Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: Розробка додатку для відстеження читацьких сесій з використанням архітектурного шаблону «модель-вид-контролер»

Виконав(ла): студент(ка) VI курсу, групи СПм-61
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)

(підпис) Франчевський М. І. (прізвище та ініціали)

Керівник (підпис) Петрик М. Р. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю. М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М. Р. (прізвище та ініціали)

Рецензент (підпис) Сверстюк А. С. (прізвище та ініціали)

Тернопіль
2024

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет _____
(повна назва факультету)

Кафедра _____
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ _____
(підпис) (прізвище та ініціали)
« » 20___ р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня _____
(назва освітнього ступеня)

за спеціальністю _____
(шифр і назва спеціальності)

студенту _____
(прізвище, ім'я, по батькові)

1. Тема роботи _____

Керівник роботи _____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «___» _____ 20___ року № _____

2. Термін подання студентом завершеної роботи _____

3. Вихідні дані до роботи _____

4. Зміст роботи (перелік питань, які потрібно розробити)

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

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

АНОТАЦІЯ

Кваліфікаційна робота на здобуття освітнього ступеню «магістр» за спеціальністю 121 – Інженерія програмного забезпечення. Тернопільський національний технічний університет ім. Івана Пулюя, факультет комп'ютерно-інформаційних систем і програмної інженерії, кафедра програмної інженерії, група СПм-61, 2024 рік.

Пояснювальна записка до кваліфікаційної роботи на здобуття освітнього ступеню «магістр» містить: 80 с., 28 рис., 3 додат., 26 бібліогр.

Тема: Розробка додатку для відстеження читацьких сесій з використанням архітектурного шаблону «модель-вид-контролер».

Дану кваліфікаційну роботу магістра присвячено розробці додатку для відстеження читацьких сесій. Використовуючи клієнт-серверну архітектуру було створено додаток з застосуванням мови програмування Kotlin у середовищі Android Studio.

Ключові слова: додаток, читацькі сесії, Firebase, Android Studio, Kotlin, клієнт-серверна архітектура, «модель-вид-контролер».

ANNOTATION

Qualification work for the master’s degree in specialty 121 - Software Engineering. Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems and Software Engineering, Software Engineering Department, SPm-61, 2024.

Explanatory note to the qualification work for the master's degree contains: 80 p., 28 fig. 3 appendix, 26 ref.

Topic: Development of an application for tracking reading sessions using the model-view-controller architectural pattern.

This master's thesis is devoted to the development of an application for tracking reading sessions. Using a client-server architecture, an application was created using the Kotlin programming language in the Android Studio environment.

Keywords: application, reading sessions, Firebase, Android Studio, Kotlin, client-server architecture, “model-view-controller”.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

API – application programming interface.

UML – unified modeling language.

MVC – архітектурний шаблон «модель-вид-контролер».

ВВ – варіант використання.

GUI – графічний інтерфейс користувача.

ПЗ – програмне забезпечення..

Читацька сесія – період взаємодії користувача з книгою.

Колекція – читацький список користувача.

Гейміфікація – впровадження ігрових елементів для підвищення залученості користувачів.

ВСТУП

Розробка додатку для відстеження читацьких сесій зумовлена зростаючим попитом на персоналізований досвід користувачів. Оскільки читання все більше зміщується в бік цифрових платформ, існує нагальна потреба в інструментах, які не лише спрощують організацію процесу читання, але й надають аналітику, щоб допомогти користувачам розвивати читацькі звички. Існуючі додатки зосереджені переважно на базовому відстеженні книг і часто не мають розширеної аналітики або комплексної підтримки відстеження читацьких сесій.

Метою дослідження є розробка додатку для відстеження читацьких сесій, який забезпечить можливість аналізу читацьких звичок, управління колекціями книг та отримання персоналізованих рекомендацій.

Для досягнення даної мети необхідно:

- оглянути предметну область та конкурентні продукти;
- обґрунтувати вибір технологій, методів та архітектури для розробки додатку;
- спроєктувати функціональні та нефункціональні вимоги до програмного забезпечення;
- реалізувати ключові функціональні можливості;
- інтегрувати додаток із зовнішніми сервісами, такими як Firebase Realtime Database і Google Books API;
- забезпечити тестування, оптимізацію продуктивності та безпеки додатку;

Об'єкт дослідження: процеси, пов'язані з управлінням читацькими звичками, зокрема автоматизацією та аналітикою читацьких сесій.

Предмет дослідження: архітектура, технології та методології розробки мобільного додатку для відстеження читацьких сесій. Це включає інтеграцію з

базами даних, алгоритми персоналізованих рекомендацій та методи покращення користувацького досвіду за допомогою аналітики та гейміфікації.

Методи дослідження: системний аналіз, UML-діаграми, архітектурний шаблон MVC, клієнт-серверна архітектура, Firebase Realtime Database, API інтеграція (Google Books API), середовище розробки Android Studio, програмування на мові Kotlin.

Наукова новизна даної роботи полягає у створенні нового та інноваційного підходу до відстеження читацьких сесій шляхом інтеграції:

- аналітики в режимі реального часу;
- інтеграції із зовнішніми API;
- інтуїтивно зрозумілого інтерфейсу;
- модульної архітектури;
- гейміфікації.

Запропоновані підходи спрямовані на створення продукту, який не тільки заповнює прогалину на існуючому ринку, але й встановлює нові стандарти відстеження читацьких тенденцій.

Практичне значення отриманих результатів – розробка додатку, який дозволяє відстежувати читацькі сесії, аналізувати читацькі звички та формувати персоналізовані книжкові рекомендації, забезпечуючи ефективне управління прогресом у читанні.

Практична реалізація цих результатів сприятиме цифровій трансформації читацьких звичок і створить конкурентну перевагу на ринку книжкових додатків.

1 ОГЛЯД ПРЕДМЕТНОЇ ОБЛАСТІ ДОДАТКУ ДЛЯ ВІДСТЕЖЕННЯ ЧИТАЦЬКИХ СЕСІЙ

1.1 Огляд конкурентів

Вивчення конкурентів є важливим етапом у розробці продукту. Він дозволяє оцінити динаміку ринку, конкурентів, визначити їх сильні та слабкі сторони та розробити стратегію ефективного позиціонування нового продукту. Нижче наведено аналіз основних конкурентів на ринку додатків для читання та відстеження книг:

- Goodreads;
- StoryGraph;
- Basmo;
- Bookshelf;
- Rork;
- Bookly.

Goodreads (рисунок 1.1) – це одна з найвідоміших платформ для книголюбів, що функціонує одночасно як інструмент каталогізації книг та соціальна мережа. Вона дозволяє користувачам відкривати для себе нові книги, відстежувати своє читання та спілкуватися з великою спільнотою читачів. Її база даних містить мільйони книг і рецензій, що робить її привабливим ресурсом для бібліофілів по всьому світу. Станом на 2022 рік на Goodreads було зареєстровано понад 140 мільйонів користувачів, що робить його найбільшою платформою на даному ринку [1].

Переваги:

- велика база даних книжок, рецензій та рейтингів;
- розвинені соціальні функції, включаючи книжкові клуби, дискусійні форуми та рекомендації від спільноти;

– персоналізовані книжкові рекомендації на основі історії читання користувача;
– можливість відстежувати книги, використовуючи полиці «Прочитано», «Зараз читаю» та «Читати»;
– інтеграція з Kindle для синхронізації прогресу.

Недоліки:

– застарілий та інтуїтивно незрозумілий інтерфейс;
– обмежена інформація по деталях фізичних копій книг;
– відсутність відстеження читацьких сесій у реальному часі та глибокої аналітики читацьких звичок.

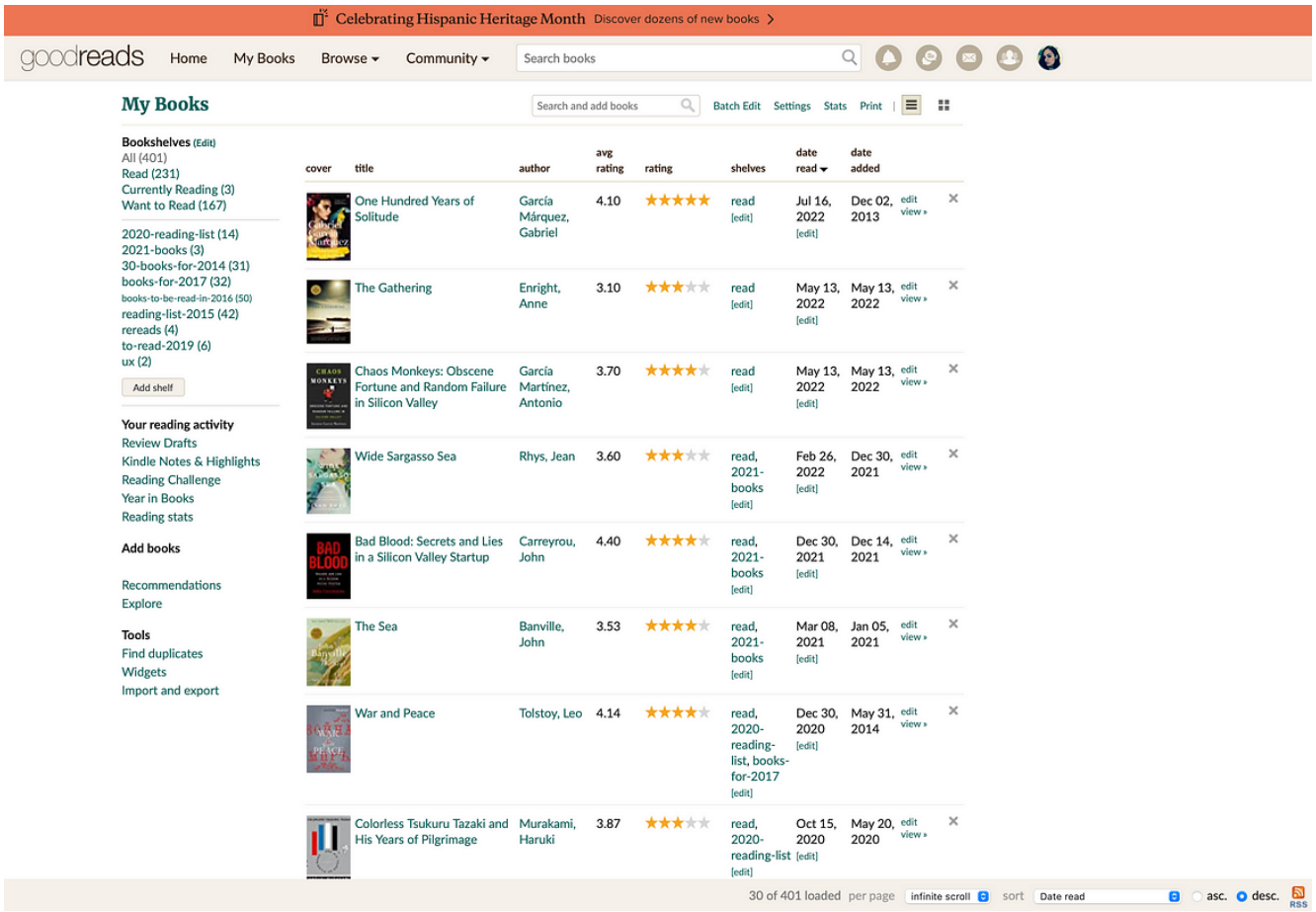


Рисунок 1.1 – Інтерфейс Goodreads

StoryGraph (рисунок 1.2) – це новіша альтернатива Goodreads, що робить акцент на рекомендаціях на основі настрою та різноманітній аналітиці для читачів.

Переваги:

- унікальні рекомендації на основі настрою, які класифікують книги за темпом, тоном та особливостями стилю;
- інтуїтивно зрозумілий, мінімалістичний інтерфейс;
- всебічна аналітика читацьких звичок, таких як розподіл за жанрами та темпом читання;

Недоліки:

- зосередженість на читанні електронних книг;
- рекомендації на основі настрою потребують постійного втручання користувача, щоб бути ефективними, що може здаватися нудним після багатьох повторювань;

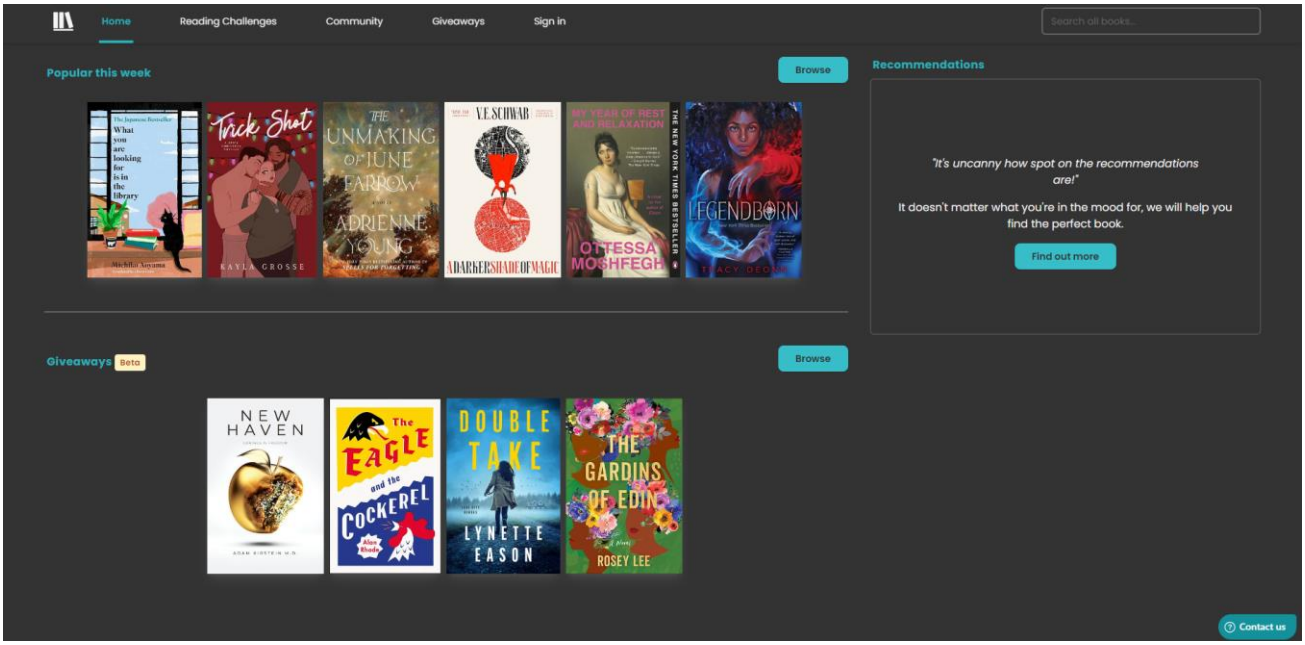


Рисунок 1.2 – Інтерфейс Storygraph

Basmo (рисунок 1.3) – це електронна бібліотека та трекер для читання, призначений для управління фізичними книжковими колекціями та моніторингу читацьких звичок.

Переваги:

– спеціалізується на відстеженні читацьких сесій фізичних копій книг з такими функціями, як таймери та аналітика читацьких сесій;
– простий і зручний дизайн, адаптований для індивідуальних читачів;
– надає інструменти для встановлення цілей читання та підтримки звички;
– пропонує відстеження прогресу та анотації до прочитаних книг.

Недоліки:

– обмежена інфраструктура підтримки, з мінімальними можливостями допомоги користувачам;
– відсутність рекомендацій або функцій пошуку книг;
– менш детальна порівняно з конкурентами аналітика.

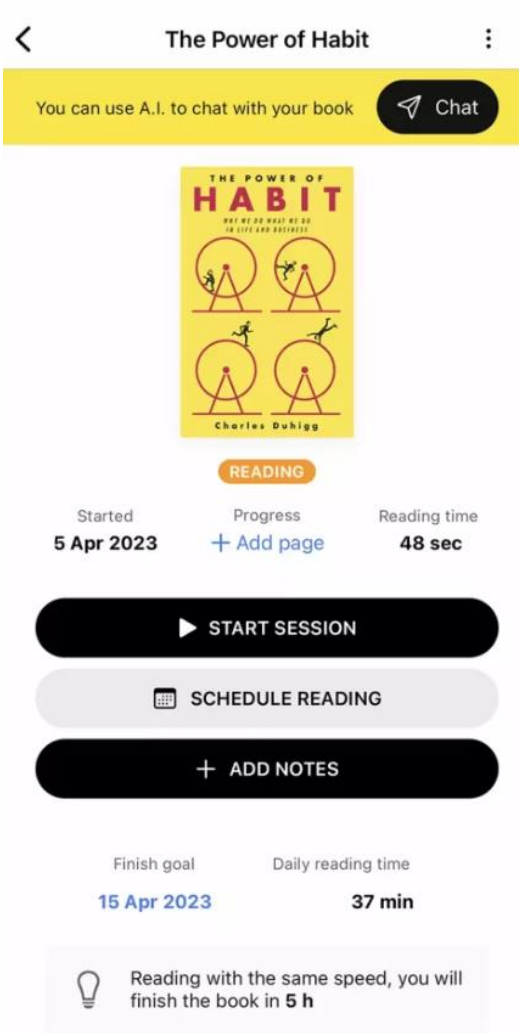


Рисунок 1.3 – Інтерфейс Basmo

Bookshelf (рисунок 1.4) – це додаток для формування читацьких звичок, що пропонує інструменти для відстеження списків книг, аналізу тенденцій та встановлення читацьких цілей.

Переваги:

- зручний і сучасний інтерфейс, який фокусується на відстеженні фізичних книг і цілей читання;
- розширена аналітика читання, наприклад, звіти про тенденції та статистику;
- функції, такі як відстеження серій, нагадування та цілі читання, що сприяють формуванню звички;
- елементи гейміфікації, такі як бейджі та візуалізація прогресу, підтримують мотивацію користувачів;
- рекомендації надаються на основі історії читання та вподобань.

Недоліки:

- менша база користувачів порівняно з більшими конкурентами, такими як Goodreads та StoryGraph;
- обмежена підтримка неанглійських мов;
- соціальні функції недостатньо розвинені, бракує надійних можливостей взаємодії;
- залежність від ручного введення для відстеження прогресу, що деякі користувачі вважають нудним.

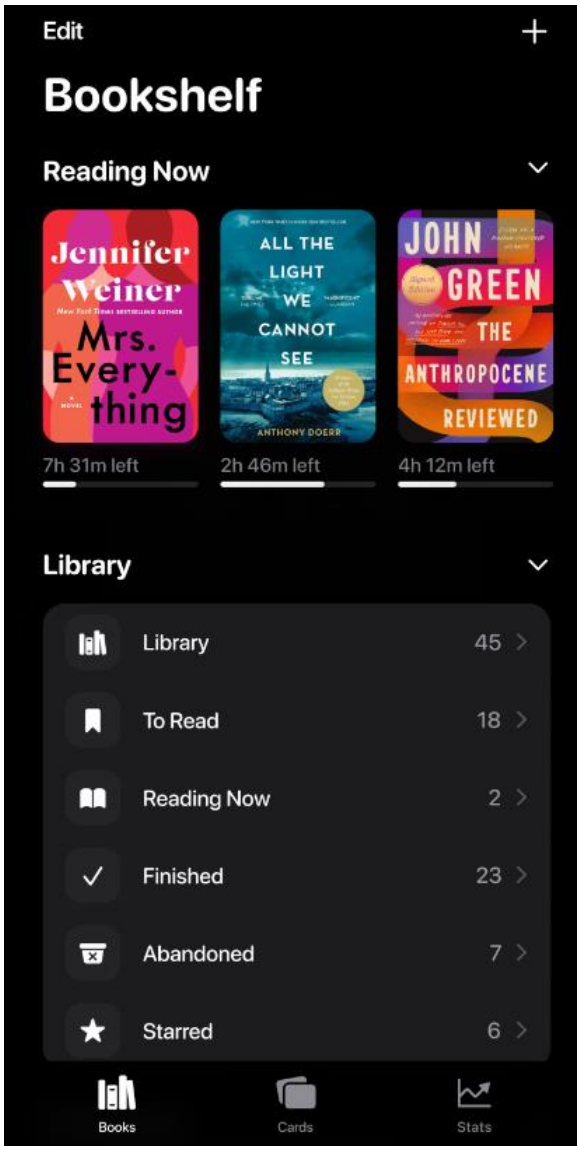


Рисунок 1.4 – Інтерфейс Bookshelf

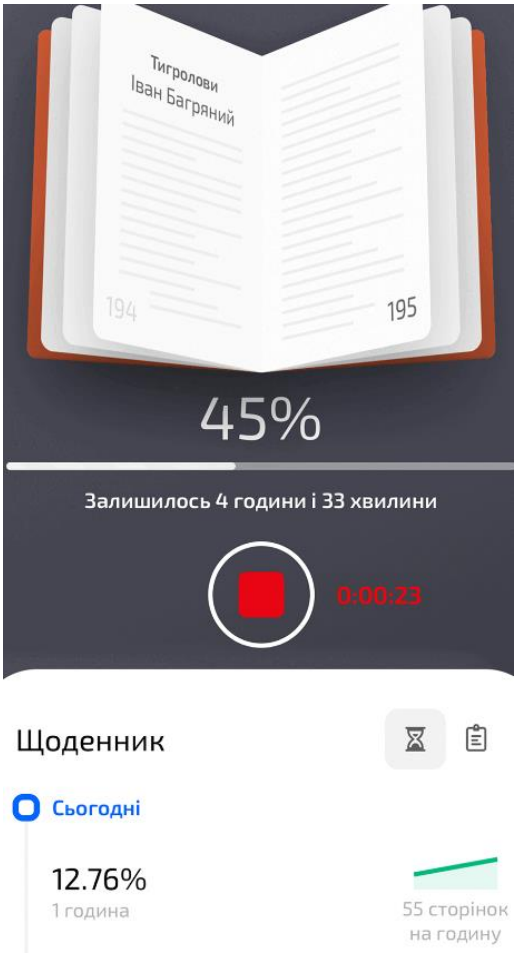
Rork (рисунок 1.5) – це мобільний додаток-щоденник читання, розроблений спеціально для відстеження прогресу в читанні та керування сесіями читання.

Переваги:

- зручні інструменти для керування та відстеження читацьких сесій, включаючи таймери та звіти про прогрес;
- орієнтована на читання фізичних копій книг, з інтуїтивно зрозумілим відстеженням сесій.
- взаємодія з іншими користувачами за допомогою базових соціальних функцій, таких як обмін інформацією про прогрес, реакції, відстеження інших користувачів.

Недоліки:

– обмежена аналітика тенденцій читацьких звичок, таких як середня швидкість читання або довгострокові тенденції;
– відсутність розширених функцій, таких як інструменти для формування звичок або постановка цілей.



Риснок 1.5 – Інтерфейс Rork

Bookly (рисунок 1.6) – це додаток для відстеження читання, розроблений для гейміфікації процесу читання з викликами, нагородами та детальною аналітикою.

Переваги:

– гейміфікований досвід заохочує читачів за допомогою значків, викликів та стрічок, щоб підтримувати мотивацію та залученість;

– дозволяє користувачам встановлювати цілі для щоденного, щотижневого або щомісячного читання та надає нагадування, щоб не відставати від графіка;
– пропонує інтеграцію з фізичними пристроями для читання книг шляхом ручного введення або сканування штрих-коду;
– створює автоматичні звіти про читання та інфографіку, які візуалізують прогрес і звички в часі.

Недоліки:

– інтерфейс може здаватися перевантаженим великою кількістю функцій, що потенційно може відлякати нових користувачів;
– соціальні функції мінімальні, немає великої спільноти чи інструментів взаємодії.

Рисунок 1.6 – Інтерфейс Bookly

У результаті проведення огляду конкурентів, було визначено, що більшість з них не мають повноцінних функцій для відстеження читацьких сесій. Ті ж з них, які реалізовують дані функції, мають свої обмеження щодо детальної аналітики, підтримки користувачів або соціальних функцій.

Зосередженість розроблюваного додатку на детальному відстеженні читання користувачів та аналітиці прогресу вигідно вирізняє його на ринку, де такі функції часто є другорядними. Позиціонуючи його як комплексне рішення для відстеження звичок читання фізичних книг з розширеною аналітикою та інтуїтивно зрозумілим інтерфейсом, розроблюваний продукт можете зайняти свою нішу і залучити читачів зацікавлених у більш детальній інформації про свої читацькі звички.

1.2 Обґрунтування вибору напрямку дослідження

Для реалізації цього проекту було обрано середовище розробки Android Studio [2]. Це офіційне середовище від Google для розробки додатків для Android, побудоване на платформі IntelliJ IDEA. Це середовище забезпечує широкий спектр інструментів, необхідних для розробки, написання коду, тестування та усунення помилок. Його спеціалізовані інструменти та функції забезпечують надійне та ефективне середовище для створення мобільних додатків, що робить Android Studio ідеальним вибором для цього проекту. Основні можливості та переваги Android Studio включають:

- редактор макетів;
- редактор коду;
- вбудований емулятор;
- інструменти моніторингу продуктивності;
- інтеграція контролю версій;
- інтеграція з Firebase.

Android Studio містить потужний візуальний редактор макетів, який дозволяє налаштовувати властивості та переглядати макети для різних розмірів екранів і пристроїв під час розробки, без необхідності запуску емулятора для тестування. Ця функція спрощує створення адаптивних користувацьких інтерфейсів.

Наявний у даному середовищі редактор коду надає розширені можливості, такі як виявлення помилок, завершення коду та інструменти рефакторингу. Він підтримує Kotlin, Java та XML, підвищуючи продуктивність розробників.

Менеджер Android Virtual Device (AVD) дозволяє тестувати додатки на віртуальних пристроях з різними розмірами екрану, роздільною здатністю та версіями Android, усуваючи необхідність наявності фізичного пристрою.

До складу середовища входять профілювальники продуктивності процесора, пам'яті та мережі, що дозволяє ефективно виявляти та усувати помилки, забезпечуючи кращу оптимізацію розроблюваних додатків.

Android Studio також надає вбудовану підтримку Git, GitHub та інших систем контролю версій, що дозволяє безперешкодно працювати у командах та керувати кодом.

Окрім цього, пряма інтеграція з інструментами Firebase дозволяє легко додавати функції автентифікації, баз даних і хмарних сховищ, що дозволяє уникнути тривалого процесу розробки власних аналогічних сервісів [3].

В цілому, екосистема Android Studio спеціально розроблена, щоб відповідати сучасним вимогам розробки для Android. Зручні та корисні інструменти, пропоновані даним середовищем, в поєднанні з простим та зрозумілим інтерфейсом роблять її найкращим вибором для розробки даного проєкту.

Основною мовою для розробки цього проекту було обрано Kotlin [4]. Це сучасна мова програмування також офіційно підтримується компанією Google. Kotlin також має високий ступінь сумісності з Java, що робить використання бібліотек та інструментів Java доступними, пропонуючи численні покращення, які спрощують розробку для Android.

Ключові особливості Kotlin:

- стислий та виразний синтаксис;

- підпрограми;
- повна сумісність з Java;
- сучасні можливості розробки.

Kotlin зменшує кількість шаблонного коду, дозволяючи писати чистий і зручний для підтримки код з меншою кількістю рядків, а вбудована безпека нуля зменшує ймовірність виникнення винятків NullPointerExceptions, які є поширеним джерелом помилок в Android-додатках.

Підпрограми Kotlin спрощують асинхронне програмування, ефективно керуючи фоновими завданнями, такими як виклики API або операції з базами даних, забезпечуючи плавний та адаптивний користувацький інтерфейс.

Використовуючи мову Kotlin можна легко викликати код на Java і навпаки, що дозволяє без проблем використовувати існуючі бібліотеки та фреймворки Java. Окрім цього, функції розширення, лямбда-вирази та функції вищого порядку дозволяють писати гнучкий та багаторазовий код.

Загалом, лаконічний синтаксис, функції безпеки, розширені інструменти для забезпечення полегшення асинхронного програмування та хороша інтеграція з Android Studio роблять Kotlin ідеальною мовою для створення складних додатків з функціями реального часу та інтерактивними користувацькими інтерфейсами.

Для автентифікації користувачів, зберігання даних у режимі реального часу та синхронізації для цього проекту було обрано хмарну платформу Firebase від Google, яка надає послуги бекенда як сервіс. Firebase спрощує внутрішні операції та забезпечує масштабованість.

Основні можливості Firebase:

- база даних Firebase в режимі реального часу;
- автентифікація Firebase;
- аналітика;
- крос-платформна підтримка.

Firebase надає хмарну базу даних NoSQL, яка дозволяє синхронізувати дані в реальному часі між пристроями. Це гарантує, що дані користувача, такі як читацькі сесії та прогрес завжди актуальні.

Окрім цього, Firebase надає безпечне та просте у використанні рішення для входу та реєстрації користувачів, підтримуючи автентифікацію за допомогою електронної пошти та пароля, а також інші методи, такі як Google Sign-In.

Firebase Analytics надає інформацію про поведінку користувачів та використання додатку в цілому, допомагаючи оптимізувати його для кращого залучення та утримання користувачів.

Firebase підтримує Android, iOS та веб-платформи, гарантуючи, що додаток може бути розширений або адаптований у майбутньому.

Отже, Firebase було обрано, через пропоновані можливості синхронізації даних у режимі реального часу та простоту інтеграції з Android Studio. Інструменти автентифікації та бази даних спрощують розробку бекенду, дозволяючи зосередитися на функціях, орієнтованих на користувача.

Таким чином, поєднання Android Studio, Kotlin та Firebase було обрано для створення надійного, масштабованого та зручного для користувача додатку. Android Studio гарантує зручне середовище для розробки, Kotlin забезпечує ефективні та зручні інструменти для реалізації даного проєкту, в той час як Firebase дозволяє безперешкодно обробляє бекенд-операції. Разом ці інструменти утворюють оптимальну для створення додатку для відстеження читацьких сесій екосистему.

1.3 Технічний аспект проблеми

Предметна область відстеження читацьких сесій включає кілька основних технічних аспектів, які безпосередньо впливають на функціональність, користувацький досвід і надійність розроблюваного додатку [5]. Ключові технічні виклики пов'язані із забезпеченням безперебійної синхронізації даних, ефективним

відстеженням читацьких сесій та наданням змістовної аналітики кінцевим користувачам. Отже, було визначено наступні основні технічні виклики:

- синхронізація даних у реальному часі;
- аутентифікація та безпека користувачів;
- відстеження та аналітика читацьких сесій;
- ефективна обробка зображень;
- продуктивність і масштабованість бази даних.

Для того, щоб надати користувачам актуальну інформацію про прогрес читання, нотатки та статистику сесій, додаток використовує базу даних у режимі реального часу. Для цього потрібна надійна інфраструктура для забезпечення безперебійної синхронізації даних між пристроєм користувача та хмарою. Будь-яка затримка в синхронізації може призвести до невідповідностей, таких як відсутність або застарілість даних про прогрес.

Безпечна робота з обліковими записами користувачів має вирішальне значення для забезпечення довіри та запобігання несанкціонованому доступу. Тому, заходи безпеки, такі як зашифрована передача даних і автентифікація на основі токенів, є важливими для захисту конфіденційної інформації користувачів.

Однією з ключових технічних вимог для цього проекту є точний запис і аналіз сесій читання. Це передбачає:

- відстеження часу початку та закінчення читацьких сесій;
- обчислення швидкості читання та прогнозування приблизного часу, необхідного для завершення книги;
- надання вичерпних звітів користувачам, включно зі статистикою їхніх читацьких звичок.

Для цього потрібні ефективні алгоритми та системи зберігання для обробки даних про читацькі сесії та отримання користувацької статистики без негативного впливу на продуктивність додатку.

Окрім цього, даний додаток часто матиме справу із зображеннями, такими як обкладинки книг, отриманими із зовнішніх API або завантаженими користувачами.

Через це виникає необхідність у їх ефективній обробці, що може бути вирішене за допомогою завантаження та кешування зображень для мінімізації повторних мережевих запитів і скорочення часу завантаження, оптимізації якості зображень для забезпечення балансу між візуальною чіткістю та продуктивністю, особливо на пристроях з обмеженими ресурсами, а також забезпечення використання іконок-плейсхолдерів для відсутніх або повільно завантажуваних зображень для підтримки безперебійної та плавної роботи користувацького інтерфейсу.

Також, оскільки додаток значною мірою покладається на Firebase Realtime Database для зберігання даних користувачів, зі збільшенням кількості користувачів база даних повинна ефективно масштабуватися, щоб впоратися зі зростаючим трафіком, зберігаючи при цьому низьку затримку. Це вимагає ретельного структурування правил бази даних та оптимізації запитів до даних, щоб гарантувати, що продуктивність залишається незмінною під час високих навантажень [6].

Окрім визначення технічних аспектів проблеми, було розглянуто та обрано методологію для розробки додатку для відстежування читацьких сесій. Для розробки даного проєкту було обрано Agile методологію. Вона ідеально доповнює проєкти, вимоги яких можуть змінюватися з часом, адже фокусуючись на ітеративній та інкрементній розробці, вона дозволяє гнучко адаптуватися до змін і швидко реагувати на відгуки користувачів. Це також дає змогу створювати та тестувати функції поступово, адаптуючись до нових ідей, коли вони виникають, та зосередитися на створенні основної функціональності, а потім вдосконалюванні або розширюванні її відповідно до нових вимог. Крім того, Agile мінімізує тиск жорстких дедлайнів, дозволяючи керувати робочим процесом більш органічно [7].

2 ПРОЄКТУВАННЯ ДОДАТКУ ДЛЯ ВІДСТЕЖЕННЯ ЧИТАЦЬКИХ СЕСІЙ

Для розробки додатку для відстеження читацьких сесій було обрано клієнт-серверну архітектуру [8]. Це широко використовувана архітектурна модель у розробці програмного забезпечення, яка розділяє обов'язки між клієнтами (користувачами інтерфейсу) і серверами (внутрішніми системами). Клієнт надсилає на сервер різноманітні запити, які той обробляє і повертає відповідні відповіді. Такий поділ забезпечує масштабованість, модульність та легкість обслуговування системи.

Ключовими компонентами клієнт-серверної архітектури є:

- клієнти;
- сервери;
- зв'язок «запит-відповідь»;
- мережеві протоколи.

Клієнт призначений для користувача, зазвичай це веб, мобільний або десктоп додаток, який відповідає за отримання даних, введених користувачем, відображення даних користувачеві, надсилання запитів на сервер для аналізу або отримання даних, надання відповідей, отриманих від сервера.

Клієнти зазвичай виконують легкі завдання і покладаються на сервер для складних операцій, таких як обробка, зберігання та обчислення даних.

Сервер виступає в ролі внутрішньої частини, обробляючи основну логіку програми, управління даними та зовнішні інтеграції. Він обробляє клієнтські запити, зберігає та управляє даними в базах даних, відправлення відповідей клієнту, забезпечення сталого та безпечного використання спільних ресурсів. Сервери призначені для ефективної обробки декількох одночасних запитів клієнтів.

Взаємодія між клієнтом і сервером відбувається за моделлю запит-відповідь. Запит – надсилання серверу структурованого запиту від клієнта, а відповідь – обробка запиту сервером і повернення відповідних даних або підтвердження. Окрім

цього, кожен запит обробляється незалежно, сервер не зберігає стан сесії між взаємодіями.

Клієнти та сервери взаємодіють через мережу, використовуючи стандартизовані протоколи, такі як HTTP/HTTPS або TCP/IP.

Основні характеристики клієнт-серверної архітектури включають: централізацію, модульність, масштабованість та безпеку.

Централізація даних та бізнес-логіки на сервері, що забезпечує узгодженість між кількома клієнтами та спрощує управління даними;

Клієнт і сервер працюють незалежно, але взаємодіють один з одним за допомогою запитів і відповідей. Таке розділення забезпечує гнучкість у розробці та обслуговуванні, адже клієнти та сервери можуть розроблятися незалежно, а клієнти можуть додаватися або оновлюватися без впливу на сервер.

Сервери можна масштабувати горизонтально (додаючи більше серверів) або вертикально (збільшуючи потужність сервера), щоб впоратися зі зростаючим числом клієнтських запитів.

Конфіденційні дані зберігаються на сервері, що зменшує ризик витоку даних на клієнтських пристроях, в той час як протоколи шифрування захищають зв'язок між клієнтами та серверами.

Недоліками клієнт-серверної архітектури є залежність від підключення – клієнти потребують стабільного мережевого з'єднання для зв'язку з сервером, що призводить до обмеженої або повністю відсутньої офлайн-функціональності при відсутності підключення до мережі. Окрім цього, сервер може стати єдиною точкою відмови або вузьким місцем в продуктивності, якщо він не розрахований на високий трафік, в той час як обслуговування потужних серверів або хмарної інфраструктури може бути дорогим, особливо для ресурсоємних додатків.

Саме тому клієнт-серверна архітектура у даному проєкті зосереджується навколо чіткого розподілу обов'язків [9]:

– клієнт (Android-додаток) відповідає за всю бізнес-логіку, взаємодію з користувачами та функціональність додатку;

- сервер (Firebase) виступає в першу чергу як служба зберігання, синхронізації та автентифікації даних.

Це забезпечує такі переваги;

- зменшення складності на стороні сервера;
- масштабованість;
- оновлення в реальному часі;
- пришвидшена ітерація;
- економічна ефективність.

Вивантажуючи бізнес-логіку на сторону клієнта, сервер зосереджується виключно на управлінні даними та автентифікації, зменшуючи його складність, що також дозволяє легко масштабувати хмарну інфраструктуру Firebase, обробляючи збільшені обсяги користувацьких даних, не вимагаючи спеціальної логіки на стороні сервера.

Синхронізація Firebase в реальному часі забезпечує безперебійну роботу користувачів з узгодженими даними на різних пристроях, а оскільки бізнес-логіка повністю знаходиться на стороні клієнта, оновлення функцій і функціональності можна робити без зміни конфігурацій на стороні сервера.

Окрім цього, сервер, орієнтований на дані, усуває необхідність у складних обчисленнях або спеціальній логіці, мінімізуючи витрати.

Проте, даний підхід також має обмеження, що стосуються безпеки даних, продуктивності та офлайн-функціональності. Зберігання та обробка всієї бізнес-логіки на стороні клієнта означає, що більш чутливі операції повинні бути ретельно захищені. Окрім цього, складні обчислення або обробка даних на стороні клієнта можуть вплинути на продуктивність на застарілих пристроях, а значна залежність від Firebase вимагає ретельної обробки офлайн-сценаріїв таких як, кешування та вирішення конфліктів синхронізації.

Для успішного проєктування додатку для відстеження читацьких сесій також необхідно визначити та проаналізувати функціональні та нефункціональні вимоги.

Під час проєктування даного програмного забезпечення були визначені наступні функціональні вимоги:

- відстеження читацьких сесій – додаток повинен надавати можливість починати, призупиняти та завершувати читацькі сесії. Він повинен фіксувати такі деталі, як книга, яку читають, кількість прочитаних сторінок та час, витрачений на читання;
- пошук та додавання книг – додаток повинен надавати користувачу можливість шукати книги через інтеграцію з Google Books API [10];
- управління колекцією – додаток повинен надвати можливість створювати, перейменовувати та видаляти колекції, а також додавати у них книги;
- управління нотатками – додаток повинен надавати користувачу можливість писати, редагувати і видаляти нотатки, пов'язані з конкретними книгами;
- відстеження прогресу читання – додаток повинен відстежувати прогрес користувача в кожній книзі і розраховувати такі показники, як швидкість читання, час, що залишився до завершення книги, і відсоток завершення;
- персоналізовані книжкові рекомендації – додаток повинен аналізувати читацькі звички та вподобання користувача, щоб пропонувати книги, адаптовані до його інтересів;
- візуалізація читацької активності – додаток повинен включати календар для відображення минулих читацьких сесій та детальної статистики;
- автентифікація користувача – додаток повинен надавати користувачам безпечну реєстрацію та вхід за допомогою Firebase Authentication.

Окрім цього, визначено нефункціональні вимоги:

- продуктивність – додаток повинен швидко завантажуватися і працювати без збоїв, навіть при управлінні великою кількістю книг, нотаток і читацьких сесій;

- надійність – додаток мусить стабільно працювати та уникати збоїв під час критичних операцій, таких як збереження прогресу читання або оновлення даних користувача;
- безпека – додаток повинен надійно зберігати дані користувача у Firebase, а вся передача даних між клієнтом і сервером повинна бути зашифрована;
- інтерфейс користувача – додаток повинен забезпечувати користувача інтуїтивно зрозумілим та візуально привабливим інтерфейсом, який спрощує навігацію і взаємодію з ключовими функціями;
- масштабованість – додаток повинен обробляти зростаючу кількість користувачів і пов'язаних з ними даних (наприклад, книг, колекцій, нотаток) без погіршення продуктивності;
- синхронізація даних – синхронізація повинна забезпечувати актуальність даних користувача на всіх пристроях з мінімальною затримкою;
- інтеграція із зовнішніми API – додаток повинен інтегруватися з API для отримання метаданих та рекомендацій щодо книг.

2.1. Розробка моделі предметної області додатку для відстеження читацьких сесій.

Процес розробки моделі предметної області додатку для відстеження читацьких сесій складається з ідентифікації та опису компонентів програмної системи. Під час проєктування системи були визначені такі основні компоненти:

- користувачі;
- книги;
- колекції;

- читацькі сесії;
- нотатки;
- рекомендації;
- аналітика.

Розроблена модель включає представлення окремих користувачів, які взаємодіють з додатком. Кожен користувач має такі атрибути, як: унікальний id користувача, облікові дані та читацькі вподобання.

Книги – основна частина додатку. Вони представлені сутностями з атрибутами, що включають: ідентифікатор книги, назву книги, автора, жанр, кількість сторінок, обкладинку.

Колекції дозволяють користувачам організовувати свої книги в довільні групи, вони мають наступні атрибути: ідентифікатор колекції, назва колекції, список книг, пов'язаних з колекцією.

Читацькі сесії відстежують взаємодію користувача з окремими книгами. Їх атрибути це: ідентифікатор сесії, ідентифікатор книги, час початку читацької сесії, кількість сторінок, що були прочитані, швидкість та тривалість сесії. Ці дані дозволять додатку обчислювати такі показники, як середня швидкість читання та приблизний час, необхідний для завершення книги.

Нотатки – це створені користувачем текстові записи, пов'язані з певними книгами. Їх атрибути включають: текст нотатки та мітку часу створення.

Система рекомендацій пропонує книги користувачам на основі жанрів, авторів або книг, з якими користувач найчастіше взаємодіє та історії читання користувача.

Окрім цього, даний додаток збирає та аналізує статистику читацьких сесій користувача, щоб надати таку інформацію, як: щомісячні тенденції, відстеження прогресу читання книг, а також досягнення цілей.

Ці аналітичні дані дозволять додатку відображати діаграми з прогресом користувача, а також мотивувати їх досягненнями та проміжними цілями.

Розроблюваний додаток зберігає всі дані, пов'язані з користувачами та книгами, у Firebase Realtime Database, яка забезпечує синхронізацію у реальному часі та масштабованість.

Також, відповідно до функціональних вимог, викомпоненти повинні мати наступні зв’язки:

- один до одного:
  - книга містить у собі інформацію про одну нотатку;
- один до багатьох:
  - користувач взаємодіє з кількома книгами, додаючи їх до колекцій або розпочинаючи читацькі сесії;
  - книга містить у собі інформацію про довільну кількість читацьких сесій;
- багато до багатьох:
  - книга може міститися у кількох колекціях, а кожна колекція може володіти кількома книгами.

2.2. Розробка бізнес-моделі додатку для відстеження читацьких сесій.

Для розробки бізнес-моделі предметної області було створено діаграму BB додатку для відстеження читацьких сесій, яка надає візуальний огляд взаємодії між акторами, що дозволяє краще висвітлити основні можливості та функціональність додатку [11]. Вона ілюструє конкретні дії, які можуть виконувати користувачі, наприклад, відстежувати читацькі сесії, керувати колекціями книг і переглядати статистику читання. Відображаючи ці взаємодії, діаграма допомагає усім зацікавленим сторонам зрозуміти, як додаток буде використовуватися в реальних умовах. Така комплексна візуалізація користувацьких взаємодій є цінним інструментом для вдосконалення проєктованого додатку та покращення загального користувацького досвіду.

Розроблену UML діаграму ВВ додатку для відстеження читацьких сесій зображено на рисунку 2.1.

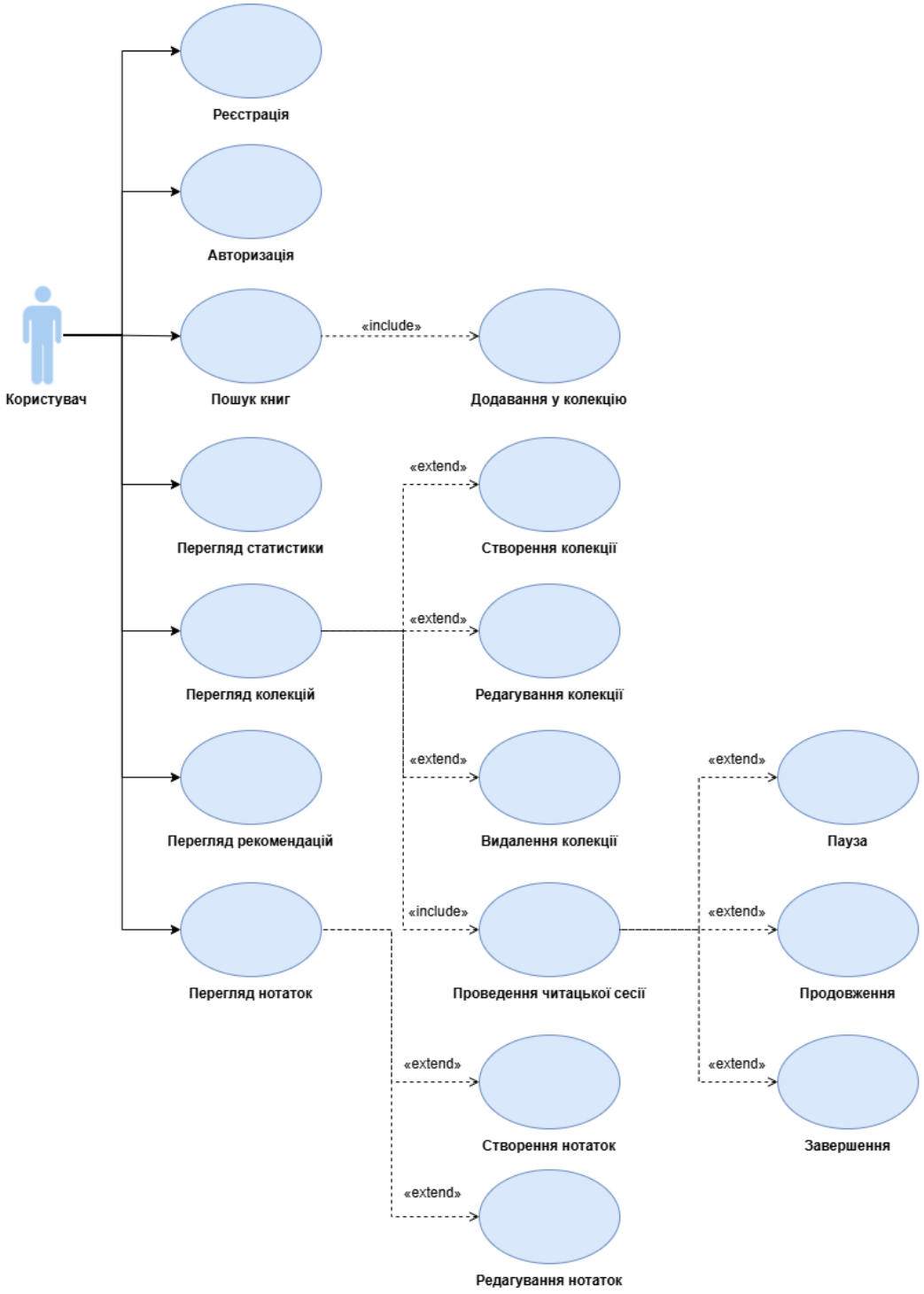


Рисунок 2.1 – Розроблена діаграма ВВ

Наступними було описано основні сценарії роботи розроблюваного додатку.

Варіант використання «Реєстрація»

Короткий опис:

Цей сценарій описує процес створення нового акаунту користувача для доступу до функціоналу додатку для відстеження читацьких сесій.

Основний потік подій:

1. Користувач натискає на кнопку «Зареєструватися».
2. Система пропонує користувачеві ввести необхідні дані.
3. Система перевіряє введені користувачем дані.
4. Якщо дані коректні, у Firebase Authentication створюється новий обліковий запис.
5. Користувач отримує повідомлення про успішну реєстрацію.

Передумови:

Користувач не повинен мати існуючого акаунту в системі.

Післяумови:

Користувач отримує доступ до всіх функцій додатку, і його дані зберігаються в Firebase.

Варіант використання «Авторизація»

Короткий опис:

Цей сценарій описує процес авторизації у систему зареєстрованого користувача.

Основний потік подій:

1. Користувач натискає на кнопку «Авторизація».
2. Система пропонує ввести електронну пошту та пароль.
3. Введені дані перевіряються через Firebase Authentication.
4. Якщо дані коректні, користувач входить до системи і отримує доступ до своїх даних.
5. У разі неправильного паролю або невірної пошти система відображає повідомлення про помилку.

Передумови:

Користувач повинен бути попередньо зареєстрований у системі.

Післяумови:

Користувач отримує доступ до свого акаунту, а також усіх своїх даних, що зберігаються у Firebase.

Варіант використання «Пошук книг»:

Короткий опис:

Цей сценарій описує пошук книг.

Основний потік подій:

1. Користувач запускає додаток для відстеження читання.
2. У головному меню користувач вибирає опцію пошуку книг.
3. Система відображає текстове поле для введення пошукового запиту.
4. Користувач вводить критерії пошуку.
5. Система надсилає запит до Google Books API.
6. Результати пошуку відображаються у вигляді списку книг із назвою, автором, зображенням обкладинки та іншими характеристиками.
7. Користувач може обрати книгу зі списку для перегляду детальної інформації або додати її до колекції.

Передумови:

Користувач під’єднаний до Інтернету для виконання пошукового запиту.

Післяумови:

Користувач отримує результати пошуку книг, які він може переглянути, додати до колекції або використовувати для створення читальної сесії.

Варіант використання «Додавання у колекцію»:

Короткий опис:

Даний ВВ описує процес додавання обраної книги до колекції користувача.

Основний потік подій:

1. Користувач вибирає книгу зі списку результатів пошуку.
2. Система відображає діалогове вікно з опціями для вибору колекції.
3. Користувач обирає існуючу колекцію.
4. Книга додається до обраної колекції.
5. Система підтверджує успішне додавання книги.

Передумови:

Користувач повинен бути зареєстрованим і увійти до свого акаунту.

Післяумови:

Книга успішно додається до обраної колекції, яка синхронізується з базою даних Firebase.

Варіант використання «Перегляд статистики»:

Короткий опис:

Цей варіант використання ілюструє процес перегляду персональної статистики читання користувача.

Основний потік подій:

1. Користувач переходить до розділу статистики у головному меню.
2. Система відображає загальні показники, такі як: кількість прочитаних книг, середня швидкість читання, час, витрачений на читання, графік прогресу читання за місяць.

Передумови:

У користувача мають бути збережені читацькі сесії для аналізу.

Післяумови:

Користувач успішно переглядає свою статистику для відстеження свого прогресу.

Варіант використання «Створення колекції»

Короткий опис:

Даний варіант описує процес створення нової колекції для організації книг користувача.

Основний потік подій:

1. Користувач натискає на кнопку «Створити колекцію».
2. Система пропонує користувачеві ввести назву колекції.
3. Система перевіряє унікальність назви колекції для даного користувача.
4. Колекція створюється і зберігається у Firebase Realtime Database.

Передумови:

Користувач повинен бути авторизований у системі.

Післяумови:

Нова колекція додається до списку колекцій користувача і стає доступною для перегляду або редагування.

Варіант використання «Проведення читацької сесії»:

Короткий опис:

Цей варіант використання описує процес запуску, паузи та завершення читацької сесії.

Основний потік подій:

1. Користувач вибирає книгу з колекції.
2. Користувач натискає «Почати сесію».
3. Система записує час початку та початкову сторінку.
4. Користувач може: поставити сесію на паузу або завершити сесію.
5. Система зберігає дані про тривалість сесії, кількість прочитаних сторінок та швидкість читання.

Передумови:

Книга повинна бути додана до колекції користувача.

Післяумови:

Дані про читацьку сесію зберігаються у Firebase та використовуються для оновлення прогресу та статистики.

Варіант використання «Перегляд рекомендацій»:

Короткий опис:

Цей варіант використання описує процес отримання персоналізованих рекомендацій книг на основі історії читання користувача.

Основний потік подій:

1. Користувач переходить до розділу рекомендацій.
2. Система аналізує дані користувача (вподобання, історія читання тощо).
3. Система надсилає запит до Google Books API для отримання релевантних рекомендацій.
4. Користувач переглядає список рекомендованих книг.
5. Користувач може обрати книгу та додати її до колекції.
6. Користувач може негативно відреагувати на рекомендацію.

Передумови:

Користувач має достатньо даних для формування рекомендацій.

Післяумови:

Користувач отримує персоналізований список книг, що відповідає його вподобанням.

2.3. Проєктування архітектури

Розробка добре структурованої архітектури має критичне значення для забезпечення безперешкодної роботи додатку та приємних вражень користувача при взаємодії з ним. Цей процес включає створення базової структури, компонентів і взаємодії, які забезпечують основні функції програми, такі як відстеження читацьких сесій, управління колекціями та створення персоналізованих рекомендацій. Ретельно розроблена архітектура не лише забезпечує ефективну роботу, але й забезпечує розширюваність, масштабованість і легкість у обслуговуванні для майбутніх удосконалень.

Архітектура додатку для відстеження читацьких сесій була розроблена з урахуванням наступних цілей:

- розділення завдань;
- управління даними в режимі реального часу;
- інтеграція зі сторонніми програмами.

Даний проєкт використовує архітектурний шаблон «модель-вид-контролер» (MVC), що розділяє бізнес-логіку, користувацький інтерфейс та управління даними [12].

У цьому шаблоні, модель відображає основну логіку та дані додатку. Вона відповідає за управління станом додатку та обробку бізнес-правил, таких як:

- отримання та зберігання даних;
- повідомлення виду про будь-які зміни в даних;

– обробку логіки додатку.

Вид (подання) – керує рівнем представлення програми та відповідає за відображення даних користувачеві та фіксацію його дій. Він відповідає за:

– представлення даних отриманих з моделі у зручний та зрозумілий спосіб;
– оновлення відображення після отримання повідомлення про зміни в моделі;
– отримання вхідних даних від користувача і їх передачу контролеру.

Контролер виступає посередником між моделлю та видом, отримує вхідні дані користувача від виду, обробляє їх та передає у модель. Обов’язки контролера:

– отримання вхідних даних від представлення та їх інтерпретація;
– оновлення моделі на основі дій користувача;
– надсилання даних з моделі назад до виду для представлення.

Серед переваг даного шаблону можна виділити такі:

– обов'язки програми чітко розділені на три окремі рівні, що полегшує управління та розуміння;
– кожен компонент можна розробляти та підтримувати незалежно, не впливаючи на інші компоненти. Наприклад, зміни в інтерфейсі користувача (View) не вплинуть на бізнес-логіку (Model);
– модульний дизайн полегшує додавання нових функцій або масштабування додатку для обробки додаткової функціональності;
– розділення компонентів спрощує модульне тестування та налагодження. Наприклад, модель можна тестувати незалежно від представлення або контролера;
– представлення та контролери можна повторно використовувати в різних частинах програми, а моделі – при створенні нових функцій.

Проте, шаблон MVC має такі недоліки, як:

– для невеликих додатків MVC може додати зайвої складності через розділення компонентів та комунікації між ними;

- реалізація MVC часто вимагає значно більшої кількості класів та взаємодій;
- для ефективного використання даного шаблону необхідний час, щоб зрозуміти, як взаємодіють компоненти і як їх ефективно реалізувати.

Завдяки використанню архітектурного шаблону MVC гарантовано стійкість архітектури до змін вимог, адже «модель-вид-контролер» дозволяє легко модифікувати існуючі функціональності та безперешкодно додавати нові без необхідності внесення змін у існуючий функціонал проєкту. Саме чітке розділення між рівнями моделі, виду та контролера гарантує, що зміни в одній частині програми не впливають безпосередньо на інші компоненти. Отже, спроєктовані класи проєкту будуть розділені на три основні групи:

- класи моделей – інкапсулюють бізнес-логіку та структури даних, зміни в представленні або логіці обробки яких можуть бути реалізовані без впливу на вид або контролери;
- вид, класи якого відображатимуть інформацію для користувача та оброблятимуть взаємодію з користувачем. Нові користувацькі інтерфейси можна створювати або змінювати незалежно від основної логіки системи;
- класи-контролери, які виступатимуть у ролі мосту між моделями та видами. Завдяки ним, будь-які зміни в робочих процесах користувачів чи поведінки додатку можуть бути оброблені без необхідності зміни інших компонентів системи.

Дана архітектура також сприяє масштабованості, гарантуючи, що окремі компоненти можуть бути розширені або замінені без порушення роботи всієї системи. Це досягається за допомогою:

- проєктування та розділенні класів системи на моделі, види та контролери таким чином, щоб зосередитися на конкретних обов'язках. Такий модульний підхід гарантує, що нові компоненти, такі як додаткові контролери, моделі або види, можуть бути легко інтегровані;

– сприянню MVC горизонтальному масштабуванню, адже вона дозволяє додавати нові функції або компоненти без порушення основної логіки, адже кожен компонент програми, у своєму базовому вигляді складається всього лиш з трьох класів – моделі, виду та контролера.

В цілому, такий підхід гарантує, що система залишається модульною та легкою в обслуговуванні.

Окрім цього, даний додаток інтегрується з Firebase Realtime Database для централізованого зберігання та синхронізації даних на кількох пристроях, що дозволяє користувачам відстежувати свій прогрес і керувати колекціями в режимі реального часу, в той час як Google Books API використовується для отримання метаданих для книг, таких як назви, автори, жанри та обкладинки, що забезпечує функціональність пошуку та рекомендацій. Це також дозволяє підтримувати все більшу кількість даних та користувачів без погіршення продуктивності, а завдяки модульній природі шаблону MVC, що дозволяє додавати нові функції з мінімальними змінами в існуючих компонентах, у розроблюваній архітектурі було досягнуто вимог масштабованості та розширюваності.

З метою створення якісної архітектури даного проєкту розроблено діаграми класів для найважливіших компонентів системи, зображені на рисунках 2.2-2.5.

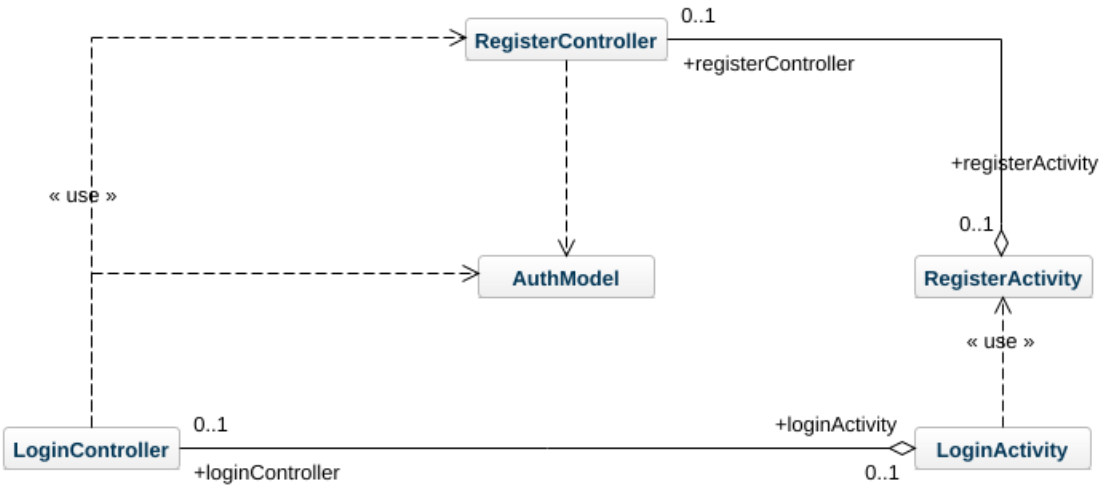


Рисунок 2.2 – Діаграма класів компоненту керування користувачами

Ця діаграма класів представляє компонент керування користувачами програми, зокрема, зображуючи взаємодії між класами, що відповідають за автентифікацію, реєстрацію та вхід у систему.

Даний компонент містить такі класи: AuthModel, RegisterActivity, RegisterController, LoginActivity та LoginController.

AuthModel представляє основну модель даних для автентифікації користувачів, інкапсулює логіку автентифікації, забезпечуючи чітке розділення між бізнес-логікою та контролерами і видом. Він містить екземпляр Firebase Authentication для керування входом і та реєстрацією, а також функції:

- registerUser() – реєструє нового користувача за допомогою Firebase Authentication та зберігає його дані в базі даних;
- loginUser() – авторизує існуючого користувача з наданими обліковими даними за допомогою Firebase Authentication.

Клас LoginController відповідає за бізнес-логіку процесу входу в систему. Він взаємодіє з AuthModel для перевірки облікових даних користувача і управління завданнями, пов'язаними з входом.

RegisterController керує логікою реєстрації користувачів. Він використовує AuthModel для перевірки та зберігання даних користувача при створенні нового облікового запису.

LoginActivity являє собою інтерфейс для входу в систему, містить поля для введення імені користувача та пароля і делегує логіку входу до LoginController.

Клас RegisterActivity – рівень інтерфейсу користувача для реєстрації. Він взаємодіє з RegisterController для обробки логіки реєстрації та відображення зворотного зв'язку з користувачем.

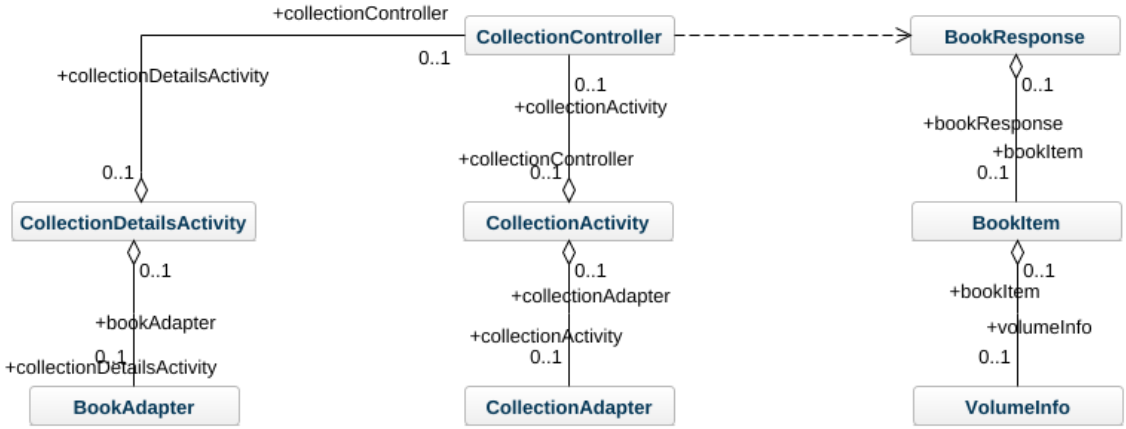


Рисунок 2.3 – Діаграма класів компоненту керування книгами та колекціями

Дана діаграма класів представляє компонент керування книгами та колекціями програми, детально описуючи взаємодію та взаємозв'язки між класами, що беруть участь в управлінні книжковими колекціями, відображаючи їхні дані та забезпечуючи взаємодію з користувачем.

Клас CollectionController – ключовий клас даного компоненту. Він обробляє всі взаємодії між класами виду і моделлю. Цей контролер гарантує, що дані колекції отримуються, оновлюються або модифікуються безперешкодно, зберігаючи подання відокремленим від шару даних.

Він містить атрибут firebaseDatabase для посилання на базу даних, та наступні методи:

- fetchAllBooks() – отримує всі книги користувача з бази даних;
- addBookToCollection() – додає певну книгу до вказаної колекції у базі даних користувача;
- removeBookFromCollection() – видаляє книгу з вказаної колекції та бази даних;
- fetchCollectionsWithCounts() – повертає список колекції з кількістю книг, збережених у ній.

CollectionActivity – інтерфейс користувача для відображення колекцій. Взаємодіє з CollectionController для отримання та відображення колекцій, а також використовує CollectionAdapter для зв'язування даних з компонентами інтерфейсу.

Клас CollectionDetailsActivity відповідає за інтерфейс для конкретної колекції, відображаючи книги, збережені у цій колекції. Він покладається на CollectionController для отримання інформації про колекцію та використовує BookAdapter для прив'язки окремих деталей книги до інтерфейсу.

CollectionAdapter – компонент інтерфейсу, що відповідає за заповнення елементів інтерфейсу CollectionActivity колекціями. Задля обробки користувацьких дій взаємодії з CollectionActivity.

Клас BookResponse являє собою структуру відповіді на виклики API, пов'язані з книгами. Містить списки об'єктів BookItem.

BookItem представляє окремий об'єкт книги, що містить такі дані, як назва, автор та кількість сторінок, включає асоціацію з VolumeInfo, яка містить метадані про книгу, такі як категорії, мову, середній рейтинг.

Рисунок 2.4 – Діаграма класів компоненту читацьких сесій та статистики користувача

Ця діаграма класів представляє компонент читацьких сесій програми, який відповідає за управління, відстеження та відображення читацьких сесій

користувачів і пов'язаної з ними статистики. Нижче наведено детальне пояснення класів та їхніх ролей у цьому компоненті.

Головним класом даного компоненту є ReadingSessionController, який відповідає за логіку для управління читацькими сесіями. Слугуючи мостом між класами виду та класом моделі, його роль, згідно шаблону MVC полягає у забезпеченні безперебійного потоку даних між цими елементами, що досягається за допомогою функцій:

- saveReadingSession() – зберігає читацьку сесію в Firebase;
- loadBookData() – завантажує з Firebase дані про читацькі сесії певної книги;
- saveProgress() – оновлює прогрес користувача (прочитані сторінки) у Firebase.
- saveRating() – зберігає оцінку користувача певної книги у Firebase;
- loadRating() – отримує збережену оцінку користувача для книги з Firebase;
- loadSessionsForMonth() – завантажує читацькі сесії за вибраний місяць;
- loadBooksForMonth() – завантажує книги та їхній прогрес за вказаний місяць;
- loadSessionsWithPageCount() – завантажує сесії разом з кількістю сторінок, щоб надати детальні дані про сесії для візуалізації статистики.

ReadingSessionModel представляє модель даних для читацьких сесій, зберігає такі деталі, як книга, що читається, час початку та закінчення сесії, кількість прочитаних сторінок, середня швидкість читання тощо. Даний клас надає дані для обробки контролеру.

Клас ReadingTrackerActivity – інтерфейс для ініціювання та керування читацькими сесіями. Він обробляє такі дії користувача, як початок, пауза, відновлення та завершення читацької сесії.

UserStatsActivity – представляє собою інтерфейс для відображення детальнішої статистики користувача на основі читацьких сесій. Він збирає та відображає такі дані, як:

– загальна кількість прочитаних книг;
– загальний час читання;
– середня швидкість читання;
– сторінки, прочитані в усіх сесіях.

Взаємодіє з ReadingSessionController для отримання попередньо розрахованої статистики, отриманої з ReadingSessionModel.

Клас DashboardUserActivity діє як основний центр або оглядовий інтерфейс для статистики та дій користувача, забезпечуючи точку входу до UserStatsActivity.

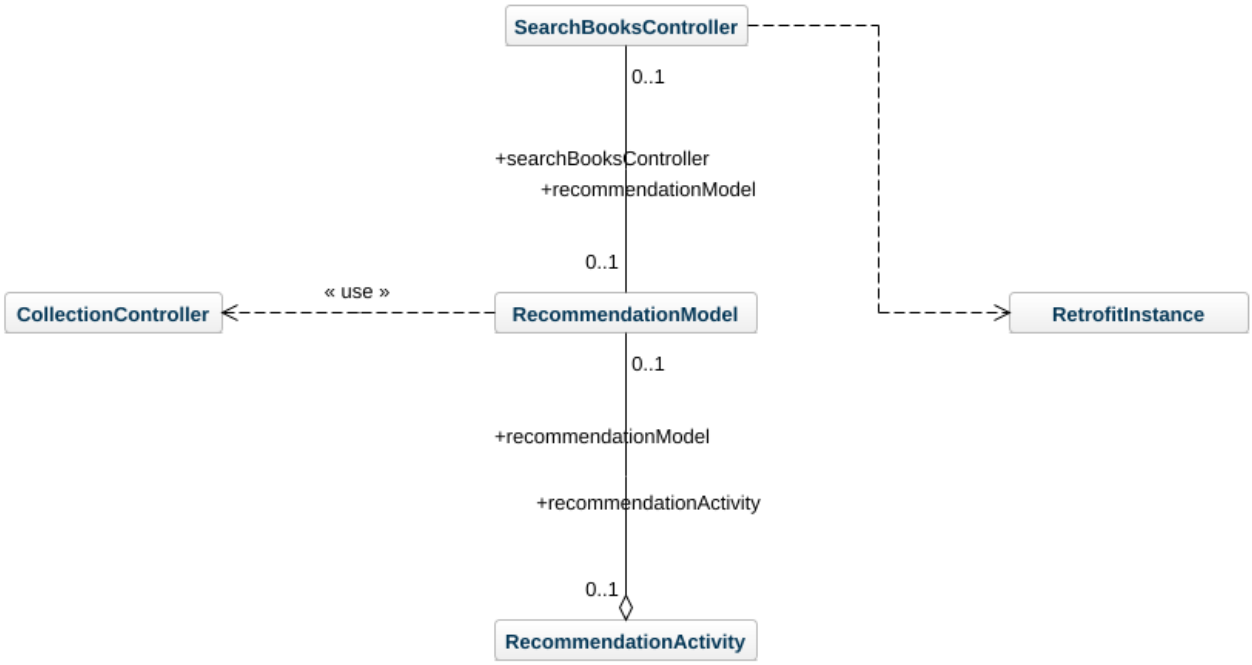


Рисунок 2.5 – Діаграма класів компоненту рекомендацій

Дана діаграма класів представляє компонент рекомендацій програми. Вона моделює взаємодію та залежності між класами, що беруть участь у створенні рекомендацій щодо книг відповідно до користувацьких вподобань завдяки інтеграції колекцій користувачів і пошуку книг.

RecommendationModel – основний клас цього компоненту. Він керує бізнес-логікою системи рекомендацій, отримує книги користувачів, аналізує їхні метадані, такі як жанри, автори, фільтрує результати та формує рекомендаційний запит на основі вподобань користувача та його історії читання. Взаємодіючи з CollectionController для отримання збережених користувачами книг, та SearchBooksController для отримання рекомендованих книг з Google Books API, RecommendationModel централізує всі операції пов'язані з формуванням рекомендацій.

Містить езкемпляри CollectionController для отримання даних про читацькі вподобання користувача, та SearchBooksController для виконання запитів на пошук книг, після формування рекомендацій. Також містить атрибути dislikedBooks – за допомогою якого відстежує книги, рекомендація яких не задовільнила користувача, та userPreferences, який зберігає динамічні вподобання користувачів, такі як:

- minRating – мінімальний поріг рейтингу;
- minPageCount та maxPageCount – діапазон довжини книг;
- preferredGenres – жанри, яким користувач надає перевагу.

RecommendationModel виконує свої обов'язки завдяки наступним методам:

- fetchRecommendations() – генерує рекомендації, аналізуючи активність користувачів, створює оптимізований пошуковий запит і за допомогою SearchBooksController виконує його;
- calculateWeightedScores() – аналізує зважені оцінки для жанрів книг чи авторів, підвищує оцінки для останніх книг, які сподобалися користувачу, для того щоб краще відповідати поточним читацьким вподобанням;
- filterBookByCriteria() – використовуючи userPreferences, здійснює динамічне фільтрування книг;
- diversifyRecommendations() – забезпечує різноманітність списку рекомендацій впроваджуючи випадковість та додаючи менш поширені результати для заохочення відкриття нових для користувача жанрів;

- dislikeBook() – додає ідентифікатор книги до набору dislikedBooks, запобігаючи її повторному рекомендуванню;
- undoDislikeBook() – видаляє ідентифікатор книги зі списку dislikedBooks;
- resetDislikedBooks() – очищає список нелюбимих книг, скидаючи всі користувацькі вподобання «не подобається»;
- updateUserPreference() – оновлює динамічні вподобання користувача.

Клас RecommendationActivity надає користувацький інтерфейс для відображення рекомендацій та управління діями користувача, використовує RecommendationModel для отримання та відображення рекомендацій.

SearchBooksController Відповідає за обробку запитів на пошук книг та рекомендацій, взаємодіючи з Google Books API через RetrofitInstance.

CollectionController – відповідає за отримання книг, що вже є в колекції користувача, для інформування логіки рекомендацій, а також за додавання книг до певних колекцій після взаємодії з користувачем у RecommendationActivity.

RetrofitInstance забезпечує мережеву конфігурацію та клієнт Retrofit API для здійснення HTTPS-запитів. Використовується SearchBooksController для виконання викликів API для отримання даних про книги.

3 КОНСТРУЮВАННЯ ДОДАТКУ ДЛЯ ВІДСТЕЖЕННЯ ЧИТАЦЬКИХ СЕСІЙ

3.1. Реалізація ключових класів

Реалізація ключових класів є фундаментальним кроком у розробці додатку для відстеження читацьких сесій. Ці класи формують основу системи, інкапсулюючи основний функціонал і структури даних, необхідні для відстеження читацьких сесій, управління колекціями та генерування персоналізованих рекомендацій. Ретельна розробка та реалізація цих класів гарантує надійність, гнучкість і задоволеність користувачів.

Клас UserStatsActivity (рисунок 3.1) відповідає за візуалізацію статистики читання користувача, такої як кількість прочитаних сторінок, завершені книги, найдовша безперервна серія читання та час, проведений за читанням. Він містить графічні елементи та відстеження багаторівневих цілей, щоб залучити користувачів і заохотити їх до покращення звичок читання.

Змінні:

- binding – зв'язує компоненти інтерфейсу, визначені в ActivityUserStatsBinding;
- readSessionController – керує завантаженням та обробкою даних читацьких сесій;
- backBtn – кнопка для переходу на попередній екран;
- milestonesDialog – діалогове вікно, у якому відображаються дані про прочитані сторінки, завершені книги, пропуски і витрачений час;
- sessionData – зберігає список читацьких сесій, отриманих для користувача;
- calendar – використовується для обчислення діапазонів дат для статистики читання;
- totalPagesRead – загальна кількість сторінок, прочитаних за всіма сесіями;

- booksFinished – кількість книг, прочитаних користувачем;
- longestStreak – найдовша послідовна серія читання користувача в днях;
- totalReadingTimeMillis – загальний час читання у мілісекундах.

Методи:

- loadReadingSessions – отримує дані читацьких сесій за поточний місяць за допомогою readSessionController, оновлює статистику і графік щомісячних сторінок після отримання даних;
- setupMonthlyPagesGraph – налаштовує і заповнює гістограму для відображення щомісячної кількості прочитаних сторінок;
- calculateAndDisplayStats – обчислює та оновлює різноманітну статистику:
    - booksFinished – завершені книги;
    - booksInProgress – книги із записаними сесіями, які ще не завершено;
    - totalPagesRead – сума всіх сторінок, прочитаних під час читацьких сесій;
    - totalReadingTimeMillis – загальний час, витрачений на читання, відформатований у годинах і хвилинах;
    - avgSpeed – середня швидкість читання в усіх сесіях;
    - longestStreak – кількість днів читання поспіль;
- calculateLongestStreak – визначає найдовшу серію днів читання поспіль, аналізуючи дати сесій;
- showMilestonesDialog – відображає детальне діалогове вікно багаторівневих цілей;
- setupPagesReadMilestone – оновлює смужки прогресу та цілі для прочитаних сторінок;
- setupBooksReadMilestone – налаштовує віхи для кількості прочитаних книг;

– setupLongestStreakMilestone – відстежує прогрес у досягненні довших періодів читання поспіль;
– setupTimeSpentMilestone – відстежує та візуалізує загальний час, витрачений на читання.

В цілому, UserStatsActivity є важливим класом для підвищення залученості та мотивації користувачів. Поєднуючи детальну статистику, відстеження етапів та інтерактивні візуалізації, він дає користувачам можливість контролювати та покращувати свої читацькі звички.

```
private fun setupMonthlyPagesGraph() {
    val barChart = findViewById<BarChart>(R.id.monthlyPagesGraphContainer)
    val entries = mutableListOf<BarEntry>()

    // Generate all months of the year in "MM-YYYY" format
    val calendar = Calendar.getInstance()
    val allMonths = mutableListOf<String>()
    for (i in 0 ≤ until < 12) {
        calendar.set(Calendar.MONTH, i)
        val monthYear = "${(i + 1).toString().padStart( length: 2,  padChar: '0')}-${calendar.get(Calendar.YEAR)}"
        allMonths.add(monthYear)
    }

    // Group sessions by month and year
    val groupedByMonth: Map<String, List<ReadingSessionModel>> = sessionData.groupBy {
        val dateParts = it.date?.split( ...delimiters "-")?.map(String::toInt)
        if (dateParts != null && dateParts.size == 3) {
            "${dateParts[1].toString().padStart( length: 2,  padChar: '0')}-${dateParts[2]}" // Month-Year as key (e.g., "11-2024")
        } else {
            null
        }
    }.filterKeys { it != null } as Map<String, List<ReadingSessionModel>> // Explicit type-cast

    // Create BarEntry for each month (default to 0 if no data)
    allMonths.forEachIndexed { index, monthYear ->
        val totalPages = groupedByMonth[monthYear]?.sumOf { it.pagesRead } ?: 0
        entries.add(BarEntry(index.toFloat(), totalPages.toFloat()))
    }

    val dataSet = BarDataSet(entries,  label: null) // Removed label
    dataSet.color = getColor(R.color.ripple_color) // Use a single color for all bars
    dataSet.valueTextColor = getColor(R.color.text_secondary) // Set bar value text color
    dataSet.valueTextSize = 12f // Improve readability
```

Рисунок 3.1 – Ключовий код класу «UserStatsActivity»

Клас RecommendationModel (рисунок 3.2) відіграє важливу роль у додатку, генеруючи персоналізовані книжкові рекомендації на основі вподобань користувача, збережених книг та історії взаємодії. Він використовує алгоритми

динамічного оцінювання для ранжування авторів і категорій, застосовує фільтри, які залежать від вподобань користувача та надає диверсифіковані рекомендації.

Змінні:

- collectionController – екземпляр CollectionController для отримання збережених користувачем книг та їхніх метаданих;
- searchBooksController – екземпляр SearchBooksController для отримання книжкових рекомендацій з Google Books API;
- dislikedBooks – змінний набір ідентифікаторів книг, які користувачеві не сподобалися, що гарантує, що вони не з'являться в майбутніх рекомендаціях;
- userPreference – зберігає динамічні налаштування користувача, зокрема, для фільтрації рекомендацій;
- minRating – мінімальний середній рейтинг для рекомендованих книг;
- minPageCount і maxPageCount – діапазон кількості сторінок для книг;
- preferredGenres – список пріоритетних жанрів для рекомендацій.

Методи:

- fetchRecommendations – отримує та обробляє персоналізовані книжкові рекомендації.
- filterBookByCriteria – перевіряє, чи відповідає книга визначеним користувачем критеріям:
  - середній рейтинг вище minRating;
  - кількість сторінок у діапазоні від minPageCount до maxPageCount.
  - книга належить до одного з обраних користувачем жанрів;
- checkPreferredGenres – перевіряє, чи перетинаються категорії книги з обраними користувачем жанрами;
- diversifyRecommendations – групує книги за їх основною категорією і гарантує, що кожна категорія буде представлена;
- dislikeBook – додає книгу до списку невподобаних;

– undoDislikeBook – видаляє книгу зі списку невподобаних;
– resetDislikedBooks –повністю очищає список нелюбимих книг;
– updateUserPreference – динамічно оновлює визначені користувачем фільтри, такі як пороги оцінок або бажані жанри.

Загалом, клас RecommendationModel є фундаментальним для досягнення мети додатку – надання персоналізованого, цікавого користувацького досвіду. Використовуючи історію взаємодії з користувачем, динамічні фільтри та різноманітні рекомендації, він гарантує, що додаток залишається корисним і привабливим для широкого кола користувачів.

```
private fun calculateWeightedScores(items: List<String>, recentBooks: List<BookItem>): Map<String, Double> {
    val recentWeights = recentBooks.flatMap { it.volumeInfo.categories ?: emptyList() }
        .groupingBy { it }
        .eachCount()
        .mapValues { (_, count) -> count * 1.5 } // Boost weights for recent interactions

    val itemCount = items.groupingBy { it }.eachCount()

    return itemCount.mapValues { (key, count) ->
        val recentWeight = recentWeights[key] ?: 0.0
        (count + recentWeight) * 100 / items.size.toDouble() // Normalize by total count
    }
}

private fun filterBookByCriteria(book: BookItem): Boolean {
    val minRating = userPreferences["minRating"] as Double
    val minPageCount = userPreferences["minPageCount"] as Int
    val maxPageCount = userPreferences["maxPageCount"] as Int

    return (book.volumeInfo.averageRating ?: 0.0) >= minRating &&
            (book.volumeInfo.pageCount ?: 0) in minPageCount ≤ .. ≤ maxPageCount &&
            checkPreferredGenres(book)
}

private fun checkPreferredGenres(book: BookItem): Boolean {
    val preferredGenres = userPreferences["preferredGenres"] as List<String>
    return if (preferredGenres.isNotEmpty()) {
        book.volumeInfo.categories?.any { it in preferredGenres } ?: false
    } else true
}
```

Рисунок 3.2 – Ключовий код класу «RecommendationModel»

Клас CollectionController (рисунок 3.3) виконує всі операції з базою даних, пов'язані з книжковими колекціями. Він надає надійний інтерфейс для створення, перейменування, видалення та керування колекціями.

Змінні:

- firebaseAuth – екземпляр FirebaseAuth, який використовується для отримання поточного ідентифікатора користувача;
- defaultCollections – попередньо визначений список назв колекцій за замовчуванням які завжди доступні для користувачів.

Методи:

- fetchCollectionsWithCounts – отримує список усіх колекцій та відповідну кількість книг у них;
- createCollection – створює нову колекцію;
- renameCollection – перейменовує існуючу колекцію, копіюючи її дані до нового ключа і видаляючи старий ключ;
- deleteCollection – видаляє вказану колекцію з бази даних;
- fetchBooksByCollection – отримує всі книги з вказаної колекції.
- addBookToCollection – додає нову книгу до вказаної колекції;
- removeBookFromCollection – видаляє певну книгу з колекції;
- updateBookPageCount – оновлює кількість сторінок книги у колекції;
- fetchThumbnail – отримує URL-адресу мініатюри певної книги у колекції;
- fetchAllBooks – отримує всі книги з усіх колекцій для поточного користувача.

Клас CollectionController є основою для керування колекціями користувачів та пов'язаними з ними книгами. Абстрагуючи операції Firebase у цілісний набір методів, він спрощує взаємодію між представленням і базою даних, забезпечуючи надійне та ефективне управління даними.

```
class SearchBooksController {

    // Callback interface to communicate with the View
    interface SearchCallback {
        fun onSuccess(bookResponse: List<BookItem>)
        fun onError(message: String)
    }

    // Search for books based on a query
    fun searchBooks(
        query: String,
        callback: SearchCallback
    ) {
        val apiService = RetrofitInstance.api

        // Prioritize title, then broader matches
        val Query = "intitle:$query OR $query"

        apiService.searchBooks(Query).enqueue(object : Callback<BookResponse> {
            override fun onResponse(call: Call<BookResponse>, response: Response<BookResponse>) {
                if (response.isSuccessful && response.body() != null) {
                    val bookItems = response.body()!!.items

                    // Post-filter books to ensure relevance
                    val filteredBooks = bookItems?.filter {
                        val pageCount = it.volumeInfo.pageCount
                        val language = it.volumeInfo.language?.lowercase() ?: ""

                        // Exclude books in russian and with 0 pages
                        language != "ru" && (pageCount == null || pageCount > 0)
                    }

                    if (filteredBooks.isNullOrEmpty()) {
                        callback.onError("No books matched the query.")
                    } else {
                        callback.onSuccess(filteredBooks)
                    }
                } else {
                    callback.onError("Failed to fetch data. Error code: ${response.code()}")
                }
```

Рисунок 3.3 – Основний код класу «SearchBooksController»

Клас ReadingTrackerActivity (рисунок 3.4) є одним з ключових класів у розроблюваній системі. Він відповідає за управління відстеженням читацьких сесій, обробку нотаток і підрахунок прогресу читання для кожної книги. Цей клас

забезпечує основний інтерфейс для користувачів для моніторингу їхніх читацьких звичок та взаємодії з їхніми сесіями читання.

Змінні:

- noteController – екземпляр класу NoteController для керування нотатками, пов'язаними з книгою;
- readingSessionController – екземпляр класу ReadingSessionController для керування даними читацької сесії;
- bookTitle – рядок, що представляє назву книги, яка відстежується;
- bookThumbnailUrl – рядок, що представляє URL-адресу мініатюри книги;
- bookPageCount – загальна кількість сторінок у книзі;
- bookProgressPages – відстежує кількість сторінок, прочитаних користувачем;
- isTracking – вказує, чи активна читацька сесія в даний момент;
- sessionStartTime – представляє мітку часу початку поточної сесії;
- totalElapsedTime: Довге значення, що зберігає загальний час, витрачений на читання під час сесії;
- memorySessions – призначений для тимчасового зберігання даних сесії в пам'яті;
- handler – екземпляр обробника для керування періодичними оновленнями інтерфейсу під час читацької сесії.

Обробники подій:

- backButton_Click – повертає користувача на попередній екран.
- hourglassButton_Click – відображає список читацьких сесій для вибраної книги;
- notepadButton_Click – відображає нотатки до вибраної книги та дозволяє користувачеві редагувати їх;
- startStopButton_Click – запускає або зупиняє читацькі сесії на основі поточного стану;

- notesInput_TextChanged – зберігає нотатки до бази даних після редагування тексту;
- ratingDisplay_Click – відкриває діалогове вікно для користувачів, щоб оцінити книгу, яку вони читають.

Допоміжні методи

- initializeViews – встановлює і прив'язує всі компоненти інтерфейсу користувача;
- loadSessionsFromFirebase – отримує збережені сесії з Firebase та оновлює інтерфейс;
- startReadingSession – ініціалізує нову читацьку сесію, встановлює таймери та оновлює інтерфейс;
- stopTimer – зупиняє таймер сесії і зберігає час, що минув;
- showStartPageDialog – пропонує користувачеві ввести початкову сторінку для читацької сесії;
- showEndPageDialog – пропонує користувачеві ввести кінцеву сторінку під час завершення сесії;
- updateProgressUI – оновлює індикатор виконання, відсоток виконання та оцінку часу, що залишився, на основі читацьких сесій;
- saveProgressToFirebase – зберігає поточний прогрес читання книги до бази даних Firebase;
- finalizeSession – завершує читацьку сесію, зберігає її в базі даних і скидає змінні сесії.

Клас ReadingTrackerActivity є ключовим для програми, надаючи основний інтерфейс для відстеження та керування читацькими сесіями. Він інтегрує обчислення прогресу читання, керування сесіями та ведення нотаток у зручний для користувача інтерфейс. Даний клас забезпечує синхронізацію усіх відповідних даних з Firebase, підтримуючи узгодженість між читацькими сесіями і пристроями.

```
private fun stopTimer() {
    if (isTracking) {
        totalElapsedTime += SystemClock.elapsedRealtime() - sessionStartTime
        sessionStartTime = 0L
        handler.removeCallbacks(updateClockRunnable)
        isTracking = false
    }
}

private fun resumeTimer() {
    if (!isTracking) {
        sessionStartTime = SystemClock.elapsedRealtime()
        handler.post(updateClockRunnable)
        isTracking = true
    }
}

private fun startReadingSession() {
    isTracking = true
    sessionStartTime = SystemClock.elapsedRealtime()
    startStopButton.text = "Stop"
    readingClock.visibility = View.VISIBLE
    handler.post(updateClockRunnable)
}

private fun finalizeSession() {
    stopTimer()

    val session = ReadingSessionModel.createSession(
        startPage = sessionStartPage,
        endPage = sessionEndPage,
        durationMillis = totalElapsedTime,
        date = getCurrentDate(),
        bookTitle = bookTitle,
        collectionKey = collectionKey
    )
```

Рисунок 3.4 – Ключовий код класу «ReadingTrackerActivity»

Таким чином, реалізація ключових класів демонструє успішну розробку додатку для відстеження читацьких сесій. Ці класи в сукупності забезпечують необхідний функціонал для управління читацькими сесіями, організації книжкових

колекцій, генерування персоналізованих рекомендацій та взаємодії з базою даних для отримання та зберігання даних про книги та вподобання користувачів. Разом вони – основні компоненти системи.

3.2 Розробка GUI

Створення графічного інтерфейсу користувача (GUI) є критично важливим компонентом сучасного процесу розробки програмного забезпечення, що надає користувачам інтуїтивно зрозумілий і візуально привабливий спосіб взаємодії зі складними системами. Добре розроблений графічний інтерфейс підвищує доступність, зручність і задоволеність користувачів, що робить його ключовим фактором успіху будь-якого програмного додатку [13].

Створення ефективного графічного інтерфейсу вимагає дотримання кількох важливих принципів. По-перше, інтерфейс повинен бути візуально привабливим, з добре продуманою графікою, іконками та макетами для створення естетично приємного досвіду. По-друге, дуже важливою є швидкість реагування, оскільки користувачі очікують плавної та безперебійної взаємодії без помітних затримок. Нарешті, простота і дизайн, орієнтований на користувача, мають першорядне значення – користувачі повинні мати можливість легко орієнтуватися в додатку, а елементи керування та інформація повинні бути логічно організовані відповідно до встановлених конвенцій і шаблонів дизайну.

Таким чином, дизайн графічного інтерфейсу – ключовий засіб для формування позитивного загального користувацького досвіду. Зосереджуючись на візуальній привабливості, швидкості реагування та зручності використання, розробники можуть створювати інтерфейси, які перевершують очікування користувачів і підвищують загальну цінність програмного забезпечення.

Для цього додатку графічний інтерфейс було розроблено за допомогою Android Studio з Kotlin, використовуючи вбудовані інструменти Android для

створення сучасних та адаптивних користувацьких інтерфейсів. Система верстки Android на основі XML дозволила розробити гнучкі та візуально привабливі інтерфейси, тоді як Kotlin забезпечив безперешкодний спосіб підключення елементів інтерфейсу до базової логіки.

Ключовою перевагою середовища розробки Android є розділення логіки системи, від її зовнішнього вигляду. Також, завдяки дотриманню архітектурного патерну Model-View-Controller (MVC) було забезпечено чітке розділення між інтерфейсом користувача програми, бізнес-логікою і базовими даними.

Крім того, Android Studio надає значний набір вбудованих компонентів, таких як кнопки, TextView та RecyclerView, які були використані для створення інтуїтивно зрозумілих та динамічних інтерфейсів. Ці компоненти були налаштовані відповідно до цілей та дизайну додатку, що забезпечило приємний користувацький досвід.

Ще однією важливою особливістю платформи Android є її надійні можливості прив'язки даних, які спрощують підключення компонентів інтерфейсу до даних додатку. Завдяки використанню цих функцій будь-які зміни в даних автоматично відображаються в інтерфейсі, що забезпечує оновлення в режимі реального часу та більш безперешкодну взаємодію з користувачем.

Завдяки використанню Android Studio, Kotlin та потужних інструментів Android, графічний інтерфейс програми був розроблений таким чином, щоб забезпечити сучасний, привабливий та зручний інтерфейс, що дозволяє користувачам легко відстежувати свої читацькі сесії, керувати книжковими колекціями та вивчати персоналізовані рекомендації. На рисунку 3.5 наведено вигляд Android Studio, призначений для розробки інтерфейсу Android-додатку та сторінку проведення читацьких сесій.

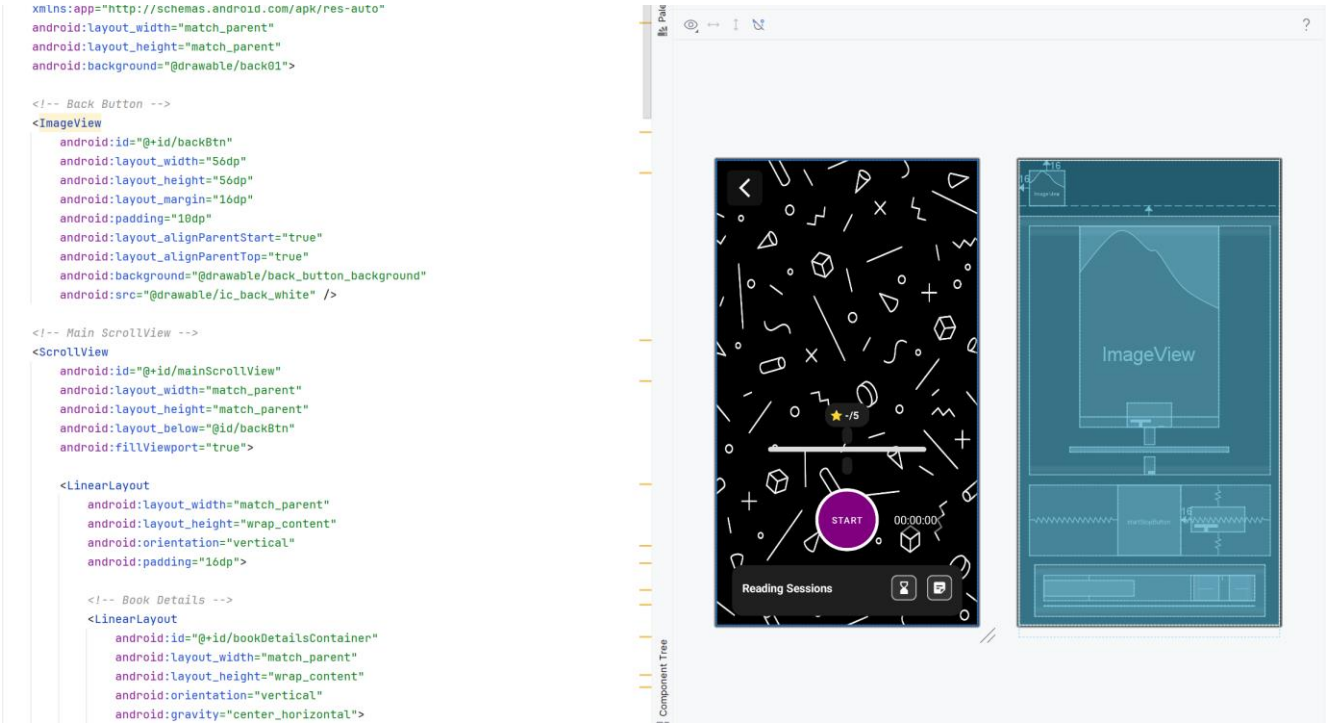


Рисунок 3.5 – Загальний вигляд розробки інтерфейсу в Android Studio

У такий спосіб також були розроблені додаткові екрани (рисунок 3.6) з універсальним дизайном для відображення списків книг, авторів і колекцій. Ці екрани використовують макети Android на основі XML та функцію прив'язки даних для динамічного наповнення контенту з бази даних під час взаємодії користувача з додатком.

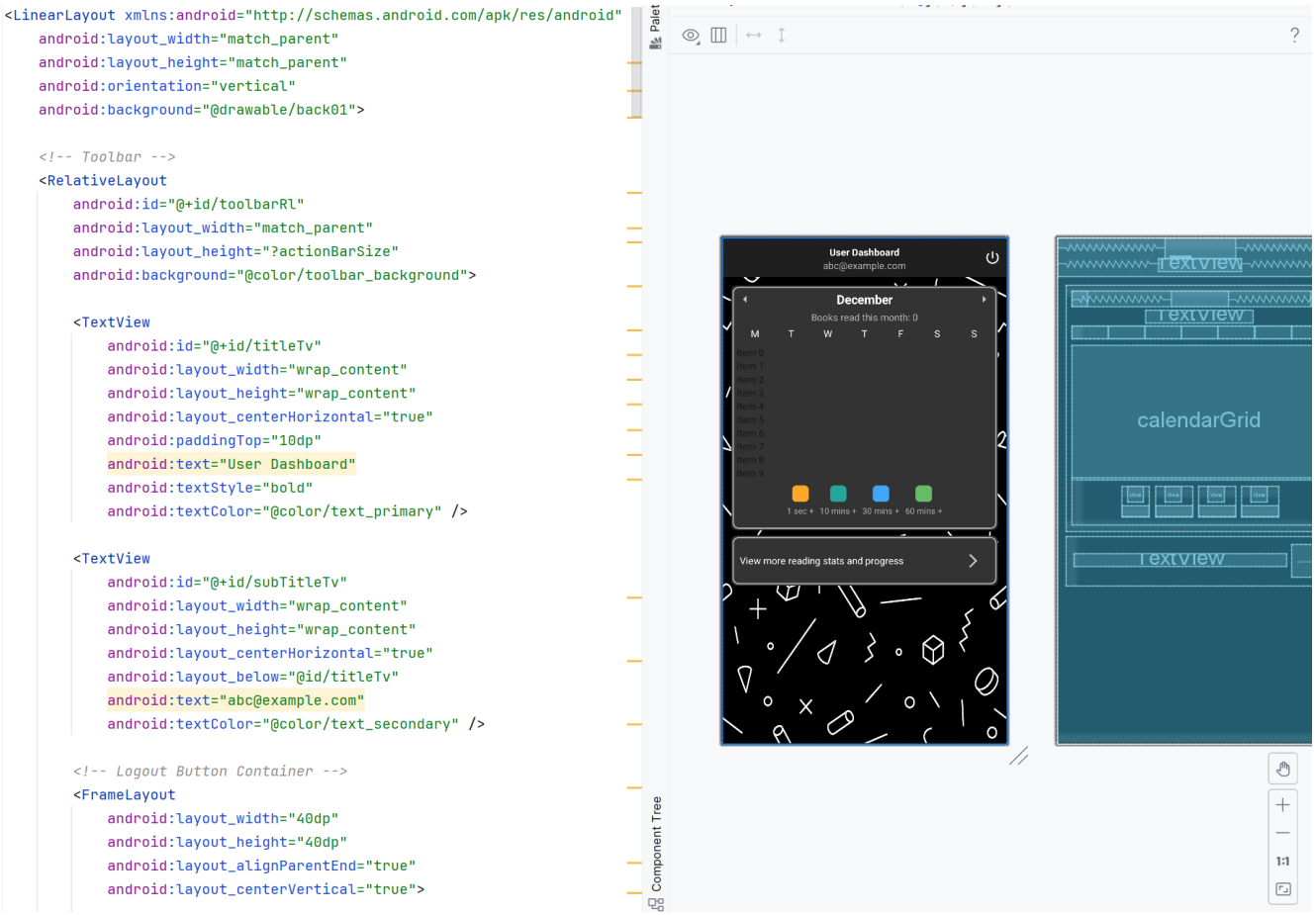


Рисунок 3.6 – Екран сторінки користувача

Цей XML-макет визначає інтерфейс інформаційної панелі користувача для розроблюваного додатку, поєднуючи функціональність з візуально привабливим дизайном. Кореневий LinearLayout забезпечує вертикальну структуру з користувацьким тлом, що малюється, для підтримання єдиного естетичного вигляду. У верхній частині RelativeLayout слугує панеллю інструментів, що відображає заголовок інформаційної панелі та електронну пошту користувача в центрі, а також кнопку виходу з системи, розташовану праворуч, зі стилізованим ефектом пульсації для інтерактивності.

Розділ календаря нижче пропонує навігацію для перегляду щомісячної читацької активності з кнопками для перемикання місяців, назвою місяця по центру і зображенням обкладинок книг, прочитаних протягом місяця. Він включає RecyclerView для динамічного заповнення днів календаря та горизонтальний рядок зі скороченими назвами днів для вирівнювання. Розділ легенди під календарем

використовує кольорові маркери для візуального представлення рівнів активності на основі часу, витраченого на читання. Знизу розділ статистики надає користувачам деталізовану статистику пов’язану з їхніми читацькими сесіями.

Таким чином, використовуючи потужні інструменти Android Studio та Kotlin, було розроблено сучасний, зручний та привабливий GUI, який залучатиме більшу аудиторію та забезпечуватиме безперебійне керування книгами та відстеження читацьких сесій.

3.3 Тестування програмного забезпечення та перевірка якості

Тестування ПЗ та контроль за його якістю є надзвичайно важливими етапами життєвого циклу розробки. Проведення тестування дозволяє систематично оцінити дану програмну систему для виявлення помилок, дефектів або вразливостей, які можуть поставити під загрозу ефективність, зручність використання чи безпеку. Забезпечення якості фокусується на досягненні загальної високої якості продукту шляхом забезпечення відповідності вимогам користувачів, галузевим стандартам і найкращим практикам. Здійснюючи ефективне тестування та проводячи вичерпні заходи, направлені на забезпечення контролю якості, буде підвищено стабільність та надійність розроблюваного продукту, що призведе до кращих вражень користувачів від користування розробленим додатком та довгострокового успіху.

Для забезпечення якості розробленого додатку для відстеження читацьких сесій було проведено комплексне тестування. Після запуску програми всі ключові елементи на сторінці реєстрації (рисунок 3.7) коректно завантажені, а користувач, ввівши свої дані та підтвердивши обраний пароль, може створити новий акаунт, і одразу автентифікуватися у додаток.

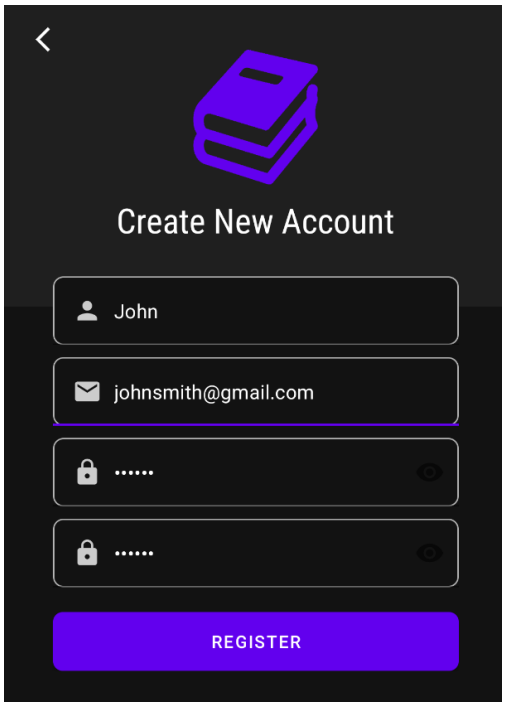


Рисунок 3.7 – Реєстрація у додатку

Після цього також було протестовано сторінку автентифікації (рисунок 3.8) у додаток, завдяки якому користувач може зайти у свій створений раніше акаунт.

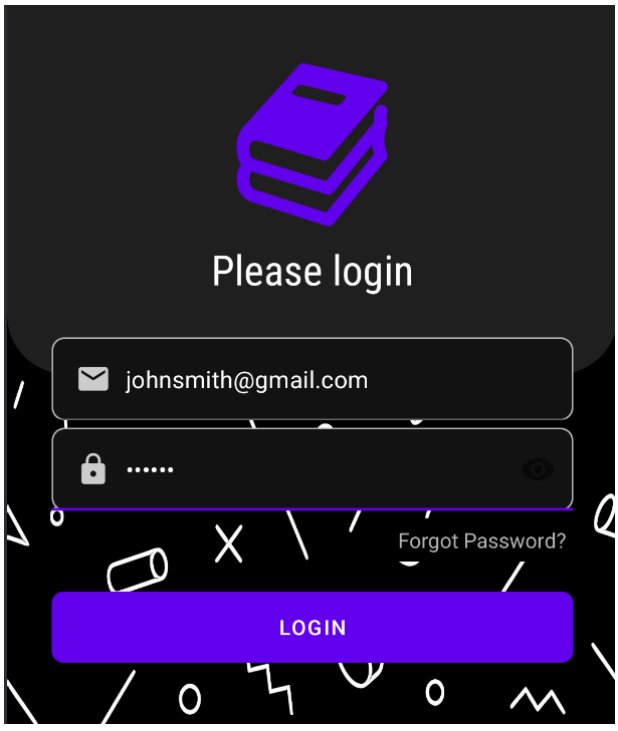


Рисунок 3.8 – Автентифікація у додаток

Після успішної автентифікації, користувач потрапляє на свою інформаційну панель (рисунок 3.9), де всі ключові елементи відображаються коректно, включаючи кнопки навігації та підсумкову інформацію. Крім того, динамічні компоненти, такі як оновлення календаря, колекції книг та статистика читання, були протестовані на предмет належної функціональності та безперебійної взаємодії.

Рисунок 3.9 – Інформаційна панель користувача

Наступним було протестовано відображення читацьких колекцій (рисунок 3.10). Вони виводяться у вигляді списку, на який можна натиснути щоби відкрити його вміст. Також кожна кнопка містить додаткову кнопку, котра викликає меню для редагування або видалення колекції.

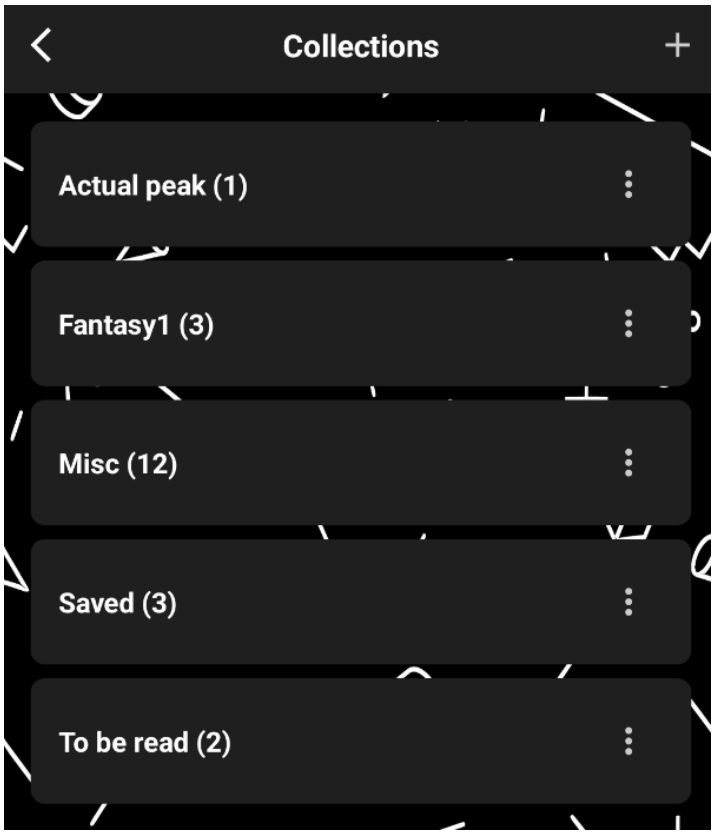


Рисунок 3.10 – Виведення списку колекцій користувача

Опісля було протестовано перегляд вмісту колекцій (рисунок 3.11). Вони виводять збережені у колекції книги у вигляді обкладинки та інформації про ступінь завершення.

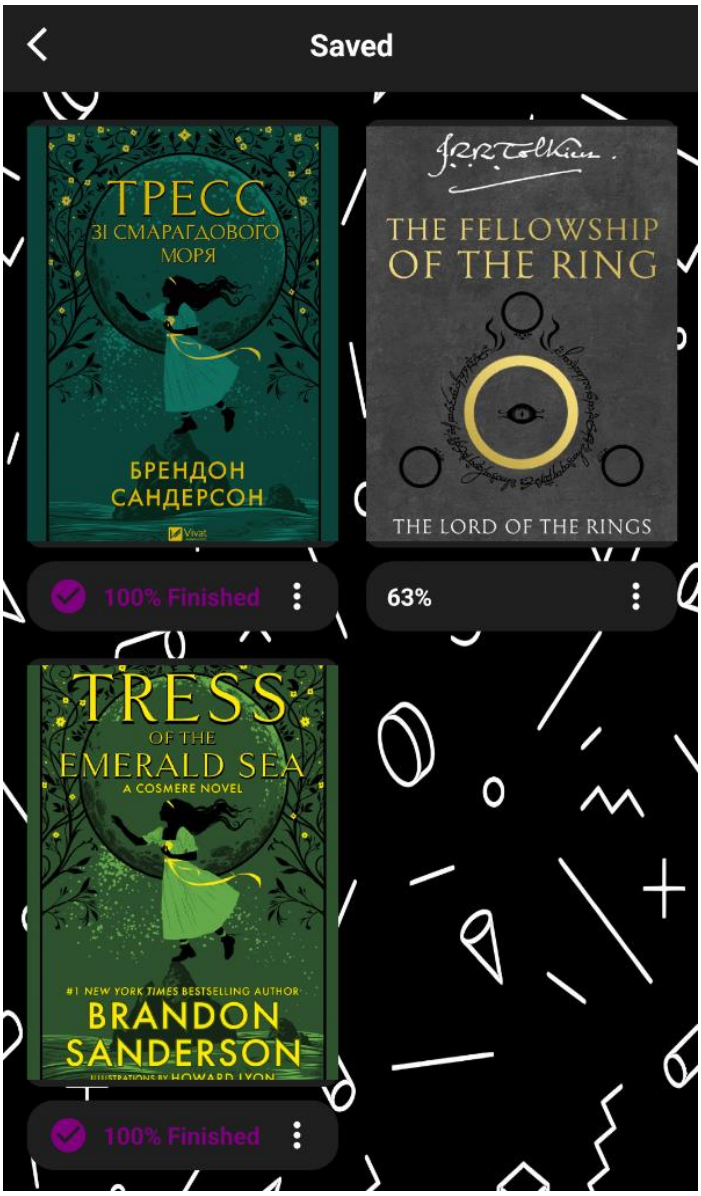


Рисунок 3.11 – Виведення вмісту колекцій

Окрім цього, було проведено тестування роботи сторінки читацьких сесій (рисунок 3.12). На цій сторінці користувач може починати, зупиняти, продовжувати та завершувати свої читацькі сесії. Окрім цього, дана сторінка виводить інформацію про наявні читацькі сесії та нотатки, загальний прогрес та приблизний час до завершення книги.

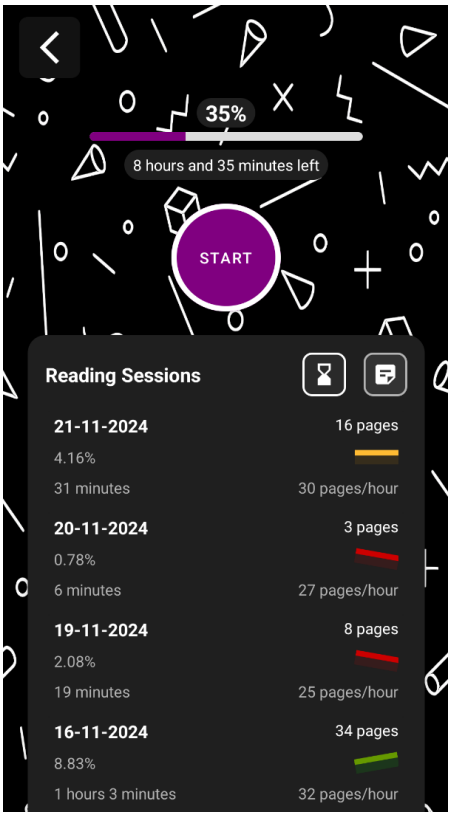


Рисунок 3.12 – Виведення інформації про читацькі сесії

Наступною було перевірено сторінку пошуку книг (рисунок 3.13), яка виводить результати до запиту користувача у вигляді списку книг, з кнопкою, що дозволяє додати обрану книгу у одну з існуючих колекцій.

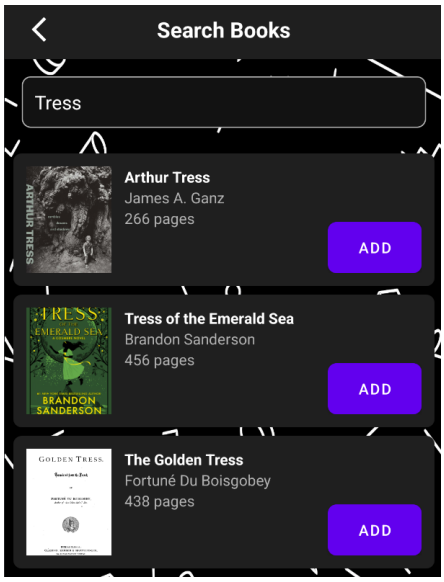


Рисунок 3.13 – Пошуку книг

Опісля було протестовано сторінку рекомендацій (рисунок 3.14), яка коректно надає користувачу персоналізовані рекомендації на основі його попередніх читацьких вподобань. На цій сторінці користувач може пролистати картку рекомендації направо щоб зберегти її до однієї з колекцій, або наліво, щоб позначити її як «не сподобалося».

Рисунок 3.14 – Виведення рекомендацій

Також було протестовано сторінку детальної статистики користувача (рисунок 3.15), яка надає коректно обчислену загальну статистику читання.

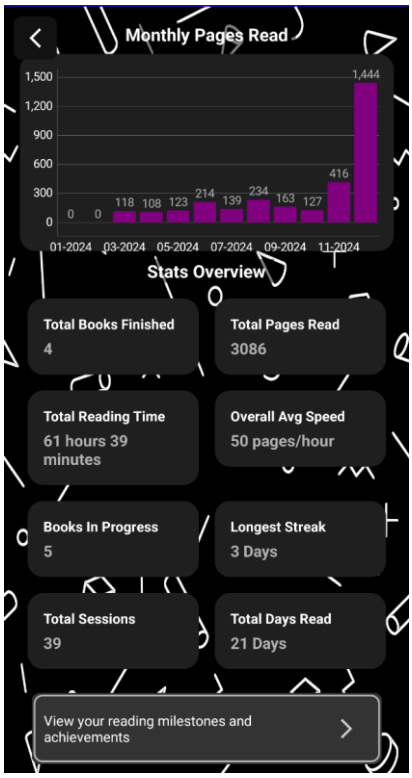


Рисунок 3.15 – Виведення детальної статистики

Після цього було протестовано діалогове меню з багаторівневими цілями (рисунок 3.16), які, завдяки елементам гейміфікації допомагають користувачам покращувати свої читацькі звички.

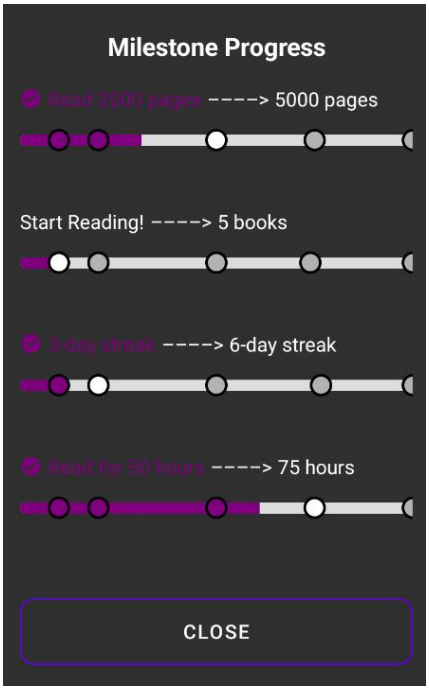


Рисунок 3.16 – Виведення багаторівневих цілей

Окрім цього, особливу увагу було приділено забезпеченню точного відображення та оновлення даних (рисунок 3.17), таких як відсоток прогресу книги та відображення читацьких сесій на календарі.

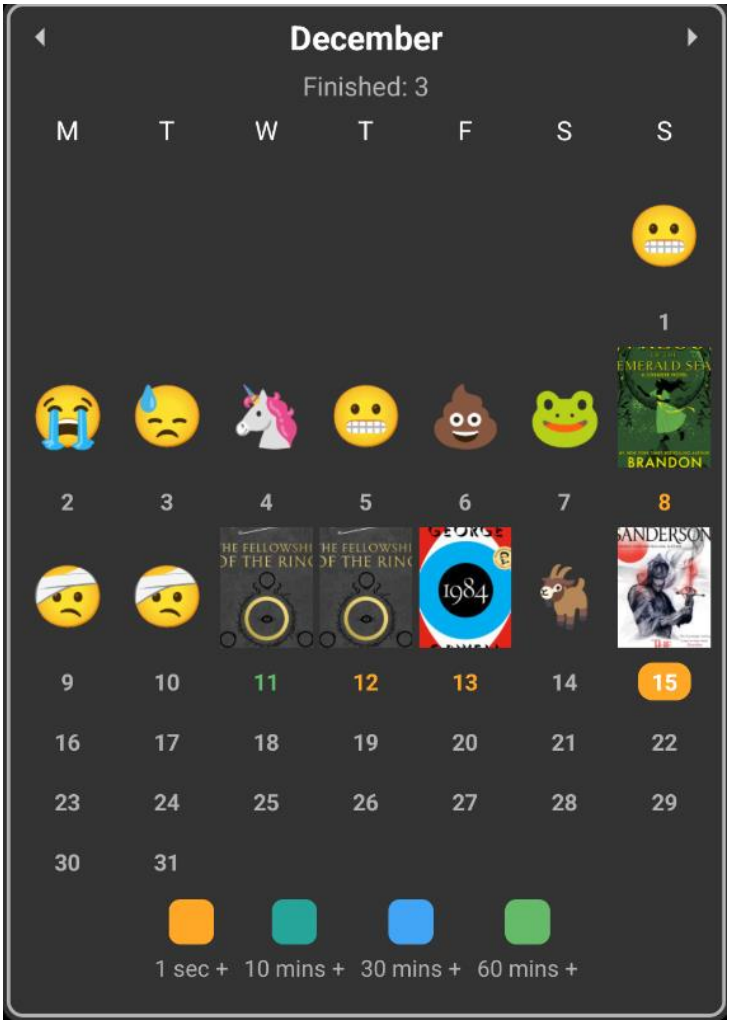


Рисунок 3.17 – Календар, оновлений відповідно до проведених читацьких сесій

Інтерактивні елементи, такі як жести та кнопки навігації, коректно працюють, чим забезпечують плавний та інтуїтивно зрозумілий користувацький досвід. Під час проведення даного тестування додаток продемонстрував високий рівень надійності та продуктивності, відповідаючи як функціональним вимогам, так і очікуванням користувачів.

4 ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Розробка додатку для відстеження читацьких сесій здійснювалася в умовах, що відповідають сучасним стандартам охорони праці та техніки безпеки. При створенні системи враховувалися вимоги нормативних документів, зокрема:

- Санітарні норми мікроклімату виробничих приміщень ДСН 3.3.6.042–99 [14];
- Наказ Міністерства соціальної політики України від 14.02.2018 №207 (НПАОП 0.00–7.15–18) [15];
- Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин (ДСанПіН 3.3.2.007–98), затверджені постановою Головного державного санітарного лікаря України від 01.12.1998 №7 [16];
- ДСТУ Б В.2.5-82:2016 "Електробезпека в будівлях і спорудах. Вимоги до захисних заходів від ураження електричним струмом" [17].

Для забезпечення комфортних умов праці під час створення програмного продукту особлива увага була приділена організації робочого місця. Відповідно до ДСН 3.3.6.042–99, робоча зона для робіт категорії «легка Іа» повинна забезпечувати оптимальні значення параметрів мікроклімату у холодний період року, зокрема:

- температура повітря у приміщенні повинна підтримуватися в діапазоні 22-24°С;
- відносна вологість повітря – у межах 40-60%;
- швидкість руху повітря не повинна перевищувати 0,1 м/с.

Дотримання цих параметрів дозволяє мінімізувати вплив несприятливих факторів на здоров'я розробника та сприяє підвищенню продуктивності праці. Робоче місце було обладнане сучасним комп’ютерним устаткуванням, яке

відповідає вимогам ДСН 3.3.6.037–99 щодо рівнів виробничого шуму, ультразвуку та інфразвуку.

Освітлення робочої зони виконувалося відповідно до НПАОП 0.00–7.15–18. Освітлення було організоване так, щоб забезпечити комфортний рівень освітленості робочого місця та створити належний контраст між екраном монітора і навколишнім середовищем, як це передбачено вимогами документа. Джерела світла розташовувалися таким чином, щоб уникнути відблисків на поверхні екрана, а яскравість екрана налаштовувалася користувачем відповідно до індивідуальних потреб для зменшення навантаження на очі.

Для уникнення перевтоми організовувались регулярні перерви на відпочинок, як це передбачено ДСанПіН 3.3.2.007–98. Відповідно до нормативів, після кожної години роботи за комп’ютером призначалися перерви тривалістю 15 хвилин, під час яких виконувалися спеціальні вправи для очей, шиї та спини. Це допомагає зменшити навантаження на зоровий апарат, покращити кровообіг та знизити ризик розвитку професійних захворювань, пов'язаних із тривалим перебуванням у статичному положенні.

У процесі роботи були враховані ключові аспекти техніки електробезпеки відповідно до ДСТУ Б В.2.5-82:2016. Комп'ютерне обладнання, яке використовувалося під час розробки програмного забезпечення, відповідало вимогам безпеки, зокрема:

- устаткування підключалося до електричної мережі лише з уземленням для мінімізації ризиків ураження струмом;
- рівні електромагнітних полів обмежено у межах допустимих норм, що не впливають на здоров'я працівників та роботу іншого обладнання;
- застосовано обладнання, яке виділяє мінімальну кількість тепла і шуму, для підтримання комфортного мікроклімату в приміщенні та зниження навантаження на організм.

Дотримання вимог цього стандарту забезпечує безпечну експлуатацію комп'ютерного обладнання та захист працівників від потенційних небезпек, пов'язаних з електричним струмом.

Окрім дотримання технічних та санітарно-гігієнічних вимог, важливим фактором під час розробки програмного забезпечення є забезпечення психофізіологічного комфорту працівника. Було створено умови для зниження рівня стресу та підвищення продуктивності за рахунок оптимізації робочого середовища. Наявність природного освітлення, регулярна вентиляція приміщення та забезпечення тиші позитивно впливають на здатність зосереджуватися протягом тривалого часу. Крім того, додаткові перерви для фізичної активності дозволяють підтримувати високу працездатність та знижують ризик професійних захворювань [18].

Організація робочого місця була виконана з урахуванням ергономічних вимог відповідно до ДСанПІН 3.3.2.007-98:

- монітор розташовано на відстані 60-70 см від очей, з урахуванням розмірів алфавітно-цифрових знаків і символів;
- висота стільця та стола налаштована таким чином, щоб забезпечити правильну поставу та запобігти перенапруженню хребта;
- кут огляду екрана був оптимізований для мінімізації навантаження на зір.

Таким чином, розробка додатку для відстеження читацьких сесій здійснювалася з дотриманням усіх нормативно-правових вимог, стандартів охорони праці та ергономічних норм. Виконання санітарно-гігієнічних вимог, організація безпечного та комфортного робочого місця, а також увага до технічних аспектів програмного забезпечення гарантують ефективність та безпеку процесу розробки даного додатку додатку.

4.2 Безпека в надзвичайних ситуаціях

Забезпечення пожежної безпеки на робочому місці є невід'ємною складовою організації процесу розробки програмного забезпечення, включаючи розробку

додатку для відстеження читацьких сесій. Відповідно до Кодексу цивільного захисту України [19] та нормативних документів, таких як Правила пожежної безпеки в Україні (НАПБ А.01.001-2014) [20], основні заходи безпеки поділяються на організаційні, технічні та експлуатаційні. Для запобігання виникненню пожежі під час роботи з комп'ютерним обладнанням та допоміжними пристроями, слід дотримуватися наступних вимог:

- запобігання утворенню горючого середовища:
    - використання негорючих або важкозаймистих матеріалів у робочому просторі;
    - зберігання горючих речовин у мінімально необхідній кількості, організація їх правильного розташування;
    - регулярне проведення провітрювання приміщення для підтримки концентрації горючих газів, пари, пилу та окисника поза межами пожежонебезпечних значень;
- запобігання виникненню джерел займання:
    - використання справного електрообладнання, що відповідає Правилам улаштування електроустановок (ПУЕ);
    - проведення регулярних оглядів електричних кабелів та пристроїв на предмет пошкоджень;
    - обладнання робочих місць засобами аварійного відключення живлення для запобігання перегріву або короткого замикання;
    - організація системи захисту від блискавок та електростатичного розряду, що є важливим для приміщень, у яких розміщується комп'ютерна техніка.

Усі робочі місця повинні бути оснащені системами протипожежного захисту та сигналізації відповідно до ДБН В.2.5-56:2014 [21], що забезпечує раннє виявлення загоряння та своєчасне реагування на надзвичайну ситуацію.

Організація заходів у надзвичайних ситуаціях визначається ДНАОП 0.00-4.14-94 [22] “Положення про розробку інструкцій з охорони праці”. У разі виникнення небезпечної ситуації, необхідно виконати наступні дії:

- Пожежа:
    - негайно повідомити про виникнення пожежі керівництву та викликати пожежну охорону;
    - відключити обладнання від електромережі для запобігання поширенню займання;
    - скористатися первинними засобами пожежогасіння для локалізації осередку займання;
- Ураження електричним струмом:
    - швидко відключити пристрій від електромережі або використати засоби з ізольованими ручками для припинення дії струму;
    - надати першу домедичну допомогу потерпілому, забезпечити доступ повітря та викликати екстрену допомогу;
- Виявлення несправностей:
    - при виявленні несправностей у роботі обладнання, що можуть призвести до аварійної ситуації, працівник зобов'язаний негайно повідомити про це відповідальну особу;
    - усунення несправностей проводиться лише після відключення пристрою від мережі та дотримання відповідних вимог безпеки.

Для мінімізації ризиків поширення вогню, робочі приміщення повинні бути обладнані системами пожежної сигналізації та аварійного освітлення, а конструкції приміщення відповідати вимогам II ступеня вогнестійкості згідно з ДБН В.1.1-7-2002 [23], що забезпечуватиме їхню здатність протистояти вогню в умовах надзвичайної ситуації.

На додаток до загальних вимог пожежної безпеки та захисту від аварійних ситуацій, впровадження заходів щодо контролю за дотриманням норм зберігання та використання електричних пристроїв значно зменшить ризики виникнення аварійних ситуацій. Це здійснюється за рахунок прокладення усіх кабельних з'єднань у відповідності до ДБН В.2.5-23:2010 [24] “Проектування електропостачання та електрообладнання будівель”, що забезпечує їхню стійкість до нагрівання та механічних пошкоджень. Для мінімізації ризику короткого

замикання слід використовувати лише сертифіковані розетки та захисні пристрої, які відповідають класу захисту IP44.

Окрім організаційних і технічних заходів пожежної безпеки, значну роль відіграє навчання працівників основам протипожежної безпеки та алгоритму дій у разі надзвичайних ситуацій, а саме організація інструктивної роботи, яка включає проведення вступного інструктажу, періодичні навчання та тренування з евакуації. Особливу увагу варто приділити ознайомленню з розташуванням засобів пожежогасіння, аварійних виходів та планів евакуації, що відповідає вимогам ДБН В.1.1-7-2002 щодо вогнестійкості конструкцій та пожежної безпеки приміщень.

Для підвищення рівня безпеки робочого середовища необхідно забезпечити дотримання вимог щодо використання електричних приладів. Усі пристрої повинні проходити регулярне технічне обслуговування для виявлення можливих дефектів і зношення. Крім того, для уникнення перенапруги в електромережі варто використовувати стабілізатори напруги та безперебійні блоки живлення (UPS), що дозволяє забезпечити безпечне завершення роботи обладнання під час раптових відключень електроенергії.

Таким чином, описані заходи з пожежної безпеки, запобігання аварійним ситуаціям та організації безпечного робочого середовища під час розробки програмного забезпечення відповідають чинним нормативним документам і стандартам. Особливу увагу приділено забезпеченню ефективного функціонування систем протипожежного захисту, дотриманню вимог до електробезпеки та організації навчання працівників алгоритмам дій у разі надзвичайних ситуацій. Зокрема, як зазначено у навчальному посібнику “Техноекологія та цивільна безпека. Частина «Цивільна безпека»” [25] та методичних рекомендаціях "Безпека в надзвичайних ситуаціях" [26], важливим є регулярне проведення інструктажів, тренувань з евакуації та перевірки працездатності протипожежного обладнання. Комплексний підхід до організації безпеки значно знижує ризики виникнення пожеж та інших надзвичайних ситуацій, забезпечуючи надійний та безпечний робочий процес.

ВИСНОВКИ

У даній кваліфікаційній роботі наведено процес розробки мобільного додатку для відстеження читацьких сесій з використання архітектурного шаблону «модель-вид-контролер».

Були досягнуті наступні результати:

1. Аналіз предметної області додатку.

Проведено комплексний аналіз поточного ландшафту мобільних додатків, пов'язаних з читанням, та виявлено прогалини в існуючих рішеннях. Цей аналіз дозволив сформувати функціональні вимоги та позиціонувати розроблений додаток як унікальний інструмент, орієнтований на відстеження сесій та персоналізовану аналітику.

2. Обґрунтування вибору архітектури.

Було обґрунтовано використання клієнт-серверної архітектури з використанням Firebase як внутрішньої служби для зберігання та синхронізації даних у режимі реального часу. Така архітектура забезпечує масштабованість, узгодженість даних між різними пристроями та безперебійну роботу користувачів.

3. Розробка моделі предметної області.

Було розроблено модель предметної області додатку, що включає такі ключові компоненти, як користувачі, книги, колекції, читацькі сесії, нотатки, рекомендації та аналітика. Модель визначає взаємозв'язки між цими компонентами для підтримки широкої функціональності та персоналізованого користувацького досвіду.

4. Реалізація основних функцій.

Для підтримки функціональності додатку було реалізовано основні класи:

- Відстеження та управління читацькими сесіями, включаючи збір даних про них в режимі реального часу, для подальшого аналізу та обчислення детальної статистики читання;

- управління колекціями для організації книг у визначені користувачем довільні категорії;
- інтеграція з Google Books API для здійснення пошуку книг та отримання їхніх метаданих;
- надання персоналізованих книжкових рекомендацій на основі вподобань користувача та історії читання.

5. Розробка GUI.

За допомогою Android Studio було розроблено графічний інтерфейс, орієнтований на користувача, з акцентом на інтуїтивну навігацію, естетичну привабливість та швидку роботу.

6. Тестування та забезпечення якості.

Для забезпечення коректності, надійності та продуктивності програмного забезпечення було проведено ретельне тестування. Функціональні та нефункціональні вимоги були перевірені, що підтвердило коректність роботи програми.

Таким чином, розроблений додаток є надійним, масштабованим рішенням, яке задовольняє потреби різних груп користувачів, поєднуючи аналітику в реальному часі, персоналізацію та зручні інтерфейси для покращення досвіду читання. Він слугує комплексною платформою для формування читацьких звичок і надання практичних знань про поведінку користувачів, створюючи основу для майбутнього розвитку технологій книжкових додатків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Кількість користувачів Goodreads. [Електронний ресурс] – Режим доступу до ресурсу: https://www.goodreads.com/blog/show/2302-goodreads-members-top-72-hit-books-of-the-year-so-far

2. Документація Android Studio. [Електронний ресурс] – Режим доступу до ресурсу: https://developer.android.com/studio

3. Документація Firebase. [Електронний ресурс] – Режим доступу до ресурсу: https://firebase.google.com/docs

4. Документація Kotlin. [Електронний ресурс] – Режим доступу до ресурсу: https://kotlinlang.org/docs/home.html

5. Петрик М. Р., Мудрик І. Я. Проєктування програмного забезпечення на основі об’єктно-орієнтованого аналізу вимог та інструментальних засобів розробки IBM Rational Software Architect / М. Р. Петрик, І. Я. Мудрик – Тернопіль : ТНТУ ім. І. Пулюя, 2022. – 56 с.

6. Real-Time Database Systems / P. Mejia Alvarez et al. Cham : Springer Nature Switzerland, 2024.

7. O’Regan G. Agile Methodology. *Undergraduate Topics in Computer Science*. Cham, 2022. P. 247–255.

8. Bass J. M. Architecture. *Agile Software Engineering Skills*. Cham, 2022. P. 111–128.

9. Difference between Data-Centric and Service-Centric Architecture. [Електронний ресурс] – Режим доступу до ресурсу: https://www.geeksforgeeks.org/difference-between-data-centric-and-service-centric-architecture/

10. Документація Google Books API. [Електронний ресурс] – Режим доступу до ресурсу: https://developers.google.com/books/docs/v1/using

11. The Unified Modeling Language. [Електронний ресурс] – Режим доступу до ресурсу: https://www.uml-diagrams.org/

12. Sarcar V. MVC Pattern. *Java Design Patterns*. Berkeley, CA, 2022. P. 593–617.

13. Stefanyshyn, V. , Stefanyshyn, I. , Pastukh, O. , Yatsyshyn, V. , Yakymenko. Accuracy of software and hardware of computer systems for human-machine interaction, I. CEUR Workshop Proceedings, 2024, 3842, pp. 178–183.

14. ДСН 3.3.6.042-99 «Санітарні норми мікроклімату виробничих приміщень». [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/va042282-99#Text

15. Наказ «Про затвердження Вимог щодо безпеки та захисту здоров'я працівників під час роботи з екранними пристроями». [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0508-18#Text

16. ДСанПіН 3.3.2.007-98 «Норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин». [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/rada/show/v0007282-98#Text

17. ДСТУ Б В.2.5-82:2016 "Електробезпека в будівлях і спорудах. Вимоги до захисних заходів від ураження електричним струмом". [Електронний ресурс] – Режим доступу до ресурсу: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=65395

18. Грибан В. Г., Негодченко О. В. Охорона праці : навч. посіб. Київ : Центр учбової літератури, 2009, 280 с.

19. Кодекс цивільного захисту України. [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/5403-17#Text

20. Наказ «Про затвердження Правил пожежної безпеки в Україні». [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0252-15#Text

21. ДСТУ ISO 11601:2019 Пожежогасіння. Вогнегасники пересувні. Експлуатаційні характеристики та конструкція. [Електронний ресурс] – Режим доступу до ресурсу: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=86823

22. Наказ «Про затвердження Положення про розробку інструкцій з охорони праці». [Електронний ресурс] – Режим доступу до ресурсу: https://zakon.rada.gov.ua/laws/show/z0226-98#Text

23. ДБН В.1.1-7:2016 Пожежна безпека об`єктів будівництва. Загальні вимоги. [Електронний ресурс] – Режим доступу до ресурсу: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=68456

24. ДБН В.2.5-23:2010 Інженерне обладнання будинків і споруд. Проектування електрообладнання об`єктів цивільного призначення. [Електронний ресурс] URL: https://online.budstandart.com/ua/catalog/doc-page.html?id_doc=25887

25. Навчальний посібник «ТЕХНОЕКОЛОГІЯ ТА ЦИВІЛЬНА БЕЗПЕКА.

ЧАСТИНА «ЦИВІЛЬНА БЕЗПЕКА»» / автор-укладач В.С. Стручок– Тернопіль: ФОП Паляниця В. А., – 156 с.

26. Методичний посібник для здобувачів освітнього ступеня «магістр» всіх спеціальностей денної та заочної (дистанційної) форм навчання «БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ» / В.С. Стручок –Тернопіль: ФОП Паляниця В. А., –156 с.

ДОДАТКИ

ДОДАТОК А

Публікація в науковому виданні

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
імені ІВАНА ПУЛЮЯ
НАУКОВЕ ТОВАРИСТВО ім. ШЕВЧЕНКА

М А Т Е Р І А Л И

XII НАУКОВО-ТЕХНІЧНОЇ КОНФЕРЕНЦІЇ

«ІНФОРМАЦІЙНІ МОДЕЛІ, СИСТЕМИ ТА ТЕХНОЛОГІЇ»

18–19 грудня 2024 року

ТЕРНОПІЛЬ
2024

УДК 004.41
М. Франчевський; М. Петрик, д.ф.-м.н., професор
(Тернопільський національний технічний університет імені Івана Пулюя, Україна)

РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ ДЛЯ ПЕРСОНАЛІЗОВАНОЇ АНАЛІТИКИ ПРОГРЕСУ ЧИТАННЯ

UDC 004.41
M. Franchevskyy; M. Petryk, Dr., Prof.

DEVELOPMENT OF A MOBILE APPLICATION FOR PERSONALIZED ANALYTICS OF READING PROGRESS

Мобільні додатки стали важливими інструментами для підвищення продуктивності та особистого розвитку, а програми-трекери для читання дають користувачам можливість відстежувати свої звички, ставити цілі та аналізувати прогрес. Мобільний додаток, призначений для персоналізованої аналітики прогресу читання, є комплексним рішенням для відстеження читацьких сесій. Він дозволяє користувачам реєструвати активність, вимірювати такі показники, як швидкість та середня швидкість читання, витрачений час, а також аналізувати шляху до завершення книг. Інтегруючи внутрішні сервіси Firebase, додаток забезпечує безпечне зберігання та синхронізацію даних, дозволяючи користувачам мати доступ до своїх даних на різних пристроях. Використання архітектурного шаблону модель-вид-контролер (MVC) забезпечує модульність, зручність у супроводі і масштабованість, що робить його ефективною основою для реалізації даного додатку [1].

Шаблон MVC є фундаментальною основою архітектури додатку. Модель інкапсулює основну бізнес-логіку, керує зберіганням даних і взаємодією з Firebase. Вид відповідає за інтерфейс користувача, забезпечуючи інтуїтивно зрозумілий і візуально привабливий досвід, а контролер виступає в ролі посередника, з'єднуючи модель і вид.

Ключовою особливістю програми є її персоналізована аналітика, така як середня швидкість читання, приблизний час, необхідний для завершення книги, а також візуальні підсумки читацьких звичок. Завдяки оновленню даних у режимі реального часу через Firebase, користувачі отримують динамічну інформацію, яка адаптується до їхньої читацької поведінки [2]. Така персоналізація спрямована на посилення залученості користувачів і сприяння формуванню стійких читацьких звичок.

Використання шаблону MVC надає низку переваг, серед яких легше тестування та обслуговування, більша гнучкість у розширенні додатку без впливу на існуючий функціонал. Процес розробки також передбачає подолання таких викликів, як забезпечення безперебійної синхронізації даних та розробка модульної структури, яка підтримує масштабованість та адаптивність.

Таким чином, даний проєкт демонструє практичне застосування архітектури MVC для розробки надійного та зручного мобільного додатку. Поєднуючи інтуїтивно зрозумілий дизайн з детальною аналітикою, додаток надає користувачам змістовну інформацію про їхній прогрес у читанні, що робить його цінним інструментом для читачів.

Література

1. Freeman E., Robson E. Head First Design Patterns: Building Extensible and Maintainable Object-Oriented Software. 2nd ed. O'Reilly Media, 2021. 669 p.
2. Android Programming: The Big Nerd Ranch Guide / B. Sills et al. 5th ed. Addison-Wesley, 2022. 688 p.

ДОДАТОК Б

Лістинг коду розробленого додатку

Лістинг 1 – основна частина коду класу «UserStatsActivity»

```
    private fun loadReadingSessions() {
            val startOfMonth = calendar.clone() as Calendar
            startOfMonth.set(Calendar.DAY_OF_MONTH, 1) // First day of
the month

            val endOfMonth = calendar.clone() as Calendar
            endOfMonth.set(Calendar.DAY_OF_MONTH,
calendar.getActualMaximum(Calendar.DAY_OF_MONTH)) // Last day of the month


readingSessionController.loadSessionsWithPageCount(startOfMonth,
endOfMonth) { sessionsWithPageCounts, error ->
                if (sessionsWithPageCounts != null) {
                    sessionData.clear()
                    sessionData.addAll(sessionsWithPageCounts.map {
it.first }) // Extract sessions only
                    calculateAndDisplayStats(sessionsWithPageCounts)
                    setupMonthlyPagesGraph()
                } else {
                    Toast.makeText(this, "Failed to load sessions:
$error", Toast.LENGTH_SHORT).show()
                }
            }
        }

        private fun setupMonthlyPagesGraph() {
            val barChart =
findViewById<BarChart>(R.id.monthlyPagesGraphContainer)
            val entries = mutableListOf<BarEntry>()

            // Generate all months of the year in "MM-YYYY" format
            val calendar = Calendar.getInstance()
            val allMonths = mutableListOf<String>()
            for (i in 0 until 12) {
                calendar.set(Calendar.MONTH, i)
                val monthYear = "${(i + 1).toString().padStart(2, '0')}-
${calendar.get(Calendar.YEAR)}"
                allMonths.add(monthYear)
            }

            // Group sessions by month and year
            val groupedByMonth: Map<String, List<ReadingSessionModel>> =
sessionData.groupBy {
                val dateParts = it.date?.split("-")?.map(String::toInt)
                if (dateParts != null && dateParts.size == 3) {
                    "${dateParts[1].toString().padStart(2, '0')}-
${dateParts[2]}" // Month-Year as key
                } else {
                    null
                }
            }.filterKeys { it != null } as Map<String,
List<ReadingSessionModel>> // Explicit type-cast
```

```
            // Create BarEntry for each month (default to 0 if no data)
            allMonths.forEachIndexed { index, monthYear ->
                val totalPages = groupedByMonth[monthYear]?.sumOf {
it.pagesRead } ?: 0
                entries.add(BarEntry(index.toFloat(),
totalPages.toFloat()))
            }

            val dataSet = BarDataSet(entries, null) // Removed label
            dataSet.color = getColor(R.color.ripple_color)
            dataSet.valueTextColor = getColor(R.color.text_secondary)
            dataSet.valueTextSize = 12f

            val data = BarData(dataSet)
            barChart.data = data
            barChart.invalidate()

            // Customize X-axis for month-year labels
            val xAxis = barChart.xAxis
            xAxis.position = XAxis.XAxisPosition.BOTTOM
            xAxis.granularity = 1f
            xAxis.setDrawGridLines(false)
            xAxis.textColor = getColor(R.color.text_primary)
            xAxis.textSize = 12f
            xAxis.valueFormatter = object : ValueFormatter() {
                override fun getFormattedValue(value: Float): String {
                    return allMonths.getOrNull(value.toInt()) ?: ""
                }
            }

            // Customize Y-axis
            val yAxis = barChart.axisLeft
            yAxis.textColor = getColor(R.color.text_primary)
            yAxis.textSize = 12f
            barChart.axisRight.isEnabled = false

            // General chart customization
            barChart.description.isEnabled = false
            barChart.legend.isEnabled = false
            barChart.animateY(1000)
        }

        private fun calculateAndDisplayStats(sessionsWithPageCounts:
List<Pair<ReadingSessionModel, Int>>) {
            // Total books finished
            booksFinished = sessionsWithPageCounts.groupBy {
it.first.bookTitle }
                .count { (_, sessionPairs) ->
                    val lastSession = sessionPairs.maxByOrNull {
it.first.timestamp }
                    lastSession?.first?.endPage ?: 0 >=
lastSession?.second ?: Int.MAX_VALUE
                }
            binding.totalBooksFinished.text = booksFinished.toString()

            // Books in progress
```

```
            val booksInProgress = sessionsWithPageCounts.groupBy {
it.first.bookTitle }
                .count { (_, sessionPairs) ->
                    val lastSession = sessionPairs.maxByOrNull {
it.first.timestamp }
                    lastSession?.first?.endPage ?: 0 <
lastSession?.second ?: Int.MAX_VALUE
                }
            binding.booksInProgress.text = booksInProgress.toString()

            // Total pages read
            totalPagesRead = sessionsWithPageCounts.sumOf {
it.first.pagesRead }
            binding.totalPagesRead.text = totalPagesRead.toString()

            // Total reading time
            totalReadingTimeMillis = sessionsWithPageCounts.sumOf {
it.first.durationMillis }
            binding.totalReadingTime.text =
formatDuration(totalReadingTimeMillis)

            // Average reading speed
            val avgSpeed =
ReadingSessionModel.calculateAverageSpeed(sessionsWithPageCounts.map {
it.first })
            binding.avgReadingSpeed.text = "$avgSpeed pages/hour"

            // Longest streak
            longestStreak = calculateLongestStreak()
            binding.longestStreak.text = "$longestStreak Days"

            // Total sessions
            binding.totalSessions.text = sessionData.size.toString()

            // Total days read
            val totalDaysRead = sessionsWithPageCounts.mapNotNull {
it.first.date }.distinct().size
            binding.daysReadThisMonth.text = "$totalDaysRead Days"
        }

        private fun calculateLongestStreak(): Int {
            val dates = sessionData.mapNotNull { it.date }
                .map { it.split("-").map(String::toInt) }
                .map { (day, month, year) -> Triple(day, month, year) }
                .sortedWith(compareBy({ it.third }, { it.second }, {
it.first }))

            var longestStreak = 0
            var currentStreak = 1

            for (i in 1 until dates.size) {
                val prev = dates[i - 1]
                val current = dates[i]

                val isConsecutive = calendar.run {
                    set(prev.third, prev.second - 1, prev.first)
                    add(Calendar.DAY_OF_YEAR, 1)
```

```
                    get(Calendar.DAY_OF_MONTH) == current.first &&
                            get(Calendar.MONTH) == current.second - 1 &&
                            get(Calendar.YEAR) == current.third
                }

                if (isConsecutive) {
                    currentStreak++
                } else {
                    longestStreak = max(longestStreak, currentStreak)
                    currentStreak = 1
                }
            }
            return max(longestStreak, currentStreak)
        }

        private fun formatDuration(durationMillis: Long): String {
            val totalMinutes = (durationMillis / (1000 * 60)).toInt()
            val hours = totalMinutes / 60
            val minutes = totalMinutes % 60
            return if (hours > 0) "$hours hours $minutes minutes" else
"$minutes minutes"
        }

        private fun showMilestonesDialog() {
            // Create and inflate the dialog
            milestonesDialog = Dialog(this)
            val dialogView =
LayoutInflater.from(this).inflate(R.layout.dialog_milestones, null)
            milestonesDialog.setContentView(dialogView)

            setupPagesReadMilestone(dialogView, totalPagesRead)
            setupBooksReadMilestone(dialogView, booksFinished)
            setupLongestStreakMilestone(dialogView, longestStreak)
            setupTimeSpentMilestone(dialogView, totalReadingTimeMillis)


milestonesDialog.window?.setBackgroundDrawableResource(R.drawable.shape_rou
nded_dialog)

            val closeDialogButton =
dialogView.findViewById<View>(R.id.closeDialogButton)
            closeDialogButton.setOnClickListener {
milestonesDialog.dismiss() }

            milestonesDialog.show()
        }

        private fun setupPagesReadMilestone(dialogView: View,
totalPagesRead: Int) {
            // Define the pages read goals
            val pagesReadGoals = listOf(1000, 2000, 5000, 7500, 10000)

            // Find the ProgressBar and dots
            val progressBar =
dialogView.findViewById<ProgressBar>(R.id.pagesReadProgressBar)
            val dots = listOf(
                dialogView.findViewById<View>(R.id.pagesReadDot1),
```

```
                dialogView.findViewById<View>(R.id.pagesReadDot2),
                dialogView.findViewById<View>(R.id.pagesReadDot3),
                dialogView.findViewById<View>(R.id.pagesReadDot4),
                dialogView.findViewById<View>(R.id.pagesReadDot5)
            )

            // Update the progress bar and position dots
            updateProgressBar(progressBar, totalPagesRead,
pagesReadGoals)
            positionDots(progressBar, dots, pagesReadGoals)

            // Find the views in the dialog
            val checkIcon =
dialogView.findViewById<ImageView>(R.id.pagesReadCheckIcon)
            val goalReachedText =
dialogView.findViewById<TextView>(R.id.pagesReadGoalReachedText)
            val arrow =
dialogView.findViewById<TextView>(R.id.pagesReadArrow)
            val nextGoalText =
dialogView.findViewById<TextView>(R.id.pagesReadNextGoal)
            val congratulationText =
dialogView.findViewById<TextView>(R.id.pagesReadCongratulationsText)

            // Determine the completed and next goals
            val completedGoals = pagesReadGoals.filter { totalPagesRead
>= it }
            val nextGoalIndex = completedGoals.size
            val nextGoal = pagesReadGoals.getOrNull(nextGoalIndex)

            // Update the milestone text and icon
            if (completedGoals.isNotEmpty()) {
                goalReachedText.text = "Read ${completedGoals.last()}
pages"

goalReachedText.setTextColor(dialogView.context.getColor(R.color.ripple_col
or))
                checkIcon.visibility = View.VISIBLE
            } else {
                goalReachedText.text = "Start Reading!"

goalReachedText.setTextColor(dialogView.context.getColor(R.color.text_prima
ry))
                checkIcon.visibility = View.GONE
            }

            // Update the next goal text
            if (nextGoal != null) {
                arrow.visibility = View.VISIBLE
                nextGoalText.visibility = View.VISIBLE
                nextGoalText.text = "$nextGoal pages"

nextGoalText.setTextColor(dialogView.context.getColor(R.color.text_primary)
)
                congratulationText.visibility = View.GONE
            } else {
                arrow.visibility = View.GONE
                nextGoalText.visibility = View.GONE
```

```
                congratulationText.visibility = View.VISIBLE
                congratulationText.text = "🎉 Congratulations on
finishing all page milestones! 🏆"

congratulationText.setTextColor(dialogView.context.getColor(R.color.ripple_
color))
            }

            // Update the color of the dots based on progress
            for ((index, goal) in pagesReadGoals.withIndex()) {
                when {
                    totalPagesRead >= goal -> {
                        // Goal completed

dots[index].setBackgroundResource(R.drawable.dot_completed)
                    }
                    index == 0 && totalPagesRead < pagesReadGoals[index]
-> {
                        // First goal active

dots[index].setBackgroundResource(R.drawable.dot_active)
                    }
                    index > 0 && totalPagesRead >= pagesReadGoals[index -
1] -> {
                        // Next goal active

dots[index].setBackgroundResource(R.drawable.dot_active)
                    }
                    else -> {
                        // Inactive for other goals

dots[index].setBackgroundResource(R.drawable.dot_inactive)
                    }
                }
            }
        }

        private fun updateProgressBar(
            progressBar: ProgressBar,
            currentProgress: Int,
            milestoneGoals: List<Int>
        ) {
            // Set the max value of the progress bar to the last
milestone goal
            progressBar.max = milestoneGoals.last()

            // Calculate the progress as a percentage of the total
            val progressValue = if (currentProgress >=
milestoneGoals.last()) {
                milestoneGoals.last()
            } else {
                currentProgress
            }

            progressBar.progress = progressValue
        }
```

```
        private fun positionDots(
            progressBar: ProgressBar,
            dots: List<View>,
            milestoneGoals: List<Int>
        ) {
            progressBar.viewTreeObserver.addOnGlobalLayoutListener {
                // Total width of the ProgressBar (minus padding)
                val progressBarWidth = progressBar.width -
progressBar.paddingStart - progressBar.paddingEnd

                // Maximum value of the progress bar
                val maxGoal = milestoneGoals.last().toFloat()

                // Position each dot based on its goal
                dots.forEachIndexed { index, dot ->
                    val goalValue = milestoneGoals[index].toFloat()
                    val positionPercentage = goalValue / maxGoal

                    // Calculate the exact pixel position within the
progress bar
                    val dotPosition = progressBar.paddingStart +
(progressBarWidth * positionPercentage).toInt()

                    // Position the dot
                    dot.translationX = (dotPosition - (dot.width /
2)).toFloat()
                }
            }
        }
```

Лістинг 2 – основна частина коду класу «RecommendationModel»

```
    fun fetchRecommendations(callback: RecommendationCallback) {
            collectionController.fetchAllBooks { success, userBooks,
error ->
                if (!success || userBooks.isEmpty()) {
                    callback.onError("Save and rate more books to receive
recommendations.")
                    return@fetchAllBooks
                }

                // Calculate weighted scores for categories and authors
                val categoryScores = calculateWeightedScores(
                    userBooks.flatMap { it.volumeInfo.categories ?:
emptyList() },
                    recentBooks = userBooks.takeLast(5)
                )
                val authorScores = calculateWeightedScores(
                    userBooks.flatMap { it.volumeInfo.authors ?:
emptyList() },
                    recentBooks = userBooks.takeLast(5)
                )

                // Get top-ranked categories and authors
                val topCategories =
categoryScores.entries.sortedByDescending { it.value }
                    .map { it.key }
```

```
                    .take(2) // Top 2 categories
                val topAuthors = authorScores.entries.sortedByDescending
{ it.value }
                    .map { it.key }
                    .take(3) // Top 3 authors

                // Generate query using diversified results

searchBooksController.generateRecommendationQuery(topCategories,
topAuthors, object : SearchBooksController.SearchCallback {
                    override fun onSuccess(bookResponse: List<BookItem>)
{
                        // Filter disliked books and apply dynamic
filters
                        val filteredBooks = bookResponse.filter {
                            it.id !in dislikedBooks &&
filterBookByCriteria(it)
                        }
                        val diversifiedBooks =
diversifyRecommendations(filteredBooks)

                        if (diversifiedBooks.isEmpty()) {
                            callback.onError("No matching books found.
Save and rate more books.")
                        } else {
                            callback.onSuccess(diversifiedBooks)
                        }
                    }

                    override fun onError(message: String) {
                        callback.onError("Error fetching recommendations:
$message")
                    }
                })
            }
        }

        private fun calculateWeightedScores(items: List<String>,
recentBooks: List<BookItem>): Map<String, Double> {
            val recentWeights = recentBooks.flatMap {
it.volumeInfo.categories ?: emptyList() }
                .groupingBy { it }
                .eachCount()
                .mapValues { (_, count) -> count * 1.5 } // Boost weights
for recent interactions

            val itemCount = items.groupingBy { it }.eachCount()

            return itemCount.mapValues { (key, count) ->
                val recentWeight = recentWeights[key] ?: 0.0
                (count + recentWeight) * 100 / items.size.toDouble() //
Normalize by total count
            }
        }

        private fun filterBookByCriteria(book: BookItem): Boolean {
            val minRating = userPreferences["minRating"] as Double
```

```
            val minPageCount = userPreferences["minPageCount"] as Int
            val maxPageCount = userPreferences["maxPageCount"] as Int

            return (book.volumeInfo.averageRating ?: 0.0) >= minRating &&
                    (book.volumeInfo.pageCount ?: 0) in
minPageCount..maxPageCount &&
                    checkPreferredGenres(book)
        }

        private fun checkPreferredGenres(book: BookItem): Boolean {
            val preferredGenres = userPreferences["preferredGenres"] as
List<String>
            return if (preferredGenres.isNotEmpty()) {
                book.volumeInfo.categories?.any { it in preferredGenres }
?: false
            } else true
        }

        private fun diversifyRecommendations(books: List<BookItem>):
List<BookItem> {
            val booksByCategory = books.groupBy {
it.volumeInfo.categories?.firstOrNull() ?: "Misc" }
            val diversifiedBooks = mutableListOf<BookItem>()

            val quota = 2 // Ensure at least 2 books per category
            booksByCategory.forEach { (_, group) ->
                diversifiedBooks.addAll(group.sortedByDescending {
it.volumeInfo.averageRating }
                    .take(quota)
                    .shuffled() // Add randomness
                )
            }

            // Add a few random books outside preferences for discovery
            val serendipitousBooks = books.filter {
it.volumeInfo.categories?.none { category ->
                category in userPreferences["preferredGenres"] as
List<String>
            } == true }.shuffled().take(2)

            return (diversifiedBooks + serendipitousBooks).distinctBy {
it.id }
        }
```

Лістинг 3 – основна частина коду класу «SearchBooksController»

```
// Search for books based on a query
    fun searchBooks(
        query: String,
        callback: SearchCallback
    ) {
        val apiService = RetrofitInstance.api

        // Prioritize title, then broader matches
        val Query = "intitle:$query OR $query"
```

```
            apiService.searchBooks(Query).enqueue(object :
Callback<BookResponse> {
                override fun onResponse(call: Call<BookResponse>,
response: Response<BookResponse>) {
                    if (response.isSuccessful && response.body() != null)
{
                        val bookItems = response.body()!!.items

                        // Post-filter books to ensure relevance
                        val filteredBooks = bookItems?.filter {
                            val pageCount = it.volumeInfo.pageCount
                            val language =
it.volumeInfo.language?.lowercase() ?: ""

                            // Exclude books in russian and with 0 pages
                            language != "ru" && (pageCount == null ||
pageCount > 0)
                        }

                        if (filteredBooks.isNullOrEmpty()) {
                            callback.onError("No books matched the
query.")
                        } else {
                            callback.onSuccess(filteredBooks)
                        }
                    } else {
                        callback.onError("Failed to fetch data. Error
code: ${response.code()}")
                    }
                }

                override fun onFailure(call: Call<BookResponse>, t:
Throwable) {
                    callback.onError("Failed to fetch data:
${t.message}")
                }
            })
        }

        fun generateRecommendationQuery(
            categories: List<String>?,
            authors: List<String>?,
            callback: SearchCallback,
            maxResults: Int = 10
        ) {
            val apiService = RetrofitInstance.api

            // Generate individual category and author queries
            val topCategory = categories?.firstOrNull()?.let {
"subject:$it" }
            val topAuthor = authors?.firstOrNull()?.let { "inauthor:$it"
}

            // Combine category and author queries
            val Query = when {
                topCategory != null && topAuthor != null -> "$topCategory
OR $topAuthor"
```

```
                topCategory != null -> topCategory
                topAuthor != null -> topAuthor
                else -> ""
            }

            if (Query.isBlank()) {
                callback.onError("No categories or authors available to
generate recommendations.")
                return
            }

            Log.d("RecommendationQuery", "Generated Query: $Query")

            // Make the API call with maxResults
            apiService.searchBooks(Query, maxResults).enqueue(object :
Callback<BookResponse> {
                override fun onResponse(call: Call<BookResponse>,
response: Response<BookResponse>) {
                    if (response.isSuccessful && response.body() != null)
{
                        Log.d("responseBodyLog", "response body:
${response.body()}")
                        val bookItems = response.body()!!.items

                        if (bookItems.isNullOrEmpty()) {
                            callback.onError("No books matched the
recommendation query.")
                        } else {
                            callback.onSuccess(bookItems)
                        }
                    } else {
                        callback.onError("Failed to fetch
recommendations. Error code: ${response.code()}")
                    }
                }

                override fun onFailure(call: Call<BookResponse>, t:
Throwable) {
                    callback.onError("Failed to fetch recommendations:
${t.message}")
                }
            })
        }
```

Лістинг 4 – основна частина коду класу «ReadingTrackerActivity»

```
    private fun setupBookDetails() {
            val highQualityThumbnailUrl =
bookThumbnailUrl?.replace("zoom=1", "zoom=0")
            if (!highQualityThumbnailUrl.isNullOrEmpty()) {
                Glide.with(this)
                    .load(highQualityThumbnailUrl.replace("http://",
"https://"))
                    .placeholder(R.drawable.ic_image_placeholder)
                    .error(R.drawable.ic_image_placeholder)
                    .into(bookThumbnail)
            } else {
```

```
bookThumbnail.setImageResource(R.drawable.ic_image_placeholder)
            }
        }

        private fun showStartPageDialog() {
            val lastSession = memorySessions[bookTitle]?.maxByOrNull {
it.timestamp } // Fetch latest session
            val defaultStartPage = lastSession?.endPage ?: 1 // Default
to 1 if no sessions exist

            val dialogView =
layoutInflater.inflate(R.layout.dialog_page_input, null)
            val dialog = Dialog(this, R.style.Theme_MyApp_Dialog)
            dialog.setContentView(dialogView)

            val pageInput =
dialogView.findViewById<EditText>(R.id.pageInputEt)
            val cancelBtn =
dialogView.findViewById<Button>(R.id.cancelBtn)
            val confirmBtn =
dialogView.findViewById<Button>(R.id.confirmBtn)

            pageInput.setText(defaultStartPage.toString()) // Prepopulate
the input

            cancelBtn.setOnClickListener { dialog.dismiss() }
            confirmBtn.setOnClickListener {
                val startPage = pageInput.text.toString().toIntOrNull()
                if (startPage != null && startPage in 1..bookPageCount) {
                    sessionStartPage = startPage
                    startReadingSession()
                    dialog.dismiss()
                } else {
                    Toast.makeText(this, "Invalid page number!",
Toast.LENGTH_SHORT).show()
                }
            }
            dialog.show()
        }

        private fun showEndPageDialog() {
            // Pause timer temporarily
            stopTimer()

            val dialogView =
layoutInflater.inflate(R.layout.dialog_page_input, null)
            val dialog = Dialog(this, R.style.Theme_MyApp_Dialog)
            dialog.setContentView(dialogView)

            val pageInput =
dialogView.findViewById<EditText>(R.id.pageInputEt)
            val cancelBtn =
dialogView.findViewById<Button>(R.id.cancelBtn)
            val confirmBtn =
dialogView.findViewById<Button>(R.id.confirmBtn)
```

```
            cancelBtn.setOnClickListener {
                // Resume timer if the dialog is canceled
                resumeTimer()
                dialog.dismiss()
            }

            confirmBtn.setOnClickListener {
                val endPage = pageInput.text.toString().toIntOrNull()
                if (endPage != null && endPage in
sessionStartPage..bookPageCount) {
                    sessionEndPage = endPage
                    finalizeSession() // Finalize the session if the end
page is valid
                    dialog.dismiss()
                } else {
                    Toast.makeText(this, "Invalid page number!",
Toast.LENGTH_SHORT).show()
                }
            }

            dialog.setCancelable(true)
            dialog.show()
        }

        private fun stopTimer() {
            if (isTracking) {
                totalElapsedTime += SystemClock.elapsedRealtime() -
sessionStartTime
                sessionStartTime = 0L
                handler.removeCallbacks(updateClockRunnable)
                isTracking = false
            }
        }

        private fun resumeTimer() {
            if (!isTracking) {
                sessionStartTime = SystemClock.elapsedRealtime()
                handler.post(updateClockRunnable)
                isTracking = true
            }
        }

        private fun startReadingSession() {
            isTracking = true
            sessionStartTime = SystemClock.elapsedRealtime()
            startStopButton.text = "Stop"
            readingClock.visibility = View.VISIBLE
            handler.post(updateClockRunnable)
        }

        private fun finalizeSession() {
            stopTimer()

            val session = ReadingSessionModel.createSession(
                startPage = sessionStartPage,
                endPage = sessionEndPage,
                durationMillis = totalElapsedTime,
```

```
                    date = getCurrentDate(),
                    bookTitle = bookTitle,
                    collectionKey = collectionKey
                )

                // Update progressPages
                bookProgressPages = sessionEndPage

                // Save the updated progressPages to Firebase
                saveProgressToFirebase()

                readingSessionController.addReadingSessionToMemory(
                    bookTitle = bookTitle,
                    durationMillis = totalElapsedTime,
                    startPage = sessionStartPage,
                    endPage = sessionEndPage,
                    memorySessions = memorySessions,
                    collectionKey = collectionKey
                )

                readingSessionController.saveReadingSession(
                    collectionKey = collectionKey,
                    bookTitle = bookTitle,
                    session = session
                ) { success, error ->
                    if (success) {
                        Toast.makeText(this, "Session saved!",
Toast.LENGTH_SHORT).show()
                        updateSessionList()
                    } else {
                        Toast.makeText(this, "Error saving session: $error",
Toast.LENGTH_SHORT).show()
                    }
                }

                resetSessionVariables()
                updateProgressUI()
            }

            private fun resetSessionVariables() {
                totalElapsedTime = 0L
                sessionStartTime = 0L
                startStopButton.text = "Start"
                readingClock.visibility = View.GONE
            }

            private fun updateSessionList() {
                sessionList.removeAllViews()

                val sessions = memorySessions[bookTitle]?.sortedByDescending
{ it.timestamp } ?: emptyList()

                if (sessions.isNotEmpty()) {
                    val avgSpeed =
ReadingSessionModel.calculateAverageSpeed(sessions)
                    val lowerBound = avgSpeed * 0.95
                    val upperBound = avgSpeed * 1.05
```

```
                sessions.forEach { session ->
                    val sessionView =
layoutInflater.inflate(R.layout.item_reading_session, null)

                    val dateText =
sessionView.findViewById<TextView>(R.id.dateText)
                    val pagesReadText =
sessionView.findViewById<TextView>(R.id.pagesReadText)
                    val percentageText =
sessionView.findViewById<TextView>(R.id.percentageText)
                    val sessionTimeText =
sessionView.findViewById<TextView>(R.id.sessionTimeText)
                    val speedGraph =
sessionView.findViewById<ImageView>(R.id.speedGraph)
                    val speedText =
sessionView.findViewById<TextView>(R.id.speedText)

                    // Calculate session-specific progress percentage
                    val sessionPercentage =
ReadingSessionModel.calculateSessionProgressPercentage(
                        session.pagesRead,
                        bookPageCount
                    )

                    dateText.text = session.date
                    pagesReadText.text = "${session.pagesRead} pages"
                    percentageText.text =
"%.2f%%".format(sessionPercentage) // Display percentage with two decimals
                    sessionTimeText.text =
formatDurationToReadable(session.durationMillis)
                    speedText.text = "${session.readingSpeed} pages/hour"

                    speedGraph.setImageResource(
                        when {
                            session.readingSpeed.toDouble() > upperBound
-> R.drawable.speed_graph_up
                            session.readingSpeed.toDouble() in
lowerBound..upperBound -> R.drawable.speed_graph_flat
                            else -> R.drawable.speed_graph_down
                        }
                    )

                    // Rotate the graph based on the speed range
                    speedGraph.rotation = when {
                        session.readingSpeed.toDouble() > upperBound -> -
10f // Green: Up
                        session.readingSpeed.toDouble() in
lowerBound..upperBound -> 0f // Yellow: Flat
                        else -> 10f // Red: Down
                    }

                    // Add session view to the layout
                    sessionList.addView(sessionView)
                }
            } else {
                // Placeholder when no sessions exist
```

```
                sessionList.addView(TextView(this).apply {
                    text = "No reading sessions yet."
                    textSize = 16f

setTextColor(resources.getColor(R.color.text_secondary, theme))
                    textAlignment = View.TEXT_ALIGNMENT_CENTER
                })
            }
        }

         private fun updateProgressUI() {
            val sessions = memorySessions[bookTitle] ?: emptyList()

            // Calculate the overall progress based on the most recent
session
            val currentProgressPages =
ReadingSessionModel.getCurrentProgressPages(sessions, bookProgressPages)
            val progressPercentage =
ReadingSessionModel.calculateProgressPercentage(currentProgressPages,
bookPageCount)

            // Update the progress bar and text with an integer
percentage
            progressBar.progress = progressPercentage
            progressText.text = "$progressPercentage%"

            if (progressPercentage == 100) {
                // Hide start/stop button and clock
                startStopButton.visibility = View.GONE
                readingClock.visibility = View.GONE

                // Show rating display
                ratingDisplay.visibility = View.VISIBLE
                loadRatingFromFirebase() // Ensure the rating is updated
from Firebase
            } else {
                // Show start/stop button and clock
                startStopButton.visibility = View.VISIBLE
                //readingClock.visibility = View.VISIBLE

                // Hide rating display
                ratingDisplay.visibility = View.GONE
            }

            // Estimate the remaining time to finish the book
            remainingTimeText.text =
ReadingSessionModel.estimateRemainingTime(
                bookPageCount,
                currentProgressPages,
                sessions
            )
        }
```

ДОДАТОК В

Диск з розробленою програмою