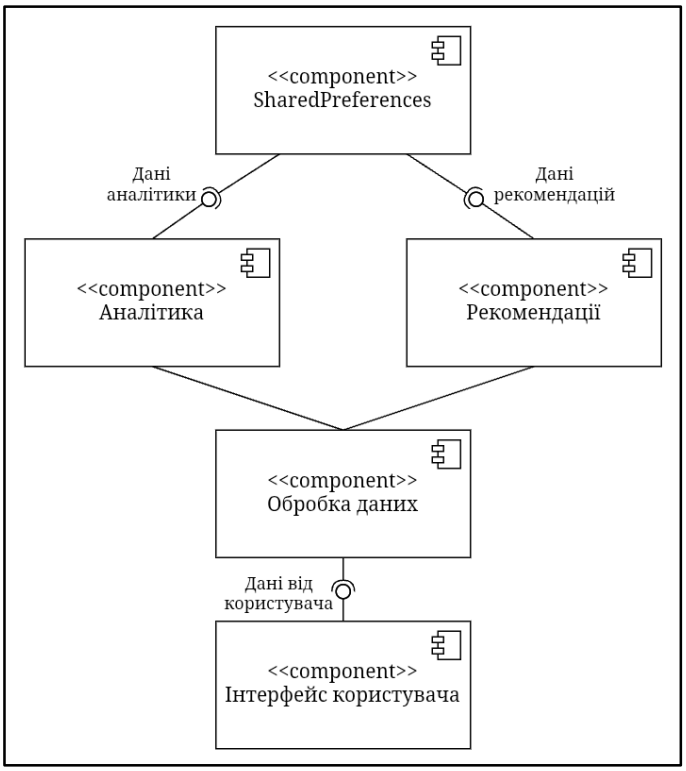


Рисунок 2.2 – Діаграма компонентів для мобільного застосунку аналітики для рекомендаційної системи читання книг

Аналітика відповідає за оброблення даних за допомогою яких формує аналітичні звіти, такі як кількість прочитаних сторінок; кількість прочитаних книг; час, витрачений на читання книги тощо.

Рекомендації формують список рекомендованих книг для користувача на основі його уподобань, а саме за рахунок виставлених оцінок.

Останнім компонентом є локальне сховище SharedPreferences. Тут зберігаються дані про книги, час і тд.

Діаграма розгортання – це діаграма, яка показує, як розгортається програмне забезпечення (ПЗ) на обчислювальні елементи.

Для мобільного застосунку аналітики для рекомендаційної системи читання книг було побудовано діаграму розгортання зображену на рисунку 2.3.

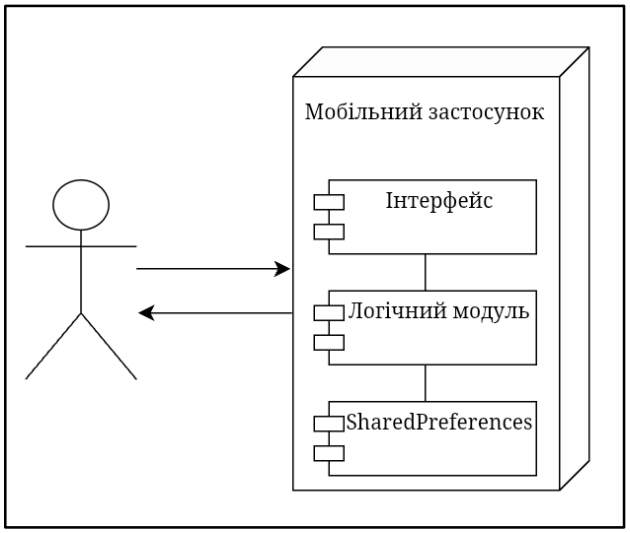


Рисунок 2.3 – Діаграма розгортання для мобільного застосунку аналітики для рекомендаційної системи читання книг

Особливістю мобільного застосунку є його спрощена архітектура. Він не вимагає підключення до мережі, щоб виконувати свої функції. Замість стандартних баз даних використовується внутрішнє сховище, яке називається SharedPreferences. Це забезпечить автономність та швидкість у використанні.

Архітектурна схема або архітектура програмного забезпечення (ПЗ), що розробляється – це схема, яка описує організацію системи та зв’язки між компонентами [16, 27].

Для мобільного застосунку аналітики для рекомендаційної системи читання книг було створено схему архітектури ПЗ (рисунок 2.4). Виділено три рівні:

– рівень інтерфейсу користувача;

– рівень домену;

– рівень даних.

Рівень інтерфейсу користувача забезпечує взаємодію з користувачем. На цьому рівні знаходяться такі компоненти, як інтерфейс, візуалізація аналітики, навігація між екранами та редагування і видалення книги.

Наступним рівнем є – рівень домену. Він забезпечує реалізацію бізнес-правил та обробку подій. На цьому рівні знаходяться управління книгами, фіксація часу читання, аналіз читання та формування рекомендацій.

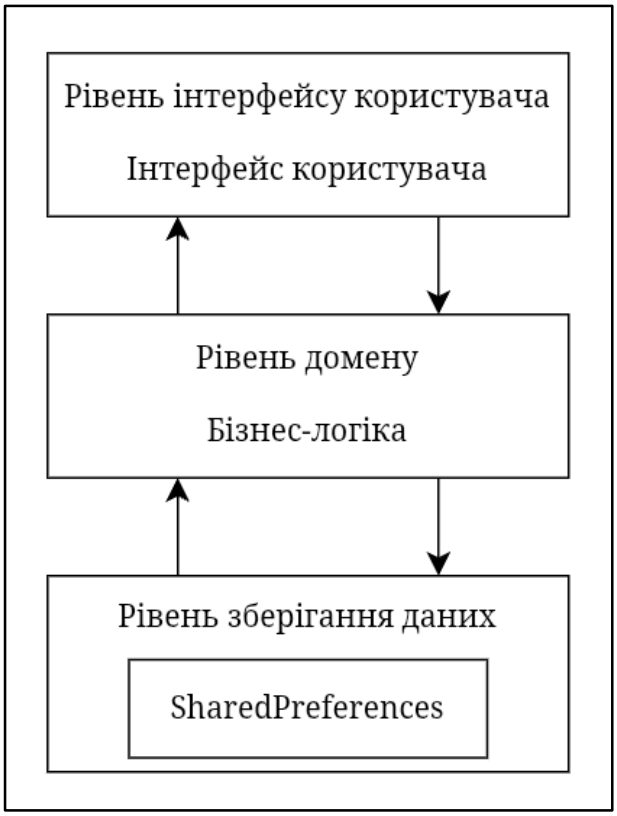


Рисунок 2.4 – Схема архітектури ПЗ мобільного застосунку аналітики для рекомендаційної системи читання книг

Останнім рівнем є рівень даних. Він забезпечує локальне зберігання даних користувача. Сюди входить компонент SharedPreferences.

2.2 Алгоритмічне забезпечення

Не менш важливим є алгоритмічне забезпечення продукту, що розробляється. Алгоритм діяльності – це модель діяльності, яка відображає послідовність виконання певних робіт, яку може виконувати як користувач, так і система (застосунок). Послідовність робіт залежить від прийнятих користувачем або системою рішень [24, 25].

Окрема діяльність (робота) позначається на діаграмі прямокутником із закругленими кутами. Прийняття рішення, згідно прийнятої графічної нотації, позначається витягнутим ромбом. Потоки керування позначаються стрілками. Від рішення виступають дві стрілки, на яких пишуть текст умови, якій вони відповідають.

Для мобільного застосунку аналітики для рекомендаційної системи було створено діаграму діяльності на якій показано, які можливі основні дії користувач може зробити для отримання рекомендацій (рисунок 2.5).

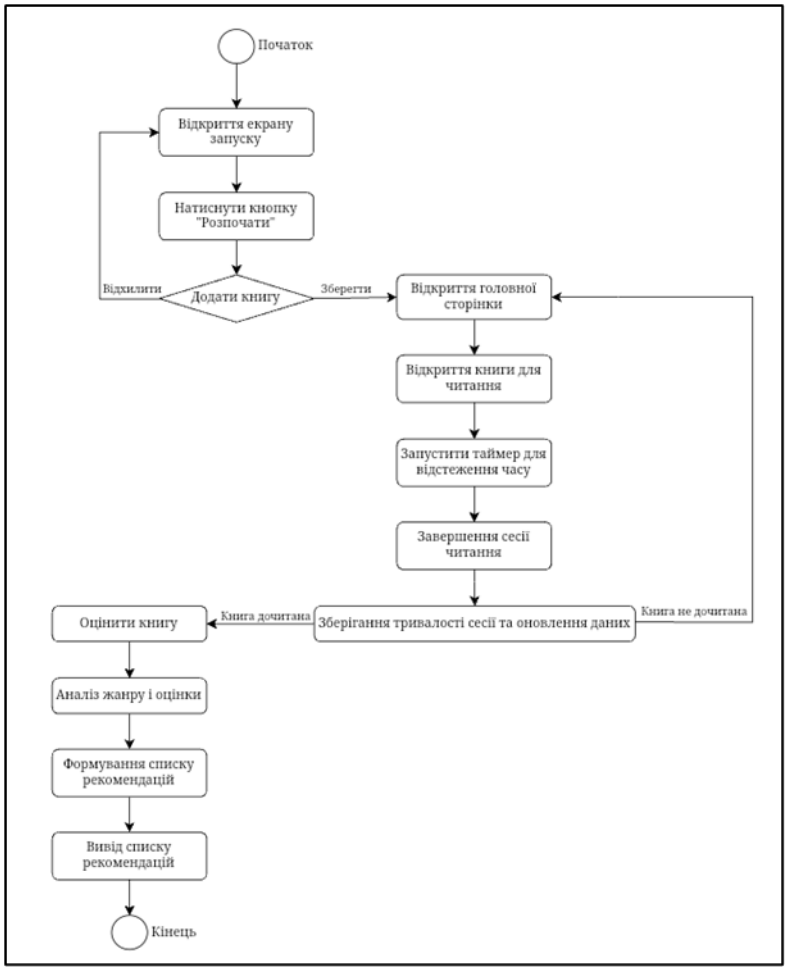


Рисунок 2.5 – Діаграма діяльності для отримання рекомендацій

Блок-схема – це графічне відображення процесу або алгоритму. Вона складається з різних наборів блоків. Вони показують процес, дані, умови та цикли. Стрілками показують напрямки, у яких рухаються потоки даних [14].

Блок-схеми використовують для того, щоб розбити складну задачу на простішу. Блок-схеми дуже поширені у використанні для програм, інженерії та бізнесу. Для мобільного застосунку аналітики для рекомендаційної системи читання книг було побудовано блок-схему логіки (рисунок 2.6). Блок-схема логіки побудована для мобільного застосунку, щоб побачити основні дії користувача та логічний зв’язок між компонентами системи.

Початок та кінець схеми позначаються овалами. Після запуску програми йде перевірка чи є додана книга. Якщо немає доданих книг у бібліотеці (новий користувач/перший вхід) управління виконанням повертає до елементу “Додати книгу”. Якщо умова виконана, то користувач переходить на наступний етап. На цьому етапі можна відстежити час читання книги, а також редагувати або видалити її. Після цього йде наступна умова: чи користувач прочитав книгу до кінця. Якщо так – відбувається побудова аналітики, після чого формуються персональні рекомендації. Якщо ні – користувач не переходить до наступного етапу.

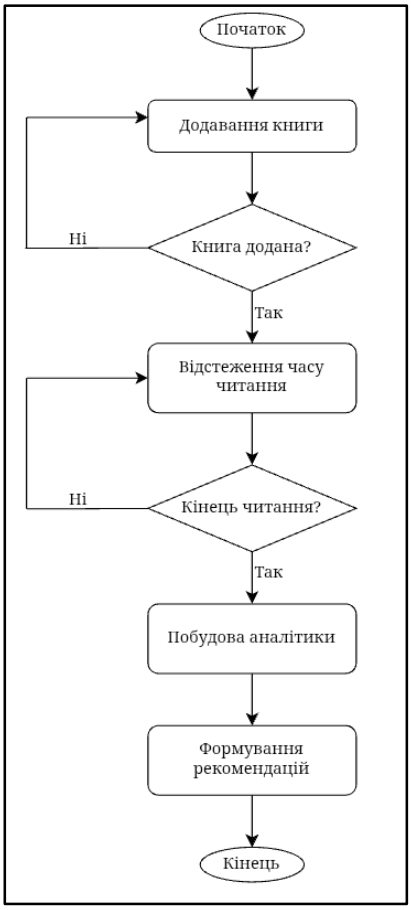


Рисунок 2.6 – Схема логіки для мобільного застосунку аналітики для рекомендаційної системи

Для виконання задачі проєктування застосунку рекомендаційної системи є доцільним використовувати модель функціонування системи. Узагальнена схема функціонування системи – це модель у графічному поданні, яка показує типові дії користувача і реакцію системи на ці дії. Саме тому для демонстрації діяльності обробки інформації було використано діаграму потоків даних щоб бачити у процесі

проєктування куди направляється та чи інша інформація, дані після дії користувача [25].

Отже, діаграма потоків даних (Data Flow Diagrams) – це діаграма, яка показує куди і які дані надходять, які вид діяльності вони обробляють, і чи зберігаються або використовуються вихідні результати іншим суб’єктом [28].

Проєктування мобільного застосунку аналітики для рекомендаційної системи читання книг було розроблено на основі процесно-орієнтованого підходу узагальнену схему функціонування системи за допомогою діаграми потоків даних (рисунок 2.7).

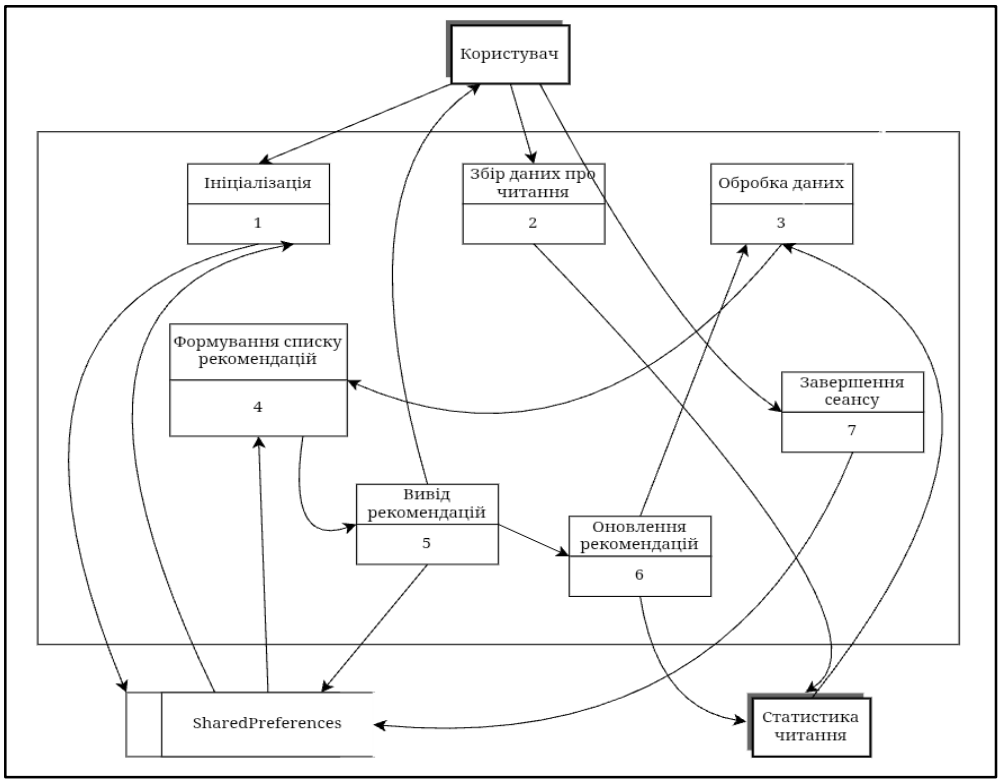


Рисунок 2.7– Діаграма потоків даних для мобільного застосунку аналітики для рекомендаційної системи читання книг

Ця діаграма будувалась на основі символів, запропонованих Гейном і Сарсоном. Їхня різниця від інших нотацій полягає в тому, що замість використання кола для процесів, використовуються прямокутники [28].

2.3 Інформаційне забезпечення

Інформаційне забезпечення (ІЗ) також відіграє важливу роль для розробки прикладного застосунку. ІЗ описує, які саме дані використовує система, звідки і куди вони надходять, та їх характер зв’язку.

Для мобільного застосунку аналітики для рекомендаційної системи читання книг було побудовано таблицю інформаційних зв’язків між елементами (таблиця 2.2).

Таблиця 2.2 – Інформаційні зв’язки між елементами системи

№	Елемент-джерело	Елемент-одержувач	Інформація, що передається	Характер зв’язку
1	Користувач	Додавання книги	Назва, автор, жанр, кількість сторінок, опис, обкладинка	Введення даних
2	Додавання книги	Shared Preferences	Збереження нової книги	Збереження
3	Відстежування часу читання	Облік прогресу	Кількість прочитаних сторінок	Передача статусу
4	Облік прогресу	Аналітика	Дані про час та обсяг читання	Внутрішня взаємодія
5	Аналітика	Вивід статистики	Аналітичні показники	Оброблена інформація
6	Shared Preferences	Формулювання рекомендацій	Список прочитаних книг, жанр, оцінки	Запит даних
7	Формування рекомендацій	Інтерфейс користувача	Рекомендовані книги	Вивід результату

Для збереження даних використовуватиметься локальне сховище даних, яке називається SharedPreferences. SharedPreferences – це API об’єкт, який вказує на файл, що містить пари ключ-значення і надає прості методи для читання та запису цих пар. Кожен файл керується платформою та може бути приватним або спільним. Цей метод використовують коли є відносно невелика колекція пар ключ-значень, яку потрібно зберегти [29].

У мобільному застосунку присутні як вхідні дані (наприклад, введення нової книги), так і вихідні дані (наприклад, вивід результатів рекомендацій).

Концептуальне, або інфологічне, проєктування – це етап розробки інформаційної системи, на якому описуються об’єкти інформаційної системи, її зв’язки та дані, які зберігаються, незалежно від способу реалізації.

Для мобільного застосунку аналітики для рекомендаційної системи читання книг було побудовано ER-модель (рисунок 2.8).

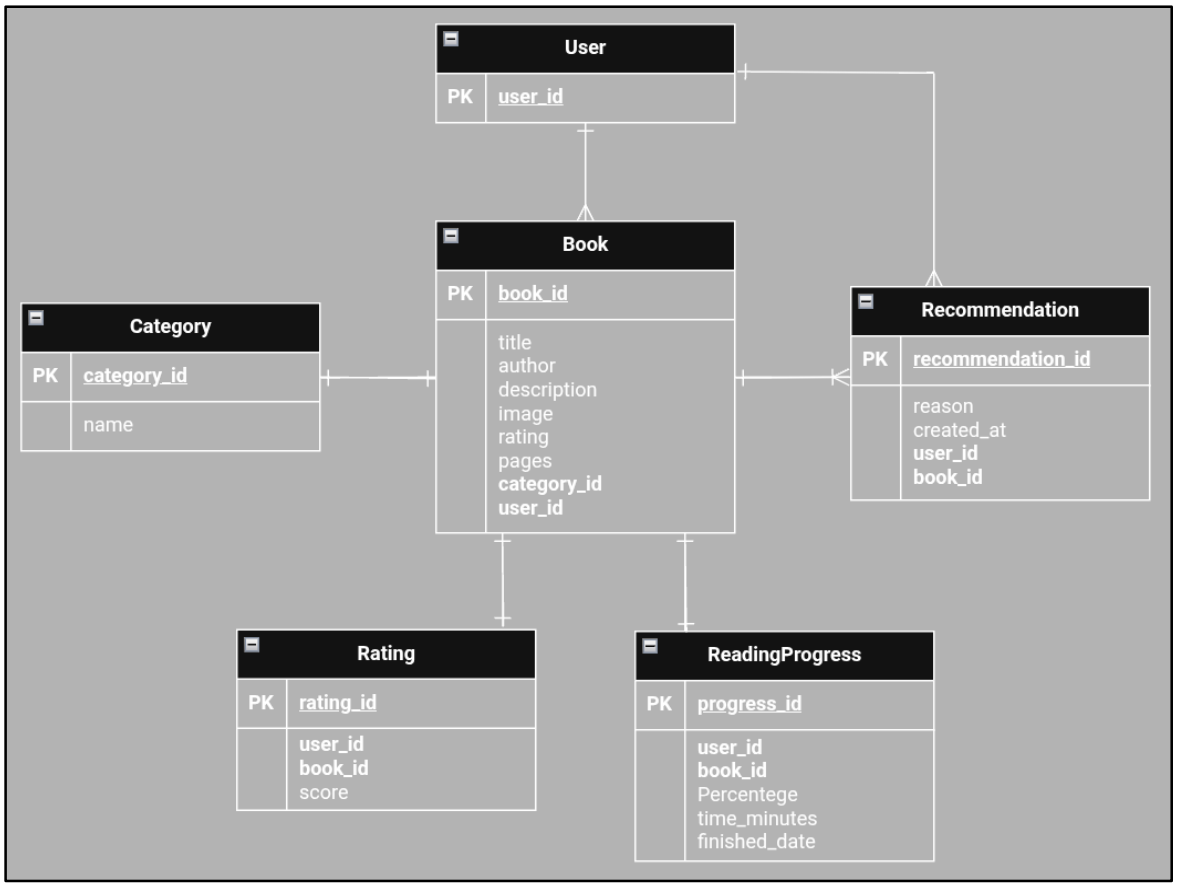


Рисунок 2.8 – ER-модель для мобільного застосунку аналітики для рекомендаційної системи

ER-модель, або модель сутність-зв’язок – це графічний підхід до проектування бази даних. Вона визначає елементи даних та їх зв’язок [30].
Сутність – це річ або об’єкт в реальному житті. Сутність має властивості, а властивості можуть мати значення. На рисунку 2.9 відображені зв’язки ER-моделі.

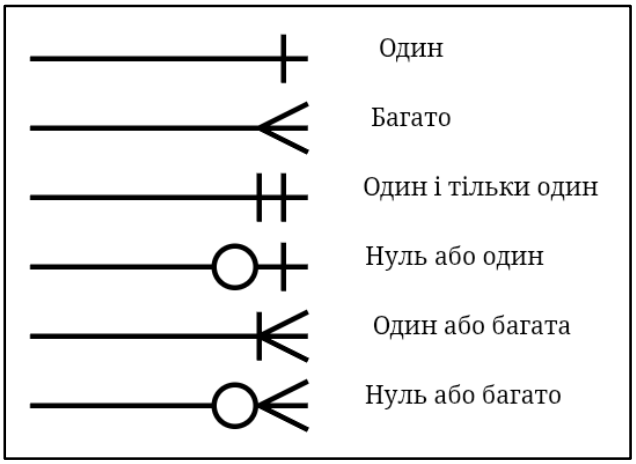


Рисунок 2.9 – Зв’язки ER-моделі

Проєктування фізичної моделі даних системи рекомендацій базується на жанрі та оцінці книг, прочитаних користувачем. Для формування рекомендацій використовується локальна база книг у застосунку, в яку попередньо було завантажено певний перелік книг. Ці книги зберігаються в SharedPreferences як окремий список рекомендованих книг, звідки система їх і бере.

3 ПРОГРАМНО-ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ ЗАСТОСУНКУ

3.1 Реалізація програмного забезпечення

Діаграма класів (class diagram) – це діаграма, яка показує сукупність декларативних або статичних елементів моделі (класи з атрибутами, операції) та їх зв’язків [31].

Класи – це опис об’єктів, які мають однакові атрибути, операції та відношення між класами. На діаграмі класи позначають прямокутниками з гострими кутами і розділеними горизонтально на дві або три частини. Зверху записується назва класу. Нижче – атрибути (-attribute), потім пишуться операції (+operation()).

Для мобільного застосунку аналітики для рекомендаційної системи було створено скорочену діаграму класів зображену на рисунку 3.1.

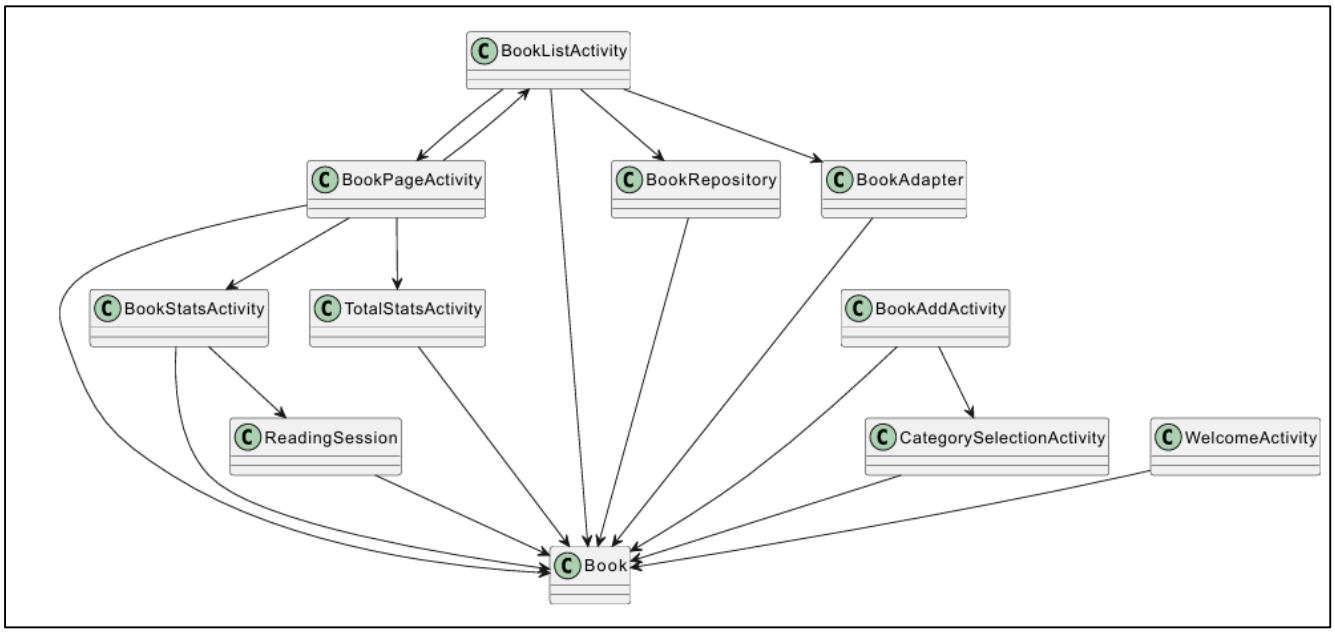


Рисунок 3.1– Скорочена діаграма класів мобільного застосунку

Для реалізації програмного забезпечення модуля використано такі програмні середовища розробки, як Figma, Android Studio, мова програмування Java, розширювана мову розмітки XML та локальне сховище даних SharedPreferences.

Середовище Figma використовувалось для створення дизайну для мобільного застосунку.

Для розробки мобільного застосунку було обрано середовище розробки Android Studio. Android Studio – це інтегроване середовище розробки виробництва Google. Це середовище забезпечує зручні інструменти для розробки застосунку на Android [32].

У класі BookListActivity знаходиться метод loadRecommendedBooks(). Цей метод відповідає за формування персональних рекомендацій.

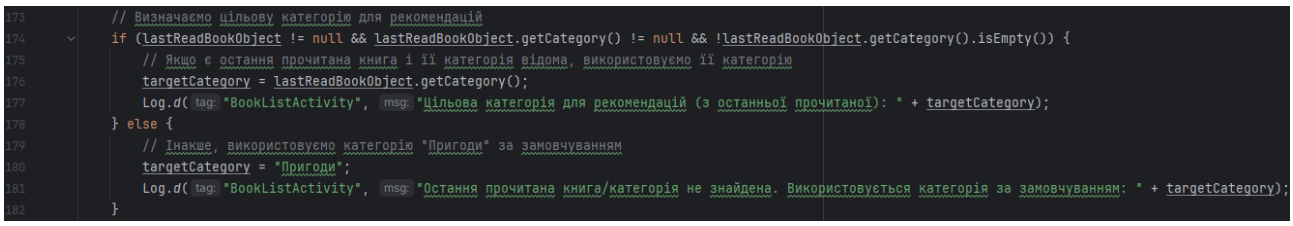
У першу чергу відкривається потік для читання файлу «recommended_books.json», після чого відбувається парсинг JSON у список об’єктів Book. Тоді отримуємо ID останньої прочитаної книги у SharedPreferences і шукаємо об’єкт останньої прочитаної книги серед завантажень користувача (рисунок 3.2).

```
139     private void loadRecommendedBooks() {
140         Log.d( tag: "BookListActivity",  msg: "loadRecommendedBooks: Початок завантаження рекомендованих книг");
141         Gson gson = new Gson();
142         Type bookListType = new TypeToken<ArrayList<Book>>() {}.getType();
143         List<Book> allBooksFromJson = new ArrayList<>();
144 
145         try {
146             // Відкриваємо потік для читання файлу "recommended_books.json" з теки assets
147             InputStream inputStream = getAssets().open( fileName: "recommended_books.json");
148             InputStreamReader reader = new InputStreamReader(inputStream, StandardCharsets.UTF_8);
149             // Розпарсуємо JSON у список об'єктів Book
150             allBooksFromJson = gson.fromJson(reader, bookListType);
151             reader.close(); // Важливо закривати потоки
152             inputStream.close();
153         } catch (IOException e) {
154             Log.e( tag: "BookListActivity",  msg: "Помилка завантаження recommended_books.json", e);
155             Toast.makeText( context: this,  text: "Не вдалося завантажити рекомендовані книги", Toast.LENGTH_SHORT).show();
156             return;
157         }
158 
159         String targetCategory = null;
160         Book lastReadBookObject = null; // Об'єкт останньої прочитаної книги
161 
162         // Отримуємо ID останньої прочитаної книги зі SharedPreferences
163         if (lastReadBookId != null) {
164             // Шукаємо об'єкт останньої прочитаної книги серед завантажених книг користувача
165             for (Book book : bookList) {
166                 if (lastReadBookId.equals(book.getId())) {
167                     lastReadBookObject = book;
168                     break;
169                 }
170             }
171         }
```

Рисунок 3.2 – Код для рекомендаційної системи

Наступним кроком визначається цільова категорія для рекомендацій. Для цього проходить перевірка: якщо є остання прочитана книга категорія якої відома

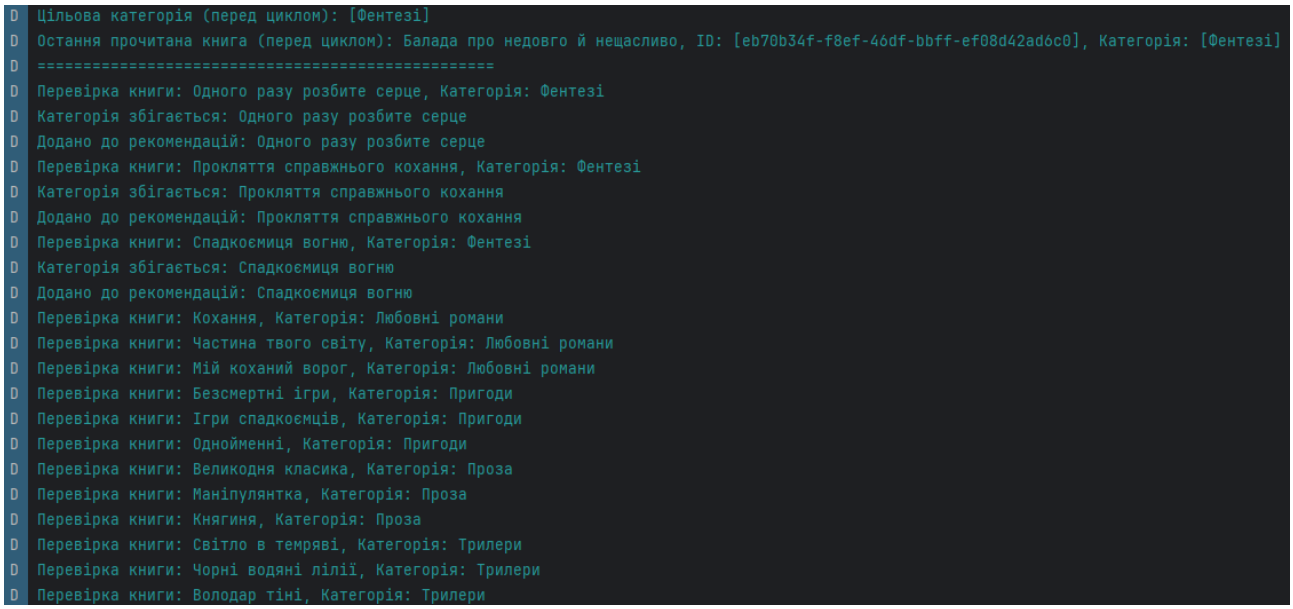
– використовується її категорія; якщо такої книги немає – рекомендуються книги з жанру «Пригоди» (рисунок 3.3).



```
// Визначаємо цільову категорію для рекомендацій
if (lastReadBookObject != null && lastReadBookObject.getCategory() != null && !lastReadBookObject.getCategory().isEmpty()) {
    // Якщо є остання прочитана книга і її категорія відома, використовуємо її категорію
    targetCategory = lastReadBookObject.getCategory();
    Log.d( tag: "BookListActivity",  msg: "Цільова категорія для рекомендацій (з останньої прочитаної): " + targetCategory);
} else {
    // Інакше, використовуємо категорію "Пригоди" за замовчуванням
    targetCategory = "Пригоди";
    Log.d( tag: "BookListActivity",  msg: "Остання прочитана книга/категорія не знайдена. Використовується категорія за замовчуванням: " + targetCategory);
}
```

Рисунок 3.3 – Код для визначення цільової категорії для рекомендацій

Після того, як було визначено цільову категорію потрібно відфільтрувати книги з JSON за цільовою категорією (рисунок 3.4).



```
D Цільова категорія (перед циклом): [Фентезі]
D Остання прочитана книга (перед циклом): Балада про недовго й нещасливо, ID: [eb70b34f-f8ef-46df-bbff-ef08d42ad6c0], Категорія: [Фентезі]
D ==================================================
D Перевірка книги: Одного разу розбите серце, Категорія: Фентезі
D Категорія збігається: Одного разу розбите серце
D Додано до рекомендацій: Одного разу розбите серце
D Перевірка книги: Прокляття справжнього кохання, Категорія: Фентезі
D Категорія збігається: Прокляття справжнього кохання
D Додано до рекомендацій: Прокляття справжнього кохання
D Перевірка книги: Спадкоємиця вогню, Категорія: Фентезі
D Категорія збігається: Спадкоємиця вогню
D Додано до рекомендацій: Спадкоємиця вогню
D Перевірка книги: Кохання, Категорія: Любовні романи
D Перевірка книги: Частина твого світу, Категорія: Любовні романи
D Перевірка книги: Мій коханий ворог, Категорія: Любовні романи
D Перевірка книги: Безсмертні ігри, Категорія: Пригоди
D Перевірка книги: Ігри спадкоємців, Категорія: Пригоди
D Перевірка книги: Однойменні, Категорія: Пригоди
D Перевірка книги: Великодня класика, Категорія: Проза
D Перевірка книги: Маніпулянтка, Категорія: Проза
D Перевірка книги: Княгиня, Категорія: Проза
D Перевірка книги: Світло в темряві, Категорія: Трилери
D Перевірка книги: Чорні водяні лілії, Категорія: Трилери
D Перевірка книги: Володар тіні, Категорія: Трилери
```

Рисунок 3.4 – Приклад фільтрації за цільовою категорією Logcat

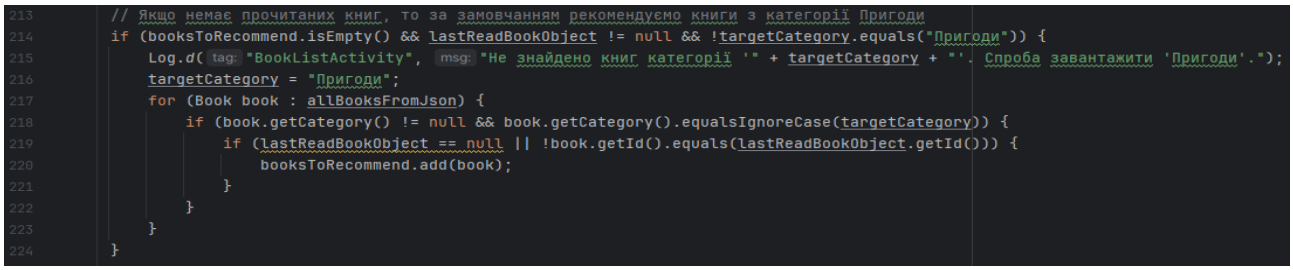
Для цього потрібно пройти перевірку чи категорія книги збігається з цільовою. Якщо вона не збігається, то ця книга пропускається (рисунок 3.5).

Якщо прочитаних книг немає, то за замовчуванням рекомендуються книги з категорії «Пригоди» (рисунок 3.6).

Останнім кроком є виведення трьох рекомендованих книг (рисунок 3.7).

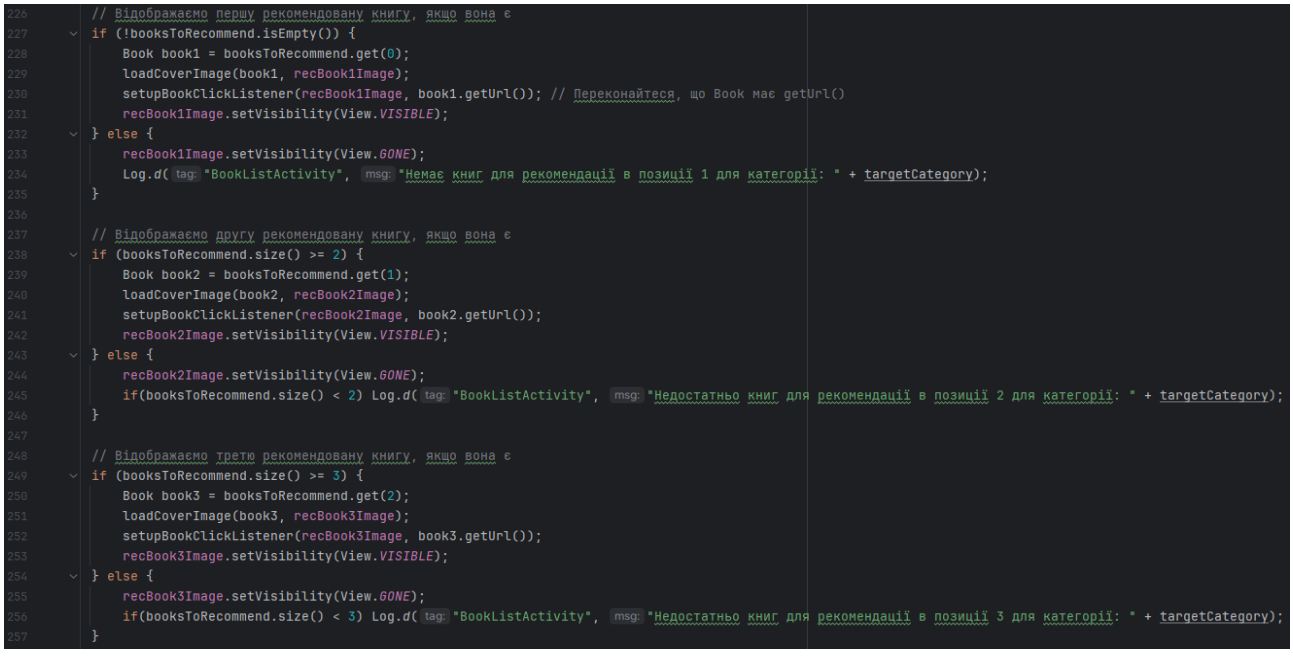
```
// Фільтруємо книги з JSON за цільовою категорією
List<Book> booksToRecommend = new ArrayList<>();
for (Book book : allBooksFromJson) {
    Log.d( tag: "BookFilter",  msg: "Перевірка книги: " + book.getTitle() + ", Категорія: " + book.getCategory());
    // Перевіряємо, чи категорія книги збігається з цільовою (ігноруючи регістр)
    if (book.getCategory() != null && book.getCategory().equalsIgnoreCase(targetCategory)) {
        Log.d( tag: "BookFilter",  msg: "Категорія збігається: " + book.getTitle());
        // Перевіряємо, чи є остання прочитана книга
        if (lastReadBookObject == null || book.getId() == null || lastReadBookObject.getId() == null || !book.getId().equals(lastReadBookObject.getId())) {
            Log.d( tag: "BookFilter",  msg: "Додано до рекомендацій: " + book.getTitle());
            booksToRecommend.add(book);
        } else {
            Log.d( tag: "BookFilter",  msg: "Книга " + book.getTitle() + " є останньою прочитаною (ID: ["+book.getId()+"]), пропускаємо.");
        }
    }
}
```

Рисунок 3.5 – Перевірка чи категорія книги з JSON збігається з цільовою



```
// Якщо немає прочитаних книг, то за замовчанням рекомендуємо книги з категорії Пригоди
if (booksToRecommend.isEmpty() && lastReadBookObject != null && !targetCategory.equals("Пригоди")) {
    Log.d( tag: "BookListActivity",  msg: "Не знайдено книг категорії '" + targetCategory + "'. Спроба завантажити 'Пригоди'.");
    targetCategory = "Пригоди";
    for (Book book : allBooksFromJson) {
        if (book.getCategory() != null && book.getCategory().equalsIgnoreCase(targetCategory)) {
            if (lastReadBookObject == null || !book.getId().equals(lastReadBookObject.getId())) {
                booksToRecommend.add(book);
            }
        }
    }
}
```

Рисунок 3.6 – Рекомендація книг за замовчуванням



```
// Відображаємо першу рекомендовану книгу, якщо вона є
if (!booksToRecommend.isEmpty()) {
    Book book1 = booksToRecommend.get(0);
    loadCoverImage(book1, recBook1Image);
    setupBookClickListener(recBook1Image, book1.getUrl()); // Переконайтеся, що Book має getUrl()
    recBook1Image.setVisibility(View.VISIBLE);
} else {
    recBook1Image.setVisibility(View.GONE);
    Log.d( tag: "BookListActivity",  msg: "Немає книг для рекомендації в позиції 1 для категорії: " + targetCategory);
}

// Відображаємо другу рекомендовану книгу, якщо вона є
if (booksToRecommend.size() >= 2) {
    Book book2 = booksToRecommend.get(1);
    loadCoverImage(book2, recBook2Image);
    setupBookClickListener(recBook2Image, book2.getUrl());
    recBook2Image.setVisibility(View.VISIBLE);
} else {
    recBook2Image.setVisibility(View.GONE);
    if(booksToRecommend.size() < 2) Log.d( tag: "BookListActivity",  msg: "Недостатньо книг для рекомендації в позиції 2 для категорії: " + targetCategory);
}

// Відображаємо третю рекомендовану книгу, якщо вона є
if (booksToRecommend.size() >= 3) {
    Book book3 = booksToRecommend.get(2);
    loadCoverImage(book3, recBook3Image);
    setupBookClickListener(recBook3Image, book3.getUrl());
    recBook3Image.setVisibility(View.VISIBLE);
} else {
    recBook3Image.setVisibility(View.GONE);
    if(booksToRecommend.size() < 3) Log.d( tag: "BookListActivity",  msg: "Недостатньо книг для рекомендації в позиції 3 для категорії: " + targetCategory);
}
```

Рисунок 3.7 – Виведення рекомендованих книг

Для зберігання даних вирішено обрати локальне сховище даних – SharedPreferences. Це зроблено для того, щоб спростити мобільний застосунок, а

також надати можливість використання офлайн, що є важливим в умовах російської агресії проти України.

3.2 Інтерфейс користувача

Для створення інтерфейсу користувача було використано середовище Figma. Після аналізу існуючих аналогів було створено шаблон мобільного застосунку аналітики для рекомендаційної системи (рисунок 3.8), але через його важку структуру було вирішено спростити дизайн для кращої реалізації застосунку (рисунок 3.9).

Рисунок 3.8 – Шаблон мобільного застосунку аналітики для рекомендаційної системи читання книг

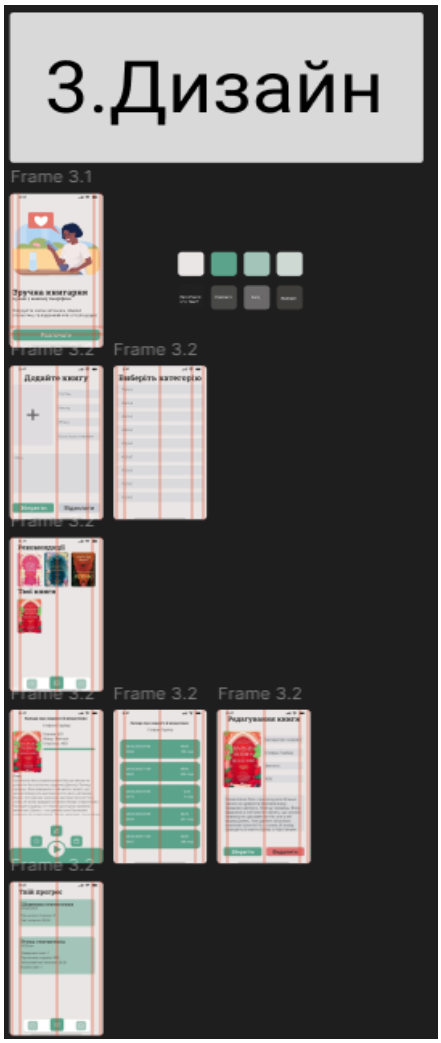


Рисунок 3.9 – Спрощений дизайн мобільного застосунку для рекомендаційної системи читання книг

Інтерфейс користувача мобільного застосунку аналітики для рекомендаційної системи є простим та не створює перешкод для дій користувача.

Діаграма станів – це діаграма, яка показує можливість переходу з одного стану в інший, і причини переходу [33].

Для мобільного застосунку аналітики для рекомендаційної системи читання книг створено діаграму станів (рисунок 3.10) на основі графічного інтерфейсу користувача.

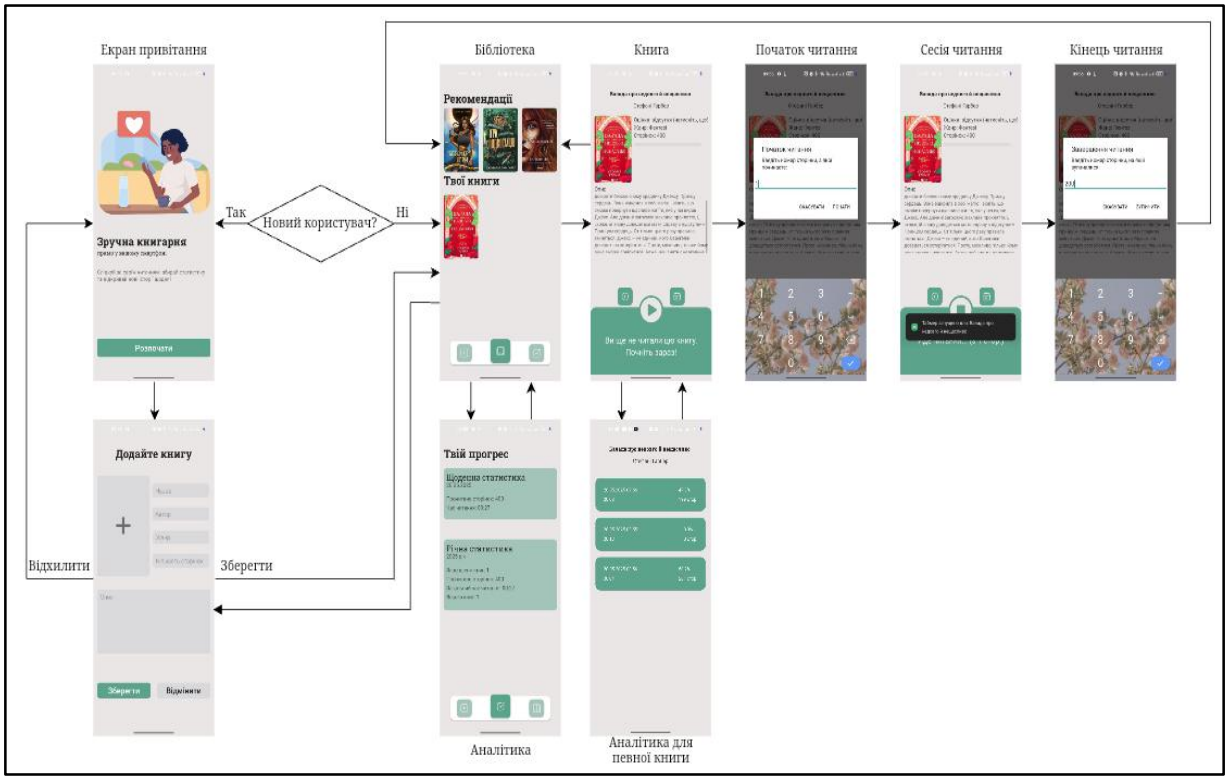


Рисунок 3.10 – Діаграма станів мобільного застосунку аналітики для рекомендаційної системи

3.3 Тестування мобільного застосунку аналітики для рекомендаційної системи читання книг

Мінімальні вимоги для зручного користування і правильної роботи мобільного застосунку аналітики для рекомендаційної системи читання книг є версія Android 7.0 і вище.

Після запуску програми з'являється вікно привітання (рисунок 3.11).

Після натискання кнопки “Розпочати” виконання перемикається у вікно (екран) додавання книги (рисунок 3.12).

На цій сторінці у користувача є різні варіанти:

– додати інформацію;

– зберегти або відхилити дію.

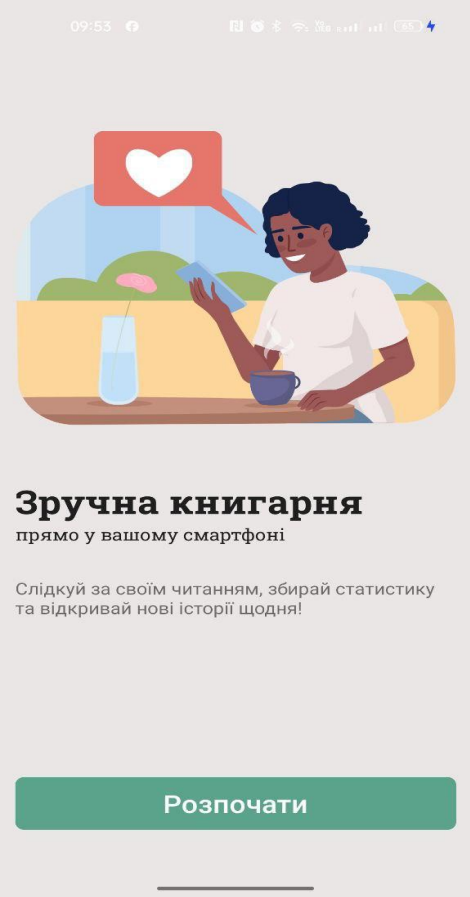


Рисунок 3.11 – Екран привітання

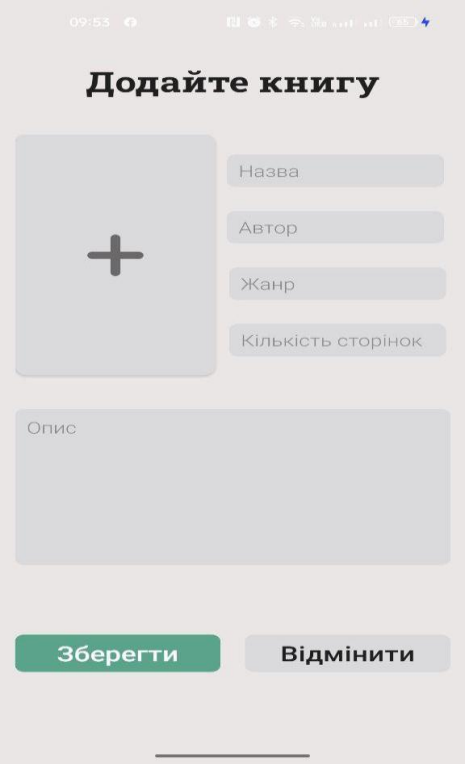


Рисунок 3.12 – Екран додавання книги

Якщо було обрано варіант «відхилити» – відбувається перехід на екран привітання (див. рисунок 3.11). Якщо варіант (рисунок 3.13) « зберегти» – перехід у бібліотеку користувача (рисунок 3.14).

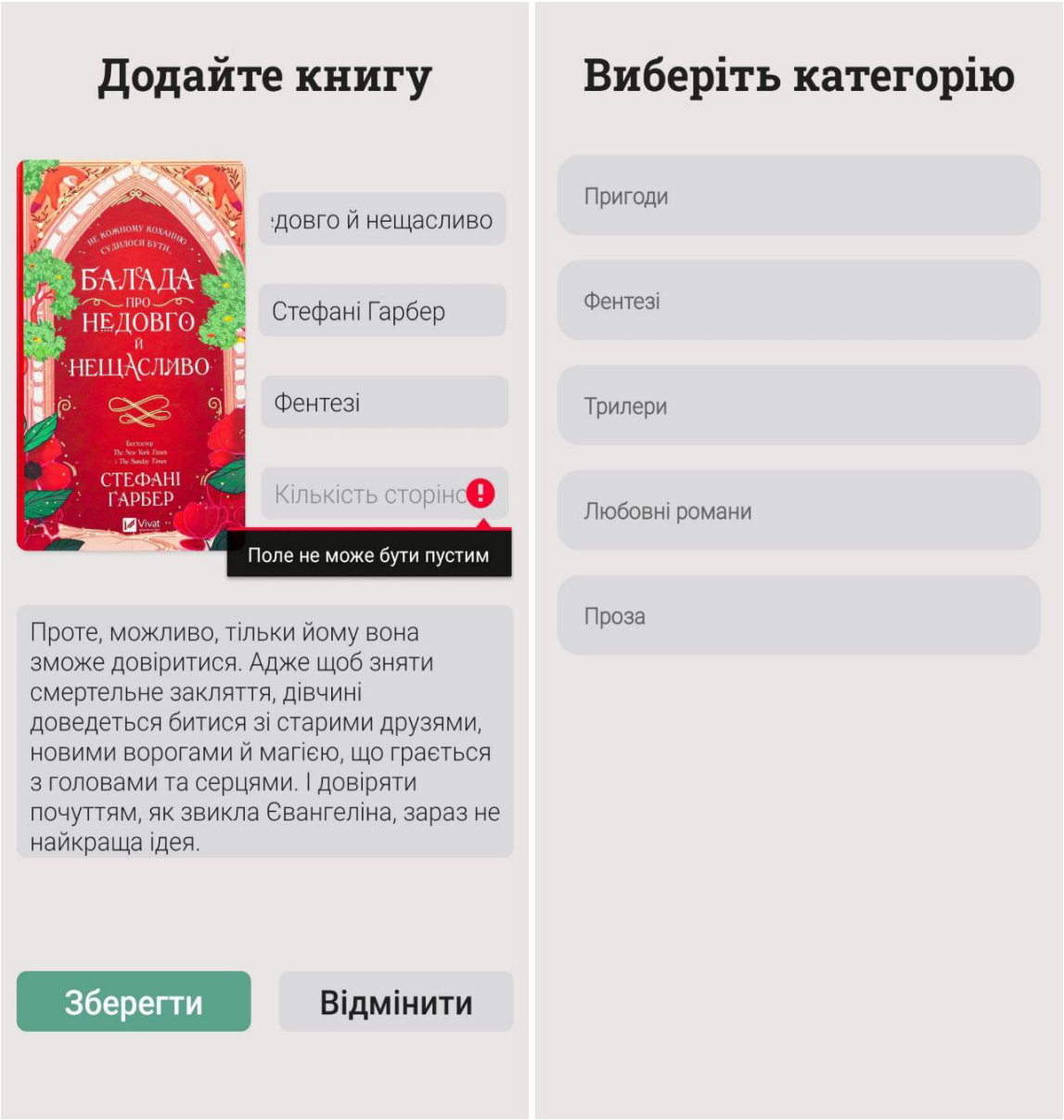


Рисунок 3.13 – Екран додавання книги (заповнення)

Важливо відмітити, що якщо не ввести всю потрібну інформацію, буде з'являтися повідомлення про необхідність її заповнення.

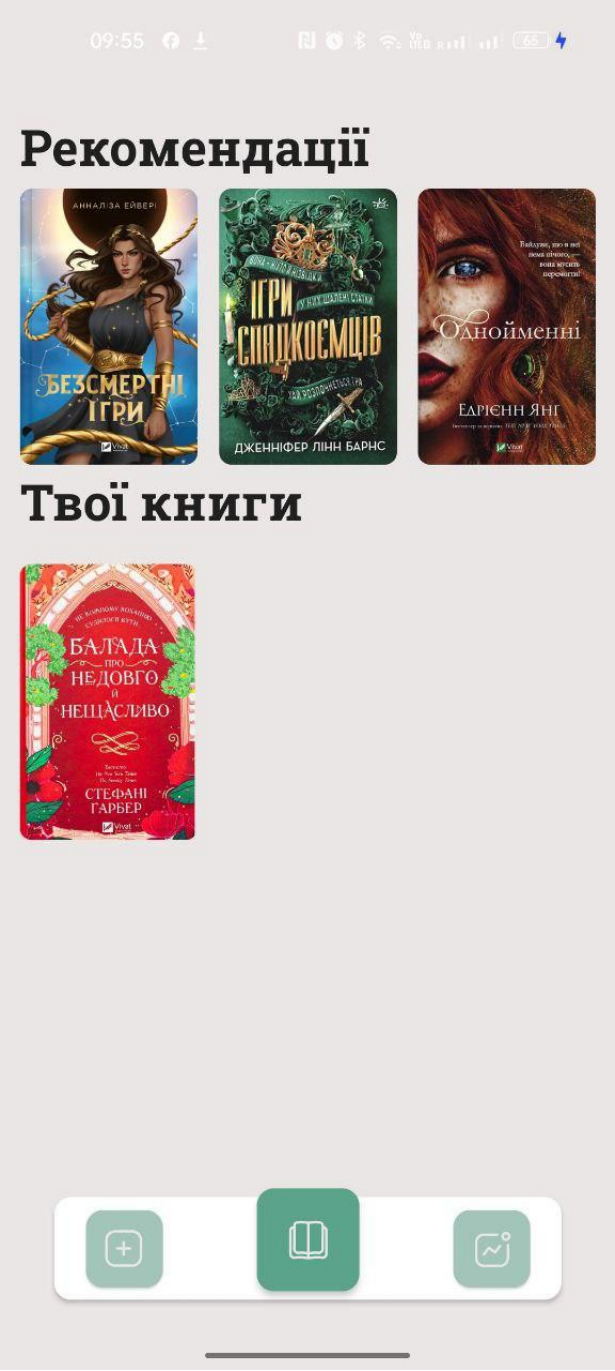


Рисунок 3.14 – Екран бібліотеки користувача

На цьому екрані можна побачити три кнопки, які знаходяться внизу:

- перша відкриває екран додавання книги (див. рисунок 3.12);
- друга – екран бібліотеки (див. рисунок 3.14);
- третя – екран аналітики (рисунок 3.15).

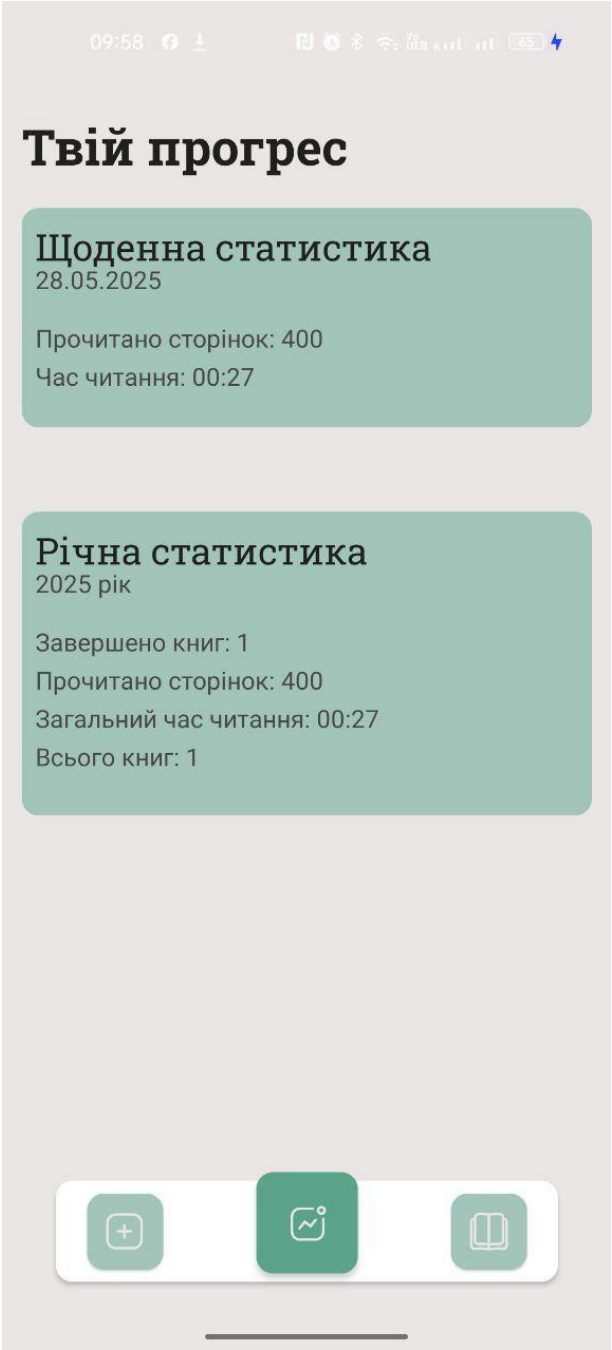


Рисунок 3.15 – Екран аналітики

Для того, щоб почати відстежувати свій час читання, потрібно на екрані бібліотеки (див. рисунок 3.14) натиснути на книгу зі списку «Твої книги». Відкривається екран для вибраної книги (рисунок 3.16).

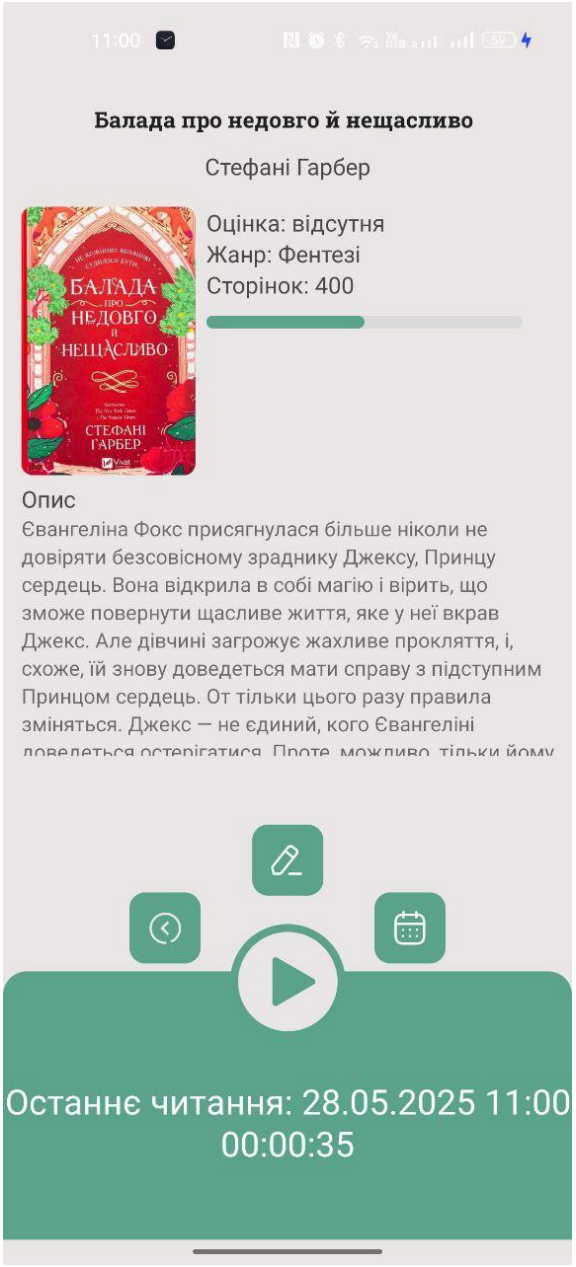


Рисунок 3.16 – Екран відстежування часу читання

На цьому екрані знаходиться додана інформація про книгу, аналітика книги і таймер читання.

Також над кнопкою, яка розпочинає відлік часу є інша кнопка, яка виконує пермикання на екран, за допомогою якого можна редагувати інформацію книги або видалити її (рисунок 3.17).

Рисунок 3.17– Екран редагування та видалення

Для перегляду аналітики для книги потрібно перейти за кнопкою, розташованою з правого боку від кнопки таймеру на екрані, зображеному на рисунку 3.16.

Після того користувачу відкривається екран аналітики (рисунок 3.18), на якому можна переглянути аналітичну інформацію стосовно цієї книги.

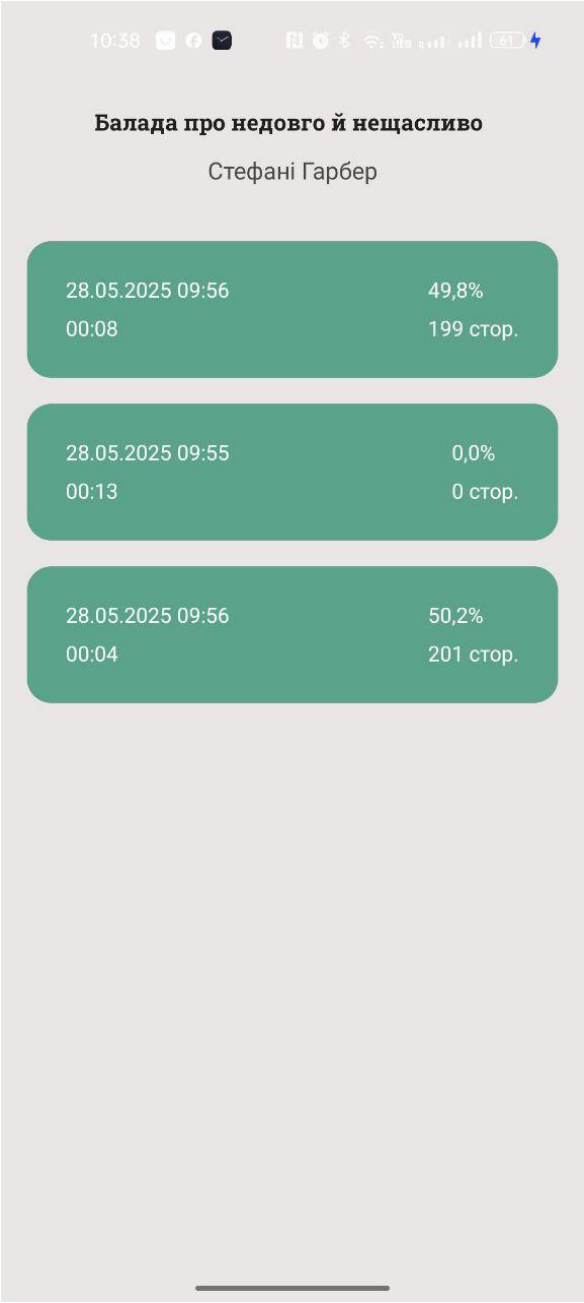


Рисунок 3.18 – Екран аналітики для книги

Щоб розпочати читання потрібно натиснути на середню кнопку на екрані книги (див. рисунок 3.16).

Після того користувач повинен ввести сторінку, з якої починає читати, а після закінчення сесії читання – ввести останню прочитану сторінку (рисунок 3.19)

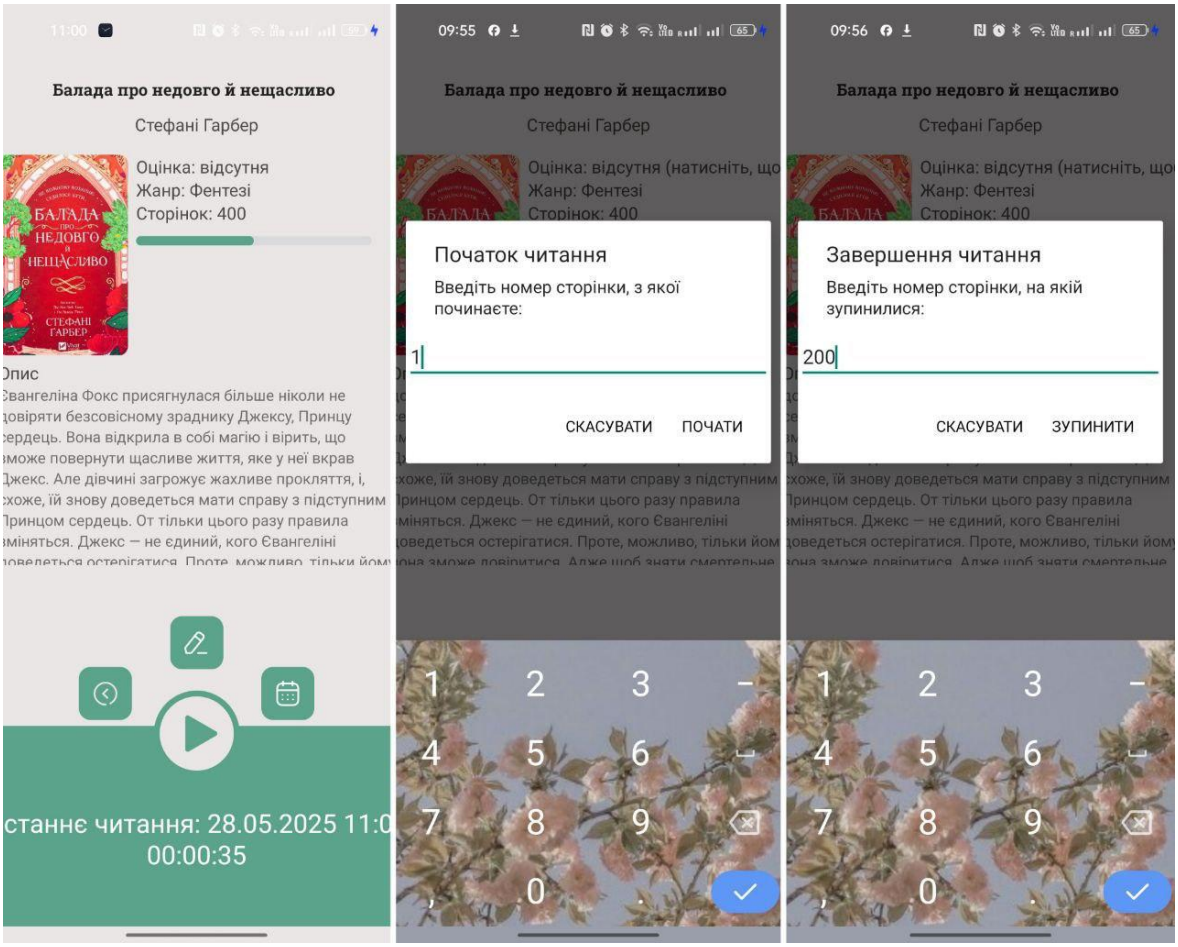


Рисунок 3.19 – Відстеження часу читання

Після закінчення сесії читання відображається актуальна шкала прогресу. Також, при завершенні читання книги, можна виставити оцінку, яка знаходиться зверху під назвою книги та автором. Щоб її змінити, потрібно натиснути на неї (рисунок 3.20).

Після оцінювання книги на екрані бібліотеки (див. рисунок 3.14) у блоці рекомендації оновлюється список з трьох книг по жанру прочитаної та оціненої книги (рисунок 3.21).

Якщо натиснути на книгу з рекомендацій, то для користувача відбувається переведення на сайт, де є можливість придбати обрану книгу.

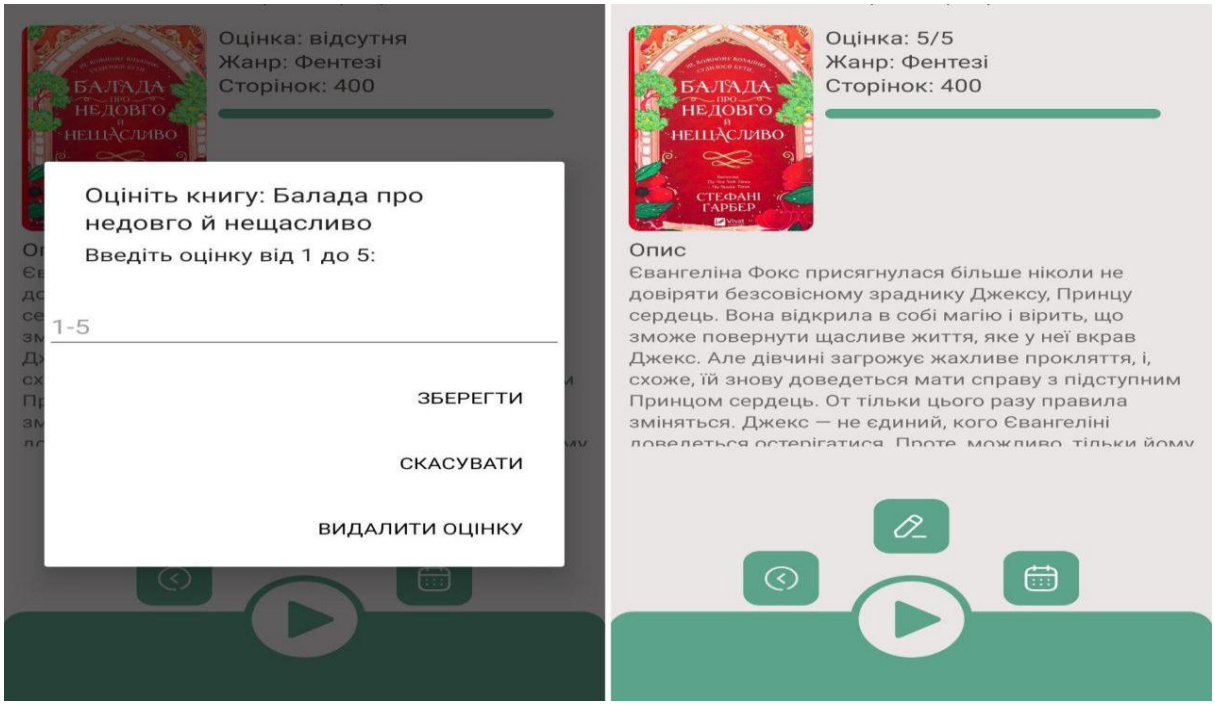


Рисунок 3.20 – Екран оцінки

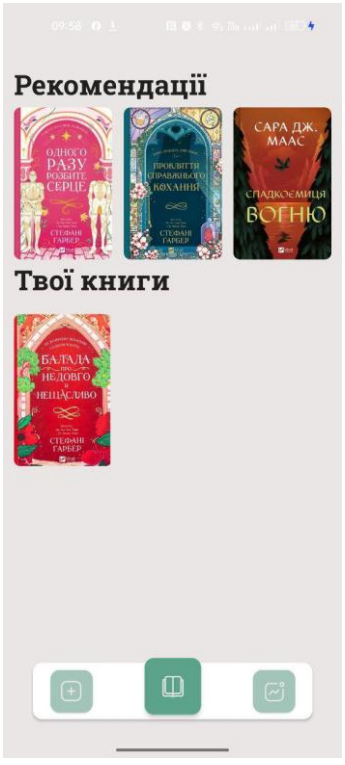


Рисунок 3.21 – Оновлення рекомендацій

ВИСНОВКИ

Результатом виконання кваліфікаційної роботи є створення мобільного застосунку, який забезпечує зручне середовище для збору та аналізу статистичних даних прогресу читання і формування на основі цих даних персональних рекомендацій. Досягнення мети кваліфікаційної роботи уможливлює отримання наступних висновків:

1. Виконано аналіз існуючих рішень мобільних застосунків у предметній області: goodreads, StoreGraph, Basmo та Rork визначено їх переваги та недоліки.

2. На основі аналізу існуючих рішень сформовано концепцію розроблення мобільного застосунку аналітики для рекомендаційної системи читання книг.

3. Розроблено архітектуру мобільного застосунку на основі таблиць та діаграм, а саме: перелік функцій та їх характеристики; інформаційні зв’язки між елементами системи; діаграма дерева функцій; діаграма варіантів використання; діаграма діяльності; блок-схема логіки; компонентна діаграма; діаграма розгортання та архітектурна схема, щоб реалізувати механізми збору, попередньої обробки та аналізу кількісних даних читання.

4. Програмно реалізовано мобільний застосунок та інтерфейс користувача, для дизайну якого використовувалось середовище Figma, середовище програмної розробки Android Studio і мова програмування – Java, для збереження даних було використано локальне сховище SharedPreferences.

5. Виконано тестування застосунку, зокрема роботу функцій інтерфейсу користувача, послідовність процедури тестування вображено відповідними знімками екрану роботи застосунку.

Програмна реалізація мобільного застосунку аналітики для рекомендаційної системи читання книг і тестування її робити, використання у проєктування інтерфейсу користувача розширюваної мови розмітки XML свідчать про практичну цінність отриманих результатів, і є підґрунтям для подальшого вдосконалення розроблено у результаті виконання кваліфікаційної роботи продукту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Трекер читання «Люблю читати» / Видавництво «Всеосвіта» [Електронний ресурс]. URL: https://vseosvita.ua/library/treker-chytannia-liubliu-chytaty-782932.html

2. Sun X., Song Y., Zhu J. Reading behavior analysis based on mobile applications: Design and evaluation . *Computers in Human Behavior*. 2019. Vol. 92. P. 428–437.

3. Chen Y., Huang T. A mobile reading tracking system based on user behavior analysis // International Journal of Human–Computer Interaction. 2020. Vol. 36, No. 4. P. 345–356.

4. Книги в Україні: як змінилася поведінка читачів під час повномасштабного вторгнення. *Forbes Україна* [Електронний ресурс].: https://forbes.ua/news/yak-chitayut-ukraintsi...

5. A systematic review and research perspective on recommender systems // *Journal of Big Data* [Електронний ресурс]. URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00592-5

6. Найкращі інструменти для розробки додатків для Android [Електронний ресурс]. URL: https://uk.androidsis.com/найкращі-інструменти-для-розробників-додатків-для-android/

7. XML introduction // Mozilla Developer Network [Електронний ресурс]. – URL : https://developer.mozilla.org/enUS/docs/Web/XML/Guides/XML_introduction

8. Function Tree Models – A Tool for Optimization // Martin Parrot [Електронний ресурс]. URL: https://martin-parrot.com/en/function-tree-models-a-tool-for-optimization/

9. An introduction to database systems [Електронний ресурс]. URL: https://lc.fie.umich.mx/~rodrigo/BD/An%20Introduction%20to%20Database%20Systems%208e%20By%20C%20J%20Date.pdf

10. Що таке аналіз конкурентів і як його проводити? // IM4UDMA [Електронний ресурс]. URL: https://im4udma.com.ua/blog/...

11. ISBN (ІСБН код). Що це і навіщо він потрібний? // ISBN Україна [Електронний ресурс]. URL: https://isbn.com.ua/isbn-code-book/

12. Основні принципи візуального дизайну в UX // Blog.ithillel.ua [Електронний ресурс]. URL: https://blog.ithillel.ua/articles/basic-principles-of-visual-design-in-ux

13. Li X., et al. Nellodee 2.0: A Quantified Self Reading App for Tracking Reading Goals. *Proceedings of the 2017 International Conference on Human-Computer Interaction*. Springer, 2017. P. 435–443.

14. Huang Y., et al. A Mobile Reading Tracking System Based on User Behavior Analysis. *IJHCI.* 2020. Vol. 36(4). P. 345–356.

15. Lee H., et al. A Systematic Survey on Android API Usage for Data-Driven Analytics with Smartphones. *arXiv* [Електронний ресурс]. URL: https://arxiv.org/abs/2104.11271

16. Srivastava N., et al. Hybrid Recommender for BookWise Using NLP and User Feedback // GitHub [Електронний ресурс]. URL: https://github.com/AJDazzle/BookWise-Book-Recommendation-App

17. Karunaratne C. One Read — A UX Case Study of a Reading Tracker App // Medium [Електронний ресурс]. URL: https://medium.com/...

18. Zhang H., et al. Adaptive Content Recommendation in Mobile Learning Applications. *ACM DL* [Електронний ресурс]. URL: https://dl.acm.org/doi/10.1145/3485447.3512256

19. Harty J., et al. Logging Practices with Mobile Analytics: An Empirical Study on Firebase // arXiv [Електронний ресурс]. URL: https://arxiv.org/abs/2104.11271

20. Gonzalez R., et al. Cloud-Based Architecture for Scalable Reading Apps . *IEEE Access*. 2021. Vol. 9. P. 12345–12356.

21. Johnson M., Shilton K. Privacy by Design in Mobile Applications: User-Centered Approaches. *Journal of the ACM*. 2018. Vol. 65(4). P. 1–25.

22. Використання підпрограм та функцій користувача [Електронний ресурс]. URL: https://project.zu.edu.ua/ReadData.php?h_id=41

23. Bennett S., McRobb S., Farmer R. Object-Oriented Systems Analysis and Design Using UML. McGraw-Hill Education, 2010. 688 p.

24. Проектування програмних засобів систем управління [Електронний ресурс]. URL: https://web.posibnyky.vntu.edu.ua/...

25. Застосування блок-схем у розробці програм // Foxminded [Електронний ресурс]. URL: https://foxminded.ua/blok-skhema/

26. Що таке діаграма компонентів в UML? // Guru99 [Електронний ресурс]. URL: https://www.guru99.com/uk/component-diagram-uml-example.html

27. Основні типи архітектури програмного забезпечення // Art of BA [Електронний ресурс]. URL: https://www.artofba.com/uk/post/main-types-of-software-architecture

28. Діаграми потоків даних (Data Flow Diagrams) // Max Zosim [Електронний ресурс]. URL: https://www.maxzosim.com/data-flow-diagrams/

29. Zapisuj proste dane dzięki usłudze SharedPreferences // Android Developers [Електронний ресурс]. URL: https://developer.android.com/training/data-storage/shared-preferences?hl=l

30. Що таке ER Modeling? Навчайтеся на прикладі // Guru99 [Електронний ресурс]. URL: https://www.guru99.com/uk/er-modeling.html

31. Проектування програмного забезпечення засобами UML [Електронний ресурс]. URL: http://mmsa.kpi.ua/sites/default/files/...

32. Android Studio: переваги та особливості [Електронний ресурс]. URL: https://qagroup.com.ua/publications/android-studio-perevagy-ta-osoblyvosti/

33. Діаграма станів \[Електронний ресурс]. URL: https://emtd.ztu.edu.ua/view/word-3021/0

34. Комар М. П., Саченко А. О., Васильків Н. М., Гладій Г. М., Коваль В. С., Ліп’яніна-Гончаренко Х. В. Методичні рекомендації до виконання кваліфікаційної роботи з освітньо-професійної програми «Комп’ютерні науки» спеціальності 122 «Комп’ютерні науки» за першим (бакалаврським) рівнем вищої освіти. Тернопіль: ЗУНУ, 2024. 52 с.

35. Загальні методичні рекомендації з підготовки, оформлення, захисту та оцінювання кваліфікаційних робіт здобувачів вищої освіти першого (бакалаврського) і другого (магістерського) рівнів. Тернопіль: ЗУНУ, 2024. 83 с.

36. Зборовська А. Р. Розробка мобільного застосунку аналітики для рекомендаційної системи читання книг. *Інтелектуальні інформаційні технології в прикладних дослідженнях (IITAR-2025)*: збірник тез доповідей студентської науково-практичної конференції (Тернопіль, 27–29 травня 2025 р.). Тернопіль: ЗУНУ, 2025. С. 152-154.

Додаток А

Програмний код для мобільного застосунку аналітики для рекомендаційної системи читання книг

class Book:

```
package com.example.bookaplication;

import android.os.Parcel;
import android.os.Parcelable;

import androidx.annotation.NonNull;

public class Book implements Parcelable{
    private String id;
    private String title;
    private String author;
    private String description;
    private String category;
    private String pages;
    private String endPage;
    private String imagePath;
    private String finished;
    private String stars;
    private String url;

    public Book(String id, String title, String author, String description, String category, String pages, String imagePath, String endPage, String stars, String finished, String url) {
        this.id = id;
        this.title = title;
        this.author = author;
        this.description = description;
        this.category = category;
        this.pages = pages;
        this.endPage = endPage;
        this.finished = finished;
        this.imagePath = imagePath;
        this.stars = stars;
        this.url = url;
    }

    protected Book(Parcel in) {
        id = in.readString();
        title = in.readString();
        author = in.readString();
        description = in.readString();
        category = in.readString();
        pages = in.readString();
        endPage = in.readString();
        imagePath = in.readString();
        finished = in.readString();
        stars = in.readString();
        url = in.readString();
    }

    public static final Creator<Book> CREATOR = new Creator<Book>() {
        @Override
        public Book createFromParcel(Parcel in) {
            return new Book(in);
        }

        @Override
        public Book[] newArray(int size) {
            return new Book[size];
        }
    };

    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getFinished() {
        return finished;
```

```
}

public void setFinished(String finished) {
    this.finished = finished;
}

public String getTitle() {
    return title;
}

public void setStars(String id) {
    this.stars = stars;
}

public String getStars() {
    return stars;
}
public String getEndPage() {
    return endPage;
}
public void setEndPage(String endPage) {
    this.endPage = endPage;
}

public void setTitle(String title) {
    this.title = title;
}

public String getAuthor() {
    return author;
}

public void setAuthor(String author) {
    this.author = author;
}

public String getDescription() {
    return description;
}

public void setDescription(String description) {
    this.description = description;
}

public String getCategory() {
    return category;
}

public void setCategory(String category) {
    this.category = category;
}

public String getPages() {
    return pages;
}

public void setPages(String pages) {
    this.pages = pages;
}

public String getImagePath() {
    return imagePath;
}
public String getUrl() { return url; }
public void setUrl(String url) { this.url = url; }

public void setImagePath(String imagePath) {
    this.imagePath = imagePath;
}

@Override
public int describeContents() {
    return 0;
}

@Override
public void writeToParcel(Parcel dest, int flags) {
    dest.writeString(id);
    dest.writeString(title);
    dest.writeString(author);
    dest.writeString(description);
```

```
        dest.writeString(category);
        dest.writeString(pages);
        dest.writeString(endPage);
        dest.writeString(imagePath);
        dest.writeString(stars);
        dest.writeString(finished);
        dest.writeString(url);
    }
}

class BookAdapter:

package com.example.bookaplication;

import android.content.Context;
import android.net.Uri;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
import android.widget.Toast;

import androidx.annotation.NonNull;
import androidx.recyclerview.widget.RecyclerView;

import com.bumptech.glide.Glide;

import java.io.File;
import java.util.ArrayList;
import java.util.List;

public class BookAdapter extends RecyclerView.Adapter<BookAdapter.BookViewHolder> {

    private Context context;
    private List<Book> bookList;

    public interface OnItemClickListener {
        void onItemClick(Book book);
    }
    private OnItemClickListener listener;

    public void setOnItemClickListener(OnItemClickListener listener) {
        this.listener = listener;
    }

    public BookAdapter(Context context, List<Book> bookList) {
        this.context = context;
        this.bookList = bookList != null ? bookList : new ArrayList<>();
    }

    @NonNull
    @Override
    public BookViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        Log.d("BookAdapter", "onCreateViewHolder викликано");
        View view = LayoutInflater.from(context).inflate(R.layout.list_item_book_block, parent, false);
        return new BookViewHolder(view);
    }

    @Override
    public void onBindViewHolder(@NonNull BookViewHolder holder, int position) {
        Log.d("BookAdapter", "onBindViewHolder викликано для позиції: " + position);
        Book book = bookList.get(position);
        holder.bind(book);
        holder.itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (listener != null) {
                    listener.onItemClick(book);
                }
            }
        });
    }

    @Override
    public int getItemCount() {
        Log.d("BookAdapter", "getItemCount: " + bookList.size());
        return bookList.size();
    }

    public void updateBooks(List<Book> newBooks) {
```

```
        Log.d("BookAdapter_Update", "---------- updateBooks ПОЧАТОК ----------");
        if (this.bookList == null) {
            Log.e("BookAdapter_Update", "ПОМИЛКА: this.bookList є NULL! Ініціалізуємо новим ArrayList.");
            this.bookList = new ArrayList<>();
        }

        this.bookList.clear();

        Log.d("BookAdapter_Update", "Розмір this.bookList ПІСЛЯ clear(): " + this.bookList.size());
        if (newBooks != null) {
            Log.d("BookAdapter_Update", "Отримано newBooks з розміром: " + newBooks.size());
            this.bookList.addAll(newBooks);
            Log.d("BookAdapter_Update", "Розмір this.bookList ПІСЛЯ addAll(): " + this.bookList.size());
        } else {
            Log.d("BookAdapter_Update", "Отримано newBooks == null. Внутрішній список залишається порожнім після clear().");
        }
        Log.d("BookAdapter", "updateBooks: новий актуальний розмір списку: " + this.bookList.size());
        Log.d("BookAdapter_Update", "---------- updateBooks КІНЕЦЬ ----------");
    }

    static class BookViewHolder extends RecyclerView.ViewHolder {
        ImageView coverImageView;

        public BookViewHolder(@NonNull View itemView) {
            super(itemView);
            coverImageView = itemView.findViewById(R.id.bookCoverImageView);
        }

        public void bind(final Book book) {

            String imagePath = book.getImagePath();
            if (imagePath != null && !imagePath.isEmpty()) {
                File imageFile = new File(imagePath);
                if (imageFile.exists()) {
                    Log.d("BookAdapter", "Завантаження зображення: " + imagePath);
                    Glide.with(itemView.getContext()).load(imageFile).into(coverImageView);
                }
            }
        }
    }
}

class BookAddActivity:

package com.example.bookaplication;

import android.app.Activity;
import android.app.ActivityOptions;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.Window;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.Toast;

import androidx.activity.result.ActivityResult;
import androidx.activity.result.ActivityResultCallback;
import androidx.activity.result.ActivityResultLauncher;
import androidx.activity.result.contract.ActivityResultContracts;
import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;
import androidx.core.view.WindowInsetsControllerCompat;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.UUID;

public class BookAddActivity extends AppCompatActivity {

    private static final String TAG = "BookAddActivity";
    private EditText editTextTitle;
```

```
private EditText editTextAuthor;
private EditText editTextDescription;
private EditText editTextPages;
private EditText editTextCategory;
private ConstraintLayout bookLogoLayout;
private ImageView bookCoverImageView;
private Uri currentSelectedImageUri = null;
private Button saveButton;
private Button cancelButton;
private ActivityResultLauncher<Intent> categorySelectionLauncher;
private ActivityResultLauncher<Intent> imageChooserLauncher;
private BookRepository bookRepository;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable(this);
    setContentView(R.layout.book_add);

    bookRepository = new BookRepository(getApplicationContext());

    editTextTitle = findViewById(R.id.bookname);
    editTextAuthor = findViewById(R.id.author);
    editTextDescription = findViewById(R.id.description);
    editTextPages = findViewById(R.id.pagescount);
    editTextCategory = findViewById(R.id.categoryselector);

    bookLogoLayout = findViewById(R.id.bookLogoLayout);
    bookCoverImageView = findViewById(R.id.bookCoverImageView);

    cancelButton = findViewById(R.id.cancelbookbutton);
    saveButton = findViewById(R.id.savebookbutton);

    categorySelectionLauncher = registerForActivityResult(
            new ActivityResultContracts.StartActivityForResult(),
            new ActivityResultCallback<ActivityResult>() {
                @Override
                public void onActivityResult(ActivityResult result) {
                    if (result.getResultCode() == Activity.RESULT_OK) {
                        Intent data = result.getData();
                        if (data != null && data.hasExtra(CategorySelectionActivity.EXTRA_SELECTED_CATEGORY)) {
                            String selectedCategory = data.getStringExtra(CategorySelectionActivity.EXTRA_SELECTED_CATEGORY);
                            editTextCategory.setText(selectedCategory);
                            Log.d(TAG, "Selected category: " + selectedCategory);
                        }
                    } else if (result.getResultCode() == Activity.RESULT_CANCELED) {
                        Log.d(TAG, "Category selection was canceled.");
                    }
                }
            });

    saveButton.setOnClickListener(v -> {
        saveBookData();
    });

    cancelButton.setOnClickListener(v -> startActivity(new Intent(BookAddActivity.this, WelcomeActivity.class),
            ActivityOptions.makeSceneTransitionAnimation(this).toBundle()));

    bookLogoLayout.setOnClickListener(v -> imageChooser());

    editTextCategory.setOnClickListener(v -> {
        Intent intent = new Intent(BookAddActivity.this, CategorySelectionActivity.class);
        ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(BookAddActivity.this);
        categorySelectionLauncher.launch(intent);
        editTextCategory.setError(null);
    });

    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.book_add), (v, insets) -> {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });

    imageChooserLauncher = registerForActivityResult(
            new ActivityResultContracts.StartActivityForResult(),
            result -> {
                if (result.getResultCode() == RESULT_OK && result.getData() != null) {
                    Uri selectedImageUri = result.getData().getData();
                    if (selectedImageUri != null) {
                        bookCoverImageView.setImageURI(selectedImageUri);
                        this.currentSelectedImageUri = selectedImageUri;
```

```
                }
            }
        });
}

private void saveBookData(){

    String title = editTextTitle.getText().toString().trim();
    String author = editTextAuthor.getText().toString().trim();
    String description = editTextDescription.getText().toString().trim();
    String category = editTextCategory.getText().toString().trim();
    String pagesStr = editTextPages.getText().toString().trim();
    String stars = "0";


    String imagePath = null;

    if (currentSelectedImageUri != null) {
        imagePath = saveImageToInternalStorage(currentSelectedImageUri, title);
        if (imagePath == null) {
            Toast.makeText(this, "Помилка збереження обкладинки", Toast.LENGTH_SHORT).show();
            return;
        }
    }

    if (isFieldEmpty(editTextTitle, title, "Назва не може бути пустою") ||
        isFieldEmpty(editTextAuthor, author, "Автор не може бути пустим") ||
        isFieldEmpty(editTextCategory, category, "Категорія не може бути пустою") ||
        isFieldEmpty(editTextPages, pagesStr, "Поле не може бути пустим") ||
        isFieldEmpty(editTextDescription, description, "Опис не може бути пустим") ||
        currentSelectedImageUri == null) {
        if (currentSelectedImageUri == null) {
            Toast.makeText(this, "Будь ласка, виберіть обкладинку для книги", Toast.LENGTH_LONG).show();
        }
        return;
    }

    String bookId = UUID.randomUUID().toString();
    Book newBook = new Book(bookId, title, author, description, category, pagesStr, imagePath, "0", stars, "false", "null");
    bookRepository.addBook(newBook);

    Log.d(TAG, "Saving book: ID=" + bookId + ", Title=" + title + ", ImagePath=" + imagePath);
    Toast.makeText(this, "Книгу '" + title + "' збережено!", Toast.LENGTH_LONG).show();

    Intent intent = new Intent(BookAddActivity.this, BookListActivity.class);
    ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(this);
    startActivity(intent, options.toBundle());
    finish();
}
private boolean isFieldEmpty(EditText editText, String text, String errorMessage) {
    if (text.isEmpty()) {
        editText.setError(errorMessage);
        editText.requestFocus();
        return true;
    }
    return false;
}

private String saveImageToInternalStorage(Uri uri, String bookTitle) {
    InputStream inputStream = null;
    OutputStream outputStream = null;

    String fileName = "cover_" + sanitizeFileName(bookTitle) + "_" + UUID.randomUUID().toString() + ".jpg";
    File directory = getDir("book_covers", MODE_PRIVATE);
    File imageFile = new File(directory, fileName);

    try {
        inputStream = getContentResolver().openInputStream(uri);
        if (inputStream == null) {
            Log.e(TAG, "Не вдалося відкрити InputStream для URI: " + uri);
            return null;
        }
        outputStream = new FileOutputStream(imageFile);
        byte[] buffer = new byte[1024];
        int length;
        while ((length = inputStream.read(buffer)) > 0) {
            outputStream.write(buffer, 0, length);
        }
        Log.d(TAG, "Зображення збережено у: " + imageFile.getAbsolutePath());
        return imageFile.getAbsolutePath();
    } catch (IOException e) {
```

```
            Log.e(TAG, "Помилка копіювання зображення у внутрішнє сховище", e);
            if (imageFile.exists()) {
                imageFile.delete();
            }
            return null;
        } finally {
            try {
                if (inputStream != null) inputStream.close();
                if (outputStream != null) outputStream.close();
            } catch (IOException e) {
                Log.e(TAG, "Помилка закриття потоків", e);
            }
        }
    }

    private String sanitizeFileName(String name) {
        if (name == null || name.isEmpty()) {
            return "untitled";
        }
        return name.replaceAll("[^a-zA-Z0-9.-]", "_").replaceAll("\\s+", "_");
    }

    void imageChooser(){
        Intent i = new Intent();
        i.setType("image/*");
        i.setAction(Intent.ACTION_GET_CONTENT);
        imageChooserLauncher.launch(Intent.createChooser(i, "Select Picture"));
    }

}

class BookEditActivity:

package com.example.bookaplication;

import android.app.ActivityOptions;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.net.Uri;
import android.os.Bundle;
import android.text.TextUtils;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ImageView;
import android.widget.Toast;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

import com.bumptech.glide.Glide;

import java.util.List;

public class BookEditActivity extends AppCompatActivity {

    private EditText editTextTitle;
    private EditText editTextAuthor;
    private EditText editTextDescription;
    private EditText editTextPages;
    private EditText editTextCategory;
    private ConstraintLayout bookLogoLayout;
    private ImageView bookCoverImageView;
    private Uri currentSelectedImageUri = null;
    private Button saveButton;
    private Button cancelButton;

    private SharedPreferences prefs;
    private static final String PREFS_NAME = "BookReadingPrefs";
    private BookRepository bookRepository;
    private Book bookToEdit;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
    EdgeToEdge.enable(this);
    setContentView(R.layout.activity_book_edit);

    editTextTitle = findViewById(R.id.bookname);
    editTextAuthor = findViewById(R.id.author);
    editTextDescription = findViewById(R.id.description);
    editTextPages = findViewById(R.id.pagescount);
    editTextCategory = findViewById(R.id.categoryselector);

    bookLogoLayout = findViewById(R.id.bookLogoLayout);
    bookCoverImageView = findViewById(R.id.bookCoverImageView);

    cancelButton = findViewById(R.id.cancelbookbutton);
    saveButton = findViewById(R.id.savebookbutton);

    bookRepository = new BookRepository(getApplicationContext());

    Intent intent = getIntent();
    if (intent != null && intent.hasExtra("book_to_edit")) {
        bookToEdit = intent.getParcelableExtra("book_to_edit");

        if (bookToEdit != null) {
            editTextTitle.setText(bookToEdit.getTitle());
            editTextAuthor.setText(bookToEdit.getAuthor());
            editTextDescription.setText(bookToEdit.getDescription());
            editTextPages.setText(bookToEdit.getPages());
            editTextCategory.setText(bookToEdit.getCategory());

            Glide.with(this)
                    .load(bookToEdit.getImagePath())
                    .into(bookCoverImageView);

            saveButton.setOnClickListener(v -> {
                saveBookChanges();
            });

            cancelButton.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {

                    bookRepository.removeBookById(bookToEdit.getId());

                    Intent intent = new Intent(BookEditActivity.this, BookListActivity.class);
                    ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(BookEditActivity.this);
                    startActivity(intent, options.toBundle());
                    finish();
                }
            });

        } else {
            Toast.makeText(this, "Помилка завантаження даних книги", Toast.LENGTH_SHORT).show();
            finish();
        }
    } else {
        Toast.makeText(this, "Немає даних для редагування", Toast.LENGTH_SHORT).show();
        finish();
    }

    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.book_edit), (v, insets) -> {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });
}

private void saveBookChanges() {
    String title = editTextTitle.getText().toString().trim();
    String author = editTextAuthor.getText().toString().trim();
    String description = editTextDescription.getText().toString().trim();
    String pagesStr = editTextPages.getText().toString().trim();
    String category = editTextCategory.getText().toString().trim();

    if (isFieldEmpty(editTextTitle, title, "Назва не може бути пустою") ||
            isFieldEmpty(editTextAuthor, author, "Автор не може бути пустим") ||
            isFieldEmpty(editTextCategory, category, "Категорія не може бути пустою") ||
            isFieldEmpty(editTextPages, pagesStr, "Поле не може бути пустим") ||
            isFieldEmpty(editTextDescription, description, "Опис не може бути пустим")) {
        return;
    }

    bookToEdit.setTitle(title);
```

```
        bookToEdit.setAuthor(author);
        bookToEdit.setDescription(description);
        bookToEdit.setCategory(category);
        bookToEdit.setPages(pagesStr);

        bookRepository.updateBook(bookToEdit);


        Intent intent = new Intent(BookEditActivity.this, BookListActivity.class);
        ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(BookEditActivity.this);
        startActivity(intent, options.toBundle());
        finish();

    }
    private boolean isFieldEmpty(EditText editText, String text, String errorMessage) {
        if (text.isEmpty()) {
            editText.setError(errorMessage);
            editText.requestFocus();
            return true;
        }
        return false;
    }
}

class BookListActivity:

package com.example.bookaplication;

import android.app.ActivityOptions;
import android.content.Intent;
import android.content.SharedPreferences;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.ImageView;
import android.widget.Toast;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;
import androidx.recyclerview.widget.GridLayoutManager;
import androidx.recyclerview.widget.RecyclerView;

import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.lang.reflect.Type;
import java.nio.charset.StandardCharsets;
import java.util.ArrayList;
import java.util.List;

import com.bumptech.glide.Glide;
import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;

public class BookListActivity extends AppCompatActivity {
    private List<Book> bookList = new ArrayList<>();
    private BookRepository bookRepository;
    private BookAdapter bookAdapter;
    private ConstraintLayout navbookadd;
    private ConstraintLayout navanalitic;
    private ImageView recBook1Image;
    private ImageView recBook2Image;
    private ImageView recBook3Image;

    private RecyclerView booksContainer;

    private String lastReadBookId;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        bookRepository = new BookRepository(getApplicationContext());
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_book_list);
        booksContainer = findViewById(R.id.booksRecyclerView);
        navbookadd = findViewById(R.id.nav_bookadd);
```

```
        booksContainer.setLayoutManager(new GridLayoutManager(this, 3));
        bookAdapter = new BookAdapter(this, bookList);
        navanalitic = findViewById(R.id.nav_analitic);
        booksContainer.setAdapter(bookAdapter);
        recBook1Image = findViewById(R.id.rec_book1_image);
        recBook2Image = findViewById(R.id.rec_book2_image);
        recBook3Image = findViewById(R.id.rec_book3_image);

        SharedPreferences prefs = getSharedPreferences(BookPageActivity.PREFS_NAME, MODE_PRIVATE);
        lastReadBookId = prefs.getString("last_read_book_id", null);

        navbookadd.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(BookListActivity.this, BookAddActivity.class);
                ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(BookListActivity.this);
                startActivity(intent, options.toBundle());
                finish();
            }
        });
        navanalitic.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(BookListActivity.this, TotalStatsActivity.class);
                ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(BookListActivity.this);
                startActivity(intent, options.toBundle());
                finish();
            }
        });
        bookAdapter.setOnItemClickListener(new BookAdapter.OnItemClickListener() {
            @Override
            public void onItemClick(Book book) {
                Intent intent = new Intent(BookListActivity.this, BookPageActivity.class);

                intent.putExtra("selected_book", book);
                Log.d("BookListActivity", "Книгу передано в BookPageActivity: " + book.getTitle());

                startActivity(intent);
            }
        });
        loadBooksAndDisplay();
        loadRecommendedBooks();
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });
    }

    private void loadBooksAndDisplay() {
        Log.d("BookListActivity", "loadBooksAndDisplay: Початок завантаження");
        List<Book> loadedBooks = bookRepository.getAllBooks();
        Log.d("BookListActivity", "loadBooksAndDisplay: З репозиторію завантажено: " + (loadedBooks != null ? loadedBooks.size() : "null") + "
книг.");
        bookList.clear();
        if (loadedBooks != null) {
            bookList.addAll(loadedBooks);
        }


        if (loadedBooks.isEmpty()) {
            Toast.makeText(this, "Список книг порожній", Toast.LENGTH_LONG).show();
        } else {
            Toast.makeText(this, "Завантажено книг: " + loadedBooks.size(), Toast.LENGTH_SHORT).show();
        }
        Log.d("BookListActivity", "loadBooksAndDisplay: Розмір currentBookList перед оновленням адаптера: " + bookList.size());

        bookAdapter.updateBooks(new ArrayList<>(bookList));
        Log.d("BookListActivity", "loadBooksAndDisplay: Метод bookAdapter.updateBooks викликано.");
    }

    private void loadRecommendedBooks() {
        Log.d("BookListActivity", "loadRecommendedBooks: Початок завантаження рекомендованих книг");
        Gson gson = new Gson();
        Type bookListType = new TypeToken<ArrayList<Book>>() {}.getType();
        List<Book> allBooksFromJson = new ArrayList<>();

        try {
            // Відкриваємо потік для читання файлу "recommended_books.json" з теки assets
            InputStream inputStream = getAssets().open("recommended_books.json");
            InputStreamReader reader = new InputStreamReader(inputStream, StandardCharsets.UTF_8);
```

```
            // Розпарсуємо JSON у список об'єктів Book
            allBooksFromJson = gson.fromJson(reader, bookListType);
            reader.close(); // Важливо закривати потоки
            inputStream.close();
        } catch (IOException e) {
            Log.e("BookListActivity", "Помилка завантаження recommended_books.json", e);
            Toast.makeText(this, "Не вдалося завантажити рекомендовані книги", Toast.LENGTH_SHORT).show();
            return;
        }

        String targetCategory = null;
        Book lastReadBookObject = null; // Об'єкт останньої прочитаної книги

        // Отримуємо ID останньої прочитаної книги зі SharedPreferences
        if (lastReadBookId != null) {
            // Шукаємо об'єкт останньої прочитаної книги серед завантажених книг користувача
            for (Book book : bookList) {
                if (lastReadBookId.equals(book.getId())) {
                    lastReadBookObject = book;
                    break;
                }
            }
        }

        // Визначаємо цільову категорію для рекомендацій
        if (lastReadBookObject != null && lastReadBookObject.getCategory() != null && !lastReadBookObject.getCategory().isEmpty()) {
            // Якщо є остання прочитана книга і її категорія відома, використовуємо її категорію
            targetCategory = lastReadBookObject.getCategory();
            Log.d("BookListActivity", "Цільова категорія для рекомендацій (з останньої прочитаної): " + targetCategory);
        } else {
            // Інакше, використовуємо категорію "Пригоди" за замовчуванням
            targetCategory = "Пригоди";
            Log.d("BookListActivity", "Остання прочитана книга/категорія не знайдена. Використовується категорія за замовчуванням: " +
targetCategory);
        }

        // Логування для відладки
        Log.d("BookFilter", "================================================");
        Log.d("BookFilter", "Цільова категорія (перед циклом): [" + targetCategory + "]");
        if (lastReadBookObject != null) {
            Log.d("BookFilter", "Остання прочитана книга (перед циклом): " + lastReadBookObject.getTitle() +
                    ", ID: [" + lastReadBookObject.getId() + "]" +
                    ", Категорія: [" + lastReadBookObject.getCategory() + "]");
        } else {
            Log.d("BookFilter", "Остання прочитана книга (перед циклом): null");
        }
        Log.d("BookFilter", "================================================");

        // Фільтруємо книги з JSON за цільовою категорією
        List<Book> booksToRecommend = new ArrayList<>();
        for (Book book : allBooksFromJson) {
            Log.d("BookFilter", "Перевірка книги: " + book.getTitle() + ", Категорія: " + book.getCategory());
            // Перевіряємо, чи категорія книги збігається з цільовою (ігноруючи регістр)
            if (book.getCategory() != null && book.getCategory().equalsIgnoreCase(targetCategory)) {
                Log.d("BookFilter", "Категорія збігається: " + book.getTitle());
                // Перевіряємо, чи є остання прочитана книга
                if (lastReadBookObject == null || book.getId() == null || lastReadBookObject.getId() == null ||
!book.getId().equals(lastReadBookObject.getId())) {
                    Log.d("BookFilter", "Додано до рекомендацій: " + book.getTitle());
                    booksToRecommend.add(book);
                } else {
                    Log.d("BookFilter", "Книга " + book.getTitle() + " є останньою прочитаною (ID: ["+book.getId()+"]), пропускаємо.");
                }
            }
        }

        // Якщо немає прочитаних книг, то за замовчанням рекомендуємо книги з категорії Пригоди
        if (booksToRecommend.isEmpty() && lastReadBookObject != null && !targetCategory.equals("Пригоди")) {
            Log.d("BookListActivity", "Не знайдено книг категорії '" + targetCategory + "'. Спроба завантажити 'Пригоди'.");
            targetCategory = "Пригоди";
            for (Book book : allBooksFromJson) {
                if (book.getCategory() != null && book.getCategory().equalsIgnoreCase(targetCategory)) {
                    if (lastReadBookObject == null || !book.getId().equals(lastReadBookObject.getId())) {
                        booksToRecommend.add(book);
                    }
                }
            }
        }

        // Відображаємо першу рекомендовану книгу, якщо вона є
        if (!booksToRecommend.isEmpty()) {
```

```
            Book book1 = booksToRecommend.get(0);
            loadCoverImage(book1, recBook1Image);
            setupBookClickListener(recBook1Image, book1.getUrl());
            recBook1Image.setVisibility(View.VISIBLE);
        } else {
            recBook1Image.setVisibility(View.GONE);
            Log.d("BookListActivity", "Немає книг для рекомендації в позиції 1 для категорії: " + targetCategory);
        }

        // Відображаємо другу рекомендовану книгу, якщо вона є
        if (booksToRecommend.size() >= 2) {
            Book book2 = booksToRecommend.get(1);
            loadCoverImage(book2, recBook2Image);
            setupBookClickListener(recBook2Image, book2.getUrl());
            recBook2Image.setVisibility(View.VISIBLE);
        } else {
            recBook2Image.setVisibility(View.GONE);
            if(booksToRecommend.size() < 2) Log.d("BookListActivity", "Недостатньо книг для рекомендації в позиції 2 для категорії: " +
targetCategory);
        }

        // Відображаємо третю рекомендовану книгу, якщо вона є
        if (booksToRecommend.size() >= 3) {
            Book book3 = booksToRecommend.get(2);
            loadCoverImage(book3, recBook3Image);
            setupBookClickListener(recBook3Image, book3.getUrl());
            recBook3Image.setVisibility(View.VISIBLE);
        } else {
            recBook3Image.setVisibility(View.GONE);
            if(booksToRecommend.size() < 3) Log.d("BookListActivity", "Недостатньо книг для рекомендації в позиції 3 для категорії: " +
targetCategory);
        }
    }


    private void loadCoverImage(Book book, ImageView imageView) {
        if (book != null && book.getImagePath() != null && !book.getImagePath().isEmpty()) {
            Glide.with(this)
                    .load(Uri.parse("file:///android_asset/"+book.getImagePath()))
                    .into(imageView);
        }
    }

    private void setupBookClickListener(ImageView imageView, final String urlString) {
        if (urlString == null || urlString.isEmpty()) {
            Log.w("BookListActivity", "URL для книги відсутній або порожній. ClickListener не встановлено для ImageView ID: " +
imageView.getId());
            imageView.setClickable(false);
            return;
        }

        imageView.setClickable(true);
        imageView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Log.d("BookListActivity_URL_CLICK", "Clicked. URL to open: [" + urlString + "]");

                if (urlString != null && !urlString.isEmpty()) {
                    Uri uri = null;
                    try {
                        uri = Uri.parse(urlString);
                    } catch (Exception e) {
                        Log.e("BookListActivity_URL_PARSE_ERROR", "Error parsing URL: " + urlString, e);
                        Toast.makeText(BookListActivity.this, "Некоректний формат URL", Toast.LENGTH_SHORT).show();
                        return;
                    }

                    if (uri == null) {
                        Log.e("BookListActivity_URL_ERROR", "Parsed URI is null for string: " + urlString);
                        Toast.makeText(BookListActivity.this, "Не вдалося обробити URL", Toast.LENGTH_SHORT).show();
                        return;
                    }

                    Log.d("BookListActivity_URL_CLICK", "Parsed URI: " + uri.toString());

                    Intent browserIntent = new Intent(Intent.ACTION_VIEW, uri);
                    if (browserIntent.resolveActivity(getPackageManager()) != null) {
                        startActivity(browserIntent);
                    } else {
                        Log.e("BookListActivity_URL_ERROR", "Не знайдено додатка для відкриття URL: " + uri.toString());
                        Toast.makeText(BookListActivity.this, "Не вдалося відкрити посилання. Немає відповідного додатка.",
```

```
Toast.LENGTH_SHORT).show();
                }
            } else {
                Log.w("BookListActivity_URL_CLICK", "Спроба відкрити порожній URL.");
                Toast.makeText(BookListActivity.this, "Посилання для цієї книги відсутнє.", Toast.LENGTH_SHORT).show();
            }
        }
    });
  }
}

class BookPageActivity:

package com.example.bookaplication;

import android.app.ActivityOptions;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.os.SystemClock;
import android.text.InputType;
import android.text.TextUtils;
import android.util.Log;
import android.view.View;
import android.widget.ImageView;
import android.widget.ProgressBar;
import android.widget.TextView;
import android.widget.Toast;

import androidx.activity.EdgeToEdge;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import android.widget.EditText;
import androidx.core.view.WindowInsetsCompat;

import com.bumptech.glide.Glide;

import org.json.JSONException;
import org.json.JSONObject;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.Locale;
import java.util.Map;
import java.util.UUID;
import java.util.concurrent.TimeUnit;

public class BookPageActivity extends AppCompatActivity {

    private ImageView buttonStart;
    private TextView textViewTimerStatus;
    private ImageView bookCoverImageView;
    private TextView bookTitleTextView;
    private TextView bookStarsTextView;
    private TextView bookDescriptionTextView;
    private TextView bookAuthorTextView;
    private TextView bookCategoryTextView;
    private ConstraintLayout backbtn;
    private ConstraintLayout statbtn;
    private ConstraintLayout buttonEditBook;
    private TextView bookPagesTextView;
    private ProgressBar progressBar;
    private boolean timerRunning = false;
    private long startTimeMillis = 0L;
    private int currentPageForSession = 0;

    private Book currentDisplayedBook;
    private Book selectedBook;
    public static final String PREFS_NAME = "BookReadingPrefs";
    private static final String BOOK_RATING_SUFFIX = "_rating";
    private static final String SESSIONS_PREFS_NAME_SUFFIX = "_Sessions";
    private static final String LAST_READ_PAGE_SUFFIX = "_lastReadPage";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_book_page);
        bookCoverImageView = findViewById(R.id.book_CoverImageView);
        bookTitleTextView = findViewById(R.id.book_title);
        bookAuthorTextView = findViewById(R.id.book_author);
        bookStarsTextView = findViewById(R.id.book_stars);
        bookDescriptionTextView = findViewById(R.id.book_description);
        bookCategoryTextView = findViewById(R.id.book_category);
        bookPagesTextView = findViewById(R.id.book_pages);
        buttonStart = findViewById(R.id.button_start);
        backbtn = findViewById(R.id.back_btn);
        buttonEditBook = findViewById(R.id.edit_btn);
        statbtn = findViewById(R.id.stat_page);
        textViewTimerStatus = findViewById(R.id.stats);
        progressBar = findViewById(R.id.progressBar);
        this.selectedBook = getIntent().getParcelableExtra("selected_book");


        if (selectedBook != null) {
            Log.d("BookPageActivity", "Отримано книгу: " + selectedBook.getTitle());

            Log.d("BookPageActivity", "onCreate: Отримано книгу: " + selectedBook.getTitle() + ", Початковий EndPage з Intent/savedState: " +
selectedBook.getEndPage());

            loadSavedEndPageForSelectedBook();

            backbtn.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Intent intent = new Intent(BookPageActivity.this, BookListActivity.class);
                    ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(BookPageActivity.this);
                    startActivity(intent, options.toBundle());
                    finish();
                }
            });

            statbtn.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Intent intent = new Intent(BookPageActivity.this, BookStatsActivity.class);
                    intent.putExtra("updated_book", selectedBook);
                    ArrayList<ReadingSession> allSessions = getAllReadingSessionsForBook();
                    intent.putParcelableArrayListExtra("reading_sessions", allSessions);
                    ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(BookPageActivity.this);
                    startActivity(intent, options.toBundle());
                }
            });

            buttonStart.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    Log.d("BookPageActivity", "Кнопка натиснута. TimerRunning: " + timerRunning);
                    if (timerRunning) {
                        promptForEndPageAndStopTimer();
                    } else {
                        promptForStartPageAndStartTimer();
                    }
                }
            });

            buttonEditBook.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    if (selectedBook != null) {
                        Intent intent = new Intent(BookPageActivity.this, BookEditActivity.class);
                        intent.putExtra("book_to_edit", selectedBook);
                        ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(BookPageActivity.this);
                        startActivity(intent, options.toBundle());
                    }
                }
            });

            Glide.with(this)
                    .load(selectedBook.getImagePath())
                    .into(bookCoverImageView);

            String bookTitle = selectedBook.getTitle();

            bookTitleTextView.setText(bookTitle);
```

```
        bookAuthorTextView.setText(selectedBook.getAuthor());
        loadBookFinishedStatus();
        progressBar.setMax(100);


        if (selectedBook.getFinished() == "false") {
            Log.d("BookPageActivity", "Книгу не закінчили");
            int lastReadPage = Integer.parseInt(selectedBook.getEndPage());
            int totalPages = Integer.parseInt(selectedBook.getPages());
            if (totalPages > 0) {
                int progressPercentage = (int) (((double) lastReadPage / totalPages) * 100);
                progressBar.setProgress(progressPercentage);
            } else {
                progressBar.setProgress(0);
            }
        } else if (selectedBook.getFinished() == "true") {
            Log.d("BookPageActivity", "Книгу закінчили");
            progressBar.setProgress(100);
        }

        Log.d("BookPageActivity", "Кількість сторінок:" + selectedBook.getEndPage());


        if (selectedBook.getStars() == null || selectedBook.getStars().isEmpty()) {
            bookStarsTextView.setText("Оцінка: відсутня");
        } else {
            bookStarsTextView.setText("Оцінка: " + selectedBook.getStars());
        }

        displayBookRating();

        bookStarsTextView.setOnClickListener(v -> showRatingDialog());

        bookDescriptionTextView.setText(selectedBook.getDescription());

        bookCategoryTextView.setText("Жанр: " + selectedBook.getCategory());

        bookPagesTextView.setText("Сторінок: " + selectedBook.getPages());

        loadLastReadingInfo();

    }
    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.book_page), (v, insets) -> {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });
    setupInitialButtonState();

}

private void logAllReadingSessions() {
    if (selectedBook == null) {
        Log.e("BookPageActivity", "Cannot log sessions, selectedBook is null.");
        return;
    }

    SharedPreferences sessionsPrefs = getSharedPreferences(getBookSessionsPrefsName(), Context.MODE_PRIVATE);
    Map<String, ?> allEntries = sessionsPrefs.getAll();

    if (allEntries.isEmpty()) {
        Log.i("BookPageActivity", "No reading sessions found for book: " + selectedBook.getTitle());
        return;
    }

    Log.i("BookPageActivity", "===== All Reading Sessions for: " + selectedBook.getTitle() + " =====");
    for (Map.Entry<String, ?> entry : allEntries.entrySet()) {
        String sessionKey = entry.getKey();
        if (sessionKey.startsWith("session_")) {
            try {
                JSONObject sessionData = new JSONObject(entry.getValue().toString());
                long startTimeMillisActual = sessionData.optLong("startTimeMillisActual", -1);
                long elapsedTimeMillis = sessionData.getLong("elapsedTimeMillis");
                int startPage = sessionData.getInt("startPage");
                int endPage = sessionData.getInt("endPage");
                String date = sessionData.getString("date");

                String formattedElapsedTime = formatMillisToHMS(elapsedTimeMillis);
                String formattedStartTime = "";
                if (startTimeMillisActual != -1) {
                    formattedStartTime = new SimpleDateFormat("dd.MM.yyyy HH:mm:ss", Locale.getDefault()).format(new Date(startTimeMillisActual));
```

```
                }


                Log.i("BookPageActivity", "Session ID: " + sessionKey);
                if (!formattedStartTime.isEmpty()) {
                    Log.i("BookPageActivity", "  Start Time: " + formattedStartTime);
                }
                Log.i("BookPageActivity", "  Duration: " + formattedElapsedTime);
                Log.i("BookPageActivity", "  Pages: " + startPage + " - " + endPage);
                Log.i("BookPageActivity", "  End Date: " + date);
                Log.i("BookPageActivity", "---------------------------------");

            } catch (JSONException e) {
                Log.e("BookPageActivity", "Error parsing session data for key: " + sessionKey, e);
            }
        }
    }
    Log.i("BookPageActivity", "=============================================");
}

private ArrayList<ReadingSession> getAllReadingSessionsForBook() {
    ArrayList<ReadingSession> sessionsList = new ArrayList<>();
    if (selectedBook == null) {
        Log.e("BookPageActivity", "Cannot get sessions, selectedBook is null.");
        return sessionsList;
    }

    SharedPreferences sessionsPrefs = getSharedPreferences(getBookSessionsPrefsName(), Context.MODE_PRIVATE);
    Map<String, ?> allEntries = sessionsPrefs.getAll();

    if (allEntries.isEmpty()) {
        Log.i("BookPageActivity", "No reading sessions found for book: " + selectedBook.getTitle());
        return sessionsList;
    }

    for (Map.Entry<String, ?> entry : allEntries.entrySet()) {
        String sessionKey = entry.getKey();
        if (sessionKey.startsWith("session_")) {
            try {
                JSONObject sessionData = new JSONObject(entry.getValue().toString());
                long startTimeMillisActual = sessionData.optLong("startTimeMillisActual", -1);
                long elapsedTimeMillis = sessionData.getLong("elapsedTimeMillis");
                int startPage = sessionData.getInt("startPage");
                int endPage = sessionData.getInt("endPage");
                String date = sessionData.getString("date");

                sessionsList.add(new ReadingSession(startTimeMillisActual, elapsedTimeMillis, startPage, endPage, date));
            } catch (JSONException e) {
                Log.e("BookPageActivity", "Error parsing session data for key: " + sessionKey, e);
            }
        }
    }
    Log.i("BookPageActivity", "Retrieved " + sessionsList.size() + " sessions for " + selectedBook.getTitle());
    return sessionsList;
}

private void saveBookFinishedStatus(boolean isFinished) {
    if (selectedBook == null || TextUtils.isEmpty(selectedBook.getTitle())) {
        Log.e("BookPageActivity", "Cannot save finished status, book or title is missing.");
        return;
    }
    SharedPreferences prefs = getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = prefs.edit();
    editor.putBoolean(getBookIdForPrefs() + "_finished_status", isFinished);
    editor.apply();
    Log.i("BookPageActivity", "Saved finished status: " + isFinished + " for book: " + selectedBook.getTitle());

    selectedBook.setFinished(String.valueOf(isFinished));
}

private void loadBookFinishedStatus() {
    if (selectedBook == null || TextUtils.isEmpty(selectedBook.getTitle())) {
        return;
    }
    SharedPreferences prefs = getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    boolean isFinished = prefs.getBoolean(getBookIdForPrefs() + "_finished_status", false);
    selectedBook.setFinished(String.valueOf(isFinished));
    Log.i("BookPageActivity", "Loaded finished status: " + isFinished + " for book: " + selectedBook.getTitle());
}

private void loadSavedEndPageForSelectedBook() {
```

```
    if (selectedBook == null || TextUtils.isEmpty(selectedBook.getTitle())) {
        return;
    }
    SharedPreferences prefs = getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    int savedPage = prefs.getInt(getLatestPageKey(), -1);

    if (savedPage != -1) {
        selectedBook.setEndPage(String.valueOf(savedPage));
        Log.i("BookPageActivity", "Завантажено збережений endPage: " + savedPage + " для книги: " + selectedBook.getTitle());
    }
}

private String getBookRatingKey() {
    if (selectedBook == null || TextUtils.isEmpty(selectedBook.getTitle())) {
        Log.e("BookPageActivity", "Cannot create rating key, book or title is missing.");
        return "unknown_book_rating";
    }
    return getBookIdForPrefs() + BOOK_RATING_SUFFIX;
}

private String getLatestPageKey() {
    if (selectedBook == null || TextUtils.isEmpty(selectedBook.getTitle())) {
        Log.e("BookPageActivity", "Cannot create rating key, book or title is missing.");
        return "unknown_book_rating";
    }
    return getBookIdForPrefs() + LAST_READ_PAGE_SUFFIX;
}

@Override
protected void onSaveInstanceState(@NonNull Bundle outState) {
    super.onSaveInstanceState(outState);
    if (selectedBook != null) {
        outState.putParcelable("selected_book_state", selectedBook);
        Log.d("BookPageActivity", "onSaveInstanceState: Збережено endPage: " + selectedBook.getEndPage());
    }
}

private void displayBookRating() {
    if (selectedBook == null) return;

    SharedPreferences prefs = getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    int savedRating = prefs.getInt(getBookRatingKey(), 0);

    if (savedRating >= 1 && savedRating <= 5) {
        bookStarsTextView.setText("Оцінка: " + savedRating + "/5");
        selectedBook.setStars(String.valueOf(savedRating));
    } else {

        String currentBookStars = selectedBook.getStars();
        if (currentBookStars == null || currentBookStars.isEmpty() || currentBookStars.equals("0")) {
            bookStarsTextView.setText("Оцінка: відсутня");
        } else {
            try {
                int ratingFromBookObject = Integer.parseInt(currentBookStars);
                if (ratingFromBookObject >= 1 && ratingFromBookObject <= 5) {
                    bookStarsTextView.setText("Оцінка: " + ratingFromBookObject + "/5");
                } else {
                    bookStarsTextView.setText("Оцінка: відсутня)");
                }
            } catch (NumberFormatException e){
                bookStarsTextView.setText("Оцінка: відсутня");
            }
        }
    }
}

private void showRatingDialog() {
    if (selectedBook == null) {
        Toast.makeText(this, "Книга не вибрана.", Toast.LENGTH_SHORT).show();
        return;
    }

    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("Оцініть книгу: " + selectedBook.getTitle());
    builder.setMessage("Введіть оцінку від 1 до 5:");

    final EditText input = new EditText(this);
    input.setInputType(InputType.TYPE_CLASS_NUMBER);
    input.setHint("1-5");
```

```
    SharedPreferences prefs = getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    int currentRating = prefs.getInt(getBookRatingKey(), 0);
    if (currentRating >= 1 && currentRating <= 5) {
        input.setText(String.valueOf(currentRating));
    }

    builder.setView(input);

    builder.setPositiveButton("Зберегти", (dialog, which) -> {
        String ratingStr = input.getText().toString();
        if (!ratingStr.isEmpty()) {
            try {
                int rating = Integer.parseInt(ratingStr);
                if (rating >= 1 && rating <= 5) {
                    saveBookRating(rating);
                    displayBookRating();
                    Toast.makeText(BookPageActivity.this, "Оцінку збережено: " + rating + "/5", Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(BookPageActivity.this, "Будь ласка, введіть число від 1 до 5.", Toast.LENGTH_SHORT).show();
                }
            } catch (NumberFormatException e) {
                Toast.makeText(BookPageActivity.this, "Будь ласка, введіть дійсне число.", Toast.LENGTH_SHORT).show();
            }
        } else {
            Toast.makeText(BookPageActivity.this, "Оцінка не може бути порожньою.", Toast.LENGTH_SHORT).show();
        }
    });

    builder.setNegativeButton("Скасувати", (dialog, which) -> dialog.cancel());

    builder.setNeutralButton("Видалити оцінку", (dialog, which) -> {
        removeBookRating();
        displayBookRating();
        Toast.makeText(BookPageActivity.this, "Оцінку видалено.", Toast.LENGTH_SHORT).show();
    });

    builder.show();

}

private void saveBookRating(int rating) {
    if (selectedBook == null) {
        Log.e("BookPageActivity", "Cannot save rating, selectedBook is null.");
        return;
    }
    SharedPreferences prefs = getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = prefs.edit();
    editor.putInt(getBookRatingKey(), rating);
    editor.apply();

    selectedBook.setStars(String.valueOf(rating));
}

private void removeBookRating() {
    if (selectedBook == null) {
        Log.e("BookPageActivity", "Cannot remove rating, selectedBook is null.");
        return;
    }
    SharedPreferences prefs = getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = prefs.edit();
    editor.remove(getBookRatingKey());
    editor.apply();

    selectedBook.setStars(null);
}

private void promptForStartPageAndStartTimer() {
    if (selectedBook == null) {
        Toast.makeText(this, "Не можу запустити таймер, книга не вибрана.", Toast.LENGTH_SHORT).show();
        return;
    }

    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("Початок читання");
    builder.setMessage("Введіть номер сторінки, з якої починаєте:");

    final EditText input = new EditText(this);
    input.setInputType(InputType.TYPE_CLASS_NUMBER);
    builder.setView(input);

    builder.setPositiveButton("Почати", (dialog, which) -> {
```

```
        String pageStr = input.getText().toString();
        if (!pageStr.isEmpty()) {
            try {
                int page = Integer.parseInt(pageStr);
                if (page < 0) {
                    Toast.makeText(BookPageActivity.this, "Номер сторінки не може бути від'ємним.", Toast.LENGTH_SHORT).show();
                    return;
                }
                int bookTotalPages = 0;
                try {
                    bookTotalPages = Integer.parseInt(selectedBook.getPages());
                } catch (NumberFormatException e) {
                    Log.e("BookPageActivity", "Invalid page number format from selectedBook.getPages(): " + selectedBook.getPages());
                    Toast.makeText(BookPageActivity.this, "Не вдалося отримати загальну кількість сторінок.", Toast.LENGTH_LONG).show();
                    return;
                }
                if (page > bookTotalPages) {
                    Toast.makeText(BookPageActivity.this, "Сторінка не може бути більшою за кількість сторінок у книзі.",
Toast.LENGTH_LONG).show();
                    return;
                }
                currentPageForSession = page;
                startTimerInternal();
            } catch (NumberFormatException e) {
                Toast.makeText(BookPageActivity.this, "Будь ласка, введіть дійсний номер сторінки.", Toast.LENGTH_SHORT).show();
            }
        } else {
            Toast.makeText(BookPageActivity.this, "Номер сторінки не може бути порожнім.", Toast.LENGTH_SHORT).show();
        }
    });
    builder.setNegativeButton("Скасувати", (dialog, which) -> dialog.cancel());
    builder.show();
}

private void startTimerInternal() {
    if (timerRunning) return;
    startTimeMillis = SystemClock.elapsedRealtime();
    timerRunning = true;
    buttonStart.setImageResource(R.drawable.stop);
    textViewTimerStatus.setText("Йде читання... (з " + currentPageForSession + " стор.)");
    Log.i("BookPageActivity", "Timer started for book: " + selectedBook.getTitle() + " from page " + currentPageForSession);
    Toast.makeText(this, "Таймер запущено для: " + selectedBook.getTitle(), Toast.LENGTH_SHORT).show();
}

private void promptForEndPageAndStopTimer() {
    if (!timerRunning) {
        Log.d("BookPageActivity", "Timer is not running, cannot stop.");
        return;
    }

    AlertDialog.Builder builder = new AlertDialog.Builder(this);
    builder.setTitle("Завершення читання");
    builder.setMessage("Введіть номер сторінки, на якій зупинилися:");

    final EditText input = new EditText(this);
    input.setInputType(InputType.TYPE_CLASS_NUMBER);
    if (currentPageForSession > 0) {
        input.setHint(String.valueOf(currentPageForSession));
    }

    builder.setView(input);

    builder.setPositiveButton("Зупинити", (dialog, which) -> {
        String pageStr = input.getText().toString();
        if (!pageStr.isEmpty()) {
            try {
                int endPage = Integer.parseInt(pageStr);
                int bookTotalPages = 0;
                if (selectedBook.getPages() != null && !selectedBook.getPages().isEmpty()) {
                    try {
                        bookTotalPages = Integer.parseInt(selectedBook.getPages());
                    } catch (NumberFormatException e) {
                        Log.e("BookPageActivity", "Invalid page number format for total pages: " + selectedBook.getPages());
                    }
                }

                if (endPage < 0) {
                    Toast.makeText(BookPageActivity.this, "Номер сторінки не може бути від'ємним.", Toast.LENGTH_SHORT).show();
                    return;
                }
                if (endPage < currentPageForSession) {
```

```
                Toast.makeText(BookPageActivity.this, "Кінцева сторінка не може бути меншою за початкову.", Toast.LENGTH_SHORT).show();
                return;
            }
            if (bookTotalPages > 0 && endPage > bookTotalPages) {
                Toast.makeText(BookPageActivity.this, "Кінцева сторінка не може бути більшою за кількість сторінок у книзі.",
Toast.LENGTH_LONG).show();
                return;
            }
            stopTimerInternal(endPage);
        } catch (NumberFormatException e) {
            Toast.makeText(BookPageActivity.this, "Будь ласка, введіть дійсний номер сторінки.", Toast.LENGTH_SHORT).show();
        }
    } else {
        Toast.makeText(BookPageActivity.this, "Номер кінцевої сторінки не може бути порожнім.", Toast.LENGTH_SHORT).show();
    }
});
builder.setNegativeButton("Скасувати", (dialog, which) -> dialog.cancel());
builder.show();
}

private void stopTimerInternal(int endPage) {
    if (!timerRunning) return;

    long elapsedTimeMillis = SystemClock.elapsedRealtime() - startTimeMillis;
    timerRunning = false;

    saveReadingSession(elapsedTimeMillis, currentPageForSession, endPage);

    String formattedTime = formatMillisToHMS(elapsedTimeMillis);
    String currentDate = new SimpleDateFormat("dd.MM.yyyy HH:mm", Locale.getDefault()).format(new Date());

    Toast.makeText(this, "Сеанс завершено. Прочитано: " + formattedTime + ". Сторінки: " + currentPageForSession + "-" + endPage,
Toast.LENGTH_LONG).show();

    buttonStart.setImageResource(R.drawable.start);
    loadLastReadingInfo();

    SharedPreferences prefs = getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = prefs.edit();
    editor.putInt(getLatestPageKey(), endPage);
    if (selectedBook != null && selectedBook.getId() != null) {
        editor.putString("last_read_book_id", selectedBook.getId());
    }
    editor.apply();

    selectedBook.setEndPage(String.valueOf(endPage));
    Log.d("BookPageActivity", "Кількість сторінок endpage у книзі: " + selectedBook.getEndPage());

    int lastReadPage = Integer.parseInt(selectedBook.getEndPage());
    int totalPages = Integer.parseInt(selectedBook.getPages());

    if (totalPages > 0) {
        int progressPercentage = (int) (((double) lastReadPage / totalPages) * 100);
        progressBar.setProgress(progressPercentage);
    } else {
        progressBar.setProgress(0);
    }

    if(lastReadPage == totalPages){
        saveBookFinishedStatus(true);
        Log.d("BookPageActivity", "Книгу " + selectedBook.getTitle() + " закінчили");
    }

    logAllReadingSessions();

    Log.d("BookPageActivity", "Timer stopped for: " + selectedBook.getTitle() + ". Elapsed: " + formattedTime + ". Pages: " + currentPageForSession +
"-" + endPage);
    currentPageForSession = 0;
}


private void saveReadingSession(long elapsedTimeMillis, int startPage, int endPage) {
    if (selectedBook == null) {
        Log.e("BookPageActivity", "Cannot save session, currentBook is null.");
        return;
    }

    SharedPreferences mainPrefs = getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    SharedPreferences.Editor mainEditor = mainPrefs.edit();
    String timeKey = getBookTimeKey();
    long currentTotalTimeMillis = mainPrefs.getLong(timeKey, 0L);
```

```
    long newTotalTimeMillis = currentTotalTimeMillis + elapsedTimeMillis;
    mainEditor.putLong(timeKey, newTotalTimeMillis);
    String dateKey = getBookLastReadDateKey();
    String currentDate = new SimpleDateFormat("dd.MM.yyyy HH:mm", Locale.getDefault()).format(new Date());
    mainEditor.putString(dateKey, currentDate);
    mainEditor.apply();

    SharedPreferences sessionsPrefs = getSharedPreferences(getBookSessionsPrefsName(), Context.MODE_PRIVATE);
    SharedPreferences.Editor sessionsEditor = sessionsPrefs.edit();
    String sessionKey = "session_" + UUID.randomUUID().toString();
    JSONObject sessionData = new JSONObject();
    try {
        sessionData.put("startTimeMillisActual", System.currentTimeMillis() - elapsedTimeMillis);
        sessionData.put("elapsedTimeMillis", elapsedTimeMillis);
        sessionData.put("startPage", startPage);
        sessionData.put("endPage", endPage);
        sessionData.put("date", currentDate);
        sessionsEditor.putString(sessionKey, sessionData.toString());
        sessionsEditor.apply();
        logAllReadingSessions();
        Log.d("BookPageActivity", "Session saved for '" + selectedBook.getTitle() + "': " + sessionData.toString());
    } catch (JSONException e) {
        Log.e("BookPageActivity", "Error creating JSON for session", e);
    }
}

private void loadLastReadingInfo() {
    if (selectedBook == null) {
        textViewTimerStatus.setText("Інформація про книгу не доступна.");
        return;
    }

    SharedPreferences sharedPreferences = getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
    String dateKey = getBookLastReadDateKey();
    String lastReadDate = sharedPreferences.getString(dateKey, null);
    long elapsed = sharedPreferences.getLong(getBookTimeKey(), 0L);

    if (lastReadDate != null) {
        textViewTimerStatus.setText("Останнє читання: " + lastReadDate + " " + formatMillisToHMS(elapsed));

    } else {
        textViewTimerStatus.setText("Ви ще не читали цю книгу. Почніть зараз!");
    }
}


private void setupInitialButtonState() {
    buttonStart.setImageResource(R.drawable.start);
}

private String getBookTimeKey() {
    return "book_time_" + getBookIdForPrefs();
}

private String getBookLastReadDateKey() {
    return "book_last_read_date_" + getBookIdForPrefs();
}

private String formatMillisToHMS(long millis) {
    if (millis < 0) {
        return "00:00:00";
    }

    long hours = TimeUnit.MILLISECONDS.toHours(millis);
    long minutes = TimeUnit.MILLISECONDS.toMinutes(millis) % TimeUnit.HOURS.toMinutes(1);
    long seconds = TimeUnit.MILLISECONDS.toSeconds(millis) % TimeUnit.MINUTES.toSeconds(1);

    return String.format(Locale.getDefault(), "%02d:%02d:%02d", hours, minutes, seconds);
}

private static class SessionData {
    private String sessionId;
    private int endPage;
    private Date sessionEndTime;

    public SessionData(String sessionId, int endPage, Date sessionEndTime) {
        this.sessionId = sessionId;
        this.endPage = endPage;
        this.sessionEndTime = sessionEndTime;
    }
```

```
        public String getSessionId() {
            return sessionId;
        }

        public int getEndPage() {
            return endPage;
        }

        public Date getSessionEndTime() {
            return sessionEndTime;
        }
    }


    private String getBookSessionsPrefsName() {
        if (selectedBook == null || TextUtils.isEmpty(selectedBook.getTitle())) {
            Log.e("BookPageActivity", "Cannot create sessions prefs name, book or title is missing.");
            return "unknown_book_sessions";
        }
        String safeTitle = selectedBook.getTitle().replaceAll("[^a-zA-Z0-9_.-]", "_");
        return "BookSessions_" + safeTitle + "_Prefs";
    }


    private String getBookIdForPrefs() {
        if (selectedBook == null || TextUtils.isEmpty(selectedBook.getTitle())) {
            Log.e("BookPageActivity", "Book or book title is null, cannot generate a unique ID for preferences.");

            return "unknown_book";
        }

        return selectedBook.getTitle().replaceAll("[^a-zA-Z0-9]", "_");
    }
}

class BookRepository:

package com.example.bookaplication;

import android.content.Context;
import android.content.SharedPreferences;
import android.util.Log;

import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;

import java.io.InputStream;
import java.io.InputStreamReader;
import java.lang.reflect.Type;
import java.util.ArrayList;
import java.util.List;
import java.util.UUID;

/**
 * Клас BookRepository відповідає за управління збереженням та отриманням об'єктів Book.
 * Він використовує SharedPreferences для збереження даних у форматі JSON.
 * Gson використовується для серіалізації та десеріалізації об'єктів Book.
 */
public class BookRepository {
    private static final String PREFS_NAME = "BookAppPrefs";
    private static final String BOOKS_KEY = "books";
    private SharedPreferences sharedPreferences;
    private Gson gson;

    /**
     * Конструктор для BookRepository.
     * @param context Контекст програми, використовується для доступу до SharedPreferences.
     */
    public BookRepository(Context context) {
        sharedPreferences = context.getSharedPreferences(PREFS_NAME, Context.MODE_PRIVATE);
        gson = new Gson();
    }

    public List<Book> getAllBooks() {
        String jsonBooks = sharedPreferences.getString(BOOKS_KEY, null);
        if (jsonBooks == null) {
            return new ArrayList<>();
        }
        Type type = new TypeToken<ArrayList<Book>>() {}.getType();
        return gson.fromJson(jsonBooks, type);
    }
```

```
    public void addBook(Book book) {
        List<Book> books = getAllBooks();
        if (book.getId() == null || book.getId().isEmpty()) {
            book.setId(UUID.randomUUID().toString());
        }
        books.add(0, book);
        saveBooks(books);
    }

    public void updateBook(Book book) {
        List<Book> books = getAllBooks();
        for (int i = 0; i < books.size(); i++) {
            if (books.get(i).getId().equals(book.getId())) {
                books.set(i, book);
                saveBooks(books);
                return;
            }
        }
    }

    public void removeBookById(String bookId) {
        if (bookId == null || bookId.isEmpty()) {
            Log.e("BookRepository", "Cannot remove book: book ID is null/empty.");
            return;
        }
        List<Book> books = getAllBooks();
        boolean removed = false;
        for (int i = 0; i < books.size(); i++) {
            if (books.get(i).getId().equals(bookId)) {
                books.remove(i);
                removed = true;
                break;
            }
        }
        if (removed) {
            saveBooks(books);
            Log.d("BookRepository", "Book with ID: " + bookId + " removed successfully.");
        } else {
            Log.w("BookRepository", "Book with ID: " + bookId + " not found for removal.");
        }
    }

    private void saveBooks(List<Book> books) {
        String jsonBooks = gson.toJson(books);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.putString(BOOKS_KEY, jsonBooks);
        editor.apply();
    }
}
```

class BookStatsActivity:

```
package com.example.bookaplication;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.text.TextUtils;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.TextView;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

import java.util.ArrayList;
import java.util.Locale;
import java.util.concurrent.TimeUnit;

public class BookStatsActivity extends AppCompatActivity {

    private TextView bookTitleTextView;
    private TextView bookAuthorTextView;
    private LinearLayout statsContainerLayout;
```

```
private Book selectedBook;
private ArrayList<ReadingSession> readingSessions;

public static final String PREFS_NAME = "BookAppPrefs";

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    EdgeToEdge.enable(this);
    setContentView(R.layout.activity_book_stats);
    statsContainerLayout = findViewById(R.id.statsContainer);
    bookTitleTextView = findViewById(R.id.book_title);
    bookAuthorTextView = findViewById(R.id.book_author);


    Intent intent = getIntent();
    if (intent != null && intent.hasExtra("updated_book")) {
        selectedBook = intent.getParcelableExtra("updated_book");
        readingSessions = getIntent().getParcelableArrayListExtra("reading_sessions");

        statsContainerLayout.removeAllViews();

        if (selectedBook != null) {
            Log.d("BookStatsActivity", "Отримано книгу: " + selectedBook.getTitle());
            bookTitleTextView.setText(selectedBook.getTitle());
            bookAuthorTextView.setText(selectedBook.getAuthor());

        } else {
            Log.e("BookStatsActivity", "Не вдалося отримати selected_book з Intent.");
        }

        if (readingSessions != null && !readingSessions.isEmpty()) {
            Log.d("BookStatsActivity", "Received " + readingSessions.size() + " reading sessions.");
            for (ReadingSession session : readingSessions) {

                Log.d("BookStatsActivity", "Session: " +
                        "Start Page: " + session.getStartPage() +
                        ", End Page: " + session.getEndPage() +
                        ", Duration (ms): " + session.getElapsedTimeMillis() +
                        ", Date: " + session.getDate() +
                        ", Actual Start Time (ms): " + session.getStartTimeMillisActual());

            }
            displayReadingSessions();
        }

    } else {
        Log.e("BookStatsActivity", "Intent не містить 'updated_book'.");
    }

    ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
        Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
        v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
        return insets;
    });
}

private void displayReadingSessions() {
    LayoutInflater inflater = LayoutInflater.from(this);

    for (ReadingSession session : readingSessions) {

        View sessionItemView = inflater.inflate(R.layout.list_item_stats_block, statsContainerLayout, false);

        TextView statDate = sessionItemView.findViewById(R.id.stat_date);
        TextView statTime = sessionItemView.findViewById(R.id.stat_time);
        TextView statPercentage = sessionItemView.findViewById(R.id.stat_percentage);
        TextView statPages = sessionItemView.findViewById(R.id.stat_pages);

        statDate.setText(session.getDate());

        statTime.setText(formatMillisToHMS(session.getElapsedTimeMillis()));

        int pagesReadInSession = session.getEndPage() - session.getStartPage();
        if (pagesReadInSession < 0) pagesReadInSession = 0;
        statPages.setText(String.format(Locale.getDefault(), "%d стор.", pagesReadInSession));



        if (selectedBook != null && Integer.parseInt(selectedBook.getPages()) > 0) {
            double percentageReadInSession = ((double) pagesReadInSession / Integer.parseInt(selectedBook.getPages())) * 100;
```

```
            if (percentageReadInSession < 0) percentageReadInSession = 0;
            statPercentage.setText(String.format(Locale.getDefault(), "%.1f%%", percentageReadInSession));
        } else {
            statPercentage.setText("N/A");
        }

        statsContainerLayout.addView(sessionItemView);
    }
}

private void displayNoSessionsMessage() {
    TextView noSessionsMsg = new TextView(this);
    noSessionsMsg.setText("Для цієї книги ще немає збережених сесій читання.");
    noSessionsMsg.setTextAlignment(View.TEXT_ALIGNMENT_CENTER);
    noSessionsMsg.setPadding(0, 50, 0, 0);
    statsContainerLayout.addView(noSessionsMsg);
}

public static String formatMillisToHMS(long millis) {
    long hours = TimeUnit.MILLISECONDS.toHours(millis);
    long minutes = TimeUnit.MILLISECONDS.toMinutes(millis) % 60;
    long seconds = TimeUnit.MILLISECONDS.toSeconds(millis) % 60;

    if (hours > 0) {
        return String.format(Locale.getDefault(), "%02d:%02d:%02d", hours, minutes, seconds);
    } else {
        return String.format(Locale.getDefault(), "%02d:%02d", minutes, seconds);
    }
}

private String getBookIdForPrefs() {
    if (selectedBook != null && !TextUtils.isEmpty(selectedBook.getTitle())) {
        return selectedBook.getTitle().replaceAll("[^a-zA-Z0-9]", "_");
    }
    return "unknown_book";
}

private String getBookLastReadDateKey() {
    return "book_last_read_date_" + getBookIdForPrefs();
}

}
```

class CategorySelectionActivity:

```
package com.example.bookaplication;

import android.app.Activity;
import android.content.Intent;
import android.content.res.AssetManager;
import android.os.Bundle;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.widget.LinearLayout;
import android.widget.TextView;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.List;

public class CategorySelectionActivity extends AppCompatActivity {

    private static final String TAG = "CategorySelection";
    public static final String EXTRA_SELECTED_CATEGORY = "extra_selected_category";

    private LinearLayout categoriesContainer;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
```

```
        setContentView(R.layout.activity_category_selection);
        categoriesContainer = findViewById(R.id.categoriesContainer);
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });

        List<String> genres = loadGenresFromAssets("genres.txt");
        populateCategories(genres);
    }

    private List<String> loadGenresFromAssets(String fileName) {
        List<String> genresList = new ArrayList<>();
        AssetManager assetManager = getAssets();
        try (InputStream inputStream = assetManager.open(fileName);
             BufferedReader reader = new BufferedReader(new InputStreamReader(inputStream))) {
            String line;
            while ((line = reader.readLine()) != null) {
                if (!line.trim().isEmpty()) {
                    genresList.add(line.trim());
                }
            }
        } catch (IOException e) {
            Log.e(TAG, "Error loading genres from assets: " + fileName, e);
        }
        return genresList;
    }

    private void populateCategories(List<String> genres) {
        if (categoriesContainer == null) {
            Log.e(TAG, "categoriesContainer is null. Cannot populate categories.");
            return;
        }
        categoriesContainer.removeAllViews();

        LayoutInflater inflater = LayoutInflater.from(this);

        for (String genre : genres) {
            View itemView = inflater.inflate(R.layout.list_item_category, categoriesContainer, false);

            TextView textViewCategoryName = itemView.findViewById(R.id.textViewCategoryName);
            if (textViewCategoryName != null) {
                textViewCategoryName.setText(genre);
            } else {
                Log.e(TAG, "TextView with ID textViewCategoryName not found in list_item_category.xml");
            }

            itemView.setOnClickListener(v -> {
                Intent resultIntent = new Intent();
                resultIntent.putExtra(EXTRA_SELECTED_CATEGORY, genre);
                setResult(Activity.RESULT_OK, resultIntent);
                finish();
            });

            categoriesContainer.addView(itemView);
        }
    }

}

class ReadingSession:

package com.example.bookaplication;

import android.os.Parcel;
import android.os.Parcelable;

public class ReadingSession implements Parcelable {
    private long startTimeMillisActual;
    private long elapsedTimeMillis;
    private int startPage;
    private int endPage;
    private String date;

    public ReadingSession(long startTimeMillisActual, long elapsedTimeMillis, int startPage, int endPage, String date) {
        this.startTimeMillisActual = startTimeMillisActual;
        this.elapsedTimeMillis = elapsedTimeMillis;
        this.startPage = startPage;
        this.endPage = endPage;
        this.date = date;
```

```
    }
    public long getStartTimeMillisActual() { return startTimeMillisActual; }
    public long getElapsedTimeMillis() { return elapsedTimeMillis; }
    public int getStartPage() { return startPage; }
    public int getEndPage() { return endPage; }
    public String getDate() { return date; }

    protected ReadingSession(Parcel in) {
        startTimeMillisActual = in.readLong();
        elapsedTimeMillis = in.readLong();
        startPage = in.readInt();
        endPage = in.readInt();
        date = in.readString();
    }

    public static final Creator<ReadingSession> CREATOR = new Creator<ReadingSession>() {
        @Override
        public ReadingSession createFromParcel(Parcel in) {
            return new ReadingSession(in);
        }

        @Override
        public ReadingSession[] newArray(int size) {
            return new ReadingSession[size];
        }
    };

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {
        dest.writeLong(startTimeMillisActual);
        dest.writeLong(elapsedTimeMillis);
        dest.writeInt(startPage);
        dest.writeInt(endPage);
        dest.writeString(date);
    }

    @Override
    public String toString() {
        return "ReadingSession{" +
                "startTimeMillisActual=" + startTimeMillisActual +
                ", elapsedTimeMillis=" + elapsedTimeMillis +
                ", startPage=" + startPage +
                ", endPage=" + endPage +
                ", date='" + date + '\'' +
                '}';
    }
}
```

class TotalStatsActivity:

```
package com.example.bookaplication;

import android.app.ActivityOptions;
import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.TextView;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import androidx.core.graphics.Insets;
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

import org.json.JSONException;
import org.json.JSONObject;

import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.Date;
```

```
import java.util.List;
import java.util.Locale;
import java.util.Map;
import java.util.concurrent.TimeUnit;

public class TotalStatsActivity extends AppCompatActivity {

    private TextView textViewTotalTimeAllBooks;
    private TextView textViewTotalPagesAllBooks;
    private TextView textViewBookCount;
    private TextView textViewTodayTime;
    private TextView textViewTodayPages;
    private TextView textViewTotalBooks;
    private TextView TodayDate;
    private TextView YearDate;
    private ConstraintLayout navbookadd;
    private ConstraintLayout navanalitic;

    private static final String SESSION_DATE_FORMAT = "dd.MM.yyyy HH:mm";

    public static final String BOOKS_PREFS = "BookReadingPrefs";
    public static final String BOOKS_LIST_KEY = "booksList";
    private BookRepository bookRepository;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.activity_total_stats);
        textViewTotalBooks = findViewById(R.id.statistic_yearbooks);
        textViewTotalPagesAllBooks = findViewById(R.id.statistic_yearpages);
        textViewTotalTimeAllBooks = findViewById(R.id.statistic_yeartime);
        textViewTodayPages = findViewById(R.id.statistic_todaypages);
        textViewTodayTime = findViewById(R.id.statistic_todaytime);
        TodayDate = findViewById(R.id.today_date);
        YearDate = findViewById(R.id.year_date);
        navanalitic = findViewById(R.id.nav_analitic);
        navbookadd = findViewById(R.id.nav_bookadd);
        textViewBookCount = findViewById(R.id.statistic_bookcount);

        navbookadd.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(TotalStatsActivity.this, BookAddActivity.class);
                ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(TotalStatsActivity.this);
                startActivity(intent, options.toBundle());
                finish();
            }
        });
        navanalitic.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(TotalStatsActivity.this, BookListActivity.class);
                ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(TotalStatsActivity.this);
                startActivity(intent, options.toBundle());
                finish();
            }
        });


        SimpleDateFormat sdf = new SimpleDateFormat("dd.MM.yyyy", Locale.getDefault());
        String date = sdf.format(new Date());

        SimpleDateFormat df = new SimpleDateFormat("yyyy", Locale.getDefault());
        String year = df.format(new Date());

        TodayDate.setText(date);
        YearDate.setText(year + " рік");
        bookRepository = new BookRepository(getApplicationContext());
        calculateAndDisplayTotalStats();
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.main), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });
    }

    private void calculateAndDisplayTotalStats() {
        List<Book> allBooks = bookRepository.getAllBooks();
        long grandTotalPagesRead = 0;
        long grandTotalMillisRead = 0;
```

```
        long todayPagesRead = 0;
        long todayMillisRead = 0;
        int finishedBooksCount = 0;
        int totalBooksInRepository = 0;

        if (allBooks == null || allBooks.isEmpty()) {
            Log.i("TotalStatsActivity", "No books found via BookRepository to calculate total stats.");
            textViewTotalBooks.setText("Завершено книг: 0");
            textViewTotalPagesAllBooks.setText("Прочитано сторінок: 0");
            textViewTotalTimeAllBooks.setText("Загальний час читання: 00:00");
            textViewTodayPages.setText("Прочитано сторінок: 0");
            textViewTodayTime.setText("Час читання: 00:00");
            textViewBookCount.setText("Всього книг: 0");
            return;
        }

        SimpleDateFormat sessionDateFormat = new SimpleDateFormat(SESSION_DATE_FORMAT, Locale.getDefault());
        Calendar todayCalendar = Calendar.getInstance();
        totalBooksInRepository = allBooks.size();

        Log.d("TotalStatsActivity", "Found " + allBooks.size() + " books via BookRepository to process.");

        for (Book book : allBooks) {
            if (book == null || book.getTitle() == null) {
                Log.w("TotalStatsActivity", "Skipping a null book or book with null title from repository.");
                continue;
            }
            SharedPreferences prefs = getSharedPreferences(BOOKS_PREFS, Context.MODE_PRIVATE);
            boolean isFinished = prefs.getBoolean(book.getTitle().replaceAll("[^a-zA-Z0-9]", "_") + "_finished_status", false);
            book.setFinished(String.valueOf(isFinished));

            if (book.getFinished() == "true") {
                finishedBooksCount++;

                Log.d("TotalStatsActivity", "Finished book found: " + book.getTitle());
            } else {
                Log.d("TotalStatsActivity", "======================");
                Log.d("TotalStatsActivity", book.getFinished());
                Log.d("TotalStatsActivity", "======================");

                Log.d("TotalStatsActivity", "Skipping unfinished book: " + book.getTitle());
            }
            Log.d("TotalStatsActivity", "Processing book: " + book.getTitle());
            ArrayList<ReadingSession> sessionsForThisBook = getReadingSessionsForSpecificBook(book);
            for (ReadingSession session : sessionsForThisBook) {
                int pagesReadInSession = session.getEndPage() - session.getStartPage();
                if (pagesReadInSession < 0) pagesReadInSession = 0;
                long timeMillisInSession = session.getElapsedTimeMillis();

                grandTotalPagesRead += pagesReadInSession;
                grandTotalMillisRead += session.getElapsedTimeMillis();

                try {
                    // Переконуємося, що session.getDate() не null і не порожній
                    if (session.getDate() != null && !session.getDate().isEmpty()) {
                        Date sessionDate = sessionDateFormat.parse(session.getDate()); // Парсимо дату сесії
                        if (sessionDate != null) {
                            Calendar sessionCalendar = Calendar.getInstance();
                            sessionCalendar.setTime(sessionDate); // Встановлюємо дату сесії в календар

                            // Порівнюємо рік та день року
                            if (todayCalendar.get(Calendar.YEAR) == sessionCalendar.get(Calendar.YEAR) &&
                                    todayCalendar.get(Calendar.DAY_OF_YEAR) == sessionCalendar.get(Calendar.DAY_OF_YEAR)) {
                                // Якщо рік і день року збігаються, це сьогоднішня сесія
                                todayPagesRead += pagesReadInSession;
                                todayMillisRead += timeMillisInSession;
                                Log.d("TotalStatsActivity", "Session for book '" + book.getTitle() + "' on " + session.getDate() + " IS TODAY. Pages: " +
pagesReadInSession);
                            } else {
                                Log.d("TotalStatsActivity", "Session for book '" + book.getTitle() + "' on " + session.getDate() + " IS NOT TODAY.");
                            }
                        }
                    } else {
                        Log.w("TotalStatsActivity", "Session for book '" + book.getTitle() + "' has null or empty date string.");
                    }
                } catch (ParseException e) {
                    Log.e("TotalStatsActivity", "Error parsing session date: '" + session.getDate() + "' for book '" + book.getTitle() + "'. Ensure format matches
SESSION_DATE_FORMAT.", e);
                }
            }
            Log.d("TotalStatsActivity", "Book: " + book.getTitle() + ", Sessions: " + sessionsForThisBook.size() +
```

```
                ", Current Total Pages: " + grandTotalPagesRead + ", Current Total Time (ms): " + grandTotalMillisRead);
        }

        textViewBookCount.setText(String.format(Locale.getDefault(), "Всього книг: %d", totalBooksInRepository));
        textViewTodayPages.setText(String.format(Locale.getDefault(), "Прочитано сторінок: %d", todayPagesRead));
        textViewTodayTime.setText(String.format(Locale.getDefault(), "Час читання: %s", formatMillisToHMS(todayMillisRead)));
        textViewTotalBooks.setText(String.format(Locale.getDefault(), "Завершено книг: %d", finishedBooksCount));
        textViewTotalPagesAllBooks.setText(String.format(Locale.getDefault(), "Прочитано сторінок: %d", grandTotalPagesRead));
        textViewTotalTimeAllBooks.setText(String.format(Locale.getDefault(), "Загальний час читання: %s",
formatMillisToHMS(grandTotalMillisRead)));
        Log.i("TotalStatsActivity", "Final Total Pages Read: " + grandTotalPagesRead);
        Log.i("TotalStatsActivity", "Final Total Time Read (ms): " + grandTotalMillisRead + " (" + formatMillisToHMS(grandTotalMillisRead) + ")");
    }

    private ArrayList<ReadingSession> getReadingSessionsForSpecificBook(Book book) {
        ArrayList<ReadingSession> sessionsList = new ArrayList<>();
        if (book == null || book.getTitle() == null) {
            Log.e("TotalStatsActivity", "Cannot get sessions, book or book title is null.");
            return sessionsList;
        }

        String safeTitle = book.getTitle().replaceAll("[^a-zA-Z0-9_.-]", "_");
        String sessionsPrefsName = "BookSessions_" + safeTitle + "_Prefs";
        Log.d("TotalStatsActivity", "Loading sessions for book '" + book.getTitle() + "' from prefs file: " + sessionsPrefsName);

        SharedPreferences sessionsPrefs = getApplicationContext().getSharedPreferences(sessionsPrefsName, Context.MODE_PRIVATE);
        Map<String, ?> allEntries = sessionsPrefs.getAll();

        if (allEntries.isEmpty()) {
            Log.i("TotalStatsActivity", "No reading sessions found for book: " + book.getTitle());
            return sessionsList;
        }

        for (Map.Entry<String, ?> entry : allEntries.entrySet()) {
            String sessionKey = entry.getKey();
            if (sessionKey.startsWith("session_")) {
                try {
                    JSONObject sessionData = new JSONObject(entry.getValue().toString());
                    long startTimeMillisActual = sessionData.optLong("startTimeMillisActual", -1);
                    long elapsedTimeMillis = sessionData.getLong("elapsedTimeMillis");
                    int startPage = sessionData.getInt("startPage");
                    int endPage = sessionData.getInt("endPage");
                    String date = sessionData.getString("date");

                    sessionsList.add(new ReadingSession(startTimeMillisActual, elapsedTimeMillis, startPage, endPage, date));
                } catch (JSONException e) {
                    Log.e("TotalStatsActivity", "Error parsing session data for key: " + sessionKey + " for book: " + book.getTitle(), e);
                }
            }
        }
        return sessionsList;
    }

    public static String formatMillisToHMS(long millis) {
        long hours = TimeUnit.MILLISECONDS.toHours(millis);
        long minutes = TimeUnit.MILLISECONDS.toMinutes(millis) % 60;
        long seconds = TimeUnit.MILLISECONDS.toSeconds(millis) % 60;

        if (hours > 0) {
            return String.format(Locale.getDefault(), "%02d:%02d:%02d", hours, minutes, seconds);
        } else {
            return String.format(Locale.getDefault(), "%02d:%02d", minutes, seconds);
        }
    }

}
```

class WelcomeActivity:

```
package com.example.bookaplication;

import android.app.ActivityOptions;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

import androidx.activity.EdgeToEdge;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.graphics.Insets;
```

```
import androidx.core.view.ViewCompat;
import androidx.core.view.WindowInsetsCompat;

import com.google.gson.Gson;
import com.google.gson.reflect.TypeToken;

import java.lang.reflect.Type;
import java.util.ArrayList;
import java.util.List;

public class WelcomeActivity extends AppCompatActivity {

    public static final String SHARED_PREFS_NAME = "BookAppPrefs";
    public static final String BOOK_LIST_KEY = "books";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        EdgeToEdge.enable(this);
        setContentView(R.layout.welcome_screen);

        if (hasBooksInStorage()) {
            Intent intent = new Intent(WelcomeActivity.this, BookListActivity.class);
            ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation(this);
            startActivity(intent, options.toBundle());
            finish();
        } else {
            Button button = findViewById(R.id.start_btn);
            button.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View v) {
                    startActivity(new Intent(WelcomeActivity.this, BookAddActivity.class),
ActivityOptions.makeSceneTransitionAnimation(WelcomeActivity.this).toBundle());
                }
            });
        }
        ViewCompat.setOnApplyWindowInsetsListener(findViewById(R.id.welcome_screen), (v, insets) -> {
            Insets systemBars = insets.getInsets(WindowInsetsCompat.Type.systemBars());
            v.setPadding(systemBars.left, systemBars.top, systemBars.right, systemBars.bottom);
            return insets;
        });
    }
    private boolean hasBooksInStorage() {
        SharedPreferences sharedPreferences = getSharedPreferences(SHARED_PREFS_NAME, MODE_PRIVATE);
        String jsonBookList = sharedPreferences.getString(BOOK_LIST_KEY, null);

        if (jsonBookList == null) {
            return false;
        }

        if (jsonBookList.trim().isEmpty() || jsonBookList.trim().equals("[]")) {
            return false;
        }

        try {
            Gson gson = new Gson();
            Type type = new TypeToken<ArrayList<Book>>() {}.getType();
            List<Book> bookList = gson.fromJson(jsonBookList, type);
            return bookList != null && !bookList.isEmpty();
        } catch (Exception e) {
            android.util.Log.e("WelcomeActivity", "Error parsing book list from SharedPreferences", e);
            return false;
        }
    }

}
```

Додаток Б

Копія опублікованих результатів

Анастасія Зборовська
Студентка групи КН-41
www.zborov.ua@gmail.com
Західноукраїнський національний університет
Тернопіль, Україна

РОЗРОБКА МОБІЛЬНОГО ЗАСТОСУНКУ АНАЛІТИКИ ДЛЯ РЕКОМЕНДАЦІЙНОЇ СИСТЕМИ ЧИТАННЯ КНИГ

Кожного дня у сучасному інформаційному світі зростає потреба у швидкому доступі до персоналізованої інформації. Зокрема у галузі читання все більше людей потребують корисні інструменти для читання. Колись люди для відслідковування своїх читацьких звичок використовували читацькі щоденники для запису свого прогресу і записування прочитаних книг. З приходом сучасних інформаційних технологій такі читачі бажають спростити цей процес, щоб цей щоденник знаходився безпосередньо у їхньому мобільному телефоні. Такі читацькі щоденники називаються Трекером [1].

Необхідно зазначити, що згідно статті “Книги в Україні: як змінилася поведінка читачів під час повномасштабного вторгнення” яку було опубліковано “Forbes.ua”, частка людей у нашій країні, які читають книги щодня, зросла удвічі [2]. Отже зробимо припущення, що для читачів, які користуються електронними книгами буде зручно мати у своєму мобільному телефоні електронну бібліотеку книг. А також не менш зручно буде отримувати відразу після прочитання книги персоналізовані рекомендації, які будуть заохочувати українців ще більше читати. Рекомендаційна система – це ефективний інструмент для фільтрації онлайн-інформації [3]. Саме тому тема мобільний застосунок аналітики для рекомендаційної системи читання книг є зараз актуальною.

Для розробки мобільних застосунків найчастіше використовують такі середовища, як Android Studio (для Android) [4] та Xcode (для IOS) [5]. Що стосується вибору мови програмування, то для Android переважно використовують Java та Kotlin. Для IOS – Swift. Також використовують React Native та Flutter [6], які дозволяють створювати додатки для Android та IOS з одного коду.

Розроблюваний застосунок вирішено спрямувати на Android-користувачів. Тому для розробки мобільного застосунку аналітики для рекомендаційної системи читання книг було обрано середовище Android Studio та мову програмування Java, а також розширювану мову розмітки XML для інтерфейсу користувача [7]. До переваг цього вибору слід зарахувати легкість роботи з даним середовищем, великий обсяг бібліотек, а також зручний перезапуск програми.

Розроблений мобільний застосунок знаходиться у кінцевій фазі реалізації. Дизайн додатку є простим і функціональним. Інтерфейс користувача інтуїтивно зрозумілий та адаптований до різних пристроїв.

Користувач мобільного застосунку аналітики для рекомендаційної системи – це людина, яка бажає вести аналітику прочитаних книг і хоче отримувати персональні рекомендації на основі її літературних уподобань.

Особливістю концепції є локальна робота застосунку. Тобто рекомендаційна система працює, використовуючи внутрішню логіку порівняння параметрів книг.

Аналітика формується на основі прочитаних і доданих у бібліотеку книг. При додаванні книги до бібліотеки потрібно вказати основні дані: назва, автор, жанр, кількість сторінок, обкладинку і, за бажанням, опис книги. Прочитавши книгу до кінця, для кращої роботи рекомендаційної системи, потрібно виставити оцінку.

Рекомендаційна система працює за рахунок збору даних у процесі читання і здійснення процедур аналітики. А саме, до основних рекомендацій включено до уваги жанри найвище оцінених і останніх 3 прочитаних книг.

На рисунку 1 можна побачити основні дії, які користувач може виконати у мобільному застосунку аналітики для рекомендаційної системи, щоб отримати рекомендації.

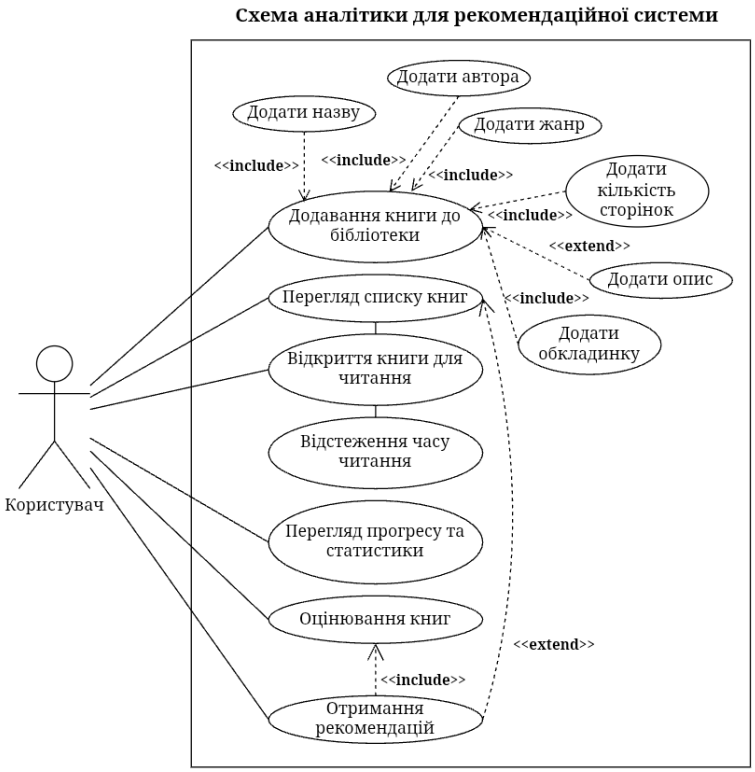


Рисунок 1 – Діаграма прецедентів

Список використаних джерел

1. Трекер читання "Люблю читати". URL: https://vseosvita.ua/library/treker-chytannia-liubliu-chytaty-782932.html
2. Книги в Україні: як змінилася поведінка читачів під час повномасштабного вторгнення. URL: https://forbes.ua/news/yak-chitayut-ukraintsi-pid-chas-vtorgnennya-chastka-lyudey-yaki-chitayut-knigi-shchodnya-zrosla-vdvichi-doslidzhennya-uik-12102023-16629?utm_source=chatgpt.com
3. A systematic review and research perspective on recommender systems. URL: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-022-00592-5

4. Найкращі інструменти для розробки додатків для Android. URL: https://uk.androidsis.com/найкращі-інструменти-для-розробників-додатків-для-android/
5. Ключові етапи створення додатка для IOS. URL: https://wezom.com.ua/ua/blog/klyuchovi-etapi-stvorennya-dodatka-dlya-ios
6. Flutter vs. React Native. Rzetelne porównanie. URL: https://mobitouch.net/pl/blog/flutter-vs-react-native-rzetelne-porownanie
7. XML introduction. URL: https://developer.mozilla.org/en-US/docs/Web/XML/Guides/XML_introduction