**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО”**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

**До захисту допущено:**
**В. о. завідувача кафедри**
_____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

# Дипломний проєкт

**на здобуття ступеня бакалавра**

**за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”**

**спеціальності 121 “Інженерія програмного забезпечення”**

на тему: Мобільний застосунок для електронної бібліотеки

Виконав : студент __4__ курсу, групи ІМ-11
(шифр групи)

_____Щегель Андрій Сергійович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____асистент Гайдай А. Р._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) асистент Нікольский С. С._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _доцент кафедри ІСТ, к.т.н. Шимкович В. М._____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2025 р.

**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**
**“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ**
**імені ІГОРЯ СІКОРСЬКОГО”**

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

**ЗАТВЕРДЖУЮ**
**В. о. завідувача кафедри**
_____ Михайло НОВОТАРСЬКИЙ
(підпис)

“__” _____2025 р.

**ЗАВДАННЯ**
на бакалаврський дипломний проєкт студента

Щегля Андрія Сергійовича

1. Тема проєкту *Мобільний застосунок для електронної бібліотеки*
керівник проєкту Гайдай Анатолій Русланович, асистент,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 23 травня 2025 року № 1705-с

2. Термін здачі студентом закінченого проєкту *2 червня 2025 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані*.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
Опис мобільних застосунків для читання електронних книг, Вибір технологій розробки, Опис розробленого програмного забезпечення, Аналіз розробленого проєкту.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) структурна схема системи, функціональна схема (ER діаграма), принципова схема (UML діаграма прецендентів).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

| Розділ | Консультант | Підпис, дата | |
|---|---|---|---|
| | | Завдання видав | Завдання прийняв |
| Нормоконтроль | Нікольский С. С. | | |
| | | | |

7. Дата видачі завдання _____

Календарний план

| № п/п | Найменування етапів дипломного проєкту | Терміни виконання етапів проєкту | Примітки |
|---|---|---|---|
| 1. | *Затвердження теми проєкту* | 27.01.2025-02.02.2025 | |
| *2.* | *Вивчення та аналіз завдання* | 03.02.2025-09.02.2025 | |
| *3.* | *Розробка архітектури та загальної структури системи* | 10.02.2025-16.03.2025 | |
| *4.* | *Розробка структур окремих підсистем* | 17.03.2025-23.03.2025 | |
| *5.* | *Програмна реалізація системи* | 24.03.2025-11.05.2025 | |
| 6. | *Оформлення пояснювальної записки* | 12.05.2025-29.05.2025 | |
| 7. | *Захист програмного продукту* | 30.05.2025 | |
| *8.* | *Передзахист* | 09.06.2025 | |
| *9.* | *Захист* | 17.06.2025 | |

Студент-дипломник _____ Андрій ЩЕГЕЛЬ
(підпис)

Керівник проєкту _____ Анатолій ГАЙДАЙ
(підпис)

## АНОТАЦІЯ

У межах даної роботи здійснено розробку мобільного застосунку для електронної бібліотеки.

Метою роботи є створення зручного інструменту для читання, зберігання та організації електронних книг на мобільному пристрої. У процесі реалізації було проаналізовано існуючі популярні рішення, виявлено їхні сильні та слабкі сторони, після чого сформовано вимоги до функціональності власного додатку.

У застосунку реалізовано можливість перегляду EPUB-файлів, збереження прогресу читання, додавання книг до особистої бібліотеки та сортування за категоріями. Додаток орієнтований на операційну систему Android, для його реалізації було використано мову програмування Kotlin.



Ключові слова: мобільний застосунок, електронна бібліотека, EPUB, Android, Kotlin.



## ANNOTATION

Within the scope of this work were made the development of a mobile application for an electronic library.

The aim of the work is to create a convenient tool for reading, storing, and organizing electronic books on a mobile device. During the development process, existing popular solutions were analyzed, their strengths and weaknesses identified, and functional requirements for the custom application were defined.

The application provides functionality for viewing EPUB files, saving reading progress, adding books to a personal library and sorting by categories. The app is designed for the Android operating system and was developed using the Kotlin programming language.



**Keywords**: mobile application, electronic library, EPUB, Android, Kotlin.

| справки | Формат | *Значення* | *Найменування* | Кіл. листів | № екземпля | *Додаток* |
|---|---|---|---|---|---|---|
| | | | Документація загальна | | | |
| | | | Знову розроблена | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.002 ТЗ*** | Мобільний застосунок для | 4 | | |
| | | | електронної бібліотеки | | | |
| | | | Технічне завдання | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.003 ПЗ*** | Мобільний застосунок для | 63 | | |
| | | | електронної бібліотеки | | | |
| | | | Пояснювальна записка | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.004 ДА*** | Мобільний застосунок для | 1 | | |
| | | | електронної бібліотеки | | | |
| | | | Структурна схема системи | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.005 ДБ*** | Мобільний застосунок для | 1 | | |
| | | | електронної бібліотеки | | | |
| | | | Функціональна схема | | | |
| | | | (ER діаграма) | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.006 ДВ*** | Мобільний застосунок для | 1 | | |
| | | | електронної бібліотеки | | | |
| | | | Принципова схема | | | |
| | | | (UML діаграма прецедентів) | | | |
| | | | | | | |
| | ***А4*** | ***ІАЛЦ.467200.007 ДГ*** | Мобільний застосунок для | 30 | | |
| | | | електронної бібліотеки | | | |
| | | | Текст програмного коду | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ***ІАЛЦ.467200.001 ОА*** | | |
| ***Зм*** | ***Лист*** | ***№ докум.*** | ***Підп*** | ***Дата*** | | | |
| ***Розроб*** | | Щегель А. С. | | | *Мобільний додаток для електронної бібліотеки* Опис альбому | Літ. | Аркуш 1 / Аркушів 1 |
| ***Перев*** | | Гайдай А. Р. | | | | | |
| | | | | | | ***КПІ ім. Ігоря Сікорського ФІОТ ІМ-11*** | |

# ТЕХНІЧНЕ ЗАВДАННЯ
# ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Мобільний застосунок для електронної бібліотеки*»

Київ – 2025

# Зміст



| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467200.003 ТЗ | | |
| Зм. | Арк. | № докум. | Підпис | Дата | | | |
| Розробив | | Щегель А. С. | | | **Мобільний застосунок для електронної бібліотеки** Технічне завдання | Літ. | Аркуш 1 / Аркушів 4 |
| Перевірив | | Гайдай А. Р. | | | | | |
| Реценз. | | Шимкович В. М. | | | | КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11 | |
| Н. Контр. | | Нікольский С. С. | | | | | |
| Затвердив | | | | | | | |

# 1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

Дане технічне завдання стосується розробки мобільного застосунку для пошуку та читання електронних книг.

Область застосування: застосунок призначений для людей, які полюбляють читати книги у вільний час та шукають зручний цифровий інструмент для доступу до електронної бібліотеки.

# 2 ПРИЧИНИ ДЛЯ РОЗРОБКИ

Підставою для розробки даного продукту є завдання для виконання бакалаврського проєкту на отримання освітньо-кваліфікаційного рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки”, кафедрою обчислювальної техніки, Національного технічного Університету України “Київський Політехнічний Інститут імені Ігоря Сікорського”.

# 3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного проєкту є розробка який забезпечить зручний доступ до електронної бібліотеки, з можливістю пошуку, читання та організації книг, орієнтованого на потреби сучасних користувачів.

# 4 ДЖЕРЕЛА РОЗРОБКИ

У процесі розробки планується використовувати такі джерела:

- Офіційна технічна документація відповідних технологій
- Науково-технічна література
- Профільні публікації, ресурси та статті в мережі Інтернет

# 5 ТЕХНІЧНІ ВИМОГИ

## 5.1 Вимоги до розробленого продукту

- Інтуїтивно зрозумілий інтерфейс для користувачів.
- Можливість реєстрації та автентифікації користувачів.
- Зручна система пошуку книг (за жанрами, назвою, автором тощо).
- Можливість додавання книг до персональної бібліотеки.
- Підтримка закладок, нотаток, рейтингу та історії читання.
- Підтримка різних форматів електронних книг (наприклад, EPUB, PDF).
- Налаштування інтерфейсу читання (розмір шрифту, кольорова тема, орієнтація тощо).

## 5.2 Вимоги до програмного забезпечення

- Операційна система Android 8+
- Android API 26+

### 5.2 Вимоги до апаратної частини

- Оперативної пам’яті не менше 4 ГБ
- Доступ до інтернету

# 6 ЕТАПИ РОЗРОБКИ

| Назва етапів виконання | Термін виконання |
|---|---|
| Затвердження теми роботи | 27.01.2025-02.02.2025 |
| Вивчення та аналіз завдання | 03.02.2025-09.02.2025 |
| Розробка архітектури та загальної структури застосунку | 10.02.2025-16.03.2025 |
| Розробка структур окремих частин застосунку | 17.03.2025-23.03.2025 |
| Програмна реалізація системи | 24.03.2025-11.05.2025 |
| Виправлення помилок | 12.05.2025-29.05.2025 |
| Оформлення пояснювальної записки | 27.01.2025-02.02.2025 |

# ПОЯСНЮВАЛЬНА ЗАПИСКА
# ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «Мобільний застосунок для електронної бібліотеки»

Київ – 2025

# Зміст

# ПЕРЕЛІК СКОРОЧЕННЬ

| | |
|---|---|
| БД | База Даних |
| EPUB | (Electronic Publication) Електронне видання |
| PDF | (Portable Document Format) Формат портативних документів |
| SQL | (Structured Query Language) Мова структурованих запитів |
| NoSQL | (Not only SQL) Не лише SQL |
| MVVM | (Model-View-ViewModel) Модель, Подання, Модель-Подання |

# ВСТУП

У сучасному цифровому світі мобільні телефони вже давно перестали бути просто засобом комунікації – сьогодні це універсальні інструменти для роботи, навчання, розваг і саморозвитку. Однією з важливих сфер їх використання є читання електронних книг. Завдяки стрімкому розвитку технологій і постійно зростаючій популярності електронних видань, усе більше користувачів надають перевагу саме цифровим форматам книжок. Це зумовлено не лише зручністю користування, а й мобільністю, економією простору, можливістю швидкого доступу до величезної кількості літератури незалежно від місця перебування читача.

Електронні книжки дозволяють мати при собі цілу бібліотеку у кишені, а з розвитком мобільних застосунків процес читання став ще комфортнішим. Проте, незважаючи на велику кількість уже існуючих програм, багато з них мають низку обмежень або не повною мірою відповідають потребам українських користувачів. Тому зростає необхідність у розробці нового програмного продукту, який би поєднував у собі широкий функціонал, зручність, локалізацію та підтримку найпопулярніших форматів електронних книг.

Дана дипломна робота присвячена створенню сучасного мобільного застосунку для читання електронних книг. Основною цільовою аудиторією продукту є користувачі, які прагнуть мати доступ до улюбленої літератури у будь-який момент – вдома, у транспорті, на відпочинку чи в навчальному процесі. Розроблений застосунок має забезпечити не лише можливість комфортного читання, а й низку додаткових функцій: пошук книжок за жанрами, назвами чи авторами, створення персональної бібліотеки, збереження прогресу читання, додавання закладок, зміна тем оформлення тощо.

Крім того, важливим аспектом є інтеграція з хмарними сервісами для синхронізації даних користувача між різними пристроями. Проєкт також має можливість для  подальшого розширення функціоналу відповідно до потреб користувачів.

Таким чином, реалізований застосунок має на меті не лише забезпечити базову функціональність для читання книжок, а й створити приємний, доступний та ефективний цифровий простір для взаємодії з літературою в електронному форматі.

# РОЗДІЛ 1. ОГЛЯД МОБІЛЬНИХ ЗАСТОСУНКІВ ДЛЯ ЧИТАННЯ ЕЛЕКТРОННИХ КНИГ

## 1.1 Загальна інформація

Зараз на ринку мобільних застосунків багато різних варіантів для комфортного читання електронних книг. Кожен з них намагається зробити все, щоб користувачам було зручно використовувати їхню платформу. Вони різняться за функціональністю, інтерфейсом і типом користувачів, на яких вони орієнтовані.

Ці застосунки можна умовно розділити на три категорії:

1. Застосунки з власною БД для книг, у яких книги можуть бути безкоштовними, доступними за підпискою або ж вимагати придбання. Ця категорія застосунків найбільш придатна для створення застосунку для електронної бібліотеки. Прикладами таких сервісів є Google Play Books, Amazon Kindle, а також український аналог – Librarius [1].

2. Платформи з певними елементами соціалізації, які поєднують функції електронної бібліотеки з можливістю взаємодії між читачами. На прикладі додатку Goodreads можна показати відмінність між цими категоріями, додаток Goodreads не має вбудованого інтерфейсу для читання книг, але є дуже розвинутою платформою для обговорення різних книг, створення відгуків, організації книг по “полицям” (за категоріями). Іншим прикладом є Wattpad, де при читанні книги можна до кожного речення залишати коментарі, що можуть бачити інші користувачі та автор книги [1].

3. Застосунки для читання локальних файлів, які не мають своєї бібліотеки книг, але дозволяють відкривати файли різних форматів (наприклад, EPUB, PDF) з пристрою користувача, та дають доступ до сторонньої безкоштовної бібліотеки. Для прикладу візьмемо PocketBook та

Moon+ Reader, вони є найкращим вибором з цієї категорії. Вони сканують файлову систему пристрою та дають можливість користувачу переглядати файли, типи яких вони підтримують, Найбільшою їх перевагою є великий вибір налаштуваннь сторінки перегляду книги, та різний функціонал, який робить перегляд книг набагато комфортнішим [1].

Кожна з трьох категорій застосунків має сильні та слабкі сторони, що створює можливість для розвитку різних застосунків, що поєднують різні функції цих категорій.

## 1.2 Існуючі рішення

Давайте детальніше розглянемо приклади наведених раніше мобільних додатків з різних категорій, розглянемо їхні позитивні та негативні сторони, та виділимо їхні особливості.

### 1.2.1 Google Play Books

Google Play Books є одним з найкращих виборів для читання електронних книг. Його популярність походить від зручного інтерфейсу та великої кількості доступної літератури. Застосунок дає змогу купувати, завантажувати, зберігати та читати книги з великої онлайн-бібліотеки, що налічує мільйони електронних книг [2].

Варто звернути особливу увагу на великий вибір налаштувань для кастомізації інтерфейсу для читання та можливість робити нотатки. Читач може змінювати розмір та стиль шрифту, також додаток має на вибір три теми, денну, нічну та сепію.

Для допомоги користувачу в знаходженні потрібного йому контенту, Google Play Books має великий вибір налаштувань для фільтрування книг, який можна побачити на рис. 1.1

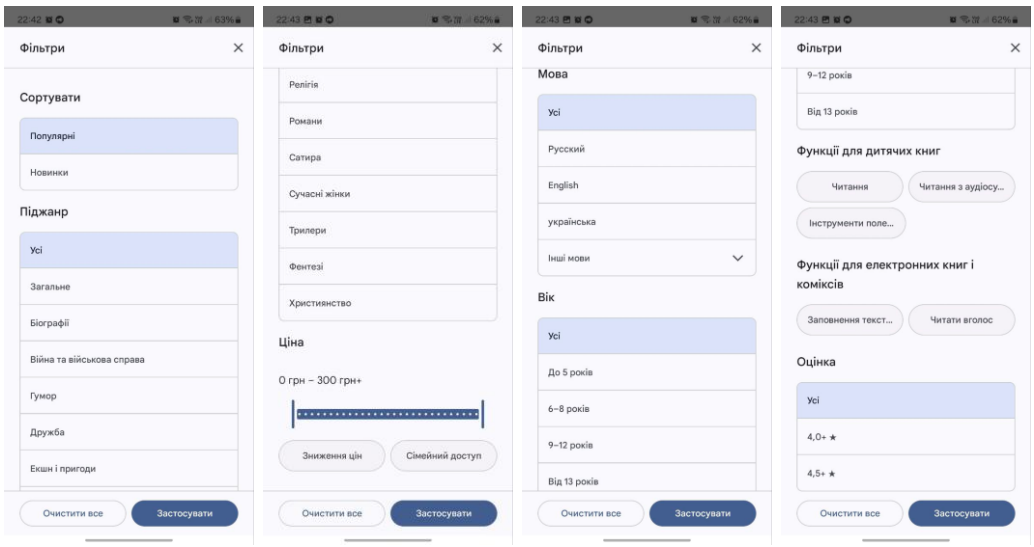



Рисунок 1.1 – Інтерфейс фільтрування книг в Google Play Books

Тепер перейдемо до недоліків. Google Play Books дозволяє фільтрувати книги за жанрами, мовами, рейтингом, але інтерфейс пошуку може здаватися складним або перевантаженим для користувача. Також може бути не зрозуміло, як потрапити до сторінки фільтрування по жанрам, так як на це немає виділеного місця.

Ще одним вагомим недоліком є комерціалізація контенту, який повинен бути у вільному доступі. Навіть твори, що належать до культурної спадщини та є загальним надбанням народу, такі як класичні українські книжки – наприклад, збірки Тараса Шевченка доступні лише за окрему плату. Це створює перешкоди для широкого доступу до національної літератури та суперечить принципам відкритої освіти й поширення культури.

### 1.2.2 Amazon Kindle

Amazon Kindle має незаперечну репутацію серед користувачів, він стоїть на одному рівні з Google Play Books та також має незліченну кількість книжок в своїй БД. Основна його привабливість для користувачів, це те що він є частиною великої екосистеми Amazon і служить гарним вибором для користувачів цієї системи [3].

Застосунок надає доступ до величезної бібліотеки, яка охоплює мільйони найменувань літератури для всіх категорій користувачів. Залежно від уподобань користувача, Kindle пропонує оформити підписку Kindle Unlimited, або купувати книги окремо. Великим недоліком цієї системи, є те, що безкоштовного доступу до книг немає. Без підписки або покупки, доступ до більшості контенту обмежений [3].

Ще одним важливим обмеженням є відсутність локалізації українською мовою. Інтерфейс Kindle наразі не підтримує українську, що може викликати труднощі у користувачів, які не володіють англійською або іншими доступними мовами. Крім того, вибір україномовних книг у Kindle Store є досить обмеженим, що значно знижує привабливість застосунку для українських читачів.

Щодо зручності користування, інтерфейс Kindle, хоча й функціональний, не є зручним для користування. Пошук, сортування та фільтрація книг не реалізовані на належному рівні, що ускладнює навігацію при наявності великої кількості збережених видань. Це може бути незручним для активних читачів, які регулярно додають нові книги до своєї колекції.

Рисунок 1.2 показує головний екран застосунку Kindle після входу користувача. Інтерфейс акцентує увагу на бібліотеці користувача та рекомендованих книгах. Угорі розташовані кнопки навігації: пошук, профіль і магазин. Середина екрана показує останні прочитані або завантажені книги у вигляді плиток із обкладинками.

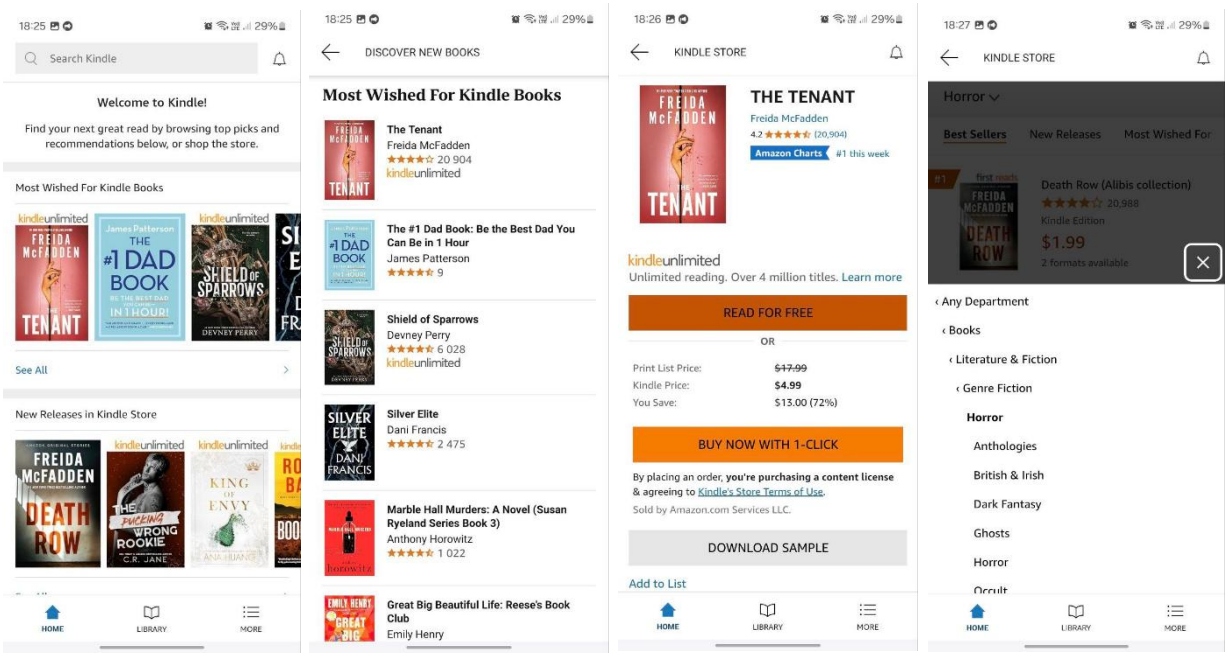



Рисунок 1.2 – Інтерфейс Amazon Kindle

### 1.2.3 Librarius

*Librarius* – це сучасний український аналог для читання електронних та аудіо книг, який пропонує доступ до великої бібліотеки українськомовної літератури. Платформа охоплює широкий спектр жанрів, таких як художня, література, класика, нон-фікшн, тощо, що дозволяє задовольнити інтереси різних категорій користувачів – від підлітків до дорослої аудиторії [4].

Цей застосунок має велику кількість книг українських авторів та українською мовою, що задовільняє потреби користувача.

Також при аналізі було виявлено велику кількість навчальної літератури, а також спеціальний розділ з нею, що істотно розширює цільову аудиторію платформи, включаючи до неї учнів шкільних навчальних закладів.

Для забезпечення якомога точнішої оцінки для книг, платформа використовує рейтинги з Goodreads для оцінювання книг.

Монетизація на платформі є досить гнучкою, для читання книг, користувач має декілька варіантів оплати, купити книгу повністю, орендувати

книгу на 14 днів за пів ціни чи заплатити за щомісячну підписку, завдяки чому можна читати більшість книг в наявності без потреби купувати кожну книгу окремо. Така варіативність дозволяє адаптувати витрати відповідно до потреб та фінансової спроможності читача.

При аналізі функціональності було виявлено певні недоліки в організації особистої бібліотеки. Інтерфейс управління прочитаними книгами не є інтуїтивним та має обмежені можливості для структурування колекції.

Рисунок 1.3 показує головний екран додатку Librarius. Інтерфейс сучасний, з яскравими візуальними елементами та чіткою навігацією. Головний банер містить рекомендації, а нижче – добірки книг за жанрами і вподобаннями. Для мобільних користувачів передбачено великі обкладинки, плавну прокрутку та навігаційну панель внизу з кнопками до головної сторінки, бібліотеки, пошуку та налаштувань. Librarius повністю локалізований українською мовою й акцентує на українських авторах та перекладах [4].

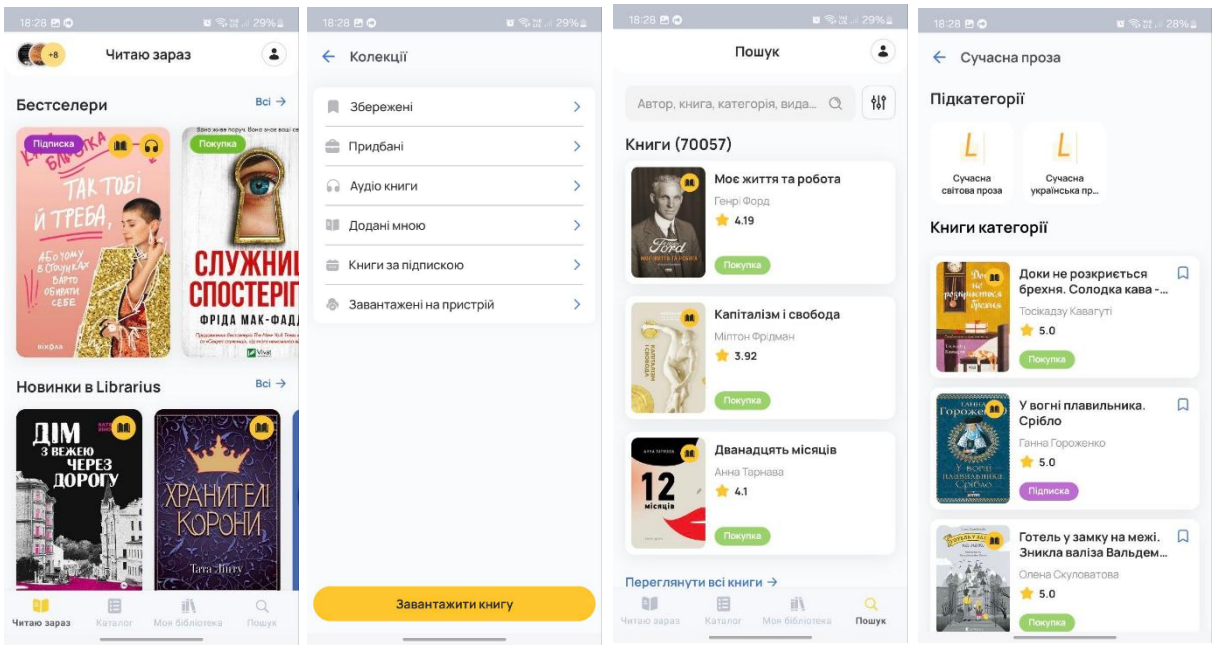



Рисунок 1.3 – Інтерфейс Librarius

### 1.2.4 Wattpad

Мобільний застосунок Wattpad – це унікальна платформа для читання та написання історій, яка поєднує у собі функціонал електронної бібліотеки та соціальної мережі для авторів і читачів [5].

Сервіс орієнтований на глобальну спільноту користувачів, даючи змогу не лише читати вже опубліковані твори, але й самостійно створювати, редагувати та поширювати власні історії у різних жанрах. Завдяки інтерактивності платформи, читачі можуть коментувати тексти, залишати вподобайки, ділитися улюбленими творами та вступати у діалог з авторами.

Особливу популярність Wattpad здобув серед молодіжної аудиторії, зокрема через велику кількість фанфіків, романів, фентезі та пригодницької літератури, які часто відображають сучасні теми. Багато авторів розпочали свій творчий шлях саме на Wattpad, а деякі з їхніх творів згодом були адаптовані до друкованих видань або навіть екранізовані [5].

Серед зручностей додатку – можливість створення власної бібліотеки, підписки на авторів, збереження прогресу читання та формування колекцій улюблених історій. Проте існують і певні обмеження: велика частина контенту доступна лише англійською мовою, а україномовний сегмент залишається малорозвиненим.

Крім того, внаслідок відкритого характеру публікацій, які не проходять професійної редактури, значна частина матеріалу може відрізнятися за рівнем якості та стилістики. Ще одним суттєвим недоліком є відсутність підтримки завантаження та перегляду власних файлів у форматах EPUB чи PDF, що робить платформу менш придатною для тих, хто хоче читати вже наявні книжки зі свого пристрою.

На рисунку 1.4 зображено головний екран застосунку Wattpad. Інтерфейс має сучасний і водночас інтуїтивно зрозумілий дизайн: користувач бачить добірку рекомендованих історій, останні прочитані твори,

персоналізовані колекції та пропозиції за жанрами. У верхній частині екрана розміщено поле пошуку, а внизу – панель навігації, що дозволяє швидко переходити між головною сторінкою, бібліотекою, списком підписок та профілем. Такий підхід забезпечує легкий доступ до основних функцій платформи та підвищує зручність користування додатком.

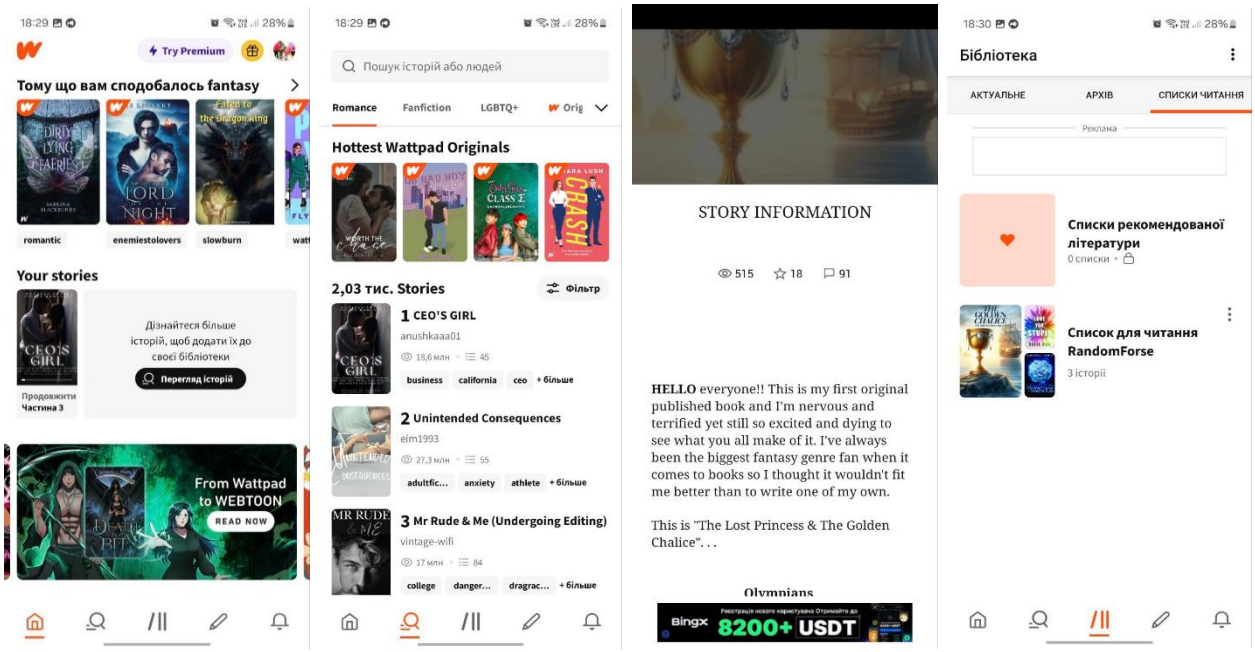



Рисунок 1.4 – Інтерфейс Wattpad

### 1.2.5 PocketBook Reader

PocketBook Reader – це універсальний мобільний застосунок для читання електронних книг, який підтримує широкий спектр форматів (EPUB, PDF, FB2, MOBI, TXT тощо) та орієнтований на зручність і гнучкість у користуванні. Цей застосунок особливо популярний серед користувачів, які володіють електронними рідерами PocketBook, але він доступний також на Android і iOS-пристроях незалежно від моделі пристрою [6].

Однією з ключових переваг PocketBook Reader є потужні інструменти для персоналізації процесу читання: користувачі можуть змінювати шрифти, колірну схему, інтерліньяж, яскравість, орієнтацію екрану та багато іншого. Реалізовано підтримку нічного режиму, текстового-to-мовного (TTS)

озвучення, синхронізації позиції читання через хмару PocketBook Cloud, а також можливість роботи з нотатками та закладками [6].

Застосунок дозволяє як відкривати локальні файли, так і синхронізувати бібліотеку з хмарними сховищами (Dropbox, Google Drive), завантажувати книги через OPDS-каталоги. Це робить його зручним інструментом як для читання вже наявних книжок, так і для отримання нових [6].

Однак серед недоліків можна назвати не зовсім інтуїтивний інтерфейс для нових користувачів, а також те, що деякі функції, зокрема синхронізація та аудіофункції, вимагають створення облікового запису PocketBook.

На рисунку 1.5 показано основний інтерфейс PocketBook Reader після відкриття книги. Видно гнучкі налаштування читання, інструменти для навігації та панель доступу до інструментів бібліотеки та змісту.

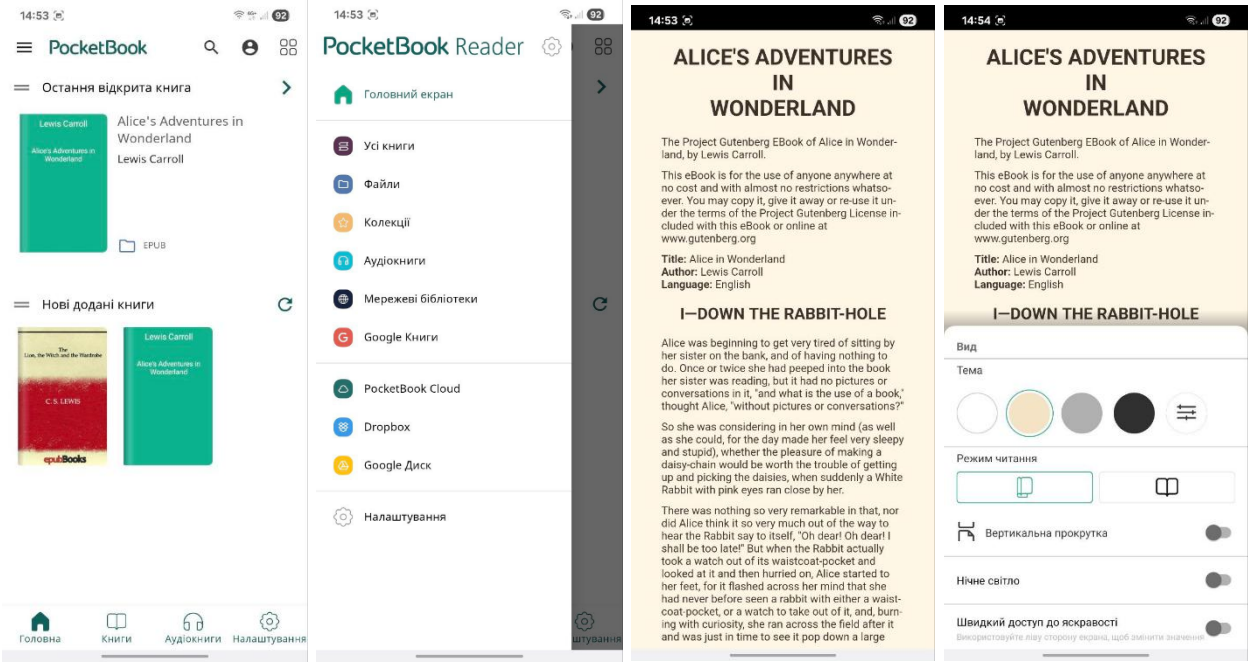



Рисунок 1.5 – Інтерфейс PocketBook Reader

## 1.3 Порівняльний аналіз

Метою цього проєкту є розробка мобільного застосунку для читання електронних книг, який буде зручним, зрозумілим та функціонально насиченим середовищем для взаємодії користувачів з літературним контентом. Відповідно до цього, будуть визначені основні недоліки цих застосунків. У таблиці 1.1 буде представлено порівняння функціональних характеристик цих застосунків.

Таблиця 1.1 – Порівняння функціоналу застосунків для читання

| Характеристика | Підтримка EPUB, PDF | Налаштування зовнішнього вигляду | Особисті бібліотеки / колекції | Соціальна взаємодія (коментарі) |
|---|---|---|---|---|
| Google Play Books | Так | Так | Так | Ні |
| Amazon Kindle | Так | Так | Так | Ні |
| Librarius | Так | Обмежене | Так | Ні |
| Wattpad | НІ | Обмежене | Так | Так |
| PocketBook Reader | Так | Так | Так | Ні |

Однак функціональні характеристики – це лише один аспект користувацького досвіду. Не менш важливими є аспекти локалізації, доступність контенту українською мовою, наявність безкоштовних книг. У таблиці 1.2 ці аспекти буде розглянуто детально, що дозволить зробити завершений висновок щодо відповідності застосунків потребам українських користувачів.

Таблиця 1.2 – Порівняння інших аспектів застосунків для читання

| Характеристика | Наявність українського інтерфейсу | Наявність книжок українською мовою | Наявність безкоштовного контенту |
|---|---|---|---|
| Google Play Books | Так | Є вибір, але він обмежений | Частково, є покупки |
| Amazon Kindle | Ні | Ні | Ні |
| Librarius | Так | Великий вибір контенту | Частково, є підписка |
| Wattpad | Частково | Є але дуже мало | Так, але присутня реклама |
| PocketBook Reader | Так | Лише файли з пристрою користувача | Повністю безкоштовний |

Сучасні мобільні застосунки для читання, хоч і пропонують велику функціональність користувачу, мають низку поширених проблем, що погіршують користувацький досвід:

- Складний або непрозорий процес додавання книг із локальної пам’яті пристрою, що вимагає від користувача додаткових дій, реєстрацій або використання сторонніх сервісів (наприклад, електронної пошти чи веб інтерфейсів).
- Недостатньо гнучка система пошуку та фільтрації книг, що ускладнює навігацію великою бібліотекою та не дозволяє швидко знайти потрібну літературу за жанром, автором чи іншими критеріями.
- Обмежений контент українською мовою, що суттєво зменшує доступність літератури для україномовної аудиторії.

- Недостатній безкоштовний доступ до книжок – більшість сервісів пропонують або обмежену кількість безкоштовного контенту, або функціонують за платною підписною моделлю.

У відповідь на ці проблеми було прийнято рішення розробити власний мобільний застосунок, який поєднуватиме в собі найкращі риси існуючих рішень, водночас усуваючи їхні основні недоліки.

Пропонований застосунок орієнтується на створення повноцінної платформи для зручного та комфортного читання, яка включатиме:

- Розширену систему пошуку та фільтрації книг за жанрами, мовою, роком випуску, автором тощо.
- Підтримку локального завантаження файлів EPUB без необхідності використовувати сторонні сервіси, з автоматичним додаванням книг до внутрішньої бібліотеки.
- Можливість повної кастомізації процесу читання, зокрема налаштування шрифтів, кольорів, міжрядкового інтервалу, режиму відображення (денний/нічний), що дозволяє адаптувати інтерфейс під індивідуальні потреби користувача.
- Фокус на україномовну аудиторію - застосунок буде мати повну українську локалізацію та забезпечить підтримку україномовного контенту.
- Простий, сучасний та зручний інтерфейс, що забезпечить швидке освоєння функціоналу без необхідності витрачати час на навчання або пошук базових функцій.

Таким чином, мобільний застосунок не лише задовольняє базову потребу у зручному читанні електронних книг, а й покращує якість взаємодії користувача з книжковим контентом завдяки широким функціональним можливостям, простоті використання та фокусу на локалізованому, доступному сервісі.

# ВИСНОВОК ДО РОЗДІЛУ 1

У даному розділі було здійснено всебічний аналіз основних типів мобільних застосунків, призначених для читання електронних книг. Зокрема, були визначені три ключові категорії таких застосунків:

1. Застосунки з великою вбудованою БД книг, які зазвичай орієнтовані на комерційні моделі, надають доступ до масштабних онлайн-бібліотек і функціонують за принципом купівлі або підписки;
2. Платформи, що дозволяють соціальну взаємодію між користувачами, створюючи своєрідне ком'юніті авторів і читачів, де основним контентом є твори, згенеровані користувачами;
3. Застосунки, що не мають власної онлайн-бібліотеки, але дозволяють завантаження та перегляд книг, що вже зберігаються на пристрої користувача, – такі рішення зазвичай зосереджені на зручності читання локального контенту.

Для кожної з цих категорій були розглянуті найпопулярніші приклади застосунків, проаналізовані їхні функціональні можливості, переваги й обмеження. Порівняльна характеристика надала змогу виявити не лише загальні риси, притаманні подібним продуктам, а й специфічні недоліки, які мають місце у кожному окремому випадку.

Було виділено декілька основних проблем, характерних для сучасних рішень у сфері мобільного читання:

Більшість існуючих рішень мають недостатню підтримку української мови для інтерфейсу та невелику кількість контенту українською мовою, що є великою незручністю для користувача, на якого націлений цей застосунок.

Результати дослідження мобільних застосунків на ринку, показали необхідність розробки більш інтуїтивно зрозумілих механізмів пошуку та сортування за категоріями в особистій колекції книг, що дозволили б

користувачам ефективно управляти бібліотекою, без складних налаштувань та тривалого процесу освоєння інтерфейсу.

Обмежена можливість додавання локальних файлів до бібліотеки також є недоліком в цих застосунках, так як цей функціонал важливим для кращого досвіду користування застосунком у користувача.

У результаті проведеного аналізу сформовано чітке уявлення про напрями, в яких доцільно вдосконалити функціональність та інтерфейс майбутнього програмного продукту.

Виявлені недоліки лягли в основу технічного завдання, яке визначає ключові критерії та вимоги до розробки нового застосунку. Головна мета цього продукту - створити сучасний, зручний і доступний інструмент для комфортного читання електронних книг, що максимально відповідатиме потребам українських користувачів.

# РОЗДІЛ 2. ВИБІР ТЕХНОЛОГІЙ РОЗРОБКИ

## 2.1 Мова програмування та середовище розробки

У процесі аналізу мов програмування, які працюють для Android, було прийнято рішення використовувати мову програмування Kotlin. Ця мова вважається покращенням на основі Java, що дозволяє розробникам створювати застосунки швидше, якісніше та з меншою кількістю помилок. Вона є найкращим вибором для розробки під платформу Android, через це було вибрано саме цю мову програмування для реалізації сучасного мобільного застосунку.

Kotlin поєднує в собі простоту та виразність синтаксису, що дозволяє писати менше коду без втрати функціональності. Завдяки цьому розробник може зосередитися на логіці застосунку, а не на зайвих технічних деталях [7].

Головною перевагою Kotlin в порівнянні з Java є безпечність у роботі з null-значеннями, що значно знижує ризик виникнення помилок під час виконання програми. Окрім того, мова повністю сумісна з Java, що дозволяє використовувати вже існуючі Java-бібліотеки, а також поєднувати обидві мови в одному проєкті, поступово мігруючи на сучасніший синтаксис Kotlin [7].

## 2.2 Середовище розробки

Для розробки застосунку використовується Android Studio, яка забезпечує весь необхідний функціонал для створення, компіляції, налагодження та тестування Android-застосунків.

Android Studio підтримує Kotlin за замовчуванням, має вбудований емулятор, що значно пришвидшує процес ручного тестування продукту.

Завдяки інтеграції з Gradle, Android Studio дозволяє ефективно керувати залежностями, що робить розробку застосунку зручнішою.

На рис. 2.1 можна побачити інтерфейс Android Studio з відкритим емулятором

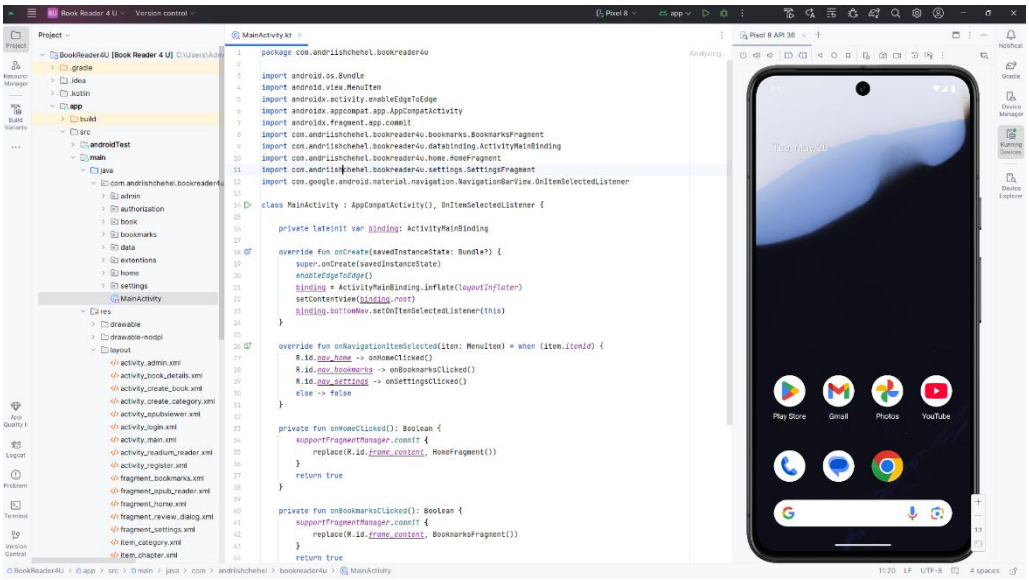


Рисунок 2.1 – Інтерфейс Android Studio

## 2.3 Робота з електронними книгами

Для надання доступу користувачу до перегляду електронних книг у мобільному застосунку було розглянуто декілька варіантів бібліотек для роботи з файлами типу EPUB.

Чому саме EPUB? Тому що це на даний момент найзручніший та найпоширеніший формат для зберігання контенту електронних книг з поміж інших (PDF, FB2 тощо).

Давайте переглянемо декілька варіантів бібліотек для роботи з файлами формату EPUB

### 2.3.1 Порівняння бібліотек для роботи з EPUB

Readium Kotlin Toolkit - потужний набір бібліотек з відкритим вихідним кодом, спеціально створений для роботи з електронними книгами коміксами, аудіо книгами написаний на Kotlin, що є великою перевагою.

Readium Kotlin Toolkit є сучасним та активно підтримуваним проєктом, що дозволяє рендерити EPUB-файли без необхідності самостійно розробляти низькорівневу логіку відображення тексту та структури книги.

Наявність в цій бібліотеці можливості не тільки обробляти файли в EPUB форматі, а й презентувати зміст файлів користувачу, та налаштовувати середовище для перегляду контенту користувачем, це величезний плюс на користь вибору цього фреймворку як основного для нашого завдання.

FolioReader - доволі зручна для роботи з EPUB файлами бібліотека. Інтеграція цієї бібліотеки досить проста і зазначена в документації. Говорячи про документацію, документація є неповною, деякі функції бібліотеки не описані взагалі, що ускладнює інтеграцію.

FolioReader як і Readium має функціонал для відтворення контенту користувачу, але тут він не є повним, наприклад немає функціоналу для відображення кількості сторінок, що залишилось читати.

FolioReader підтримує лише формат EPUB 2.0, підтримка інших форматів (EPUB 3.0 чи PDF) не присутня.

Ще одним негативним моментом є відсутність активної підтримки, проєкт вже давно не підтримується, що робить його не дуже доречним у використанні для створення нових застосунків.

epub4j – бібліотека для парсингу EPUB файлів, вона базується на бібліотеці epublib, покращуючи її функціонал та використовуючи більш сучасний підхід.

Ця бібліотека дозволяє зчитувати структуру EPUB книг, включаючи маніфест, зміст, метадані тощо. Вона доволі зручна для попереднього парсингу та аналізу контенту EPUB файлів перед виведенням на екран.

epub4j має лише функціонал парсингу та обробки EPUB файлів. Вона не виводить зміст файлу на екран, тому для перегляду змісту, потрібно реалізовувати власну систему відображення.

Бібліотека працює на низькому рівні, що потребує заглиблення в структуру EPUB, що ускладнює роботу для розробника. При такій роботі бібліотеки, вона має застарілу документацію, що аж ніяк не допомагає в подоланні цих труднощей.

Таблиця 2.1 – Порівняння роботи бібліотек

| Характеристика | Readium Kotlin Toolkit | FolioReader | epub4j |
|---|---|---|---|
| Підтримка EPUB3 | Присутня | Обмежена | Обмежена |
| Дата останнього коміту в GitHub | 15.05.2025 | 05.01.2020 | 05.12.2024 |
| Складність інтеграції | Складна | Проста | Середня |
| Підтримка візуального рендерера | Має власний рендерер | Має простий рендерер | Немає власного рендереру |

Хоча існує декілька рішень для роботи з EPUB-файлами, Readium Kotlin Toolkit вирізняється серед них оптимальним поєднанням функціональності та підтримки сучасних стандартів. На відміну від бібліотек, які лише парсять EPUB, або застарілих проєктів з обмеженою підтримкою (як FolioReader), Readium пропонує повноцінне, стабільне й масштабоване рішення для реалізації сучасного інтерфейсу читання.

## 2.4 Хмарне сховище та аутентифікація

Для забезпечення надійного зберігання даних користувачів, організації доступу до контенту та синхронізації інформації між різними пристроями було обрано платформу Firebase від компанії Google. Firebase є потужним бекенд-рішенням, яке інтегрується з Android-застосунками і пропонує широкий набір сервісів для автентифікації, зберігання, аналітики та обробки даних у реальному часі.

У межах розробки даного мобільного застосунку використовуються такі ключові сервіси Firebase:

### 2.4.1 Firebase Authentication

Забезпечує безпечну систему реєстрації та входу для користувачів. Платформа підтримує автентифікацію за допомогою електронної пошти та пароля, Google-акаунтів, Facebook, а також анонімний вхід. У нашому випадку реалізовано автентифікацію через електронну пошту, що є зручною та звичною для більшості користувачів. Firebase автоматично керує сесіями користувачів, спрощуючи реалізацію авторизації на стороні клієнта [8].

### 2.4.2 Firebase Cloud Firestore

Це масштабоване NoSQL-сховище, яке використовується для збереження структурованих даних у форматі документів і колекцій. У застосунку Firestore використовується для зберігання:

- метаданих про книги (назва, автор, жанр, дата додавання тощо);
- інформації про користувацький прогрес читання;
- списків улюблених книг та інших персоналізованих даних.

Firestore підтримує Kotlin corutones, що дозволяє при оновленні бази даних, миттєво синхронізувати дані між усіма пристроями, на яких авторизований користувач [9].

### 2.4.3 Firebase Storage

Даний сервіс призначений для зберігання великих файлів, зокрема електронних книг у форматі EPUB та різних зображень (наприклад обкладинка книги чи фото профілю). Застосунок також має можливість надавати спільний доступ до файлів за допомогою генерованих URL-посилань.

Завдяки інтеграції Firebase, досягається висока гнучкість, масштабованість та безпечність роботи з персональними даними користувача. Це рішення дозволяє уникнути необхідності створення власного серверного API, що значно скорочує час розробки та спрощує підтримку застосунку [10].

## 2.5 Архітектура застосунку

Проведемо аналіз двох найпопулярніших архітектур для розробки мобільних застосунків – MVVM (Model-View-ViewModel) та MVP (Model-View-Presenter). Обидва підходи спрямовані на розділення відповідальностей між компонентами, полегшення тестування та покращення структури коду. Однак вони мають суттєві відмінності в реалізації зв’язку між логікою та інтерфейсом користувача.

Архітектура MVVM (Model-View-ViewModel) - популярний шаблон у сучасній Android-розробці. Це офіційно підтримується Google і інтегрується з компонентами Android Jetpack, такими як ViewModel, LiveData, Data Binding і Kotlin Coroutines. MVVM спрощує управління станом інтерфейсу, підвищує

кількість повторно використаного коду, розмежує частини коду, для більш безпечної взаємодії між шарами та дає змогу тестувати застосунок [11].

Основні складові MVVM-архітектури:

- Model: відповідає за бізнес-логіку та доступ до даних. У нашому випадку це взаємодія з Firebase (Firestore, Storage, Authentication), а також сервіси, що обробляють EPUB-файли за допомогою Readium.
- View: це XML-інтерфейс користувача або фрагменти, які відображають дані. View підписується на зміни стану у ViewModel і реагує на них без прямої прив'язки до бізнес-логіки.
- ViewModel: виступає як проміжний шар між View і Model. ViewModel зберігає стан інтерфейсу, отримує дані з моделей та надає їх View у вигляді Flow. Також ViewModel обробляє дії користувача та передає їх до Model для подальшої обробки.

На рис 2.2 можна побачити схему архітектури MVVM, та яким чином розподіляються задачі всередині.

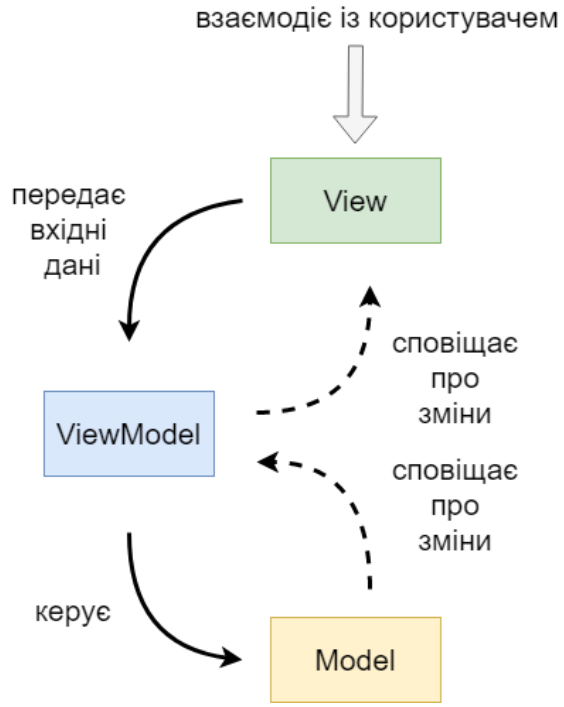



Рисунок 2.2 – Схема роботи MVVM архітектури

Архітектурний шаблон MVP (Model–View–Presenter) був одним з найпоширеніших підходів до структурування Android-додатків до появи Android Architecture Components. Основна мета MVP – забезпечити чітке розділення обов'язків між частинами застосунку, зменшити залежність між ними та спростити тестування [12].

Основні компоненти MVP:

- Model – відповідає за бізнес-логіку, обробку даних, доступ до API або бази даних. Цей компонент повністю ізольований від елементів інтерфейсу та не має уявлення про те, як ці дані відображаються.
- View – інтерфейс користувача. У контексті Android це зазвичай Activity, Fragment або кастомний View. View не містить логіки – її основна функція полягає у відображенні даних, отриманих від Presenter, та передачі дій користувача назад у Presenter через виклики методів.
- Presenter – посередник між View та Model. Presenter реалізує логіку взаємодії між даними та інтерфейсом. Він запитує дані з Model, обробляє їх і передає у View через спеціально визначений інтерфейс. Також Presenter обробляє події користувача, отримані від View. Важливо, що Presenter не має прямого доступу до Android SDK, що дозволяє ефективно його тестувати.

Після проведення аналізу архітектур, було вирішено створити порівняльну таблицю 2.2, яка дозволить чітко визначити ключові аспекти та особливості архітектурних рішень, їх переваги, недоліки та специфіку використання в мобільних застосунках. Такий підхід сприятиме кращому розумінню обґрунтованості вибору та допоможе у подальшій розробці.

Таблиця 2.2 – Порівняння архітектур MVVM та MVP

| Характеристика | MVVM | MVP |
|---|---|---|
| Розділення обов’язків | Високе, завдяки ViewModel, як посереднику між View та Model | Високе, Presenter керує логікою, View – лише візуальним відображенням |
| Обробка подій користувача | Через ViewModel, з подальшою передачею у Model | Обробляються безпосередньо у Presenter |
| Складність реалізації | Вища для початківців, потребує розуміння реактивності | Простіший для старту, але менш масштабований у великих проєктах |
| Зв’язок між View і логікою | Односторонній: ViewModel не знає про View | Двосторонній: Presenter знає про View через інтерфейс |
| Придатність до тестування | Висока, оскільки ViewModel не залежить від UI | Висока, але вимагає моків інтерфейсу View |
| Можливість повторного використання логіки | ViewModel забезпечує високу повторну використуваність між різними екранами. | Низька – Presenter зазвичай пов’язаний з конкретною View |

Для мобільного застосунку електронної бібліотеки було обрано архітектуру MVVM. Це рішення зменшить зв'язність компонентів, збільшить сумісність з технологіями Android (Kotlin, Coroutines) і покращить можливість тестування та розширення застосунку. Цей підхід забезпечує чітку структуру, полегшує процес розробки та покращує масштабованість.

# ВИСНОВОК ДО РОЗДІЛУ 2

У цьому розділі було обґрунтовано вибір основних технологій, інструментів та архітектурних рішень, що використовуються для розробки мобільного застосунку електронної бібліотеки. Зокрема, обрано мову Kotlin і середовище Android Studio як сучасний і зручний стек для створення Android-додатків. Для рендерингу електронних книг у форматі EPUB інтегровано Readium Kotlin Toolkit, що забезпечує широку функціональність для зручного читання.

Система зберігання даних реалізована на основі хмарної платформи Firebase, яка дозволяє легко організувати аутентифікацію, зберігання метаданих і самих книг. Для підтримки офлайн-режиму передбачено використання локальної бази даних Room. Архітектура проєкту базується на шаблоні MVVM, що сприяє зручному розділенню логіки, гнучкості та масштабованості застосунку. Обрані рішення забезпечують стабільну, зручну та ефективну роботу застосунку на різних пристроях.

# РОЗДІЛ 3. ОПИС РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі здійснюється детальний опис процесу створення програмного забезпечення, що реалізує функціональність електронної бібліотеки. На основі попереднього аналізу було обрано відповідні технології, інструменти та архітектурні підходи, які будемо використовувати при розробленні мобільного застосунку.

## 3.1 Застосування архітектури MVVM

У процесі розробки мобільного застосунку було прийнято рішення використати архітектурний шаблон Model–View–ViewModel (MVVM). Це рішення базується на результатах попереднього порівняльного аналізу архітектур, у якому MVVM показала себе як найбільш відповідна для реалізації нашого продукту. Архітектура MVVM дозволяє чітко відокремити логіку інтерфейсу від бізнес-логіки та доступу до даних, що значно покращує структурованість коду, полегшує налагодження, тестування й повторне використання компонентів.

У реалізації застосунку, архітектура MVVM здійснюється через розмежування коду на три основні рівні: Model, ViewModel та View.

Model містить репозиторії, які взаємодіють із джерелами даних, такими як Firebase Firestore, Storage та Authentication. Саме тут реалізовано логіку зчитування та запису даних, зокрема: отримання списку книг, завантаження обкладинок, перевірку прав доступу користувача, додавання рецензій тощо.

ViewModel виступає як посередник між логікою та інтерфейсом користувача. Цей компонент реалізує бізнес-логіку, опрацьовує дії користувача, формує дані для відображення та зберігає стан інтерфейсу,

завдяки використанню анотації @HiltViewModel, залежності до ViewModel передаються автоматично. Для передачі даних у View використовуються StateFlow, LiveData або suspend-функції з Kotlin Coroutines – залежно від задачі.

View – це компоненти, які безпосередньо відповідають за відображення інформації на екрані. У проєкті це реалізовано у вигляді Activity та Fragment, кожен з яких має свою чітко визначену відповідальність: головна сторінка, каталог книг, деталі книги, екран читання EPUB, написання рецензій тощо. View не містить бізнес-логіки, а лише реагує на зміни стану ViewModel та передає йому події користувача. Для доступу до елементів інтерфейсу застосовується View Binding, що запобігає помилкам на етапі компіляції.

Для впровадження залежностей у всіх шарах застосовано бібліотеку Hilt [13], яка є рекомендованим рішенням для Dependency Injection у Jetpack-екосистемі. Клас BookReader4UApplication позначено анотацією @HiltAndroidApp, що забезпечує ініціалізацію Hilt при старті програми. Усі залежності – репозиторії, сервіси, утиліти – надаються через відповідні Hilt-модулі, що розміщені у пакеті di.

Структура проєкту організована наступним чином:

- data – містить моделі, інтерфейси репозиторіїв, сервіси для взаємодії з Firebase, а також класи-джерела даних. Тут же знаходяться реалізації інтерфейсів з анотаціями @Inject.
- ui – представлений модулями для кожного екрану застосунку (наприклад, home, search, bookDetails, reader, library, review). Кожен модуль містить Fragment/Activity, ViewModel і пов’язані з ними ресурси. Така організація забезпечує ізоляцію відповідальностей та полегшує масштабування.
- di – включає Hilt-модулі (@Module, @InstallIn) для надання екземплярів Firebase, репозиторіїв, логерів, валідаційних класів тощо. Це дозволяє централізовано управляти конфігурацією залежностей.

- util – допоміжні класи, наприклад, розширення (extension functions), обробники помилок, конвертери даних тощо.

Така реалізація MVVM у поєднанні з Hilt забезпечує високу модульність, легке тестування, розширюваність функціональності та зменшення зв'язаності між компонентами. На рис. 3.1 представлено структуру проєкту, яка наочно демонструє відповідність коду архітектурним принципам, описаним вище.

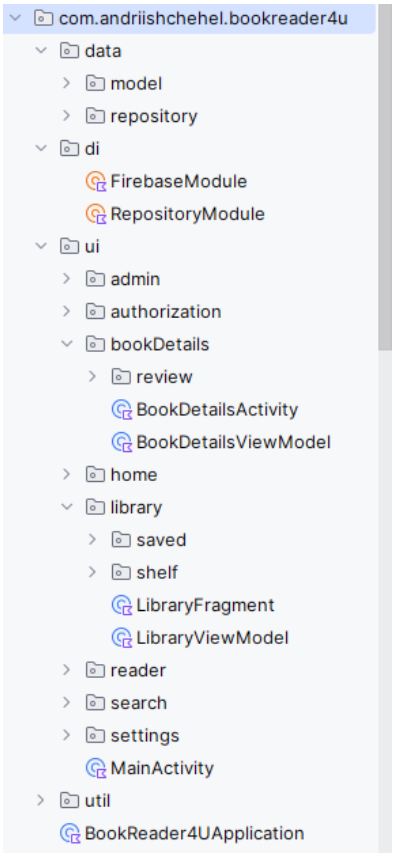



Рисунок 3.1 – Структура проєкту

## 3.2 База даних

Для зберігання даних у мобільному застосунку використовується хмарне сховище Firebase Firestore, яке забезпечує гнучку та масштабовану NoSQL-структуру, орієнтовану на документи. Основні дані представлені у вигляді колекцій та підколекцій документів, що дозволяє ефективно

організувати взаємозв’язки між об’єктами та забезпечити високий рівень продуктивності при читанні й записі даних.

Для представлення нашої бази даних, створимо ER-діаграму, яка демонструє основні сутності та зв’язки між ними. Ця діаграма представлена на рисунку 3.2

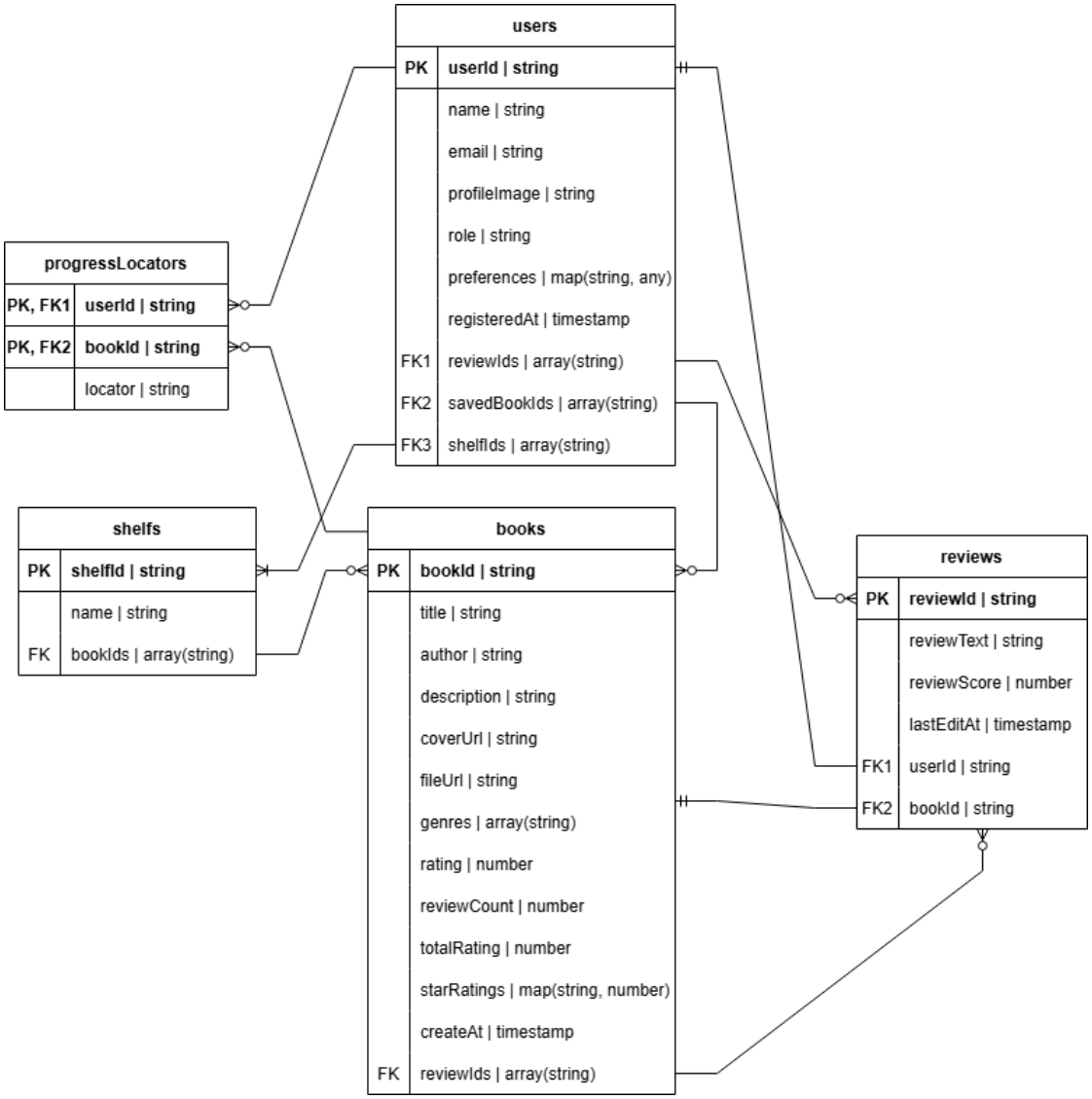



Рисунок 3.2 – ER діаграма логічної структури БД

Так як Cloud Firestore є документно-орієнтованою NoSQL базою даних, то навіть за наявності ER-діаграми не всі зв’язки між сутностями є очевидними. Структура документів у Firestore є гнучкою, що дозволяє

зберігати складені об’єкти, масиви та вкладені підколекції. Проте це також вимагає чіткого розуміння того, які саме поля зберігаються у кожній колекції.

Нижче на таблицях 3.1 – 3.5 наведено опис основних полів документів для колекцій з бази даних:

Таблиця 3.1 – Опис полів колекції users

| Поле | Тип | Опис |
|---|---|---|
| name | string | Ім’я або псевдонім користувача |
| email | string | Адреса електронної пошти користувача, за нею користувач входить в свій акаунт |
| profileImage | string | Посилання на зображення профілю, зберігається в Firebase storage |
| role | string | Назва ролі користувача |
| preferences | map(string, any) | Мапа, що зберігає всі налаштування читання користувача (тема, розмір шрифту) |
| registeredAt | timestamp | Дата реєстрації користувача на сайті |
| reviewIds | array of strings | Список ідентифікаторів, для позначення всіх рев’ю, які створив користувач. |
| savedBookIds | array of strings | Список ідентифікаторів, для позначення всіх книжок, які користувач помітив, як збережені |
| shelfIds | array of string | Список ідентифікаторів, які вказують на підколекцію, де зберігається інформація про всі полиці, створені користувачем |

Колекція users зберігає особисті дані зареєстрованих користувачів і є ключовим елементом структури бази даних, оскільки користувач взаємодіє з іншими об'єктами. Поля reviewIds, savedBookIds і shelfIds забезпечують

логічні зв’язки з іншими колекціями, формуючи персоналізовану структуру бібліотеки для кожного користувача.

Таблиця 3.2 – Опис полів колекції books

| Поле | Тип | Опис |
|---|---|---|
| title | string | Назва книги |
| author | string | Ім’я автора |
| description | string | Опис книги |
| coverUrl | string | Посилання на обкладинку книги, зберігається в Firebase storage |
| fileUrl | string | Посилання на файл книги в форматі EPUB, зберігається в Firebase storage |
| genres | array of strings | Список жанрів книги, з спеціальними назвами, які потім програма перетворює на локалізовані назви жанрів. |
| rating | number | Середня оцінка книги, створена користувачами |
| reviewCount | number | Кількість оцінок книги |
| totalRating | number | Сума всіх оцінок книги, потрібна для обчислення середньої оцінки книги |
| starRatings | map of (string, number) | Мапа, що зберігає кількість оцінок для кожного рівня зірок (від 1 до 5) |
| createdAt | timestamp | Дата завантаження книги в бібліотеку |
| reviewIds | array of strings | Список ідентифікаторів, які вказують на колекцію, де зберігається інформація про всі відгуки, про цю книгу |

Колекція books відображає електронні книги в застосунку, включаючи назву, автора, опис, обкладинку, жанри та рейтинг. Поля reviewCount,

totalRating і starRatings формують статистику відгуків користувачів. Масив reviewIds дозволяє швидко отримати всі рецензії конкретної книги.

Таблиця 3.3 – Опис полів колекції reviews

| Поле | Тип | Опис |
|---|---|---|
| reviewText | string | Текст відгуку, не обов’язковий |
| reviewScore | number | Оцінка книги від 1 до 5 |
| lastEditAt | timestamp | Дата та час створення відгуку |
| userId | reference | Ідентифікатор, який вказує на користувача, який створив відгук |
| bookId | reference | Ідентифікатор, який вказує на книгу, для якої був створений цей відгук |

Колекція reviews містить відгуки користувачів про книги, включаючи текстові коментарі та оцінки. Вона є важливою складовою системи зворотного зв’язку, що дозволяє іншим користувачам ознайомитися з враженнями читачів. Поля userId і bookId зв'язують відгук з відповідним користувачем і книгою, що дозволяє легко формувати списки рецензій для кожної книги та профілю користувача.

Таблиця 3.4 – Опис полів колекції shelfs

| Поле | Тип | Опис |
|---|---|---|
| name | string | Назва колекції, яку користувач створив при створенні полиці, чи стандартна – Несортовані |
| bookIds | array of strings | Список ідентифікаторів книжок, які користувач може додати до своїх колекцій |

Підколекція shelfs є важливою складовою частиною персоналізованої бібліотеки користувача. Кожна полиця має унікальну назву, що дозволяє користувачеві створювати тематичні добірки книг, такі як “Прочитано”, “Читаю зараз”, “Улюблене” або інші власні категорії. Крім назви, кожна полиця містить масив ідентифікаторів книг (bookIds), що дозволяє відображати книжкові добірки у вигляді структурованих списків. Така гнучка організація даних значно покращує користувацький досвід, дозволяючи ефективно працювати з особистими книжковими добірками, сортувати та фільтрувати книги згідно з уподобаннями. Підколекція shelfs логічно належить до users і зберігається як вкладена колекція у Firestore, що дозволяє легко реалізувати вкладену ієрархію без необхідності складних зовнішніх зв’язків.

Таблиця 3.5 – Опис полів колекції progressLocators

| Поле | Тип | Опис |
|---|---|---|
| locator | string (JSON) | Позиція читання у форматі Readium Locator — містить інформацію про розділ, позицію, scroll-точку тощо. |
| updatedAt | timestamp | Час останнього оновлення прогресу читання. |

Та наостанок не менш важлива колекція progressLocators, вона зберігає прогрес читання кожного користувача для конкретної книги. Ідентифікатором документа в цій колекції є комбінація userId_bookId, що забезпечує унікальність запису для кожного користувача й книги. Основне поле — locator, яке зберігає JSON-об’єкт, сформований бібліотекою Readium, що відповідає точному місцю зупинки читання. Завдяки цьому користувач може легко продовжити читання з того місця, де зупинився.

## 3.3 Реалізація модулів застосунку

Мобільний застосунок складається з функціональних модулів, кожен з яких відповідає за реалізацію окремої частини користувацького досвіду. Модульна структура забезпечує масштабованість застосунку, спрощує його підтримку і тестування, а також сприяє повторному використанню компонентів в інших частинах проєкту. Нижче наведено опис основних модулів.

### 3.3.1 Модуль пошуку книг

Модуль пошуку є важливою частиною користувацького інтерфейсу, що дозволяє швидко знайти потрібну книгу серед великого каталогу. Його реалізація передбачає використання декількох параметрів пошуку, фільтрації та сортування, що значно підвищує зручність користування застосунком.

Пошук здійснюється у ViewModel, яка приймає вхідні параметри та формує відповідні запити до Firestore через репозиторій. Основні параметри пошуку включають:

Пошук за ключовим словом - користувач може ввести назву книги або ім’я автора. Через те, що Firestore не підтримує повноцінного текстового пошуку (зокрема, операції часткового співпадіння чи пошуку підрядка), дана функція реалізована на клієнтській стороні. Спочатку застосунок завантажує з бази даних набір книг, що відповідають базовим критеріям (наприклад, жанр або рейтинг), після чого ViewModel виконує фільтрацію отриманого списку за введеним текстом. Кожен елемент перевіряється на наявність часткового співпадіння ключового слова в полях title або author. Такий підхід дозволяє реалізувати пошук без складних індексів, проте вимагає контролю обсягу вибірки для запобігання зайвому навантаженню на клієнт.

Фільтрація за жанром - користувач має можливість обрати один або кілька жанрів із заздалегідь визначеного списку для звуження результатів пошуку. Ця фільтрація виконується на серверній стороні, що зменшує навантаження на базу даних.

Виключення небажаних жанрів - для підвищення персоналізації реалізовано функціональність виключення книг із певних жанрів, які користувач вказує як небажані. Через те, що Firestore не підтримує запити з оператором NOT IN для масивів, дана логіка реалізується на клієнтській стороні. Після завантаження книг із бази ViewModel здійснює додаткову фільтрацію: із загального списку виключаються ті книги, у яких хоча б один жанр збігається з переліком жанрів, зазначених як небажані.

Фільтрація за мінімальним рейтингом - доступна можливість встановити мінімальне значення оцінки (наприклад, не нижче 3.5). Це дозволяє відсіяти книги з низьким середнім рейтингом. У репозиторії формується запит, що перевіряє поле rating >= X.

Сортування результатів – користувач може обрати порядок відображення результатів за одним із параметрів:

- за назвою (title)
- за датою додавання (createdAt)
- за середнім рейтингом (rating)

Також для кожного з них є функція сортування за спаданням та зростанням.

Складність реалізації полягає в обмеженнях Firestore щодо комбінованих запитів і сортування. Для фільтрування застосовується динамічне формування запиту в репозиторії з урахуванням того, які фільтри активні. Наприклад, якщо вмикається фільтрація за рейтингом і сортування за датою, запит будується за обома параметрами із попередньо створеними індексами Firestore.

### 3.3.2 Модуль інформації про книгу

Модуль, що відповідає за відображення повної інформації про вибрану книгу, є одним із основних компонентів застосунку. Цей екран забезпечує взаємодію користувача з книжковим контентом: він надає детальну інформацію про видання, а також забезпечує доступ до подальших дій – таких як читання, збереження в обране, написання або редагування відгуку. Доступ до цього модуля можливий зі списку книжок на головній сторінці, у результатах пошуку або через перегляд бібліотеки користувача.

Сторінка детальної інформації про книгу включає в себе такі основні компоненти:

- Назва книги, ім’я автора та рік видання – базові метадані, що дозволяють ідентифікувати книгу та зорієнтувати користувача.
- Опис книги – текстовий блок із розгорнутим анотаційним описом, який дає змогу користувачу скласти уявлення про зміст видання до початку читання.
- Жанри – відображені у вигляді кольорових локалізованих тегів, які дозволяють швидко зрозуміти тематичну належність книги. Ця інформація також використовується у модулі фільтрації при пошуку.
- Обкладинка книги – завантажується з Firebase Storage і слугує візуальним маркером, допомагаючи впізнати видання та підтримувати візуальну привабливість інтерфейсу.
- Рейтинг книги – середнє значення, обчислене на основі всіх оцінок користувачів. Поруч із ним також представлена зіркова шкала – графічна візуалізація розподілу оцінок у діапазоні від 1 до 5 зірок, реалізована у вигляді прогрес-барів. Це дозволяє швидко оцінити загальний рівень задоволеності читачів.
- Інтерактивні кнопки – дають змогу перейти до перегляду книги (EPUB-читання), додати її в обране або залишити власний відгук.

- Останні рецензії користувачів – блок, що містить список нещодавніх коментарів до книги від інших користувачів. Вони допомагають скласти уявлення про загальне сприйняття книги аудиторією.

Особливу увагу в межах цього модуля приділено функціональності роботи з рецензією поточного користувача. При відкритті сторінки системою автоматично перевіряється, чи існує вже рецензія від цього користувача (визначається за унікальною комбінацією userId та bookId). Якщо рецензія знайдена – вона підтягується з бази даних і відображається у спеціальному блоці інтерфейсу, де доступна для редагування. У разі її відсутності пропонується чиста форма для створення нової.

Такий підхід дозволяє:

- забезпечити наявність єдиного унікального відгуку від кожного користувача для кожної книги;
- запобігти дублюванню рецензій;
- підтримувати коментарі в актуальному стані;
- надати швидкий доступ до власної думки користувача, виділивши її серед інших.

### 3.3.3 Модуль читання книги

Однією з ключових функцій мобільного застосунку є можливість повноцінного читання електронних книг у форматі EPUB. Для реалізації цього функціоналу було інтегровано бібліотеку Readium Kotlin Toolkit, яка забезпечує багатий набір можливостей для роботи з EPUB-файлами на Android-платформі.

Цей модуль реалізовано у вигляді окремого Fragment, що відповідає за завантаження та відображення вмісту книги. При відкритті книги користувач автоматично перенаправляється на цей екран, де йому надається повноцінне середовище для комфортного читання.

Функціональні можливості модуля включають:

Завантаження та рендеринг EPUB-файлу з Firebase Storage. Після натискання кнопки “Читати” відбувається завантаження файлу та ініціалізація EpubNavigatorFragment, який обробляє структуру документа, таблицю змісту, сторінки, стилі оформлення тощо.

Масштабування та адаптація вигляду тексту – користувач може змінювати розмір шрифту, вибирати між світлою та темною темами, а також підлаштовувати відступи. Це забезпечує персоналізований досвід читання для кожного користувача.

Запам’ятовування позиції – застосунок автоматично зберігає місце, на якому зупинився користувач. При повторному відкритті книги застосунок відновлює перегляд саме з тієї сторінки, де читання було припинено.

Таким чином, модуль читання книги забезпечує повноцінний, гнучкий і зручний інтерфейс для взаємодії з EPUB-документами, що є критично важливим для електронної бібліотеки.

### 3.3.4 Модуль полиць

Модуль персональної бібліотеки користувача є важливою складовою застосунку. Він дозволяє створювати власні тематичні добірки книг, структурувати бібліотеку за вподобаннями та швидко отримувати доступ до обраних матеріалів.

Архітектурно модуль реалізований згідно з принципами MVVM. Усі взаємодії відбуваються через ViewModel, яка звертається до репозиторію для отримання даних з Firestore. Репозиторій відповідає за завантаження структури користувацьких полиць shelfs, а також обробку операцій створення, оновлення та видалення.

Ключовий функціонал модуля включає:

Завантаження колекцій полиць shelfs, що зберігаються для кожного користувача окремо. Кожна колекція містить ідентифікатори полиць, які згруповані за тематиками або за призначенням (наприклад, “Хочу прочитати”, “Улюблене”, “Прочитано”).

Можливість створювати нові полиці, даючи їм унікальні назви. При створенні полиці користувач обирає її назву. Нові полиці зберігаються у підколекції shelfs.

Редагування назв полиць, що дозволяє гнучко адаптувати структуру бібліотеки у процесі використання. Користувач може змінити назву будь-якої полиці, а зміни автоматично синхронізуються з Firestore та оновлюються у всіх пов’язаних компонентах інтерфейсу.

Видалення полиць, разом з усіма асоційованими книгами, при цьому оновлюється як вміст shelfs, так і візуальне представлення бібліотеки.

Додавання та вилучення книг з будь-якої полиці. Книга може належати до декількох полиць одночасно. Для цього використовуються масиви ідентифікаторів (bookIds) у кожному документі полиці, які оновлюються через репозиторій.

Окрему увагу приділено тому, щоб усі дані в модулі оновлюються автоматично завдяки використанню Flow. Будь-які зміни, які користувач вносить до своїх полиць (навіть із іншого пристрою), миттєво відображаються в інтерфейсі без потреби вручну оновлювати екран.

Такий підхід дозволяє реалізувати зручну, динамічну та персоналізовану бібліотеку, яка повністю адаптується під користувацький досвід і відображає принципи сучасного дизайну інтерфейсів мобільних застосунків.

# ВИСНОВОК ДО РОЗДІЛУ 3

У цьому розділі було детально розглянуто ключові аспекти реалізації програмного забезпечення мобільного застосунку для електронної бібліотеки..

Особливу увагу приділено організації бази даних у середовищі Firebase Firestore. Описано структуру основних колекцій, принципи зберігання та взаємозв'язків між сутностями, з урахуванням особливостей документно-орієнтованої NoSQL моделі. Такий підхід дозволив досягти високої гнучкості, масштабованості та швидкої синхронізації даних у режимі реального часу.

Окремо проаналізовано реалізацію функціональних модулів застосунку, зокрема: пошуку книг, перегляду деталей книги, читання EPUB-файлів, роботи з персональними полицями та рецензіями користувачів. Кожен модуль побудовано відповідно до принципів повторного використання та орієнтації на зручність кінцевого користувача.

Таким чином, реалізована програмна архітектура та функціональність забезпечують надійну, продуктивну й масштабовану основу для сучасного електронного книжкового сервісу, що відповідає як технічним вимогам.

# РОЗДІЛ 4 АНАЛІЗ РОЗРОБЛЕНОГО ПРОЄКТУ

У цьому розділі буде здійснено аналіз функціональних можливостей, інтерфейсу та продуктивності створеного мобільного застосунку. Метою є оцінка відповідності реалізованого продукту поставленим вимогам, виявлення його сильних і слабких сторін, а також визначення потенційних напрямів для подальшого вдосконалення.

## 4.1 Демонстрація роботи проєкту

Для забезпечення повного охоплення функціональності мобільного застосунку було побудовано діаграму прецедентів (use case diagram), яка зображена на рис. 4.1. Вона відображає взаємодію користувача з системою, на ній представлено всі основні сценарії роботи з застосунком.


Рисунок 4.1 – Діаграма прецедентів для користувача

### 4.1.1 Вхід до системи

Функціональність входу до системи є однією з базових і необхідних для забезпечення персоналізованої взаємодії користувача із застосунком. Реалізація процесу авторизації здійснюється за допомогою Firebase Authentication, що дозволяє швидко та безпечно керувати обліковими записами користувачів.

Після встановлення застосунку користувач має два шляхи взаємодії:

- авторизація з уже існуючими обліковими даними;
- реєстрація нового акаунта за допомогою електронної пошти та пароля.

Інтерфейс авторизації в застосунку можна побачити на рис. 4.2, а інтерфейс реєстрації нового акаунту, можна побачити на рис. 4.

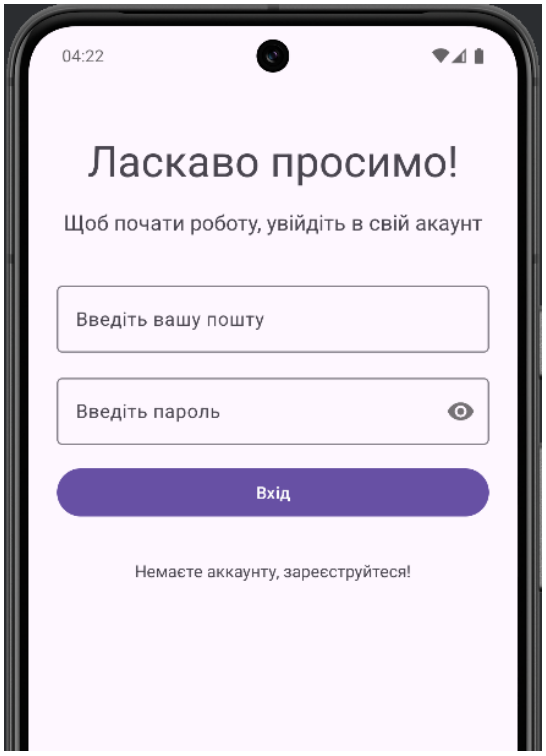



Рисунок 4.2 – Сторінка авторизації

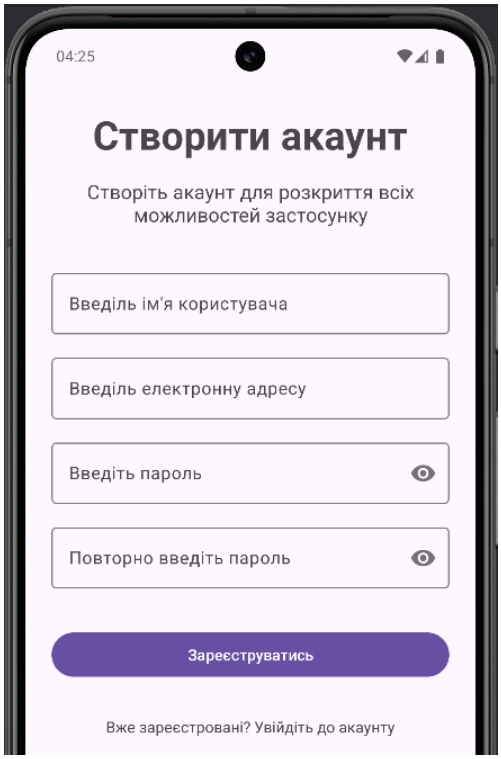



Рисунок 4.3 – Сторінки реєстрації

Інтерфейс авторизації користувача, є першою сторінкою, яку бачить користувач, якщо користувач зареєстрований, він вводить свої дані, а якщо немає, нажимає на текст нижче для переходу до сторінки реєстрації.

У кожної з цих сторінок, є поля куди користувач повинен вводити свої дані, для покращення користувацького досвіду було зроблено виведення помилки, яку зробив користувач.

На рис 4.4, 4.5 наведено приклади користувацьких помилок, на сторінці авторизації та реєстрації, відповідно

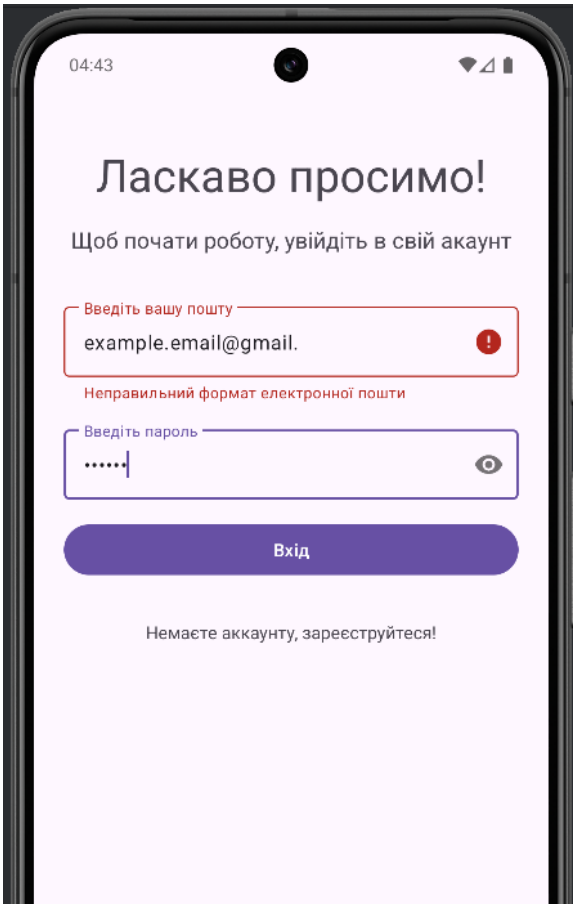



Рисунок 4.4 – Валідація даних при вході

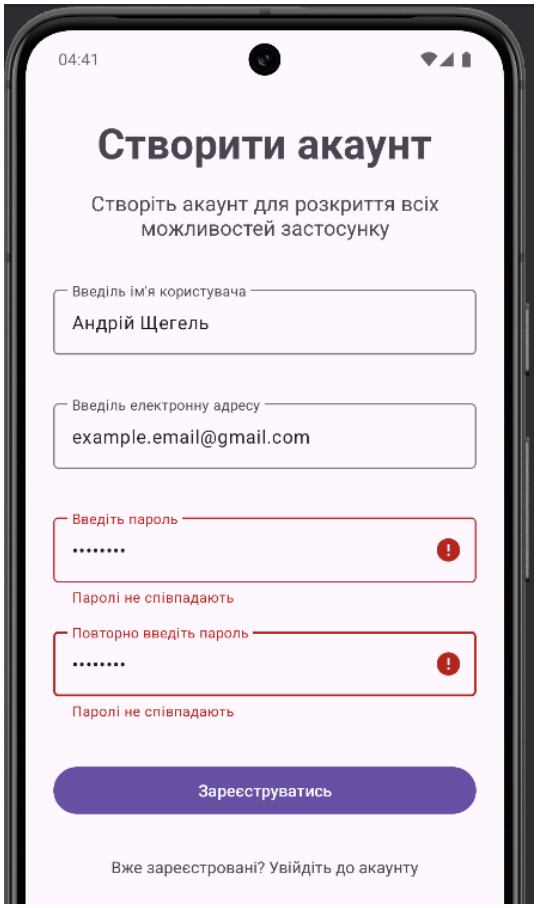



Рисунок 4.5 – Валідація даних при реєстрації

Також для збереження часу користувачів, було реалізовано функціонал збереження сесії – повторний вхід без збереження пароля.

Таким чином, модуль входу до системи забезпечує не лише безпечну автентифікацію, а й зручний старт взаємодії користувача з функціоналом електронної бібліотеки. Його реалізація є основою для доступу до всіх персоналізованих функцій застосунку.

### 4.1.2 Головна сторінка

Після успішної авторизації користувач потрапляє на домашню сторінку застосунку – екран, який надає швидкий доступ до найактуальніших матеріалів. Цей екран формує перше враження про систему, тому при його реалізації особливу увагу було приділено дизайну, структурі та релевантності контенту.

Головна сторінка відображає такі основні розділи:

- Новинки – блок, що містить останні додані книги. Джерелом даних є поле createdAt у колекції books, що дозволяє формувати добірку за датою додавання. Це дозволяє користувачам швидко ознайомитися з новими надходженнями.
- Популярні книги – перелік найвищо оцінених книг у застосунку. Рейтинг формується на основі поля rating у кожному документі книги. У цьому розділі зазвичай виводиться фіксована кількість книг з найвищими оцінками.
- Жанрові добірки – наприклад, “Детективи”, “Фентезі”, “Психологія” тощо. Кожна добірка формується окремим запитом до Firestore з фільтрацією за полем genres. Це дозволяє швидко знаходити літературу в улюблених категоріях.

Усі блоки відображаються у вигляді горизонтальних RecyclerView (каруселей), що дозволяє зручно гортати книги всередині кожної категорії. Натискання на обкладинку відкриває детальну інформацію про відповідну книгу.

На рис 4.6 можемо побачити інтерфейс представлення декількох розділів, які допоможуть користувачу швидше дізнатись про книги в нашій електронній бібліотеці.

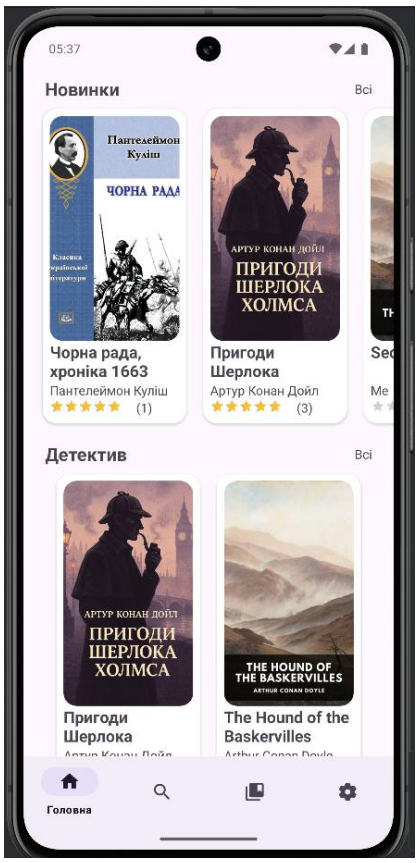



Рисунок 4.6 – Інтерфейс головної сторінки

Головна сторінка слугує центральним навігаційним елементом, який забезпечує перший контакт користувача з вмістом бібліотеки, стимулює подальшу взаємодію із застосунком і підвищує залученість.

### 4.1.3 Інформація про книгу

Екран із повною інформацією про книгу є одним з центральних елементів взаємодії користувача із застосунком. Цей екран відкривається при натисканні на будь-яку обкладинку книги зі списку – чи то з головної сторінки, результатів пошуку, особистих полиць, чи бібліотеки. Його призначення – надати користувачеві вичерпну інформацію про вибране видання, а також забезпечити доступ до додаткових дій: читання, додавання в обране, створення відгуку тощо.

Для зручності користувача реалізовано автоматичне визначення, чи вже є ця книга в обраних або чи існує особиста рецензія – у такому випадку кнопки змінюють стан (наприклад, "Редагувати відгук").

Для отримання більш повної інформації по цій сторінці, див пункт 3.3.2.

На рисунку 4.7 зображено інтерфейс сторінки книги, де показано розміщення інформаційних блоків, доступ до основних функцій та дизайн, що відповідає стилю застосунку. Візуальна структура забезпечує орієнтацію в контенті без перевантаження користувача інформацією.

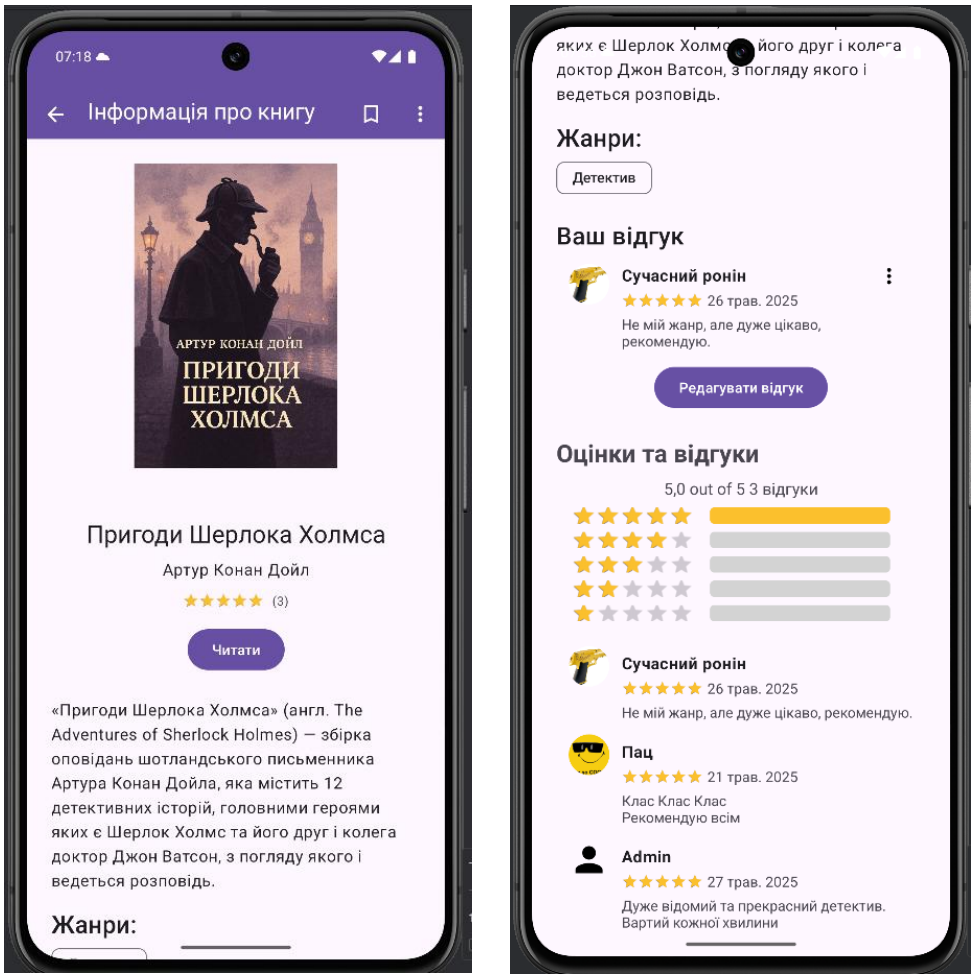



Рисунок 4.7 – Вигляд сторінки інформації про книгу

### 4.1.4 Читання книги

Функціональність читання книги є ключовою частиною застосунку, яка безпосередньо реалізує основне призначення електронної бібліотеки – надання доступу до літературного контенту. Після натискання кнопки “Читати” на сторінці книги, користувач переходить на екран перегляду вмісту файлу у форматі EPUB.

Більше про реалізацію цього модуля можна знайти в пункті 3.3.3, де описано більше про інтеграцію та можливості цього модуля.

Цей модуль представлений мінімалістичним, функціональним виглядом, який фокусується на зручності читання та уникає зайвих відволікаючих елементів. У нижній або верхній частині інтерфейсу доступні елементи управління, які дають змогу швидко змінювати розмір шрифту, тему оформлення або перейти до змісту книги.

Цей екран автоматично зберігає позицію читання, забезпечуючи користувачу зручне повернення до місця, на якому було припинено читання. Такий підхід забезпечує комфортне та гнучке використання застосунку незалежно від типу контенту чи індивідуальних уподобань читача.

Далі подивимось на виконання:

Для прикладу візьмемо два рисунки. На рисунку 4.8 представлено приклад стандартного вигляду екрану для читання книги у мобільному застосунку – інтерфейс оформлений у світлій темі з типовим шрифтом і розмірами, що підходить для загального використання. Такий вигляд за замовчуванням орієнтований на нових користувачів або тих, хто не змінював налаштування читання.

Для порівняння, на рисунку 4.9 зображено персоналізований вигляд цього ж екрану – користувач активував тему сепії для більш комфортного читання, а також збільшив розмір шрифту для зручнішого сприйняття тексту, ще й виставив міжрядковий інтервал.

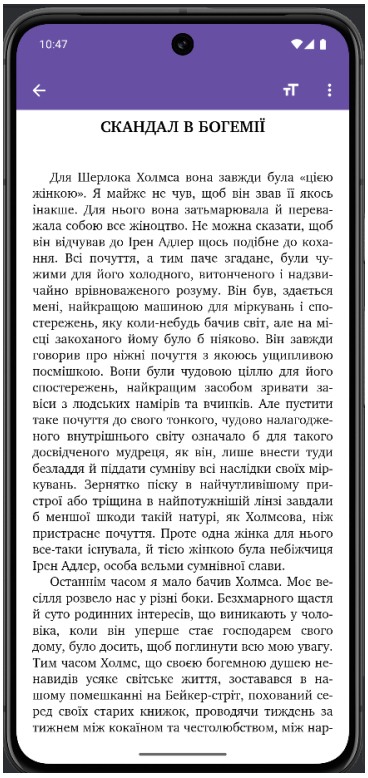



Рисунок 4.8 – Стандартний вигляд інтерфейсу читання книги

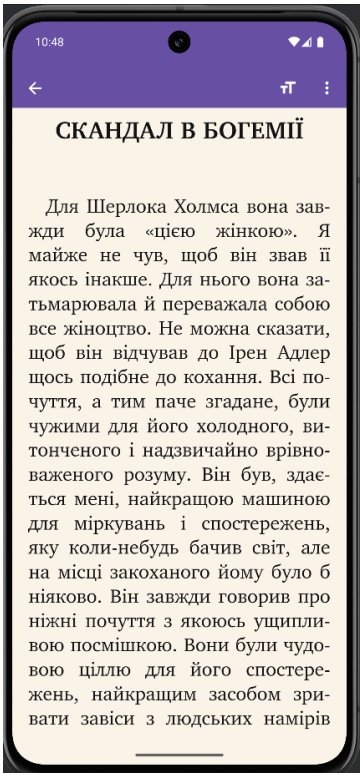



Рисунок 4.9 – Персоналізований вигляд інтерфейсу читання книги

Завдяки таким можливостям кастомізації, застосунок дозволяє кожному користувачу адаптувати інтерфейс читання до власних уподобань, створюючи комфортне середовище для тривалого читання. Це особливо важливо для користувачів з різним рівнем зору або специфічними вимогами до інтерфейсу, що підвищує доступність застосунку.

### 4.1.5 Пошук книг

Функціональність пошуку книг є одним із найважливіших інструментів у навігації по великому каталогу електронної бібліотеки. Вона дозволяє користувачам швидко знаходити потрібні книги за різними критеріями, такими як назва, автор або жанр. На головному екрані пошуку користувач може ввести ключове слово, що відразу активує фільтрацію доступних книг на

клієнтській стороні – адже Firestore не підтримує повноцінного текстового пошуку.

Окрім базового пошуку, застосунок підтримує розширені можливості фільтрації, що дозволяють користувачеві отримати більш точні та віповідні запиту результати. Зокрема, доступна опція вибору одного або кількох жанрів, виключення небажаних жанрів із результатів, встановлення мінімального порогового значення рейтингу книги, а також сортування за різними критеріями – назвою, датою додавання або середньою оцінкою. Для більш детального пояснення функцій фільтрації див. пункт 3.3.1.

Приступимо до аналізу цього модуля.

На рисунку 4.10 наведено початковий інтерфейс пошуку книг, пошук книг проводиться що разу після переходу до вкладки. На рис 4.11 наведено приклад пошуку, по назві книги.

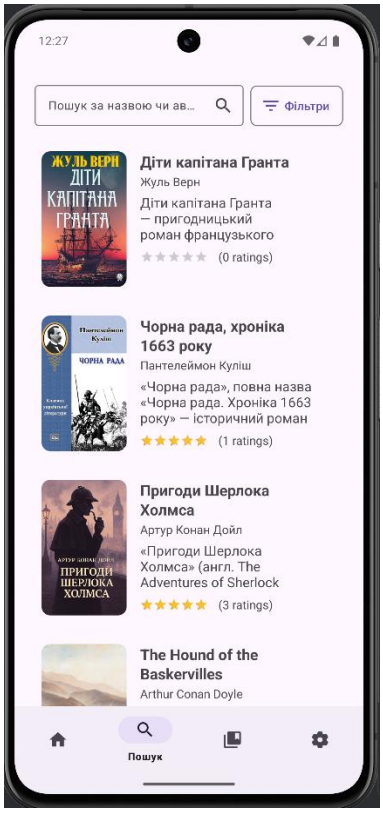



Рисунок 4.10 – Початковий стан інтерфейсу

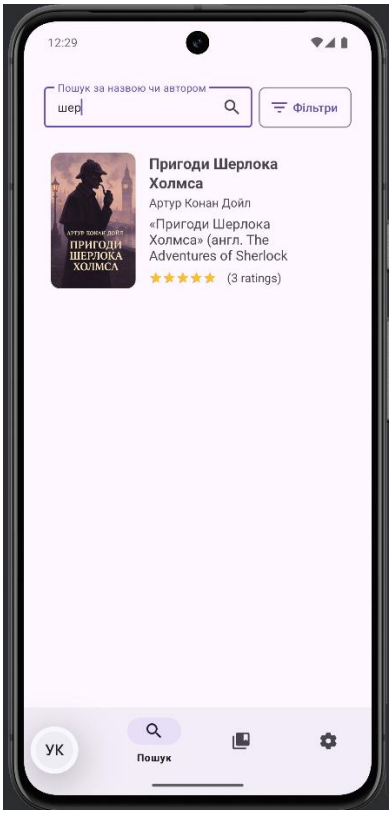



Рисунок 4.11 – Інтерфейс з результатом пошуку по назві

Далі розглянемо фільтри, які користувач може застосувати для пошуку.

На рисунку 4.12 зображено початковий вигляд інтерфейсу фільтрації, у якому користувач може побачити всі доступні параметри пошуку. У цьому стані жоден із фільтрів ще не застосовано, що дозволяє виконати пошук за ключовим словом без додаткових обмежень. Такий підхід зручний для загального перегляду бібліотеки або першого ознайомлення з контентом.

На рисунку 4.13 представлено приклад, коли користувач обирає певні фільтри, обирає жанр, який йому подобається я, виключає один із небажаних та встановлює мінімальну оцінку книги.

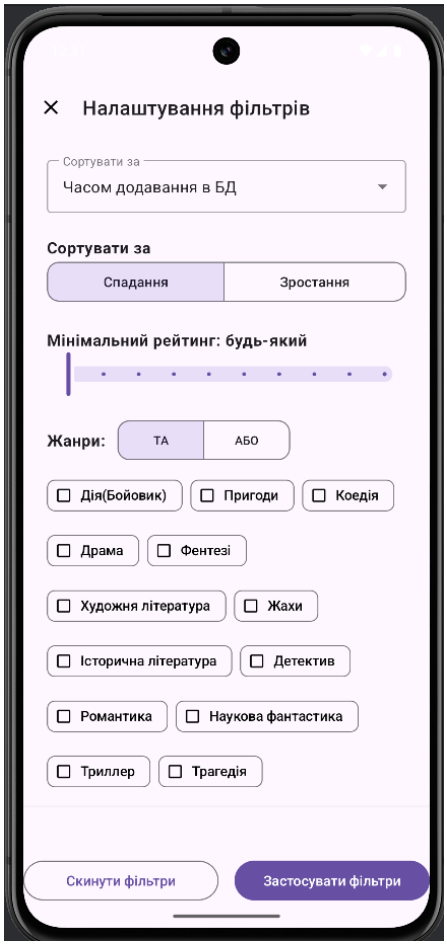



Рисунок 4.12 – Початкові налаштування фільтрів

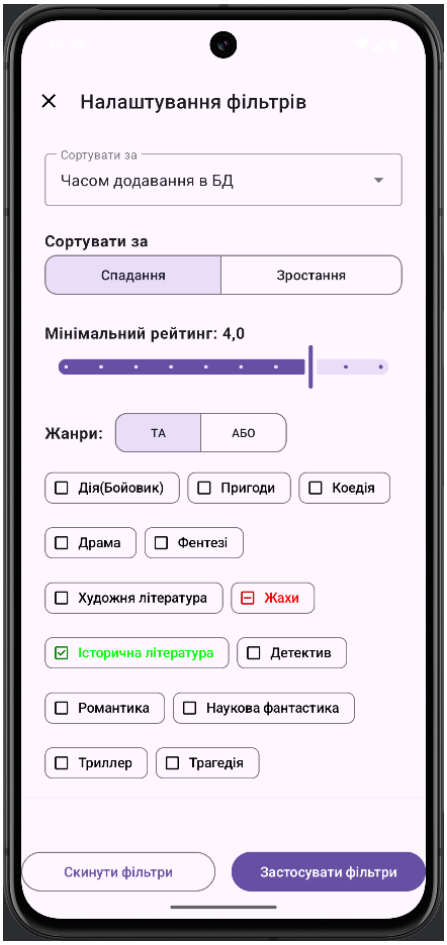



Рисунок 4.13 – Налаштування фільтрів для пошуку по вподобаннях

Після налаштування критеріїв, користувач натискає кнопку “Застосувати фільтри”, внаслідок чого система формує запит і виводить оновлений список результатів, що відповідають усім обраним параметрам. Це забезпечує зручність і точність пошуку, адаптованого до індивідуальних вподобань користувача. Результат запиту можна побачити на рис. 4.14.

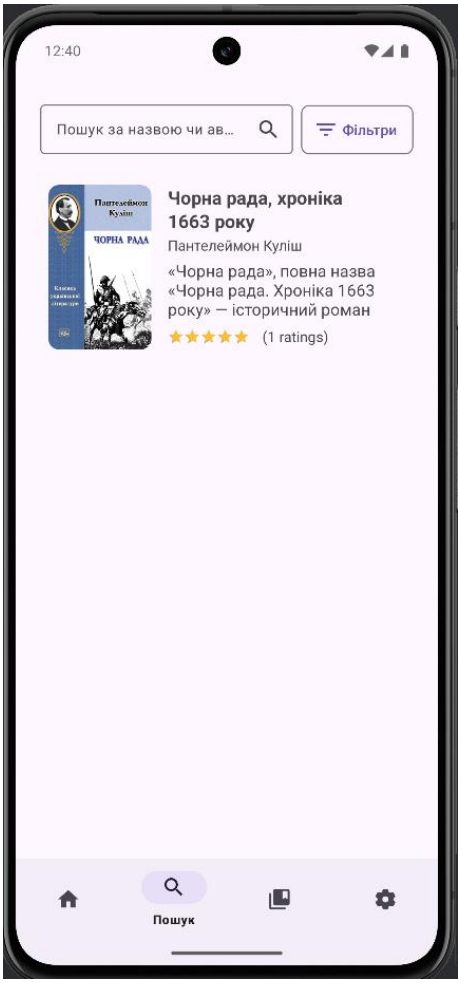



Рисунок 4.14 – Результат пошуку книг за заданими фільтрами

Застосовуючи вибрані фільтри, система провела пошук у колекції книг і сформувала відповідь, яка повністю відповідає заданим критеріям. Єдиною виявленою книгою стала "Чорна рада", що належить до вибраних жанрів, має відповідний середній рейтинг та була додана в межах встановленого періоду. Цей результат свідчить про ефективність реалізованого механізму фільтрації.

Слід зазначити, що кількість знайдених книг наразі обмежена через те, що база даних ще перебуває в процесі наповнення. У майбутньому, зі збільшенням кількості завантажених видань, результати пошуку стануть більш різноманітними, а користувачі зможуть швидше знаходити твори, які відповідають їхнім перевагам. Незважаючи на це, вже зараз інтерфейс пошуку демонструє свою гнучкість і здатність точно підбирати релевантні матеріали.

### 4.1.5 Моя бібліотека та колекції

"Моя бібліотека" є розділом застосунку, який дозволяє користувачеві організовувати свій читацький простір за допомогою тематичних колекцій і швидкого доступу до важливих книг. Цей функціонал забезпечує краще взаємодію з контентом і сприяє створенню впорядкованого середовища для читання.

Окрім тематичних полиць, модуль бібліотеки включає ще три важливі списки:

Збережені книги – перелік видань, які користувач позначив як обрані. Це дає змогу швидко повертатися до вподобаного матеріалу без необхідності повторного пошуку.

Завантажені книги – перелік книг, що збережені локально на пристрої для читання в офлайн-режимі. Цей список є особливо корисним у випадках обмеженого або відсутнього доступу до Інтернету.

Продовження читання – секція, яка відображає останні відкриті книги та дозволяє миттєво перейти до сторінки, на якій було зупинено читання. Застосунок автоматично зберігає позицію користувача під час кожного сеансу, що забезпечує безшовний досвід використання.

Усі дані зберігаються у Firestore та синхронізуються в реальному часі. Зміни в колекціях або списках автоматично оновлюються в інтерфейсі завдяки

Kotlin Flow, без ручного оновлення екрана. Це дозволяє користувачам бачити актуальний стан бібліотеки на всіх пристроях.

На рисунку 4.15 детально представлено інтерфейс розділу "Моя бібліотека", який включає ключові компоненти персоналізованої взаємодії користувача з книжковим контентом. Зокрема, демонструється загальний вигляд бібліотеки з секціями збережених, завантажених і нещодавно відкритих книг, що забезпечують швидкий доступ до актуального контенту.

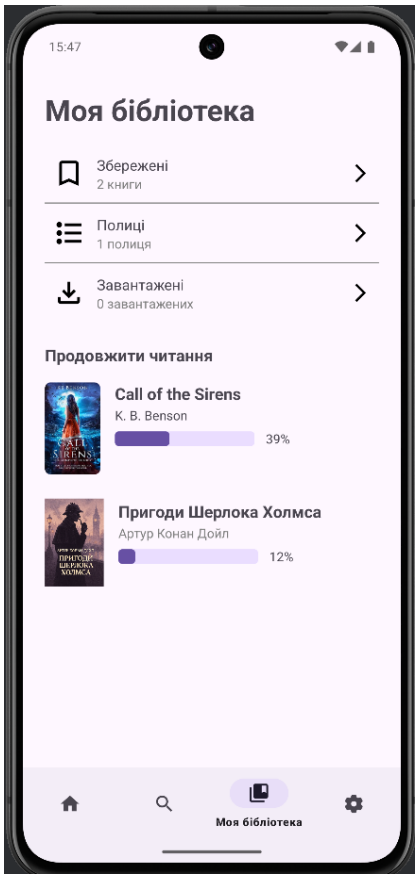



Рисунок 4.15 – Інтерфейс сторінки “Моя біліотека”

На рисунках 4.16 та 4.17 представлено інтерфейс модуля користувацьких полиць, що є складовою персональної бібліотеки кожного користувача. Принципи його роботи, архітектуру та обробку даних описано у пункті 3.3.4. У цьому підрозділі зосередимо увагу на візуальній частині реалізації: розглянемо список створених полиць, та відображення книг у них.


Рисунок 4.16 – Інтерфейс списків полиць

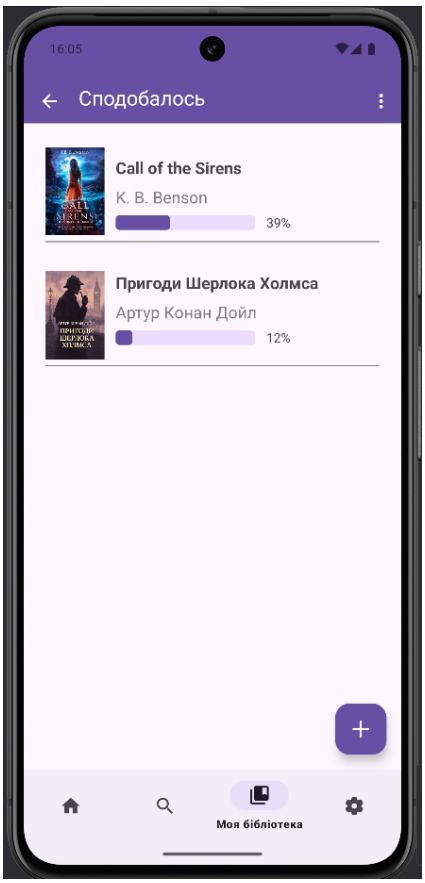



Рисунок 4.17 – Інтерфейс книг в полицях

## 4.2 Рекомендації щодо подальшого вдосконалення

Хоча розроблений мобільний застосунок BookReader4U уже реалізує основні функції електронної бібліотеки та забезпечує зручний інтерфейс для користувача, існує ряд напрямів, які можна розглянути для подальшого покращення функціональності та користувацького досвіду.

Створення повноцінного пошуку за текстом. В даний момент пошук за назвою та автором виконується на клієнтській стороні, що обмежує масштабованість при збільшенні бази даних. У майбутньому доцільно інтегрувати сторонній сервіс повнотекстового пошуку, наприклад Algolia або ElasticSearch, який дозволить реалізувати більш точні та швидкі пошукові

запити з підтримкою автодоповнення, синонімів та сортування за релевантністю.

Підтримка інших форматів книг. Наразі підтримується лише формат EPUB. Варто розглянути додаткову підтримку PDF, FB2 чи аудіокниг для розширення цільової аудиторії.

Незважаючи на хмарну структуру збереження даних, для зручності користувачів бажано реалізувати повноцінний офлайн-режим із локальним кешуванням завантажених книг, обраних полиць і рецензій.

Поточна версія не включає систему рекомендацій. Запровадження алгоритмів машинного навчання або базових фільтрів на основі жанрових вподобань, історії читання та рейтингу дозволить створювати персоналізовані добірки, що підвищить залученість користувача.

Подальша оптимізація інтерфейсу для різних розмірів екранів, включаючи планшети, а також реалізація жестів для полегшення навігації покращить загальне враження від роботи з додатком.

Загалом, ці рекомендації спрямовані на підвищення функціональності, продуктивності та залученості користувачів. Їх реалізація дозволить розширити можливості застосунку та зробити його більш конкурентоспроможним на ринку мобільних рішень для читання електронних книг.

# ВИСНОВОК ДО РОЗДІЛУ 4

У цьому розділі було здійснено аналіз розробленого мобільного застосунку за його функціональністю, інтерфейсом користувача та реалізованими модулями. За допомогою діаграми прецедентів і прикладів інтерфейсів розглянуто основні сценарії взаємодії користувача з системою: реєстрація та вхід до акаунту, перегляд головної сторінки з добірками книг, детальна інформація про книгу, читання EPUB-файлів, пошук літератури за різними критеріями та робота з персональною бібліотекою.

Особливу увагу приділено модулям читання, рецензування та управління книжковими полицями, які формують індивідуальний досвід користувача та дозволяють максимально зручно працювати з електронною бібліотекою. Інтерфейси застосунку адаптовано для зручного мобільного використання, а логіка взаємодії з Firebase реалізована згідно з архітектурним шаблоном MVVM.

У підрозділі 4.2 також запропоновано можливі напрями подальшого вдосконалення, які охоплюють як технічні аспекти (покращення пошуку), так і функціональні (система рекомендацій, підтримка інших форматів книг). Здійснений аналіз дозволяє оцінити сильні сторони застосунку та визначити вектори для його майбутнього розвитку.

# ВИСНОВКИ

У межах дипломного проєкту було розроблено мобільний застосунок, який реалізує функціональність сучасної електронної бібліотеки з підтримкою читання книг у форматі EPUB. Під час роботи над проєктом проведено повний цикл розробки програмного забезпечення – від аналізу вимог і вибору технологій до реалізації функціональних модулів та підготовки рекомендацій щодо подальшого вдосконалення.

У теоретичній частині проєкту було проаналізовано особливості мобільних електронних бібліотек, вимоги до архітектури подібних застосунків, а також проведено обґрунтований вибір технологічного стеку: Kotlin, Android SDK, Firebase, Readium Kotlin Toolkit тощо.

Застосунок побудовано за архітектурним підходом MVVM, що забезпечує хорошу підтримуваність, масштабованість і тестованість проєкту. Було детально описано структуру бази даних Firestore та реалізовано логіку обробки даних із використанням ViewModel і репозиторіїв.

У результаті проведеної роботи вдалося досягти поставленої мети: створити функціональний, сучасний, адаптивний і зручний у користуванні мобільний застосунок для читання книг. Також сформульовано напрямки для подальшого розвитку проєкту – серед них: інтеграція повнотекстового пошуку, система рекомендацій, розширення формату підтримуваних файлів.

Розроблений застосунок повністю виконує умови технічного завдання та має значний потенціал для розвитку.

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ


1. Челяк О. Від класики до платформ із ШІ: 12 застосунків для поціновувачів читання. *Суспільне Медіа*. URL: https://suspilne.media/culture/919537-vid-klasiki-do-platform-iz-si-12-zastosunkiv-dla-pocinovuvaciv-citanna/ (дата звернення: 12.05.2025).
2. Google Play Books app overview. *The DAISY Consortium*. URL: https://daisy.org/guidance/info-help/guidance-training/reading-systems/google-play-books-app-overview/ (date of access: 12.05.2025).
3. Free Kindle reading apps for iOS, Android, Mac, and PC. *Amazon.com*. URL: https://www.amazon.com/b?ie=UTF8&node=16571048011 (date of access: 13.05.2025).
4. Про нас та контакти. *Librarius*. URL: https://librarius.pro/about-us-and-contacts (дата звернення: 13.05.2025).
5. Wattpad - Where stories live. *Wattpad*. URL: https://www.wattpad.com/ (date of access: 14.05.2025).
6. PocketBook Reader - додаток для читання pdf, epub. *Офіційний веб сайт компанії PocketBook*. URL: https://pocketbook.com.ua/uk-ua/app-ua (дата звернення: 14.05.2025).
7. Android's Kotlin-first approach. *Android Developers*. URL: https://developer.android.com/kotlin/first (date of access: 20.05.2025).
8. Firebase authentication. *Firebase*. URL: https://firebase.google.com/docs/auth (date of access: 20.05.2025).
9. Cloud Firestore. *Firebase*. URL: https://firebase.google.com/docs/firestore (date of access: 20.05.2025).
10. Cloud Storage for Firebase. *Firebase*. URL: https://firebase.google.com/docs/storage (date of access: 20.05.2025).
11. Işık O. C. Introduction to MVVM Architecture. Medium. URL: https://medium.com/@onurcem.isik/introduction-to-mvvm-architecture-5c5558c3679 (date of access: 22.05.2025).

12. Vijayan B. Architectural Pattern - Model–view–presenter (MVP). DEV Community. URL: https://dev.to/binoy123/architectural-pattern-model-view-presenter-mvp-28hl (date of access: 23.05.2025).
13. Dependency injection with Hilt | App architecture. *Android Developers*. URL: https://developer.android.com/training/dependency-injection/hilt-android (date of access: 26.05.2025).

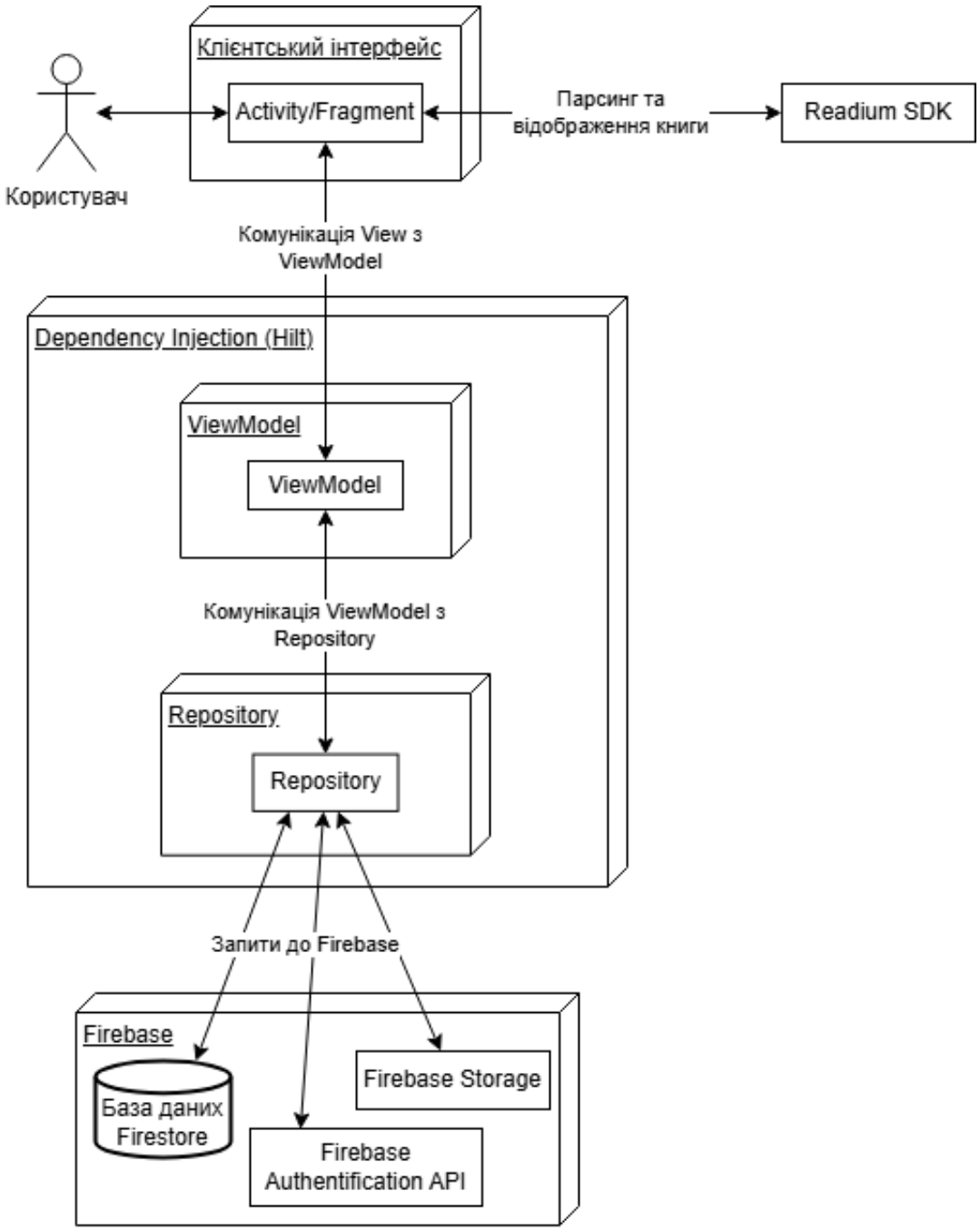
# ДОДАТОК А

Мобільний застосунок для електронної бібліотеки

Структурна схема системи

Аркушів 1

Київ - 2025 р.




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467200.004 ДА | | |
| Зм. | Арк. | № докум. | Підпис | Дата | | | |
| Розробив | | Щегель А. С. | | | **Мобільний застосунок для електронної бібліотеки** Структурна схема системи | Літ. | Аркуш 1 / Аркушів 1 |
| Перевірив | | Гайдай А. Р. | | | | | |
| Реценз. | | Шимкович В. М. | | | | КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11 | |
| Н. Контр. | | Нікольский С. С. | | | | | |
| Затвердив | | | | | | | |

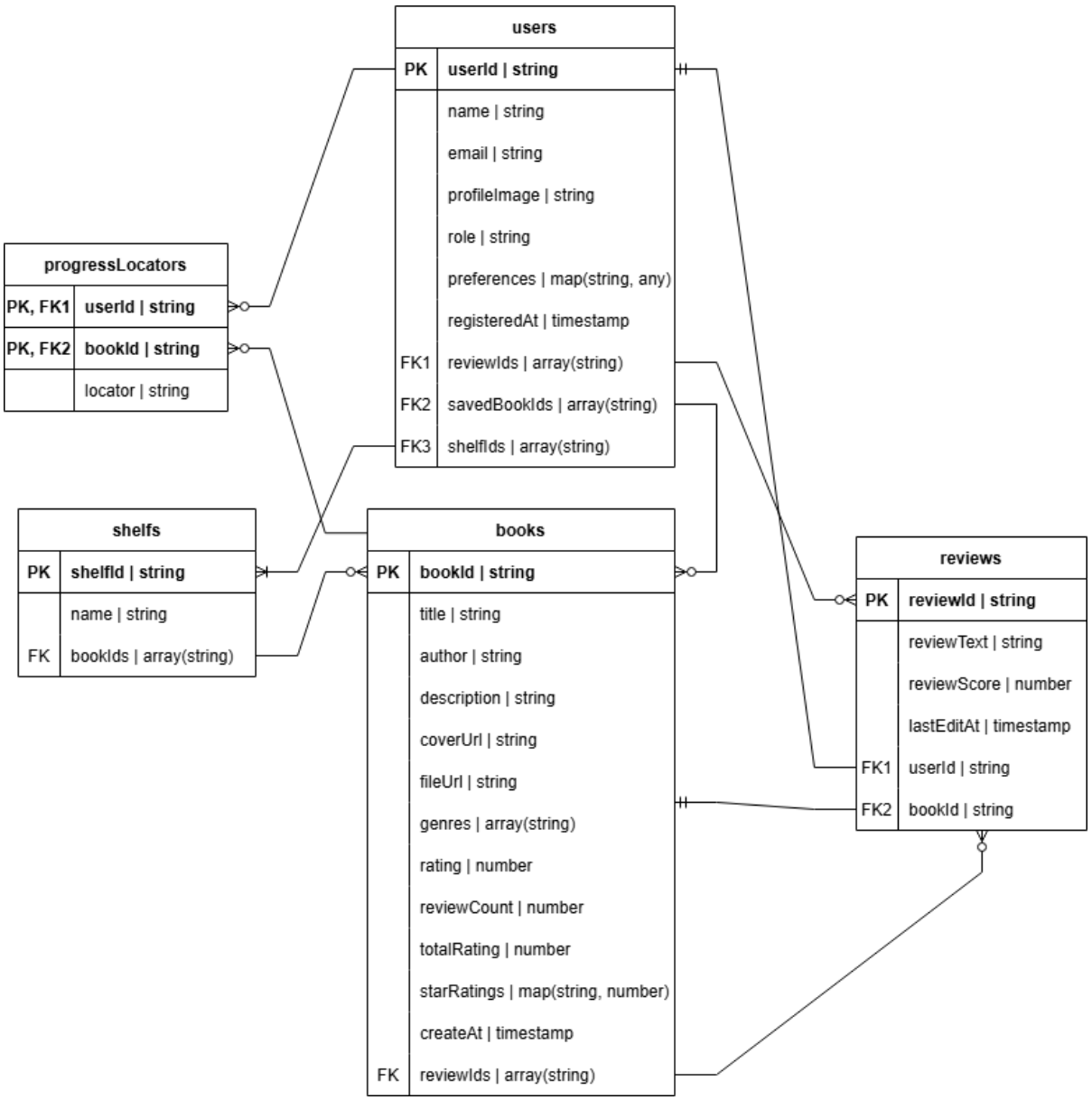
# ДОДАТОК Б

Мобільний застосунок для електронної бібліотеки

Функціональна схема (ER діаграма)

Аркушів 1

Київ - 2025 р.




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467200.005 ДБ | | |
| Зм. | Арк. | № докум. | Підпис | Дата | | | |
| Розробив | | Щегель А. С. | | | **Мобільний застосунок для електронної бібліотеки** | Літ. | Аркуш 1 / Аркушів 1 |
| Перевірив | | Гайдай А. Р. | | | | | |
| Реценз. | | Шимкович В. М. | | | Функціональна схема (ER діаграма) | КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11 | |
| Н. Контр. | | Нікольский С. С. | | | | | |
| Затвердив | | | | | | | |

# ДОДАТОК В

Мобільний застосунок для електронної бібліотеки

Принципова схема (UML діаграма прецедентів)

Аркушів 1

Київ - 2025р.


| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | ІАЛЦ.467200.006 ДВ | | |
| Зм. | Арк. | № докум. | Підпис | Дата | | | |
| Розробив | | Щегель А. С. | | | **Мобільний застосунок для електронної бібліотеки** | Літ. | Аркуш 1 / Аркушів 1 |
| Перевірив | | Гайдай А. Р. | | | | | |
| Реценз. | | Шимкович В. М. | | | Принципова схема (UML діаграма прецедентів) | КПІ ім. Ігоря Сікорського, ФІОТ, ІМ-11 | |
| Н. Контр. | | Нікольский С. С. | | | | | |
| Затвердив | | | | | | | |

# ДОДАТОК Г

Мобільний застосунок для електронної бібліотеки

Текст програмного коду

Аркушів 30

Київ - 2025р.

```
// BookReader4UApplication
@HiltAndroidApp
class BookReader4UApplication : Application()

// di/FirebaseModule
@InstallIn(SingletonComponent::class)
@Module
object FirebaseModule {

    @Provides
    @Singleton
    fun provideFirestoneInstance(): FirebaseFirestore{
        return FirebaseFirestore.getInstance()
    }

    @Provides
    @Singleton
    fun provideAuthInstance(): FirebaseAuth{
        return FirebaseAuth.getInstance()
    }
}

// di/RepositoryModule
@InstallIn(SingletonComponent::class)
@Module
object RepositoryModule {

    @Provides
    @Singleton
    fun provideBaseRepository(
        auth: FirebaseAuth,
        database: FirebaseFirestore
    ): BaseRepository {
        return BaseRepositoryImp(database, auth)
    }

    @Provides
    @Singleton
    fun provideAuthRepository(
        auth: FirebaseAuth,
        baseRepository: BaseRepository
    ): AuthRepository {
        return AuthRepositoryImp(auth, baseRepository)
    }

    @Provides
    @Singleton
    fun provideBookDetailsRepository(
        baseRepository: BaseRepository
    ): BookDetailsRepository {
        return BookDetailsRepositoryImp(baseRepository)
    }

// data/repository/BaseRepositoryImp
class BaseRepositoryImp(
    private val database: FirebaseFirestore,
```

```
    private val auth: FirebaseAuth,
) : BaseRepository {

    override fun fetchCurrentUserId(): Result<String> {
        return auth.currentUser?.uid?.let { Result.success(it) }
            ?: Result.failure(IllegalStateException("User not logged in"))
    }

    override suspend fun createDocument(
        collectionId: String,
        data: Any
    ): Result<String> = try {
        val reviewRef = database.collection(collectionId).add(data).await()
        Result.success(reviewRef.id)
    } catch (e: Exception) {
        Result.failure(e)
    }

    override suspend fun fetchDocument(
        collectionId: String,
        documentId: String
    ): Result<DocumentSnapshot> = try {
        Result.success(database.collection(collectionId).document(documentId).get().await())
    } catch (e: Exception) {
        Result.failure(e)
    }

    override suspend fun updateDocument(
        collectionId: String,
        documentId: String,
        updates: Map<String, Any>
    ): Result<Unit> = try {
        database.collection(collectionId).document(documentId).update(updates).await()
        Result.success(Unit)
    } catch (e: Exception) {
        Result.failure(e)
    }

    override suspend fun deleteDocument(
        collectionId: String,
        documentId: String
    ): Result<Unit> = try {
        database.collection(collectionId).document(documentId).delete().await()
        Result.success(Unit)
    } catch (e: Exception) {
        Result.failure(e)
    }

    override suspend fun setDocument(
        collectionId: String,
        documentId: String,
        data: Any
    ): Result<Unit> = try {
        database.collection(collectionId).document(documentId).set(data).await()
        Result.success(Unit)
    } catch (e: Exception) {
        Result.failure(e)
    }
}
```

```
// ui/auth/LoginActivity
@AndroidEntryPoint
class LoginActivity : AppCompatActivity() {
    private val viewModel: LoginViewModel by viewModels()
    private lateinit var binding: ActivityLoginBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        binding = ActivityLoginBinding.inflate(layoutInflater)
        setContentView(binding.root)

        setListeners()
    }

    override fun onStart() {
        super.onStart()
        if (viewModel.fetchCurrentUserId().isSuccess) {
            val intent = Intent(this, MainActivity::class.java)
            startActivity(intent)
            finish()
        }
    }

    private fun setListeners() {
        binding.textViewNavToRegister.setOnClickListener {
            val intent = Intent(this, RegisterActivity::class.java)
            startActivity(intent)
        }

        binding.buttonLogin.setOnClickListener {
            val email = binding.editTextLoginEmail.text.toString()
            val password = binding.editTextLoginPassword.text.toString()

            lifecycleScope.launch {
                viewModel.login(email, password).fold(
                    onSuccess = {
                        processSingIn()
                    },
                    onFailure = {
                        binding.inputLayoutLoginEmail.error =
                            getString(R.string.error_wrong_email_or_password)
                        binding.inputLayoutLoginPassword.error =
                            getString(R.string.error_wrong_email_or_password)
                    }
                )
            }
        }

        binding.editTextLoginEmail.setOnFocusChangeListener { _, hasFocus ->
            if (!hasFocus) {
                val email = binding.editTextLoginEmail.text.toString()
                when (viewModel.validateEmail(email)) {
                    "email has wrong pattern" -> binding.inputLayoutLoginEmail.error =
                        getString(R.string.error_wrong_email_pattern)
                }

            }
        }
```

```
        binding.editTextLoginEmail.doAfterTextChanged {
            binding.inputLayoutLoginEmail.error = null
        }

        binding.editTextLoginPassword.setOnFocusChangeListener { _, hasFocus ->
            if (!hasFocus) {
                val password = binding.editTextLoginPassword.text.toString()
                when (viewModel.validatePassword(password)) {
                    "password < 6" -> binding.inputLayoutLoginPassword.error =
                        getString(R.string.error_password_length)
                }
            }
        }

        binding.editTextLoginPassword.doAfterTextChanged {
            binding.inputLayoutLoginPassword.error = null
        }
    }

    private fun processSingIn() {
        lifecycleScope.launch {
            viewModel.checkUserStatus().fold(
                onSuccess = { role ->
                    if (role == "admin") {
                        val intent = Intent(this@LoginActivity, AdminActivity::class.java)
                        startActivity(intent)
                    } else {
                        val intent = Intent(this@LoginActivity, MainActivity::class.java)
                        startActivity(intent)
                    }
                },
                onFailure = {
                    Toast.makeText(this@LoginActivity, it.localizedMessage, Toast.LENGTH_LONG).show()
                    return@launch
                }
            )
        }
    }
}

// ui/auth/LoginViewModel
@HiltViewModel
class LoginViewModel @Inject constructor(
    private val repository: AuthRepository
) : ViewModel() {

    fun fetchCurrentUserId(): Result<String> {
        return repository.fetchCurrentUserId()
    }

    suspend fun login(email: String, password: String): Result<Unit> {
        return if (isValidInput(email, password)) {
            repository.login(email, password)
        } else {
            Result.failure(IllegalArgumentException("Not valid input"))
        }
    }

    suspend fun checkUserStatus(): Result<String> {
        val userId = fetchCurrentUserId().getOrElse { return Result.failure(it) }
```

```
        return repository.currentUserRole(userId)
    }

    private fun isValidInput(email: String, password: String): Boolean {
        return (validatePassword(password).isNullOrEmpty() && validateEmail(email).isNullOrEmpty())
    }

    fun validatePassword(password: String): String? {
        return if (password.length < 6) {
            "password < 6"
        } else null
    }

    fun validateEmail(email: String): String? {
        return if (!Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
            "email has wrong pattern"
        } else null
    }
}

// data/repository/AuthRepositoryImp
class AuthRepositoryImp(
    private val auth: FirebaseAuth,
    private val baseRepository: BaseRepository
) : AuthRepository {

    override fun fetchCurrentUserId(): Result<String> {
        return baseRepository.fetchCurrentUserId()
    }

    override suspend fun login(email: String, password: String): Result<Unit> = try {
        auth.signInWithEmailAndPassword(email, password).await()
        Result.success(Unit)
    } catch (e: Exception) {
        Result.failure(e)
    }

    override suspend fun currentUserRole(userId: String): Result<String> {
        val resultDoc = baseRepository.fetchDocument("users", userId).getOrElse {
            return Result.failure(it)
        }
        val role = resultDoc.getString("role") ?: return Result.failure(IllegalStateException("Role field is missing in
database"))
        return Result.success(role)
    }

    override suspend fun register(name: String, email: String, password: String): Result<Unit> {
        val userId = createUser(email, password).getOrElse {
            return Result.failure(it)
        }

        val user = CreateUser(name = name, email = email)
        return baseRepository.setDocument(
            collectionId = "users",
            documentId = userId,
            data = user,
        )
    }

    private suspend fun createUser(email: String, password: String): Result<String> = try {
```

```
            val user = auth.createUserWithEmailAndPassword(email, password).await()
            val uid = user.user?.uid
            if (uid != null) {
                Result.success(uid)
            } else {
                Result.failure(IllegalStateException("UID is null"))
            }
        } catch (e: Exception) {
            Result.failure(e)
        }
    }

    // ui/auth/RegisterActivity
    @AndroidEntryPoint
    class RegisterActivity : AppCompatActivity() {
        private lateinit var binding: ActivityRegisterBinding
        private val viewModel: RegisterViewModel by viewModels()

        override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            enableEdgeToEdge()
            binding = ActivityRegisterBinding.inflate(layoutInflater)
            setContentView(binding.root)

            setListeners()

        }

        private fun setListeners() {
            binding.textViewNavToLogin.setOnClickListener {
                val intent = Intent(this, LoginActivity::class.java)
                startActivity(intent)
            }

            binding.buttonRegister.setOnClickListener { onRegisterClicked() }

            binding.editTextRegisterEmail.setOnFocusChangeListener { _, hasFocus ->
                if (!hasFocus) {
                    val email = binding.editTextRegisterEmail.text.toString()
                    when (viewModel.validateEmail(email)) {
                        "email has wrong pattern" -> binding.inputLayoutRegisterEmail.error =
                            getString(R.string.error_wrong_email_pattern)
                    }
                }
            }

            binding.editTextRegisterEmail.doAfterTextChanged {
                binding.inputLayoutRegisterEmail.error = null
            }

            binding.editTextRegisterUsername.setOnFocusChangeListener { _, hasFocus ->
                if (!hasFocus) {
                    val username = binding.editTextRegisterUsername.text.toString()
                    when (viewModel.validateUsername(username)) {
                        "username < 3" -> binding.inputLayoutRegisterUsername.error =
                            getString(R.string.error_username_length)
                    }
                }
            }
```

```
        binding.editTextRegisterUsername.doAfterTextChanged {
            binding.inputLayoutRegisterUsername.error = null
        }

        binding.editTextRegisterPassword.setOnFocusChangeListener { _, hasFocus ->
            if (!hasFocus) {
                val password = binding.editTextRegisterPassword.text.toString()
                when (viewModel.validatePassword(password)) {
                    "password < 6" -> binding.inputLayoutRegisterPassword.error =
                        getString(R.string.error_password_length)
                }
            }
        }

        binding.editTextRegisterPassword.doAfterTextChanged {
            binding.inputLayoutRegisterPassword.error = null
        }

        binding.editTextRegisterConfirmPassword.setOnFocusChangeListener { _, hasFocus ->
            if (!hasFocus) {
                val password = binding.editTextRegisterConfirmPassword.text.toString()
                when (viewModel.validatePassword(password)) {
                    "password < 6" -> binding.inputLayoutRegisterConfirmPassword.error =
                        getString(R.string.error_password_length)
                }
            }
        }

        binding.editTextRegisterConfirmPassword.doAfterTextChanged {
            binding.inputLayoutRegisterConfirmPassword.error = null
        }
    }

    private fun onRegisterClicked() {
        val username = binding.editTextRegisterUsername.text.toString()
        val email = binding.editTextRegisterEmail.text.toString()
        val password = binding.editTextRegisterPassword.text.toString()
        val confirmPassword = binding.editTextRegisterConfirmPassword.text.toString()
        when {
            username.isEmpty() -> {
                binding.inputLayoutRegisterUsername.error = getString(R.string.error_empty_field)
                return
            }

            email.isEmpty() -> {
                binding.inputLayoutRegisterEmail.error = getString(R.string.error_empty_field)
                return
            }

            password.isEmpty() -> {
                binding.inputLayoutRegisterPassword.error = getString(R.string.error_empty_field)
                return
            }

            confirmPassword.isEmpty() -> {
                binding.inputLayoutRegisterConfirmPassword.error =
                    getString(R.string.error_empty_field)
                return
            }
        }
```

```
        when (viewModel.isValidInput(username, email, password, confirmPassword)) {
            "passwords not match" -> {
                binding.inputLayoutRegisterPassword.error =
                    getString(R.string.error_match_passwords)
                binding.inputLayoutRegisterConfirmPassword.error =
                    getString(R.string.error_match_passwords)
            }

            "check inputs" -> {
                Toast.makeText(this, getString(R.string.error_check_fields), Toast.LENGTH_LONG)
                    .show()
            }
        }
        lifecycleScope.launch {
            viewModel.register(username, email, password).fold(
                onSuccess = {
                    val intent = Intent(this@RegisterActivity, LoginActivity::class.java)
                    startActivity(intent)
                    finish()
                },
                onFailure = {
                    Toast.makeText(this@RegisterActivity, it.localizedMessage, Toast.LENGTH_LONG)
                        .show()
                }
            )
        }
    }
}

// ui/auth/RegisterViewModel
@HiltViewModel
class RegisterViewModel @Inject constructor(
    private val repository: AuthRepository
) : ViewModel() {

    fun isValidInput(username: String, email: String, password: String, confirmPassword: String): String? {
        if (password != confirmPassword) return "passwords not match"
        if (validatePassword(password).isNullOrEmpty() && validateEmail(email).isNullOrEmpty() &&
validateUsername(username).isNullOrEmpty()) {
            return "check inputs"
        }
        return null
    }

    fun validatePassword(password: String): String? {
        return if (password.length < 6) {
            return "password < 6"
        } else null
    }

    fun validateUsername(username: String): String? {
        return if (username.length < 3) {
            return "username < 3"
        } else null
    }

    fun validateEmail(email: String): String? {
        return if (!Patterns.EMAIL_ADDRESS.matcher(email).matches()) {
            return "email has wrong pattern"
        } else null
```

```
    }

    suspend fun register(username: String, email: String, password: String): Result<Unit> {
        return repository.register(username, email, password)
    }
}

// ui/MainActivity
@AndroidEntryPoint
class MainActivity : AppCompatActivity(), OnItemSelectedListener {

    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)
        binding.bottomNav.setOnItemSelectedListener(this)
        binding.bottomNav.selectedItemId = R.id.nav_home
    }

    override fun onNavigationItemSelected(item: MenuItem) = when (item.itemId) {
        R.id.nav_home -> onHomeClicked()
        R.id.nav_search -> onSearchClicked()
        R.id.nav_library -> onLibraryClicked()
        R.id.nav_settings -> onSettingsClicked()
        else -> false
    }

    private fun onHomeClicked(): Boolean {
        supportFragmentManager.commit {
            replace(R.id.frame_content, HomeFragment())
        }
        return true
    }

    private fun onSearchClicked(): Boolean {
        supportFragmentManager.commit {
            replace(R.id.frame_content, SearchFragment())
        }
        return true
    }

    private fun onLibraryClicked(): Boolean {
        supportFragmentManager.commit {
            replace(R.id.frame_content, LibraryFragment())
        }
        return true
    }

    private fun onSettingsClicked(): Boolean {
        supportFragmentManager.commit {
            replace(R.id.frame_content, SettingsFragment())
        }
        return true
    }
}

// ui/home/HomeFragment
```

```
@AndroidEntryPoint
class HomeFragment : Fragment() {

    private lateinit var binding: FragmentHomeBinding
    private lateinit var newBooksAdapter: BookAdapter
    private lateinit var mysteryBooksAdapter: BookAdapter
    private val viewModel: HomeViewModel by viewModels()

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
    ): View {
        binding = FragmentHomeBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
        super.onViewCreated(view, savedInstanceState)
        buildNewBooks()
        buildMysteryBooks()
    }

    private fun buildNewBooks() {
        newBooksAdapter = BookAdapter(emptyList())
        binding.rvNewBooks.adapter = newBooksAdapter
        binding.rvNewBooks.addItemDecoration(DefaultItemDecorator(24))

        newBooksAdapter.setOnItemClickListener(object : BookAdapter.OnItemClickListener {
            override fun onItemClick(position: Int) {
                val book = newBooksAdapter.getItem(position)
                val intent = Intent(context, BookDetailsActivity::class.java)
                intent.putExtra("Book", book)
                startActivity(intent)
            }
        })

        lifecycleScope.launch {
            viewModel.getLatestBooks()
                .flowWithLifecycle(viewLifecycleOwner.lifecycle, Lifecycle.State.STARTED)
                .collectLatest { books ->
                    newBooksAdapter.setBooks(books)
                }
        }
    }

    private fun buildMysteryBooks() {
        mysteryBooksAdapter = BookAdapter(emptyList())
        binding.rvMysteryBooks.adapter = mysteryBooksAdapter
        binding.rvMysteryBooks.addItemDecoration(DefaultItemDecorator(24))

        mysteryBooksAdapter.setOnItemClickListener(object : BookAdapter.OnItemClickListener {
            override fun onItemClick(position: Int) {
                val book = mysteryBooksAdapter.getItem(position)
                val intent = Intent(context, BookDetailsActivity::class.java)
                intent.putExtra("Book", book)
                startActivity(intent)
            }
        })

        lifecycleScope.launch {
            viewModel.getMysteryBooks().collectLatest { books ->
```

```
                mysteryBooksAdapter.setBooks(books)
            }
        }
    }
}

// ui/home/HomeViewModel
@HiltViewModel
class HomeViewModel @Inject constructor(
    private val repository: HomeRepository
): ViewModel() {

    fun getLatestBooks(): Flow<List<Book>> {
        return repository.getLatestBooks()
    }
    fun getMysteryBooks(): Flow<List<Book>> {
        return repository.getMysteryBooks()
    }
}

// data/repository/HomeRepositoryImp
class HomeRepositoryImp(
    private val database: FirebaseFirestore,
): HomeRepository {
    override fun getLatestBooks(): Flow<List<Book>> {
        return database.collection("books")
            .orderBy("createAt", Query.Direction.DESCENDING)
            .limit(20).snapshots()
            .map { snapshot -> snapshot.documents.mapNotNull { doc -> parseBook(doc) } }
    }

    override fun getMysteryBooks(): Flow<List<Book>> {
        return database.collection("books")
            .whereArrayContains("genres", "mystery")
            .limit(20).snapshots()
            .map { snapshot -> snapshot.documents.mapNotNull { doc -> parseBook(doc) } }
    }

    private fun parseBook(doc: DocumentSnapshot): Book {
        val reviews =
            (doc.get("reviews") as? List<*>)?.filterIsInstance<String>() ?: emptyList()
        val starRatings = (doc.get("starRatings") as? Map<*, *>)?.mapNotNull {
            val star = (it.key as? String)?.toIntOrNull()
            val count = (it.value as? Number)?.toInt()
            if (star != null && count != null) star to count else null
        }?.toMap() ?: emptyMap()

        return Book(
            bookId = doc.id,
            image = doc.getString("image") ?: "",
            title = doc.getString("title") ?: "",
            author = doc.getString("author") ?: "",
            rating = doc.getDouble("rating")?.toFloat() ?: 0f,
            description = doc.getString("description") ?: "",
            genres = (doc.get("genres") as? List<*>)?.filterIsInstance<String>() ?: emptyList(),
            reviews = reviews,
            reviewCount = doc.getLong("reviewCount") ?: 0L,
            fileUrl = doc.getString("fileUrl") ?: "",
            starRatings = starRatings
        )
```

```
    }
}

// ui/book/BookDetailsActivity
@AndroidEntryPoint
class BookDetailsActivity : AppCompatActivity() {

    private val viewModel: BookDetailsViewModel by viewModels()
    private lateinit var binding: ActivityBookDetailsBinding
    private lateinit var book: Book

    private var reviewText = ""
    private var existingReviewId = ""
    private var reviewScore = 0f

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityBookDetailsBinding.inflate(layoutInflater)
        setContentView(binding.root)
        ViewCompat.setOnApplyWindowInsetsListener(binding.toolbarBookDescription) { v, insets ->
            val topInset = insets.getInsets(WindowInsetsCompat.Type.systemBars()).top
            v.updatePadding(top = topInset)
            insets
        }
        extractBookFromIntent()
        setupUI()
        setupListeners()
        setupFragmentManagers()
        displayReviews()

        lifecycleScope.launch {
            viewModel.getUserReview(book.bookId).fold(
                onSuccess = {
                    showUserReview(it)
                },
                onFailure = {
                    if (it !is NoSuchElementException) {
                        showToast(it.localizedMessage ?: "Unknown message")
                    }
                }
            )
        }
    }

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        menuInflater.inflate(R.menu.toolbar_book_details, menu)

        val saveItem = menu.findItem(R.id.button_book_details_saved)
        val moreItem = menu.findItem(R.id.menu_other_options)

        val whiteColor = ContextCompat.getColor(this, android.R.color.white)
        saveItem.icon?.setTint(whiteColor)
        moreItem.icon?.setTint(whiteColor)
        lifecycleScope.launch {
            viewModel.isSaved(book.bookId).onSuccess {
                val iconRes = if (it) R.drawable.bookmark_saved else R.drawable.icon_bookmark_border
                val icon = ContextCompat.getDrawable(this@BookDetailsActivity, iconRes)
                icon?.setTint(whiteColor)
                saveItem.setIcon(icon)
            }.onFailure {
```

```
                showToast(it.localizedMessage ?: "Unknown error")
            }
        }

        return true
    }

    override fun onOptionsItemSelected(item: MenuItem): Boolean {
        when (item.itemId) {
            R.id.button_book_details_saved -> {
                lifecycleScope.launch {
                    viewModel.changeState(book.bookId).onSuccess {
                        val whiteColor = ContextCompat.getColor(this@BookDetailsActivity, android.R.color.white)
                        val iconRes = if (it == "saved") R.drawable.bookmark_saved else
R.drawable.icon_bookmark_border
                        val icon = ContextCompat.getDrawable(this@BookDetailsActivity, iconRes)
                        icon?.setTint(whiteColor)
                        item.setIcon(icon)
                    }.onFailure {
                        showToast(it.localizedMessage ?: "Failed to update saved state")
                    }
                }
            }
        }
        return super.onOptionsItemSelected(item)
    }


    private fun extractBookFromIntent() {
        book = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
            intent.getParcelableExtra("Book", Book::class.java)
        } else {
            @Suppress("DEPRECATION")
            intent.getParcelableExtra("Book") as? Book
        } ?: run {
            showToast("Error: book not found")
            finish()
            return
        }
        viewModel.computeRatingProgress(book)
    }

    private fun setupUI() {
        with(binding) {
            ivBookDescriptionCover.load(book.image)
            tvBookDescriptionTitle.text = book.title
            tvBookDescriptionAuthor.text = book.author
            rbBookDescriptionRating.rating = book.rating
            tvRatingCount.text = "(${book.reviewCount})"
            tvBookDescriptionDescription.text = book.description

            if (book.genres.isNotEmpty()) {
                setupCategoryChips(book.genres)
            } else {
                tvBookDescriptionGenres.visibility = View.GONE
                chipGroupBookDescriptionGenres.visibility = View.GONE
            }
        }

        // Set the toolbar as the action bar
```

```
        setSupportActionBar(binding.toolbarBookDescription)
        // Enable the back button
        supportActionBar?.setDisplayHomeAsUpEnabled(true)
        supportActionBar?.setDisplayShowHomeEnabled(true)
        binding.toolbarBookDescription.setNavigationOnClickListener {
            onBackPressedDispatcher.onBackPressed()
        }
    }

    private fun setupCategoryChips(genres: List<String>) {
        genres.forEach { genre ->
            val genreName = GenresHelper.getGenreName(this, genre)
            binding.chipGroupBookDescriptionGenres.addView(
                Chip(this).apply {
                    text = genreName
                    isCheckable = false
                    setOnClickListener {
                        val intent = Intent(context, SearchGenresActivity::class.java)
                        intent.putExtra("Genre", genre)
                        startActivity(intent)
                    }
                }
            )
        }
    }

    private fun setupListeners() {
        binding.buttonBookDescriptionRead.setOnClickListener {
            if (book.fileUrl.isEmpty()) {
                showToast("Not specified file path")
                return@setOnClickListener
            }

            val intent = Intent(this, ReadiumActivity::class.java).apply {
                putExtra("EPUB_URL", book.fileUrl)
            }
            startActivity(intent)
        }

        binding.rbBookDescriptionRatingInteract.setOnRatingBarChangeListener { _, rating, fromUser ->
            if (fromUser) {
                val dialog =
                    ReviewSubmitDialogFragment(rating, book.bookId, reviewText, existingReviewId)
                dialog.show(supportFragmentManager, ReviewSubmitDialogFragment.TAG)
                binding.rbBookDescriptionRatingInteract.rating = 0f
            }
        }

        binding.rbBookDescriptionRatingInteract.setOnRatingBarChangeListener { _, rating, fromUser ->
            if (fromUser) {
                val dialog =
                    ReviewSubmitDialogFragment(rating, book.bookId, reviewText, existingReviewId)
                dialog.show(supportFragmentManager, ReviewSubmitDialogFragment.TAG)
                binding.rbBookDescriptionRatingInteract.rating = 0f
            }
        }

        binding.buttonBookDescriptionReview.setOnClickListener {
            val dialog =
                ReviewSubmitDialogFragment(reviewScore, book.bookId, reviewText, existingReviewId)
```

```
            dialog.show(supportFragmentManager, ReviewSubmitDialogFragment.TAG)
        }

        binding.ibBookDescriptionMenuDeleteReview.setOnClickListener { view ->
            PopupMenu(this@BookDetailsActivity, view).apply {
                menuInflater.inflate(R.menu.menu_review_delete, menu)

                setOnMenuItemClickListener { menuItem ->
                    when (menuItem.itemId) {
                        R.id.action_delete_review -> {
                            lifecycleScope.launch {
                                viewModel.deleteUserReview(existingReviewId).getOrElse {
                                    showToast(it.message ?: "Unknown message")
                                    return@launch
                                }
                                hideUserReviewSection()
                            }
                            true
                        }

                        else -> false
                    }
                }
                show()
            }
        }

        binding.tvBookDescriptionMoreReviews.setOnClickListener {
            val dialog = ReviewListDialogFragment(book.bookId)
            dialog.show(supportFragmentManager, ReviewListDialogFragment.TAG)
        }
    }

    private fun hideUserReviewSection() {
        binding.layoutBookDescriptionUserReview.visibility = View.GONE
        binding.rbBookDescriptionRatingInteract.visibility = View.VISIBLE
        binding.tvBookDescriptionRating.text = getString(R.string.rate_this_book)
        binding.buttonBookDescriptionReview.text = getString(R.string.button_create_review)
        reviewText = ""
        existingReviewId = ""
        reviewScore = 0f
    }

    private fun showUserReview(review: UserReviewUI) {
        with(binding) {
            // Make review section visible
            layoutBookDescriptionUserReview.visibility = View.VISIBLE
            rbBookDescriptionRatingInteract.visibility = View.GONE

            // Display proper text for button and title
            tvBookDescriptionRating.text = getString(R.string.your_review)
            buttonBookDescriptionReview.text = getString(R.string.button_edit_review)

            // Display name of the user
            tvBookDescriptionUsername.text = review.username

            // Display review score
            rbBookDescriptionUserReviewIndicator.rating = review.rating

            // Get and format timestamp of the review
```

```
            tvBookDescriptionUserReviewDate.text = SimpleDateFormat(
                "dd MMM yyyy",
                Locale.getDefault()
            ).format(review.timestamp.toDate())

            // Load profile image
            ivBookDescriptionProfileRound.load(review.profileImage) {
                crossfade(true)
                placeholder(R.drawable.default_profile_image_40)
                error(R.drawable.default_profile_image_40)
            }

            // Display review text
            tvBookDescriptionUserReviewText.text = review.text
            tvBookDescriptionUserReviewText.makeExpandable()
        }

        // Store review text and ID for editing
        reviewText = review.text
        existingReviewId = review.reviewId
        reviewScore = review.rating
    }

    private fun setupFragmentManagers() {
        supportFragmentManager.setFragmentResultListener("review_result", this) { _, bundle ->
            val review = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
                bundle.getParcelable("review", UserReviewUI::class.java)
            } else {
                @Suppress("DEPRECATION")
                bundle.getParcelable("review")
            } ?: return@setFragmentResultListener


            lifecycleScope.launch {
                val userData = viewModel.getUserData().getOrElse {
                    showToast(it.message ?: "Unknown message")
                    return@launch
                }
                showUserReview(
                    UserReviewUI(
                        username = userData.name,
                        profileImage = userData.profileImage,
                        reviewId = review.reviewId,
                        timestamp = review.timestamp,
                        rating = review.rating,
                        text = review.text,
                    )
                )
            }

        }
    }


    private fun displayReviews() {
        with(binding) {
            if (book.reviewCount != 0L) {
                llBookDescriptionRatingDetails.visibility = View.VISIBLE
                llBookDescriptionReviewContainer.visibility = View.VISIBLE
                loadRatingDetails()
```

```
            loadLatestReviews()

            tvBookDescriptionAverageRating.text = when {
                book.reviewCount == 1L -> getString(
                    R.string.rating_single,
                    book.rating,
                    book.reviewCount
                )

                book.reviewCount in 2..4 -> getString(
                    R.string.rating_two_to_four,
                    book.rating,
                    book.reviewCount
                )

                else -> getString(R.string.rating_multiple, book.rating, book.reviewCount)
            }
            tvBookDescriptionMoreReviews.visibility =
                if (book.reviewCount > 3) View.VISIBLE else View.GONE
        } else {
            llBookDescriptionRatingDetails.visibility = View.GONE
            llBookDescriptionReviewContainer.visibility = View.GONE
        }
    }
}

private fun loadRatingDetails() {
    viewModel.ratingProgress.observe(this) { progresses ->
        val progressBars = listOf(
            binding.pbBookDescription5StarProgress,
            binding.pbBookDescription4StarProgress,
            binding.pbBookDescription3StarProgress,
            binding.pbBookDescription2StarProgress,
            binding.pbBookDescription1StarProgress
        )

        progresses.forEachIndexed { index, value ->
            progressBars[index].progress = value
        }
    }
}

private fun loadLatestReviews() {
    val reviewIds = book.reviews.takeLast(3).reversed()

    val container = binding.llBookDescriptionReviewContainer
    val reviewViews = listOf(
        container.getChildAt(0),
        container.getChildAt(1),
        container.getChildAt(2)
    )

    reviewViews.forEach { it.visibility = View.GONE }

    reviewIds.forEachIndexed { index, reviewId ->
        if (index < reviewViews.size) {
            val reviewView = reviewViews[index]
            reviewView.visibility = View.VISIBLE

            val usernameText = reviewView.findViewById<TextView>(R.id.tv_review_username)
```

```
            val dateText = reviewView.findViewById<TextView>(R.id.tv_review_date)
            val ratingBar = reviewView.findViewById<RatingBar>(R.id.rb_review_indicator)
            val reviewContent = reviewView.findViewById<TextView>(R.id.tv_review_text)
            val profileImage = reviewView.findViewById<ImageView>(R.id.iv_review_profile_round)
            lifecycleScope.launch {
                val review = viewModel.getReview(reviewId).getOrElse {
                    showToast(it.message ?: "Unknown message")
                    return@launch
                }
                usernameText.text = review.username
                ratingBar.rating = review.rating
                dateText.text = SimpleDateFormat(
                    "dd MMM yyyy",
                    Locale.getDefault()
                ).format(review.timestamp.toDate())
                if (review.text.isNotEmpty()) {
                    reviewContent.text = review.text
                } else {
                    reviewContent.visibility = View.GONE
                }

                profileImage.load(review.profileImage) {
                    crossfade(true)
                    placeholder(R.drawable.default_profile_image_40)
                    error(R.drawable.default_profile_image_40)
                }
            }

        }
    }

    private fun showToast(message: String) {
        Toast.makeText(this, message, Toast.LENGTH_SHORT).show()
    }
}

// ui/book/BookDetailsViewModel
@HiltViewModel
class BookDetailsViewModel @Inject constructor(
    private val repository: BookDetailsRepository
) : ViewModel() {

    val ratingProgress = MutableLiveData<List<Int>>() // порядок: [5★, 4★, ..., 1★]
    private var isBookSaved = false

    fun computeRatingProgress(book: Book) {
        val reviewCount = book.reviewCount
        if (reviewCount == 0L) {
            ratingProgress.value = listOf(0, 0, 0, 0, 0)
            return
        }

        val result = (5 downTo 1).map { stars ->
            val count = book.starRatings[stars] ?: 0
            val progress = (count * 100) / reviewCount
            when {
                progress == 0L -> 0
                progress <= 4L -> 4
                else -> progress.toInt()
```

```
            }
        }

        ratingProgress.value = result
    }

    suspend fun getUserReview(bookId: String): Result<UserReviewUI> {
        return repository.fetchUserReview(bookId)
    }

    suspend fun getUserData(): Result<UserProfile> {
        return repository.getCurrentUserProfile()
    }

    suspend fun getReview(reviewId: String): Result<UserReviewUI> {
        return repository.fetchReview(reviewId)
    }

    suspend fun deleteUserReview(reviewId: String): Result<Unit> {
        return repository.deleteReview(reviewId)
    }

    suspend fun isSaved(bookId: String): Result<Boolean> {
        return repository.isSaved(bookId).map {
            isBookSaved = it
            it
        }
    }

    suspend fun changeState(bookId: String): Result<String> {
        if (isBookSaved) {
            removeFromSaved(bookId).onFailure {
                return Result.failure(it)
            }
            isBookSaved = false
            return Result.success("removed")
        } else {
            addToSaved(bookId).onFailure {
                return Result.failure(it)
            }
            isBookSaved = true
            return Result.success("saved")
        }
    }

    private suspend fun addToSaved(bookId: String): Result<Unit> {
        return repository.addToSaved(bookId)
    }

    private suspend fun removeFromSaved(bookId: String): Result<Unit> {
        return repository.removeFromSaved(bookId)
    }
}

// data/repository/BookDetailsRepositoryImp
class BookDetailsRepositoryImp(
    private val baseRepository: BaseRepository
) : BookDetailsRepository {

    override suspend fun fetchUserReview(bookId: String): Result<UserReviewUI> {
```

```
        val userId = baseRepository.fetchCurrentUserId().getOrElse {
            return Result.failure(it)
        }
        val userResult = baseRepository.fetchDocument("users", userId)
        val userDoc = userResult.getOrElse {
            return Result.failure(it)
        }
        val reviewIds = (userDoc.get("reviews") as? List<*>)?.filterIsInstance<String>()
        if (reviewIds.isNullOrEmpty()) {
            return Result.failure(NoSuchElementException("No reviews found for user"))
        }

        for (reviewId in reviewIds) {
            val reviewResult = baseRepository.fetchDocument("reviews", reviewId)
            val reviewDoc = reviewResult.getOrElse {
                return Result.failure(it)
            }
            val currentBookId = reviewDoc.getString("reviewTo")
            if (currentBookId == bookId) {
                val username = userDoc.getString("name") ?: ""
                val imageUrl = userDoc.getString("profileImage") ?: ""
                val timestamp = reviewDoc.getTimestamp("timestamp") ?: Timestamp.now()
                val rating = reviewDoc.getDouble("reviewScore")?.toFloat() ?: 0f
                val text = reviewDoc.getString("reviewText") ?: ""
                return Result.success(
                    UserReviewUI(
                        username = username,
                        profileImage = imageUrl,
                        reviewId = reviewId,
                        timestamp = timestamp,
                        rating = rating,
                        text = text
                    )
                )
            }
        }
        return Result.failure(NoSuchElementException("No reviews found for user"))
    }

    override suspend fun getCurrentUserProfile(): Result<UserProfile> {
        val userId = baseRepository.fetchCurrentUserId().getOrElse {
            return Result.failure(it)
        }

        val userDoc = baseRepository.fetchDocument("users", userId).getOrElse {
            return Result.failure(it)
        }

        val name = userDoc.getString("name") ?: ""
        val image = userDoc.getString("profileImage") ?: ""
        return Result.success(UserProfile(name, image))
    }

    override suspend fun fetchReview(reviewId: String): Result<UserReviewUI> {
        val result = baseRepository.fetchDocument("reviews", reviewId)
        val reviewDoc = result.getOrElse {
            return Result.failure(it)
        }
        val reviewerId = reviewDoc.getString("reviewFrom") ?: ""
        val timestamp = reviewDoc.getTimestamp("timestamp") ?: Timestamp.now()
```

```
        val rating = reviewDoc.getDouble("reviewScore")?.toFloat() ?: 0f
        val reviewText = reviewDoc.getString("reviewText") ?: ""
        val userResult = baseRepository.fetchDocument("users", reviewerId)
        val userDoc = userResult.getOrElse {
            return Result.failure(it)
        }
        val username = userDoc.getString("name") ?: ""
        val profileImage = userDoc.getString("profileImage") ?: ""

        return Result.success(
            UserReviewUI(
                reviewId = reviewId,
                username = username,
                profileImage = profileImage,
                timestamp = timestamp,
                rating = rating,
                text = reviewText
            )
        )
    }

    override suspend fun deleteReview(reviewId: String): Result<Unit> {
        val result = baseRepository.fetchDocument("reviews", reviewId)
        val reviewDoc = result.getOrElse {
            return Result.failure(it)
        }
        val userId = reviewDoc.getString("reviewFrom") ?: ""
        val bookId = reviewDoc.getString("reviewTo") ?: ""
        val reviewScore = reviewDoc.getLong("reviewScore") ?: 0L

        val userUpdate = mapOf("reviews" to FieldValue.arrayRemove(reviewId))
        val bookUpdates = mapOf(
            "reviews" to FieldValue.arrayRemove(reviewId),
            "reviewCount" to FieldValue.increment(-1),
            "totalRating" to FieldValue.increment(-reviewScore.toDouble()),
            "starRatings.${reviewScore.toInt()}" to FieldValue.increment(-1)
        )

        baseRepository.deleteDocument("reviews", reviewId).getOrElse {
            return Result.failure(it)
        }
        baseRepository.updateDocument("users", userId, userUpdate).getOrElse {
            return Result.failure(it)
        }
        baseRepository.updateDocument("books", bookId, bookUpdates).getOrElse {
            return Result.failure(it)
        }
        updateAverageRatings(bookId).getOrElse {
            return Result.failure(it)
        }
        return Result.success(Unit)
    }

    private suspend fun updateAverageRatings(bookId: String): Result<Unit> {
        val bookDoc = baseRepository.fetchDocument("books", bookId).getOrElse {
            return Result.failure(it)
        }
        val reviewCount = bookDoc.getLong("reviewCount") ?: 0L
        val totalRating = bookDoc.getLong("totalRating") ?: 0L
        if (reviewCount != 0L) {
```

```
            val bookUpdate = mapOf("rating" to (totalRating.toFloat() / reviewCount))
            baseRepository.updateDocument("books", bookId, bookUpdate).getOrElse {
                return Result.failure(it)
            }
        }
        return Result.success(Unit)
    }

    override suspend fun isSaved(bookId: String): Result<Boolean> {
        val uid = baseRepository.fetchCurrentUserId().getOrElse {
            return Result.failure(it)
        }
        val userDoc = baseRepository.fetchDocument("users", uid).getOrElse {
            return Result.failure(it)
        }
        val savedList = (userDoc.get("savedBookIds") as? List<*>) ?: emptyList<Any>()
        val savedIds = savedList.filterIsInstance<String>()
        return Result.success(bookId in savedIds)
    }

    override suspend fun addToSaved(bookId: String): Result<Unit> {
        val uid = baseRepository.fetchCurrentUserId().getOrElse {
            return Result.failure(it)
        }
        val userUpdate = mapOf("savedBookIds" to FieldValue.arrayUnion(bookId))
        return baseRepository.updateDocument("users", uid, userUpdate)
    }

    override suspend fun removeFromSaved(bookId: String): Result<Unit> {
        val uid = baseRepository.fetchCurrentUserId().getOrElse {
            return Result.failure(it)
        }
        val userUpdate = mapOf("savedBookIds" to FieldValue.arrayRemove(bookId))
        return baseRepository.updateDocument("users", uid, userUpdate)
    }
}

// ui/search/SearchFragment
@AndroidEntryPoint
class SearchFragment : Fragment() {
    private lateinit var binding: FragmentSearchBinding
    private lateinit var searchAdapter: SearchAdapter
    private var orderBy: String = ""
    private var descendingOrder = true
    private var minRating = 0.5f
    private var useAndLogic = true
    private var included: ArrayList<String> = arrayListOf()
    private var excluded: ArrayList<String> = arrayListOf()
    private var searchJob: Job? = null
    private val viewModel: SearchViewModel by viewModels()

    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View {
        binding = FragmentSearchBinding.inflate(inflater, container, false)
        return binding.root
    }

    override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
```

```
        super.onViewCreated(view, savedInstanceState)
        if (orderBy.isEmpty()) {
            orderBy = resources.getStringArray(R.array.order_by)[0]
        }
        setupListeners()
        lifecycleScope.launch {
            search()
        }
    }

    private fun setupListeners() {
        binding.buttonFilter.setOnClickListener {
            val dialog = FilterDialogFragment(
                orderBy,
                descendingOrder,
                minRating,
                useAndLogic,
                included,
                excluded
            )
            dialog.show(childFragmentManager, FilterDialogFragment.TAG)
        }

        binding.searchEditText.addTextChangedListener(object : TextWatcher {
            override fun beforeTextChanged(s: CharSequence?, start: Int, count: Int, after: Int) {}

            override fun onTextChanged(s: CharSequence?, start: Int, before: Int, count: Int) {}

            override fun afterTextChanged(s: Editable?) {
                searchJob?.cancel()
                searchJob = CoroutineScope(Dispatchers.Main).launch {
                    delay(1000)
                    search()
                }
            }

        })

        childFragmentManager.setFragmentResultListener(
            FilterDialogFragment.FILTER_REQUEST_KEY,
            this
        ) { _, bundle ->
            orderBy = bundle.getString(FilterDialogFragment.KEY_ORDER_BY) ?: ""
            descendingOrder = bundle.getBoolean(FilterDialogFragment.KEY_ORDER)
            minRating = bundle.getFloat(FilterDialogFragment.KEY_MIN_RATING)
            useAndLogic = bundle.getBoolean(FilterDialogFragment.KEY_AND_LOGIC)
            included =
                bundle.getStringArrayList(FilterDialogFragment.KEY_INCLUDED_GENRES) ?: arrayListOf()
            excluded =
                bundle.getStringArrayList(FilterDialogFragment.KEY_EXCLUDED_GENRES) ?: arrayListOf()
            lifecycleScope.launch {
                search()
            }
        }
    }

    private suspend fun search() {
        val translatedOrderBy = translateOrderBy(orderBy)
        val search = binding.searchEditText.text.toString()
        showList(
```

```
            viewModel.search(
                translatedOrderBy,
                descendingOrder,
                search,
                minRating,
                useAndLogic,
                included,
                excluded
            )
        )
    }

    private fun translateOrderBy(orderBy: String): String {
        val orderOptions = resources.getStringArray(R.array.order_by)
        return when (orderBy) {
            orderOptions[0] -> "createAt"
            orderOptions[1] -> "title"
            orderOptions[2] -> "rating"
            orderOptions[3] -> "reviewCount"
            else -> orderOptions[1]
        }
    }

    private fun showList(list: List<Book>) {
        if (list.isNotEmpty()) {
            binding.rvSearchDisplay.visibility = View.VISIBLE
            binding.tvSearchError.visibility = View.GONE
            searchAdapter = SearchAdapter(list)
            binding.rvSearchDisplay.adapter = searchAdapter
            searchAdapter.setOnItemClickListener(object : SearchAdapter.OnItemClickListener {
                override fun onItemClick(position: Int) {
                    val intent = Intent(context, BookDetailsActivity::class.java)
                    intent.putExtra("Book", list[position])
                    startActivity(intent)
                }
            })
        } else {
            binding.rvSearchDisplay.visibility = View.GONE
            binding.tvSearchError.visibility = View.VISIBLE
        }
    }
}

// ui/search/SearchViewModel
@HiltViewModel
class SearchViewModel @Inject constructor(
    private val repository: SearchRepository
) : ViewModel() {

    suspend fun search(
        orderBy: String,
        descendingOrder: Boolean,
        search: String,
        minRating: Float,
        useAndLogic: Boolean,
        included: List<String>,
        excluded: List<String>
    ): List<Book> {
        return repository.search(
            orderBy,
```

```
            descendingOrder,
            search,
            minRating,
            useAndLogic,
            included,
            excluded
        )
    }
}

// data/repository/SearchRepository
class SearchRepositoryImp(
    private val database: FirebaseFirestore,
) : SearchRepository {

    override suspend fun search(
        orderBy: String,
        descendingOrder: Boolean,
        search: String,
        minRating: Float,
        useAndLogic: Boolean,
        included: List<String>,
        excluded: List<String>,
    ): List<Book> {
        val snapshot = createQuerySnapshot(included, minRating, orderBy, descendingOrder)
        return processSearchResults(
            snapshot,
            search = search,
            excluded = excluded,
            included = included,
            useAndLogic = useAndLogic
        )
    }

    private suspend fun createQuerySnapshot(
        included: List<String>,
        minRating: Float,
        orderBy: String,
        descendingOrder: Boolean
    ): QuerySnapshot {
        var query: Query = database.collection("books")
        if (included.isNotEmpty()) {
            query = query.whereArrayContainsAny("genres", included)
        }
        if (minRating >= 1) {
            query = query.whereGreaterThanOrEqualTo("rating", minRating)
        }
        return query.orderBy(
            orderBy,
            if (descendingOrder) Query.Direction.DESCENDING else Query.Direction.ASCENDING
        ).get().await()
    }

    private fun processSearchResults(
        snapshot: QuerySnapshot,
        search: String,
        excluded: List<String>,
        included: List<String>,
        useAndLogic: Boolean
    ): List<Book> {
```

```
        val bookList = mutableListOf<Book>()
        for (doc in snapshot) {
            val title = doc.getString("title") ?: ""
            val author = doc.getString("author") ?: ""
            if (!title.contains(search, true) && !author.contains(search, true)) continue

            val genres = doc.get("genres") as? List<*> ?: emptyList<Any>()
            val genreStrings = genres.filterIsInstance<String>()
            val excludeOK = excluded.none { genreStrings.contains(it) }

            if (useAndLogic) {
                val includeOK = included.all { genreStrings.contains(it) }
                if (includeOK && excludeOK) bookList.add(getBookInfo(doc))
            } else {
                if (excludeOK) bookList.add(getBookInfo(doc))
            }
        }
        return bookList
    }

    private fun getBookInfo(doc: DocumentSnapshot): Book {
        val reviews = (doc.get("reviews") as? List<*>)?.filterIsInstance<String>() ?: emptyList()
        val starRatings = when (val data = doc.get("starRatings")) {
            is Map<*, *> -> data.mapNotNull { (key, value) ->
                val star = (key as? String)?.toIntOrNull()
                val count = (value as? Number)?.toInt()
                if (star != null && count != null) star to count else null
            }.toMap()

            else -> emptyMap()
        }


        return Book(
            bookId = doc.id,
            image = doc.getString("image") ?: "",
            title = doc.getString("title") ?: "",
            author = doc.getString("author") ?: "",
            rating = doc.getDouble("rating")?.toFloat() ?: 0f,
            description = doc.getString("description") ?: "",
            genres = (doc.get("genres") as? List<*>)?.filterIsInstance<String>()
                ?: emptyList(),
            reviews = reviews,
            reviewCount = doc.getLong("reviewCount") ?: 0L,
            fileUrl = doc.getString("fileUrl") ?: "",
            starRatings = starRatings,
        )
    }
}

// ui/bookDetails/review/ReviewSubmitDialogFragment
@AndroidEntryPoint
class ReviewSubmitDialogFragment(
    private var reviewScore: Float,
    private val bookId: String,
    private var reviewText: String,
    private var existingReviewId: String,
) : DialogFragment() {

    private lateinit var binding: DialogFragmentReviewEditBinding
```

```
private val viewModel: ReviewSubmitViewModel by viewModels()

private val startingScore = reviewScore

override fun onViewCreated(view: View, savedInstanceState: Bundle?) {
    super.onViewCreated(view, savedInstanceState)

    binding.ratingBarReview.rating = reviewScore
    binding.editTextReview.setText(reviewText)

    binding.toolbarReview.setNavigationOnClickListener { dismiss() }
    binding.buttonConfirmReview.setOnClickListener { submitReview() }
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setStyle(STYLE_NORMAL, android.R.style.ThemeOverlay)
}

override fun onCreateView(
    inflater: LayoutInflater,
    container: ViewGroup?,
    savedInstanceState: Bundle?
): View {
    binding = DialogFragmentReviewEditBinding.inflate(inflater, container, false)
    return binding.root
}

private fun submitReview() {
    reviewScore = binding.ratingBarReview.rating
    reviewText = binding.editTextReview.text.toString()

    if (reviewScore == 0f) {
        showSnackbar("Please rate this book from 1 to 5")
    }
    lifecycleScope.launch {
        val review = viewModel.submitUserReview(
            reviewScore,
            reviewText,
            bookId,
            existingReviewId,
            startingScore
        ).getOrElse {
            showSnackbar(it.localizedMessage ?: "Unknown message")
            return@launch
        }
        sendResultBack(review)
    }
}

private fun sendResultBack(review: UserReviewUI) {
    val bundle = Bundle().apply {
        putParcelable("review", review)
    }
    parentFragmentManager.setFragmentResult("review_result", bundle)
    dismiss()
}

private fun showSnackbar(message: String) {
    view?.let { Snackbar.make(it, message, Snackbar.LENGTH_LONG).show() }
```

```
    }

    companion object {
        const val TAG = "ReviewEditDialog"
    }
}

// ui/bookDetails/review/ReviewSubmitViewModel
@HiltViewModel
class ReviewSubmitViewModel @Inject constructor(
    private val repository: ReviewRepository
) : ViewModel() {

    suspend fun submitUserReview(
        reviewScore: Float,
        reviewText: String,
        bookId: String,
        existingReviewId: String,
        startingScore: Float
    ): Result<UserReviewUI> {

        val userId = repository.fetchCurrentUserId().getOrElse {
            return Result.failure(it)
        }
        val review = CreateReview(
            reviewFrom = userId,
            reviewTo = bookId,
            reviewScore = reviewScore,
            reviewText = reviewText,
            timestamp = Timestamp.now(),
        )
        return repository.submitUserReview(existingReviewId, review, startingScore)
    }
}


// data/repository/ReviewRepositoryImp
class ReviewRepositoryImp(
    private val database: FirebaseFirestore,
    private val baseRepository: BaseRepository
) : ReviewRepository {

    override fun fetchCurrentUserId(): Result<String> {
        return baseRepository.fetchCurrentUserId()
    }

    override suspend fun fetchReviews(bookId: String): Result<List<Review>> {
        val reviews = mutableListOf<Review>()
        val query = database.collection("reviews").whereEqualTo("reviewTo", bookId).get().await()
        for (reviewDoc in query.documents) {
            val timestamp = reviewDoc.getTimestamp("timestamp") ?: Timestamp.now()
            val date = SimpleDateFormat(
                "dd MMM yyyy",
                Locale.getDefault()
            ).format(timestamp.toDate())

            val reviewerId = reviewDoc.getString("reviewFrom") ?: continue

            val userDoc = baseRepository.fetchDocument("users", reviewerId).getOrElse {
                return Result.failure(it)
```

```
            }
            val reviewerName = userDoc.getString("name") ?: ""
            val reviewerAvatar = userDoc.getString("profileImage") ?: ""

            reviews.add(
                Review(
                    reviewerName = reviewerName,
                    reviewerAvatar = reviewerAvatar,
                    rating = reviewDoc.getDouble("reviewScore")?.toFloat() ?: 0f,
                    text = reviewDoc.getString("reviewText") ?: "",
                    date = date,
                )
            )
        }
        return Result.success(reviews)
    }

    override suspend fun submitUserReview(
        reviewId: String,
        review: CreateReview,
        startingScore: Float
    ): Result<UserReviewUI> {
        var currentReviewId = reviewId
        if (currentReviewId.isNotEmpty()) {
            setReview(currentReviewId, review, startingScore).getOrElse {
                return Result.failure(it)
            }
        } else {
            currentReviewId = createReview(review).getOrElse {
                return Result.failure(it)
            }
        }
        val userDoc = baseRepository.fetchDocument("users", review.reviewFrom).getOrElse {
            return Result.failure(it)
        }
        val username = userDoc.getString("name") ?: ""
        val imageUrl = userDoc.getString("profileImage") ?: ""

        updateAverageRatings(review.reviewTo).getOrElse {
            return Result.failure(it)
        }
        return Result.success(
            UserReviewUI(
                reviewId = currentReviewId,
                username = username,
                profileImage = imageUrl,
                timestamp = review.timestamp,
                rating = review.reviewScore,
                text = review.reviewText
            )
        )
    }

    private suspend fun createReview(review: CreateReview): Result<String> {
        val reviewId = baseRepository.createDocument("reviews", review).getOrElse {
            return Result.failure(it)
        }

        val userId = review.reviewFrom
        val bookId = review.reviewTo
```

```
        val starKey = review.reviewScore.toInt().toString()

        val userUpdate = mapOf("reviews" to FieldValue.arrayUnion(reviewId))
        val bookUpdates = mapOf(
            "reviews" to FieldValue.arrayUnion(reviewId),
            "starRatings.$starKey" to FieldValue.increment(1),
            "totalRating" to FieldValue.increment(review.reviewScore.toLong()),
            "reviewCount" to FieldValue.increment(1),
        )

        baseRepository.updateDocument("users", userId, userUpdate).getOrElse {
            return Result.failure(it)
        }
        baseRepository.updateDocument("books", bookId, bookUpdates).getOrElse {
            return Result.failure(it)
        }
        return Result.success(reviewId)
    }

    private suspend fun setReview(
        reviewId: String,
        review: CreateReview,
        startingScore: Float
    ): Result<Unit> {
        val scoreChangedBy = (review.reviewScore - startingScore)

        if (scoreChangedBy != 0f) {
            val bookUpdates = mapOf(
                "starRatings.${review.reviewScore.toInt()}" to FieldValue.increment(1),
                "starRatings.${startingScore.toInt()}" to FieldValue.increment(-1),
                "totalRating" to FieldValue.increment(scoreChangedBy.toLong())
            )
            baseRepository.updateDocument("books", review.reviewTo, bookUpdates).getOrElse {
                return Result.failure(it)
            }
        }

        baseRepository.setDocument("reviews", reviewId, review).getOrElse {
            return Result.failure(it)
        }
        return Result.success(Unit)
    }

    private suspend fun updateAverageRatings(bookId: String): Result<Unit> {
        val bookDoc = baseRepository.fetchDocument("books", bookId).getOrElse {
            return Result.failure(it)
        }
        val reviewCount = bookDoc.getLong("reviewCount") ?: 0L
        val totalRating = bookDoc.getLong("totalRating") ?: 0L
        if (reviewCount != 0L) {
            val bookUpdate = mapOf("rating" to (totalRating.toFloat() / reviewCount))
            baseRepository.updateDocument("books", bookId, bookUpdate).getOrElse {
                return Result.failure(it)
            }
        }
        return Result.success(Unit)
    }
}
```