DEVELOPMENT OF A BOOK RECOMMENDER SYSTEM

By

Ling Shi Shi

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)
Faculty of Information and Communication Technology
(Kampar Campus)

JUNE 2025

DEVELOPMENT OF A BOOK RECOMMENDER SYSTEM

By

Ling Shi Shi

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)
Faculty of Information and Communication Technology
(Kampar Campus)

JUNE 2025

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and appreciation to my supervisor, Ts Dr Phan Koo Yuen who has given me this bright opportunity to develop book recommender system. It is my first step to establish a career in mobile application field. A million thanks to you.

Finally, I must say thanks to my parents and my family for their love, support, and continuous encouragement throughout the course.

COPYRIGHT STATEMENT

This Final Year Project proposal is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ABSTRACT

This project aims to create a hybrid book recommendation system that addresses key challenges such as usability and customization. Existing applications frequently fall short of providing tailored and intuitive experiences, leaving a gap for customers who want more control over their preferences. By combining collaborative filtering, which leverages user behavior like reading history and ratings, with content-based filtering that analyzes book metadata such as genre, author, and title, the system solves limitations such as data sparsity and the cold start problem. The report's literature review part includes reviews of applications such as Goodreads, Meet New Books, and StoryGraph. A review of existing applications demonstrates the need for a more flexible and user-centered approach. The proposed solution is a mobile application built with Flutter, Python, Firebase, and PostgreSQL that dynamically adjusts recommendations depending on ongoing user feedback. The system uses weighted hybrid methodologies to provide accurate, diverse and personalized book suggestions, with the goal of increasing user satisfaction and transforming the book discovery experience.

Area of Study (Minimum 1 and Maximum 2): Recommender System, Machine Learning

Keywords (Minimum 5 and Maximum 10): Book Recommend, Content Based, Collaborative Filtering, Hybrid Approach, Mobile Application

TABLE OF CONTENTS

APPENDIX A

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

CB	Content Based
CF	Collaborative Filtering
NLP	Natural Language Processing
MVC	Model-View-Controller
RMSE	Root Mean Squared Error
RAD	Rapid Application Development
TF-IDF	Term Frequency-Inverse Document Frequency
SVD	Singular Value Decomposition

CHAPTER 1: Introduction

1.1 Project Background

Finding the right book might be likened to unearthing a hidden treasure in the wide world of literature. Nevertheless, readers frequently have the difficult task of choosing their next book due to the abundance of categories, authors, and themes to choose from. Enter the book recommender system is an effective tool that makes use of the amount of data found in digital libraries to help readers in their quest. Book recommender systems evaluate user preferences, performance histories, and book attributes using complex algorithms. These systems deliver tailored recommendations based on user interests by learning about individual tastes and preferences. The future of reading and literary is greatly influenced by recommender systems, whose role is becoming growing more important as digital libraries grow, and users demand personalization.

In this age of changing with the times, machine learning and artificial intelligence are advancing at a rapid pace, leading to the development of recommender systems. Thanks to these technologies, the system can continuously learn from user interactions and improve its recommendations. A comprehensive grasp of user preferences is ensured by the diversity of algorithms, which range from content-based approaches that examine book features to collaborative filtering methods that compare user behaviors.

Collaborative filtering, which anticipates user preferences by finding similarities in users' past activities, is where recommender systems got their start. Conversely, content-based filtering makes predictions about user preferences based on the characteristics and features of items that the user has already shown interest in. The strengths and differences of diverse machine learning approaches now employed in the development of book recommender systems are thoroughly examined in this article. We carried out an extensive search in reputable academic sources like Google Scholar to carry out this study and at least 500 research papers related to this subject were found. Our goal in examining these articles is to find efficient algorithms, novel and creative approaches that will influence personalized book recommendations in the future. Through many papers, it was finally decided to use a hybrid algorithm combining

content based and collaborative filtering algorithms to develop the book recommendation system. Furthermore, our goal is to acquire a more profound comprehension of how machine learning methods might be applied to enhance the precision, relation, and user contentment of book recommendations in the current digital era.

1.2 Problem Statement

1.2.1 Data sparsity

The reason for data sparsity is that users often only rate a small number of products. [1] Due to the widespread application of management systems, the amount of information is increasing, and people are having to invest more time and effort in finding the right information. In addition, a lot of books are hard for users to find or leave comments on because of their complexity and diversity of books in the market. There is relatively little space used by new users for the database table. It is harder to apply recommendations effectively if a user has only made a few visits and has only rated or commented on one or two books. When the number of users is proportionally higher than the available ratings, matrices of user item ratings could be extremely large and sparse because most users do not leave ratings for many items. [2] This results in insufficient data to identify similar users or items, reducing the quality of recommendations.

1.2.2 Cold Start

The term "cold start" describes the recommendation system's incapacity to recommend products for new users or new products. The recommendation algorithm can determine a user's propensity for a product based on their personal profile and the books they have read in the past. However, for new users, they have yet to rate any of the programs. [3] Since it lacks sufficient behavioral data and book content information, the system cannot evaluate previous user behaviors to recommend books to new users that match the new user's preferences (called cold visitors), lacks the behaviors of other users who have specific preferences like the customer (called gray sheep), and there may be new products added to the system (called cold products).[4] It is also more challenging to extract and categorize the content of new books. Since they are unable to connect with other users, new users do not receive any recommendations when they sign up. The

recommendation system's accuracy should be raised by implementing enhancement strategies.

1.2.3 Scalability

The difficulty of managing big, dynamic datasets is referred to as scalability, and it calls for reliable, effective methods and systems. It has to do with how many users and items the system is intended to support. A significant amount of processing power is needed to determine user similarity in order to provide suggestions because there are millions of users and goods on the internet.[3] Handling massive volumes of data, such a matrix with potentially 30 million users and 50 million items, is more challenging for the system.[4] In order to improve recommendation system performance and maintain a balance between computing time and accuracy, scalability is mostly used to address the issue of increasing computing time.[5]

1.3 Motivation

The objective of this paper is to propose a version of an improved existing book recommendation system using hybrid techniques. This paper will thoroughly investigate existing algorithms to improve and develop a machine learning model that provides accurate, diverse and personalized recommendations.

Traditional recommendation models such as collaborative filtering and content-based filtering will be used as a basis for evaluating user preferences and similarities. Although traditional recommendation methods have been widely used, each method has its limitations. A hybrid book recommendation system combines the strengths of traditional approaches by leveraging user behavioral data as well as book attributes to increase user satisfaction, improve discovery, and adapt to a variety of user needs. The proposed system improves the overall efficiency of the model and provides the personalized and intuitive experience that users expect. Not only that, but this paper also aims to better understand the overall picture of readers' preferences in Malaysia.

1.4 Research Objectives

The objectives of this project are as follows:

1. To prepare and preprocess datasets on books and ratings from available datasets and organize data prepared for recommendations:
    - Gather relevant information about books, tags, and user ratings from available datasets and organize that data into a dataset prepared for recommendations.
    - Clean and preprocess the dataset by removing duplicates, handling any missing values, and normalizing the ratings in a manner that retains consistency throughout the dataset and provides for analysis.
    - Extract relevant features from the dataset, such as a book's genre, author, and tags, as well as capturing any patterns in user ratings for the dataset, which will be input into a recommendation algorithm.

2. To develop and evaluate different recommendation models including content-based filtering, collaborative filtering, and hybrid approaches:
    - Implement content-based filtering, which uses attributes of books like genre, author, and tags to recommend books that are similar to a user's input.
    - Implement collaborative filtering, which uses the user-item rating matrix to find similarities either between users or books items to help make ratings accurately based only on user information.
    - Combine both content-based and collaborative filtering approaches into the hybrid model and evaluate all models with RMSE in terms of accuracy and performance of recommendations.

3. To design and develop a mobile application using Flutter for book recommendations with personalized user experience:
    - Design and develop an app that uses Flutter that is clean and easy to use to allow users to navigate, browse, and interact with information presented in the book recommendations app.

- Implement key functionalities like secure login and registration, browsing books, rating books, and allowing users to save books to the user's personal library.
- Integrate a hybrid recommendation engine into the application so users can see curated, personalized book recommendations according to their personal preferences and rating history.

1.5 Project Scope and Direction

This project aims to develop a powerful and personalized book recommendation system using a hybrid approach combining content-based filtering and collaborative filtering algorithms to replace users' manual book search in traditional libraries. Cold start and data sparsity issues will be resolved by the system by combining a content-based method that emphasizes book metadata with collaborative filtering that examines user preferences and interactions to deliver users more varied and relevant book recommendations.

Furthermore, machine learning will be utilized in the development of a lightweight web application. Users of the app will be able to peruse suggested books, give them ratings and reviews. To guarantee smooth user engagement with the system, a user-friendly and responsive user interface will be created. The proposed system will manage user data in accordance with ethical standards and privacy legislation, guaranteeing users' anonymity and privacy at all times.

By enhancing the real-time aspect of suggestions, the purposed method would guarantee that customers receive timely and useful recommendations without having to wait around for a lengthy period. This is especially crucial to preserving user satisfaction and engagement because delayed recommendations might frustrate users and make them utilize the system less frequently.

The system will be fully developed, tested, and deployed to ensure that it meets performance standards. It is anticipated that the project will be finished in September 2025.

1.6 Contributions

1.6.1 Knowledge of the body

Everyone's life can be profoundly enhanced, their memory improved, as well as language and knowledge base strengthened through reading. However everyone has a finite brainpower, we can never learn everything. Accepting book recommendations will unintentionally introduce us to books we were unaware of, which will open our minds to the possibility of experiencing new things. Any university's library serves as a research and information hub as well as a vital source of educational materials for students. [6] The book recommendation system is like an electronic library that collects all kinds of books in the world and contains knowledge in various fields. Through an analysis of user-book interactions, including ratings and reviews, the recommendation system can discern user preferences and make recommendations. Behavioral economics, consumer psychology, and market research can all benefit from this analysis. The book recommendation system's technology and insights can also be applied to other industries, including music, movies, and news articles. This will increase the application of recommendation technology by providing a platform for the creation of recommendation systems in other fields.

1.6.2 Practical implications

This paper proposes multiple contributions. Users can easily apply the recommendation system in a straightforward manner. Interface operations enable the easy construction of tailored recommendation apps by introducing best practices into intuitive, intelligent scenarios. The system offers three modes to facilitate Internet information flow and build a platform more tailored to the user's interests and personality, such as guess you like, similar user, and popular recommendations. In addition, targeted suggestions are made based on user similarities and current popularity of books that match the features and topics of books that users may enjoy. This improves user experience overall and raises satisfaction and engagement levels. Users can also find books in the

recommender system that they are unable to locate through conventional browsing. These algorithms expose users to a greater choice of information, encouraging exploration and chance discovery, by suggesting books of comparable interests or categories. Rana et al. [7] computed value of RMSE for recommender system based on collaborative filtering is 1.504. The system will be built using a hybrid algorithm to improve the accuracy of recommending books that match user interests.

1.7 Report Organization

The following chapters provide details about this research. Chapter 1 provides an overview of the project, including the problem statement, objectives, project scope, and contributions. In Chapter 2, seven articles and three mobile application which is similar with book recommender app is reviewed. Chapter 3 discusses the proposed methodology and explained details using diagrams. Chapter 4 focuses on system design, with block diagrams, component specifications, and implementation guidelines. Chapter 5 details the system implementation, including hardware and software setup as well as configuration. Chapter 6 talks about the system evaluation and discussion. Lastly, Chapter 6 will talk about conclusion and recommendation to improve this project.

CHAPTER 2

Literature Reviews

2.1 Previous works

2.1.1 Recommender Systems Challenges and Solutions Survey

The content-based filtering method in the paper [8] recommends items by their descriptions, user tastes, and matching items similar to those in the ratings list. First, the content analyser needs to preprocess the information to extract structured and relevant information. After that, the data items analysed by feature extraction techniques are fed into the profile learner and filtering components. User profiles are constructed using the collected user preference data and generalize the data. The profile learner of the web recommender implements a feedback method that combines positive and negative example vectors into prototype vectors that represent user profiles. The filtering component computes the cosine similarity between the prototype vectors and the item vectors. Finally, it matches and recommends new and potentially interesting items to the user based on the user's profile. However, this method requires analysing and testing all item characteristics for recommendation. It also does not take into account user ratings of the item and does not include an assessment of the quality of the product.

2.1.2 A survey of active learning in collaborative filtering recommender systems

The collaborative filtering recommender system described in this study [9] leverages user ratings to produce recommendations for new items in which the user may be interested. Ratings are maintained in a m × n matrix, indicating the number of users and items. If a user rated an item, it is converted into data and assigned to the corresponding rating. When a new user signs up for the system or adds a new item, a new empty row or column is added to this matrix. This system uses the interaction and similarity between users and between items to compute recommendations. The system predicts the target user's ratings for unrated things, ranks them based on the expected ratings,

and recommends the items with the highest predicted ratings to the user. However, this solution has a cold start difficulty. The system does not have enough user ratings to provide appropriate suggestions. As a result, the algorithm is unable to appropriately recommend current items to new users or new items to current users. In addition to the cold start, this system also faces the problem of sparsity of rating data. If users only rate a few items, it is difficult for the system to have enough data to generate accurate recommendations.

2.1.3 Automatic recommendation system based on hybrid filtering algorithm

[3] described a hybrid system-based book recommendation system that predicts recommendations. The proposed approach combines collaborative filtering with content-based filtering in three stages. By comparing user profiles, it finds persons in the first phase who are comparable to the active user. In order to retrieve the user profile and locate similar users, it employs collaborative filtering. The user's interactions are preserved to the user profile via the ontology. The content-based filtering technique detects and generates suggestions based on similarities between user profiles and items in the book collection. In the following stage, it selects the candidate's item for each similar user based on vectors of the item contents and the user's profile. The final stage involves recommending items to the intended user based on the prediction value for each item, which was calculated using the Resnick prediction equation. The results of the experimental section show that the proposed hybrid filtering technique outperforms both content-based and traditional collaborative filtering. The proposed hybrid approach copes with the problems of over-specialization, content extraction, and new users or new items. However, this research work was conducted on an offline dataset and cannot examine the true acceptance of users on the provided recommendations.

2.1.4 Book Recommendation System through Content Based and Collaborative Filtering Method

[10] describes a book recommendation system that combines characteristics of content-based filtering, collaborative filtering, and association rule mining to provide efficient recommendations. It describes the book’s content and user's purchase history of a book

and recommends books with similar content based on a series of book features. The Equivalence Class Clustering and Bottom-up Lattice Traversal (ECLAT) algorithm is an efficient method for identifying frequent item groups. It discovers the elements at the bottom and makes appropriate suggestions. Compared to other methods, ECLAT is more efficient and timesaving because it searches the database only once for the full dataset. First, it scans the dataset to identify all the available books. Then it performs data preprocessing, which includes extracting the data required for mining. The books are classified according to categories and subcategories. Each class contains various types of books written by different authors. Content-based and collaborative filtering are applied to the identified books before recommending them to the user. However, it needs the development of appropriate recommendation modules based on user interests, as well as the coordination and implementation of content-based and collaborative filtering and user trust. Furthermore, the system is only accessible to individuals who are educated and familiar with computers and the internet.

2.1.5 Introducing Hybrid Technique for Optimization of Book Recommender System

The system proposed in [11] uses a hybrid technique that combines collaborative filtering and content-based techniques to predict recommended books. The hybrid technique uses collaborative filtering techniques to filter out users who have rated books. Then it filters the users using demographic features. The content-based technique compares the filtered users to find similar users and recommends the types of books that the users have viewed and rated by similar users. This reduces the number of user comparisons. This technique filters recommendations based on user profiles and demographic characteristics without considering user preference data, solving the cold start problem for new users or books. The ontology is also utilized to save user profiles, which simplifies user analysis and eliminates the complexity of scanning logs. However, different users have different tastes or preferences that can directly affect the system's recommendations. In addition, the number of ratings obtained is minimal in comparison to the number that has to be anticipated, hence suggestion accuracy drops as the number of users increases.

2.1.6 College Library Personalized Recommendation System Based on Hybrid Recommendation Algorithm

[12] developed a customized recommendation system for college libraries by combining collaborative and content-based filtering methods. The user-item rating matrix and clustering are utilized to address the data sparsity issue based on book and patron attributes. Additionally, the issue of new item cold start can be successfully resolved by this technique. By conducting comparative experiments on the dataset, the results show that the hybrid approach is more capable of making more accurate recommendations than a single approach. Finally, the design of personalized book recommendation system is implemented using Spark big data platform to confirm the feasibility of this algorithm in practical applications. However, the quantity of books read by users influences the recommendation algorithm. A smaller training set cannot fully express the user's preference.

2.1.7 A Hybrid Book Recommender System Based on Table of Contents (ToC) and Association Rule Mining

[13] described a hybrid book recommendation system that combines association rule mining with content-based and collaborative filtering methods. This system recommendation is divided into phases. Preprocessing is the initial step in extracting book features. During the registration process, users are required by the system to provide their areas of interest. As soon as the user begins engaging with the system, it registers their ratings for the books they have read or viewed as well as automatically and implicitly keeps track of their activity in order to create a profile of them. The PHP library PDF Parser is used to extract book content and information. The CF approach uses the user's ratings on a specific item to predict the user's interest in the next step. It compares the user's evaluation to the assessments of other users in order to identify similar users. By taking into account the ratings of their friends, the ratings of books that the new user has not yet reviewed are predicted. The CB approach then uses TF-IDF to analyse the book content and recommend books that are relevant to the user's interests. After performing the CB and CF techniques, the books that the reader can study next are determined by the system using association rule mining on the transactions whose categories match the books. Stronger rules are obtained by adjusting

the support and confidence parameters. The issues of sparse ratings, limited content analysis, and cold start can be resolved with this strategy. However, this system may have the same synonymy and polysemy problems as traditional recommender systems.

2.2 Comparison of Reviewed Techniques

2.2.1 Comparison of Reviewed Techniques using Machine Learning Methods

Paper Title	**Techniques Involved**	**Advantages**	**Disadvantages**
Recommender Systems Challenges and Solutions Survey [8]	Content-based filtering	The system did not use the user's information to recommend things. The framework can propose new items to consumers based on the similarity of the item's details.	This method requires analysing and testing all item characteristics for recommendation. It does not consider user ratings of the item and an assessment of the quality of the product.
A survey of active learning in collaborative filtering recommender systems [9]	Collaborative filtering, active learning	This approach may work very well in offline assessments since it can significantly increase the system's accuracy with the least amount of quantity of evaluations. Active learning approach chooses	This approach encounters the issue of cold start. There are not enough user ratings for it to produce reliable recommendations. As a result, neither new nor old items may be appropriately

		and implements standards for gathering information. It allows for improved suggestion generation and more accurately represents users' preferences	recommended by the system to new or existing users. The sparsity of rating data is another issue this system must deal with. It is challenging for the algorithm to have enough information to produce reliable suggestions if users only rate a small number of goods.
Introducing Hybrid Technique for Optimization of Book Recommender System [11]	Collaborative filtering and content-based techniques.	Content-based techniques reduce the number of user comparisons. Filtering recommendations based on user profiles and demographic information eliminates the cold start issue for new users and books. Ontologies simplify the user analysis task and reduces the	The recommendations of the system are influenced by the tastes or preferences of different users. The number of ratings obtained may be small and decrease the recommendation accuracy as the number of users increases.

		overhead of parsing log files.	

Table 2.2.1: Comparison of Reviewed Techniques

Each of the above-mentioned recommender system techniques has advantages and disadvantages specific to it. While content-based filtering works well for suggesting new products, it has trouble evaluating products based on user reviews and quality standards.[8] When paired with active learning, collaborative filtering can produce suggestions that are more accurate. Nonetheless, there are certain challenges related to data sparsity and cold start.[9] By combining the best features of both methods, hybrid approaches lessen many of their drawbacks. Hybrid systems can overcome the challenges of using a single algorithm by utilizing item similarity and user preferences, which increases their adaptability to a range of situations like book recommendations.[11] The goal of this study is to create recommender systems using hybrid algorithms that integrate collaborative and content-based filtering skills to generate suggestions that are more accurate, scalable, and personalized.

2.2.2 Comparison of similar applications:

Goodreads	**Meet New Books**	**StoryGraph**
Collaborative -Rate at least 20 books -Learn personal tastes from ratings, then generates recommendations -Compare book with rating each book rated by others to see what other books you have in common with them	Collaborative Filtering -Match book preferences with those of other readers and recommend books they're enjoying.	Mood-Based Collaborative Filtering -StoryGraph incorporates the mood of books (e.g., "lighthearted," "dark," "reflective") into its recommendations. It identifies users with similar preferences in both books and moods.

Content-based filtering -Classified by genre - Extract relevant features from books, such as genre, author, keywords, and description, create user profiles, and compare and match.	Content-based Filtering - Recommend books that are similar to the ones users recently viewed - Analyse the books placed on that shelf to identify common features - Predict user tastes and tailor personalized book recommendations based on features of rated or saved books, such as genre and theme	Natural Language Processing (NLP) for Mood Tagging -Analyse book descriptions, reviews, and other textual content to automatically tag books with appropriate moods, themes, and other attributes.
Clustering algorithms - Represent users or books as feature vectors, such as user ratings, book types. Then use clustering algorithms to group similar users or books together and recommend books of the same type as the books the user likes.	Natural Language Processing (NLP) - NLP techniques are used to analyse text data such as book descriptions and reviews to better understand the theme, genre, and sentiment of each title. This helps in tagging books with relevant keywords and sentiments.	Content-Based Filtering with Mood and Pace -Emphasizes matching books to users based on detailed metadata, including mood, pace (e.g., "fast-paced," "medium-paced"), and content (e.g., "character-driven," "plot-driven"). This helps to tailor recommendations more closely to a user's specific reading preferences.

Table 2.2.2: Comparison of similar applications

Book recommendation systems in different applications use a combination of methods such as collaborative filtering, content-based filtering and natural language processing to recommend personalized suggestions. This project selects several features of these applications and refers to Goodreads' collaborative filtering-based recommendation system design method. The system can analyse the characteristics of books from the books that users have rated and then compare and recommend with users who have left similar reviews. In addition, the project plans to identify user preferences by referring to the bookshelf organization of Meet New Books. The algorithm generates individualized recommendations based on the user-created shelf organization. StoryGraph stands out by incorporating mood and pace into its recommendations. The technology offers personalized recommendations based on the user's current mood and pace. In addition to these applied methods, it is also recommended to provide quizzes that analyse and tally users' answers before they search for books. The analysis is used to derive the user's book preferences and characteristics and to target recommendations to the user.

2.3 Limitation of Previous Studies

The content-based method mentioned in [8] is difficult to analyse and test all product features. Additionally, it ignores user reviews, which makes it challenging to assess the quality of the products. The collaborative algorithm that is discussed in [9] has a cold start issue and insufficient user ratings. Both new and experienced users cannot be properly recommended new or old things by the system. Sparse rating data is an issue for the system if users only rate a small number of products. The study that was referenced in [3] reveals a major assessment constraint because it was carried out with an offline dataset. Due to this restriction, the study was unable to evaluate the real level of user approval of the recommendations, which is essential information for determining the recommendation system's actual efficacy in practice. Based on user interests, the recommendation module covered in [10] must be put into practice. This entails adding a trust mechanism that represents users' faith in the system in addition to putting collaborative and content-based filtering into practice. Furthermore, The system may be less accessible to a larger audience because it is intended primarily for educated users who are familiar with computers and the Internet. The accuracy of

recommendations in the system described in [11] is challenged by the diversity of user preferences and tastes. The limited amount of user ratings available for training compared to the enormous number of predictions needed is another factor impeding the system's success. The approach outlined in [12] is heavily reliant on the user's book reading volume. Recommendations from a smaller training set might be less accurate because it does not fully represent the range of a user's preferences. Finally, the system in [13] has issues that are comparable to those with conventional recommendation systems, particularly with issues like synonymy and polysemy.

2.4 Proposed Solutions

This project proposes a hybrid method that combines collaborative and content-based filtering algorithms to develop a robust and tailored book recommendation system. To solve the cold start issue, the content-based filtering component will examine book properties like author, genre, and keywords to generate comprehensive user profiles based on reading history. Collaborative filtering will use similar users' preferences to make tailored recommendations to those who have enough interaction data. The system makes sure that customers always receive a variety of relevant recommendations, even in situations where there is insufficient data, by striking a balance between these two methods.

To combine the suggestions from the two methods, the hybrid system will use a mixed or weighted approach. It employs a weighted average to combine the ratings of content-based and collaborative filtering modules, and it modifies the weights according to the data that is available. For instance, it can prioritize collaborative filtering for users with rich interaction data and give new users with little interaction history a higher content-based filtering weight. This flexibility guarantees the system's continued accuracy and efficacy across a range of user scenarios. Furthermore, by avoiding over-specialization, the hybrid approach maintains the proposals' appeal and novelty. Because of this, it is the best option for ensuring that recommendations for books are made based on user preferences on platforms like libraries and reading communities.

CHAPTER 3

Proposed Method/Approach

3.1 Methodologies and General Work Procedures

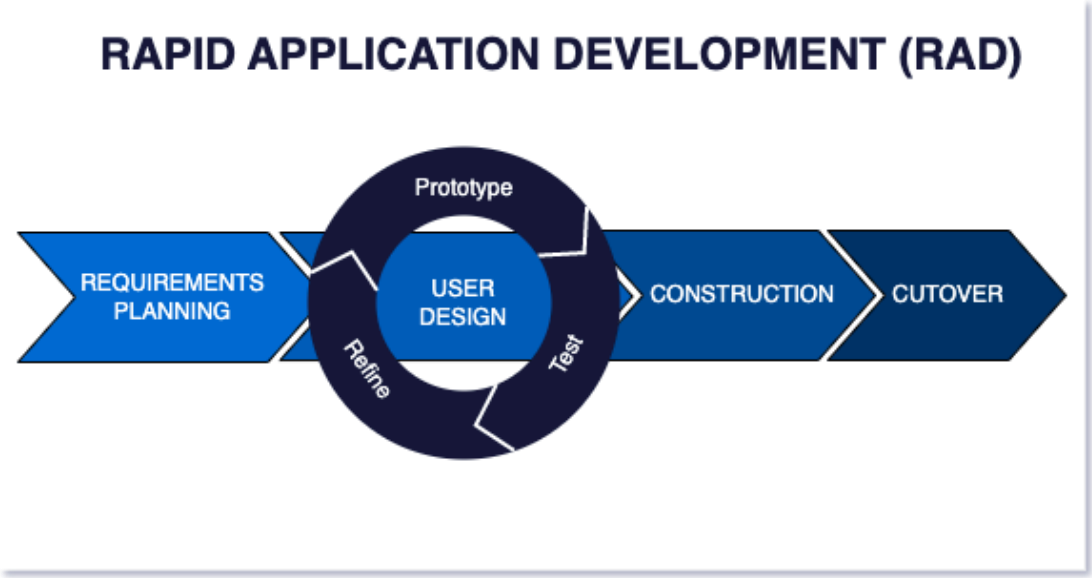


Figure 3.1 Rapid Application Development (RAD) Methodology

The Rapid Application Development (RAD) approach will be implemented in this project. RAD is a user-centered, flexible software development approach that starts with prototypes and then evolves according to user feedback. It is an agile strategy that emphasizes providing practical software in a short period of time through a prototype, refine, and test cycle, with a focus on incorporating user feedback throughout the process. RAD ensures that the development process is user-centered, efficient, and responsive, which is ideal for the dynamic nature of personalized recommendation systems.

The project's development began with creating a simple version of the system, which included features such as user login, book browsing, rating, and simple recommendations. After developing the initial system prototype, it was tested with sample users to gather feedback on functionality and user experience. More features,

such as hybrid recommendation integration, user interface optimization, and tailored recommendations, will be introduced and refined as the system develops.

3.2 System Methodology/Approach

3.2.1 System Architecture Diagram

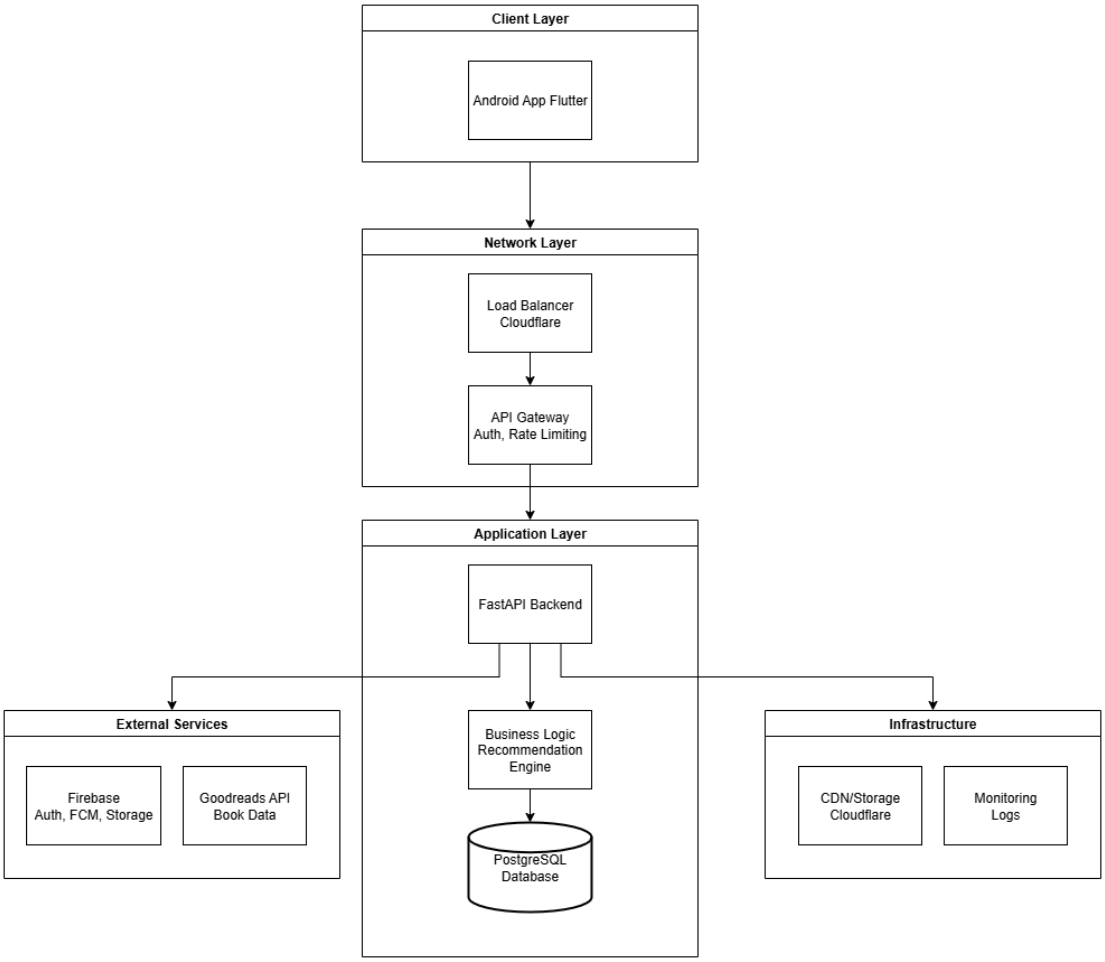


Figure 3.2.1 System Architecture Diagram of Book Recommender System

The system architecture diagram gives a detailed insight of the structure of the book recommendation system, separated into five separate layers and displaying the data flow in each layer. The diagram uses a layered architecture pattern for maintainability, scalability, and security, while also representing how different components interact in the ecosystem.

The client layer represents the user-facing applications that provide the initial interface for users interacting with the book recommender system. The client layer consists mainly of Android applications developed in Flutter. The Flutter framework allows for cross-platform consistency and reusability of code, opening a path for developing iOS clients in the future as well as web clients available through browsers. As these clients would work closely with the book recommender system API, there is potential for further developments in the future. This layer focuses on providing a clean, compact user interface that allows for user interaction and is responsive so the user can easily and seamlessly navigate the environment.

The network layer is like a skilled security guard at the entrance of a building who also manages the flow of traffic in the entrance. This layer has two jobs: to allow only authorized users on the network (security), and to distribute traffic across servers (load balancing) so that users do not overload one server. An example of this could be at the end of the semester around exam week when thousands of students all try to access the library website. In this case, the network layer is distributing traffic across multiple servers because the library website will crash if everyone tries to get on it at the same time. This layer essentially is verifying if everyone is who they say they are and isn't trying to hack the system or overload the system by making too many requests.

The application layer is the component where most of the intelligence occurs, as it is the brains of the entire system. This project's model uses this layer as a storage component for the FastAPI backend, the recommendation engine for the application, and the PostgreSQL database. When a user selects a rating of a book or searches for a book, this layer handles the request, pulls back the needed information, and returns the information to the user. The recommendation engine is in the application layer of the model that learns from the user's previously selected favourite books and then returns the new recommendation book selections from those patterns. The PostgreSQL database in the application layer of the model holds all book information, user-generated ratings, book reviews, and the associated queries to handle the triggering of various data capabilities.

The external services layer represents how the system utilizes services that add operational functionality in the external system. This system incorporates Firebase for user login and push notifications and an API from Goodreads to grab more extensive information. This means that the system does not necessarily have to build everything from the ground up, it can use services that other companies provide. Firebase manages all the complications of authentication, and Goodreads provides detailed book information including title, author, category, and community ratings.

The infrastructure layer is most like the plumbing or electrical systems in a building, which the user cannot see, but functionally, everything depends on it working properly. Infrastructure layer includes a CDN which helps images and files load quickly based on a user's location, and monitoring systems to track the performance of the system. Infrastructure can make images load almost instantly because it may be stored on a server physically closer to the user.

3.2.2 Use Case Diagram

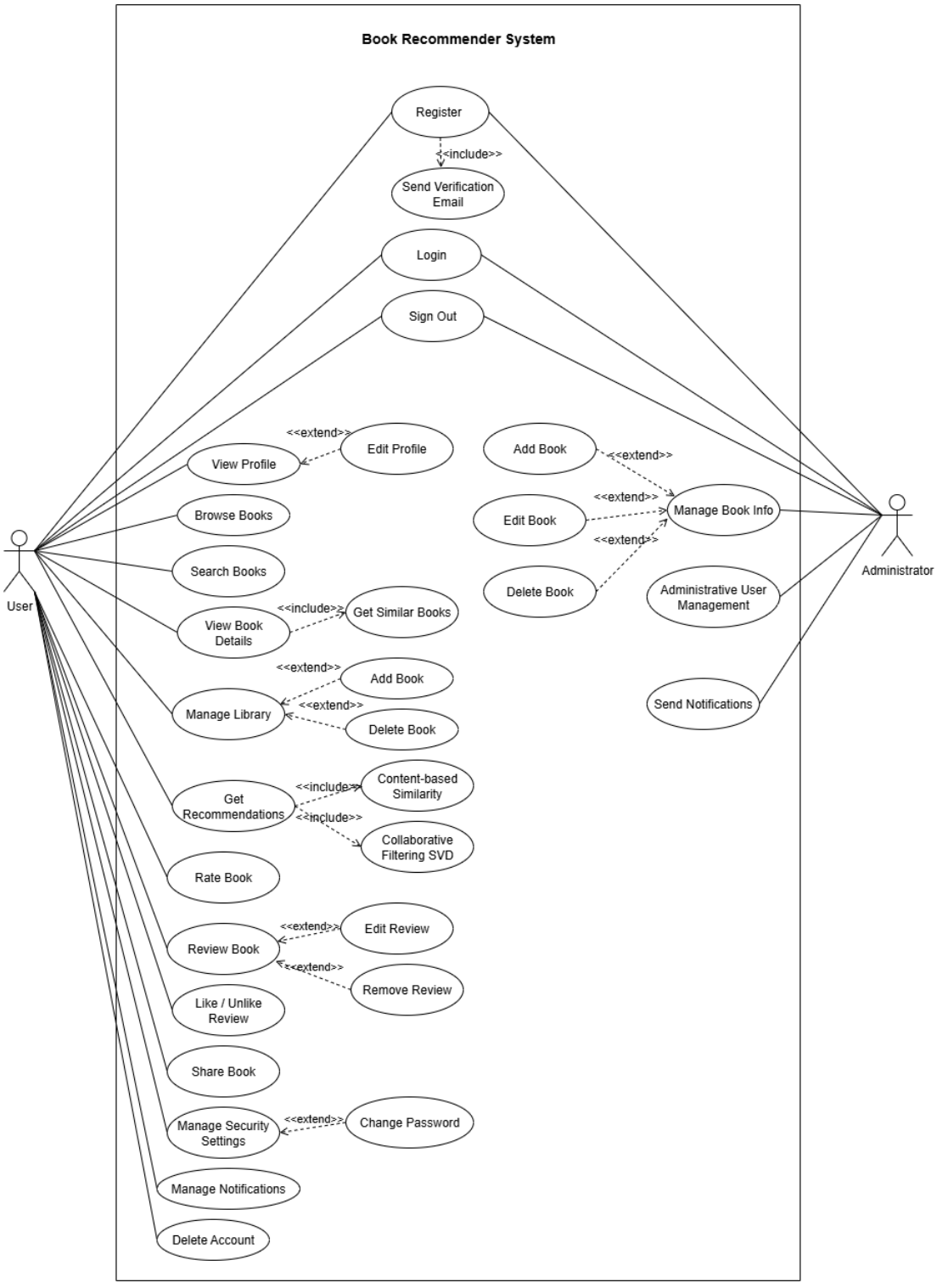


Figure 3.2.2 Use Case Diagram of Book Recommender System

Figure 3.2.2 shows the use case of the system. The use case includes of two actor which are the user and administrator.

3.2.2.1 Register

This use case allows user and administrator to register new account.

3.2.2.2 Send Verification Email

This use case sends user and administrator email with a verification link to confirm their email address.

3.2.2.3 Login

This use case allows user and administrator to log into their existing account.

3.2.2.4 Sign Out

This use case allows user and administrator to sign out their account.

3.2.2.5 View Profile

This use case display user profile, preferences, and basic statistics.

3.2.2.6 Edit Profile

This use case allows user to update avatar, nickname, bio, and anonymity.

3.2.2.7 Browse Books

This use case allows user to scroll through the book lists and explore available titles.

3.2.2.8 Search Books

This use case allows user to find books based on keywords, titles, authors, or genres.

3.2.2.9 View Book Details

This use case enables the user to access detailed information about a certain book, such as its description and rating.

3.2.2.10 Get Similar Books

This use case gives system a target book, computes cosine similarity over content features and return the most similar book titles.

3.2.2.11 Manage Library

This use case enables the user to see and manage saved books.

3.2.2.12 Add Book

This use case allows user to add a book into one of personal library categories.

3.2.2.13 Delete Book

This use case allows user to delete a book from personal library categories.

3.2.2.14 Get Recommendations

This use case allows user to get personalized book suggestions based on past ratings, preferences, and behaviours.

3.2.2.15 Content-based Similarity

This use case computes similarity via CountVectorizer to calculate the similarity of title, normalized authors, tags and rank by cosine similarity.

3.2.2.16 Collaborative Filtering SVD

This use case predicts ratings for unseen books using matrix factorization (SVD) trained on user–book ratings, then recommends higher estimated ratings first.

3.2.2.17 Rate Book

This use case allows user to submit or update a rating and contributes to the collaborative model.

3.2.2.18 Review Book

This use case allows user to leave a textual review for the book.

3.2.2.19 Edit Review

This use case allows user to manage textual review for the book.

3.2.2.20 Remove Review

This use case allows user to delete review and clear dependent likes safely.

3.2.2.21 Like / Unlike Review

This use case allows user to toggle like on reviews and may trigger a notification to the review author.

3.2.2.22 Share Book

This use case creates content suitable for sharing a book externally.

3.2.2.23 Manage Security Settings

This use case allows user to update name, email, password, and other profile information.

3.2.2.24 Change Password

This use case allows user to update the current password by sending an email with password reset link.

3.2.2.25 Manage Notifications

This use case allows user to register or unregister device tokens and manage push preferences.

3.2.2.26 Delete Account

This use case allows user to remove account permanently and all related data (likes, reviews, library, ratings, preferences, notifications), respecting foreign-key order.

3.2.2.27 Manage Book Info

This use case enables administrators to manage book information in the database.

3.2.2.28 Add Book

This use case allows administrator to add book in the database.

3.2.2.29 Edit Book

This use case allows administrator to edit book information in the database.

3.2.2.30 Delete Book

This use case allows administrator to delete book in the database.

3.2.2.31 Administrative User Management

This use case allows administrator to audit users, disable or re-enable accounts, and enforce security actions when needed.

3.2.2.32 Send Notifications

This use case triggers server-side notification delivery to targeted users or devices.

3.2.3 Activity Diagram

3.2.3.1 User Activity Diagram

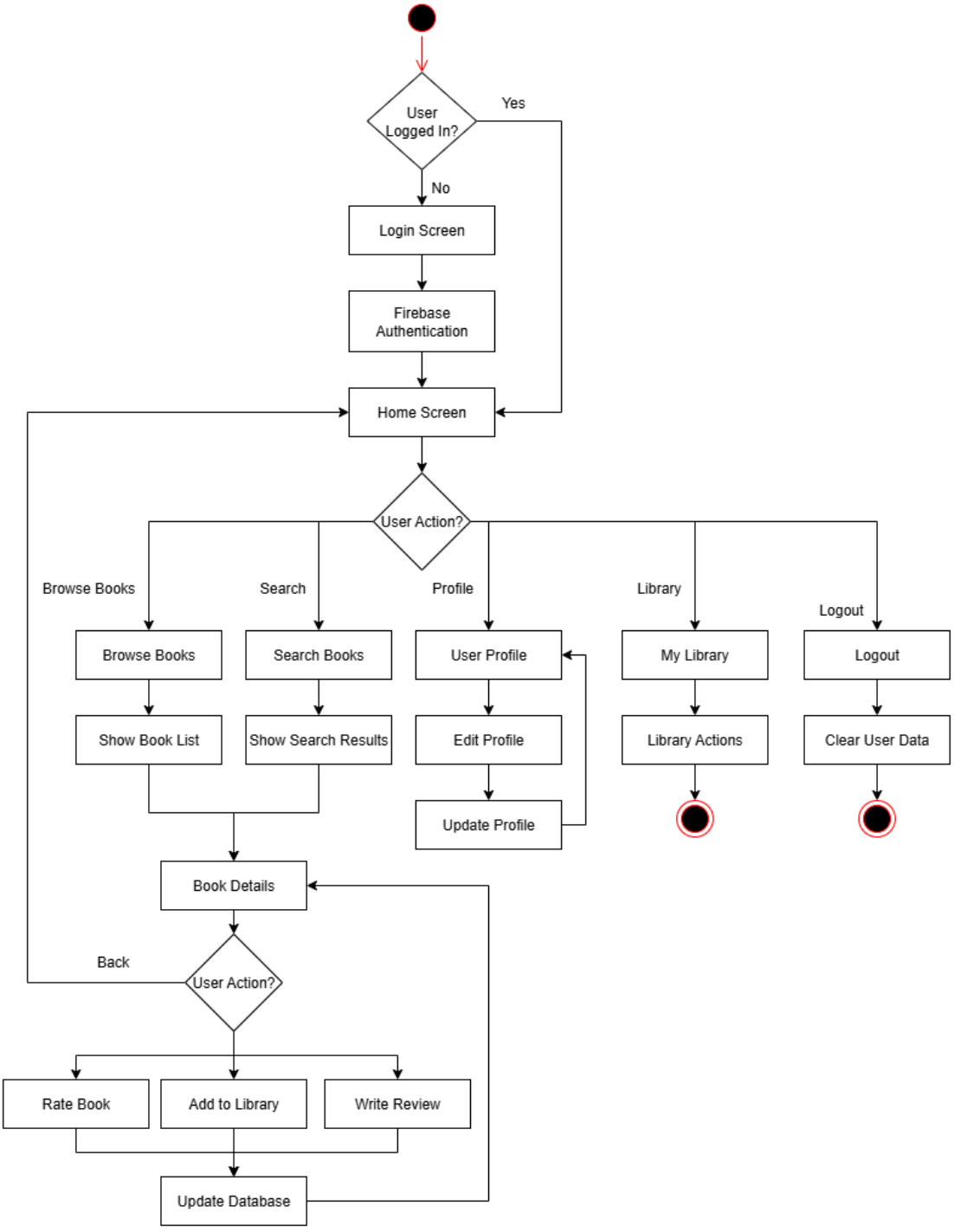


Figure 3.2.3.1 User Activity Diagram

The user activity diagram provides a comprehensive representation of the primary user flow within book recommender system. At the start, the processes will be initiated by user authentication. If the user is not already authenticated, they will be presented with

the login screen to authenticate using Firebase. Once the user has successfully authenticated, the user will then be directed to the home screen, where all functionality of the app will be accessed. The home screen will offer the user a choice of five different activities: to browse books, to search for a specific book title, to view their own library, to access their profile or to log out of the application. When the user browses books and searches for books, they will be presented with a list of books and search results along with the book detail screens. If the user wants to modify the book via rating, review, or adding to a library, the user will perform commands to do so and the updated data will sync to the backend to maintain consistency across the app. Lastly, at any time in the user flow, the user can choose to conclude their session by logging out or closing the application.

3.2.3.2 Authentication Activity Diagram

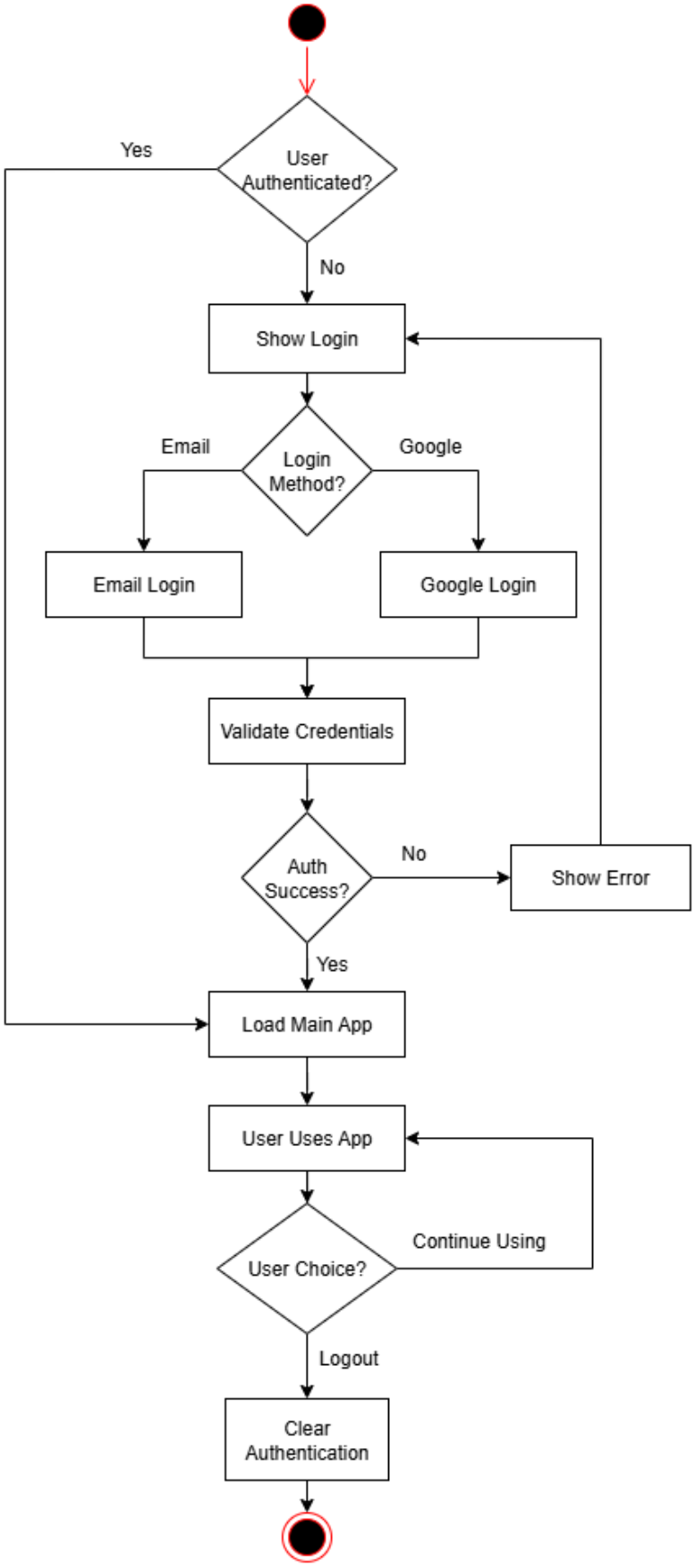


Figure 3.2.3.2 Authentication Activity Diagram

The authentication activity diagram outlines the simplified authentication process. When the user first opens the app, the system will check if the user is authenticated. If the user is not authenticated, the system will present the user with two options to log in, either email with password or Google Sign-In. The authentication process authenticates the user with Firebase, and if they successfully authenticate, the main application is loaded. Inside the app, the user is presented with two options: continue utilizing the app that creates a feedback loop, or log out of the app which cleanly clears the authentication state and closes the app. This process allows for both secure access to the application and a secure way to end the session.

3.2.3.3 Book Rating Activity Diagram

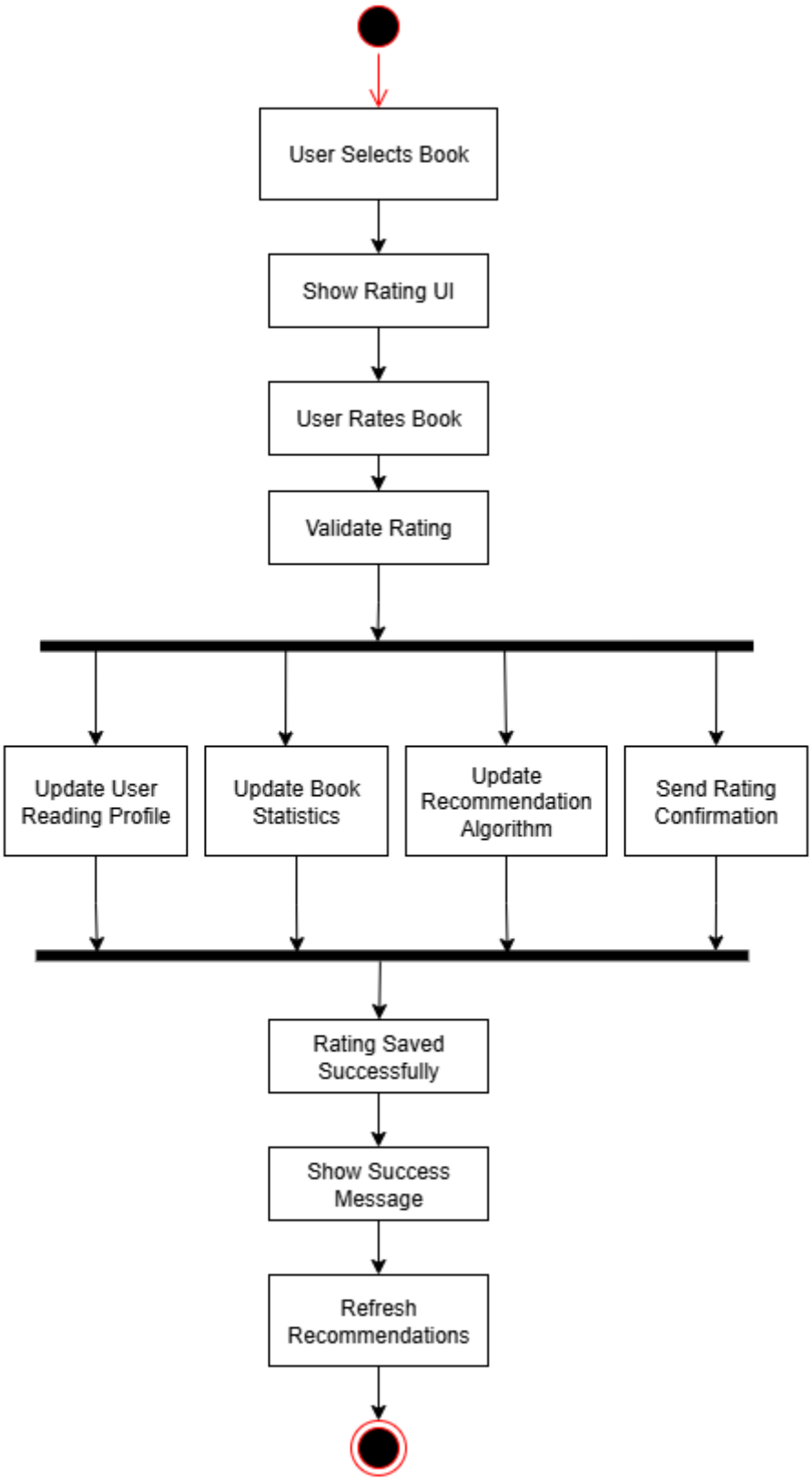


Figure 3.2.3.3 Book Rating Activity Diagram

The activity diagram for book rating shows the effective method that recommender system uses to process book ratings from users using concurrent processing patterns. When a user rates a book on a 1-5 scale, the system does not simply record the ratings as a single value. Instead, it sends instructions to invoke processes that happen in parallel. If they're not all happening simultaneously, the system cannot guarantee consistency and provide feedback instantaneously.

This process begins when a user selects a book to rate. The user submits the rating via the rating interface. The system then verifies that the rating meets the system requirement. At this point, the system employs a fork operation to initiate four independent parallel processes. Update user reading profile identifies the user's reading habits and preferences, thus updates the user's personalized recommendation profile regarding this new rating. Update book statistics reevaluate the book's average rating, total number of ratings, and popularity statistics that the system uses to help determine recommendations for users. Update recommendation algorithm takes all of this new user rating information to update the machine learning models the system utilizes for content-based and collaborative filtering. Finally, send rating confirmation immediately sends feedback to the user to confirm their rating has been submitted.

All four processes run simultaneously, ensuring all updates successfully finish on each backend process before proceeding to the next step. This process of joining is critical because incomplete updates may lead to inconsistent data states or failed user experiences. After all the processes that can run parallel have finished, the system confirms to the user that their rating has been saved and then refreshes the recommendation engine with the new data, allowing the user to receive updated personalized suggestions based on their most recent rating.

3.2.3.4 Book Recommendation Activity Diagram

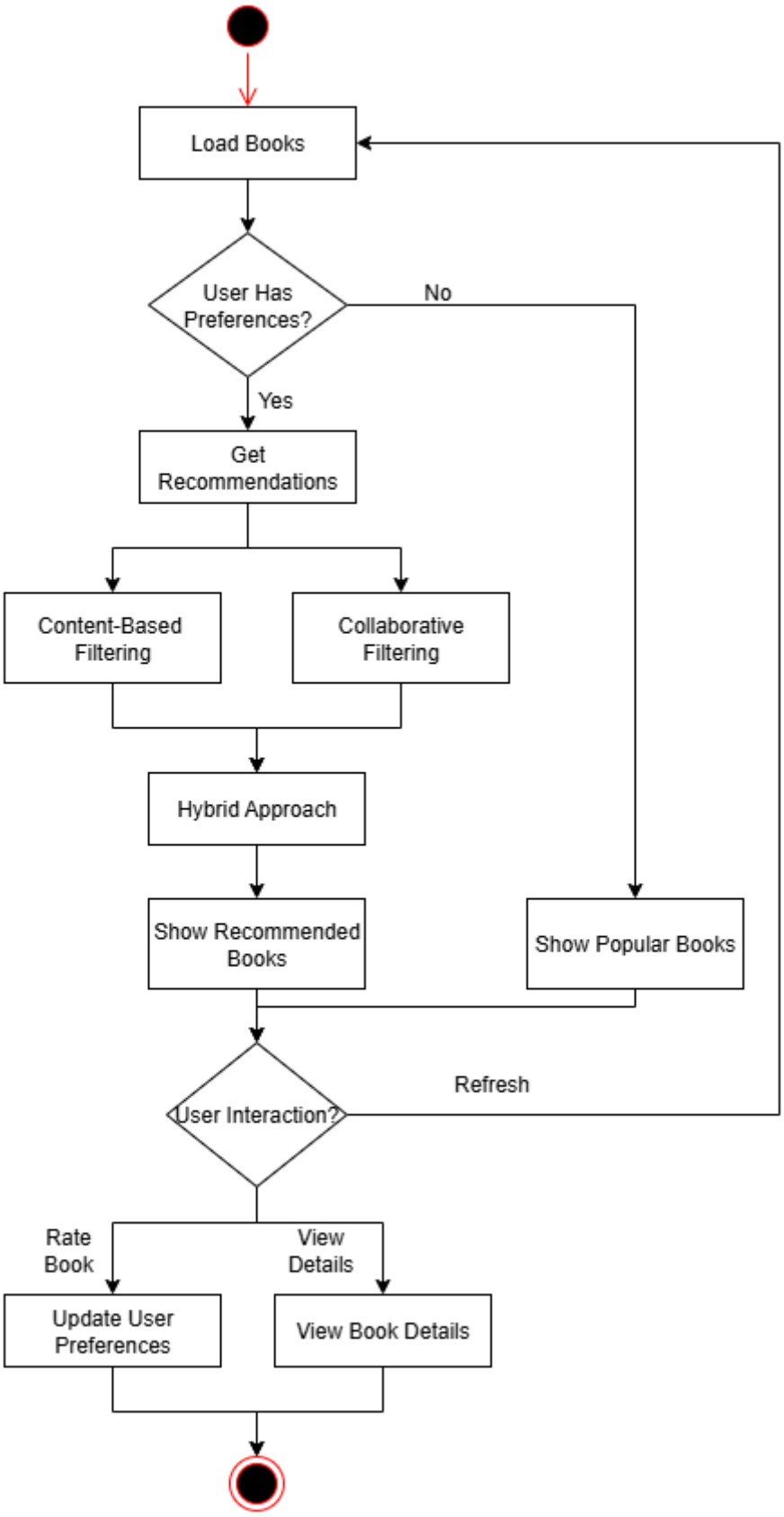


Figure 3.2.3.4 Book Recommendation Activity Diagram

The book recommendation activity diagram illustrates how recommender system provides personalized book suggestions to users. When users open the home screen, the system loads books based on user preferences. For new users without established preferences, the system displays popular books to provide immediate content. For existing users, the system uses a hybrid recommendation technique that combines two algorithms: content-based filtering and collaborative filtering approach. Content-Based Filtering examines book characteristics such as genres, tags, and authors to identify books that are comparable to those that the user has rated highly. Collaborative Filtering recognizes people with similar reading habits and recommends books that they have previously enjoyed. The Hybrid Approach uses both methods, including weighted scoring, to produce more accurate and diversified recommendations. When users rate books, view details, or interact with recommendations, the system updates their preferences, which influences future recommendations. Users can refresh recommendations or close the application at any time, with the close action properly terminating the activity flow. This creates a comprehensive learning system that improves recommendation accuracy over time.

3.2.4 Sequence Diagram

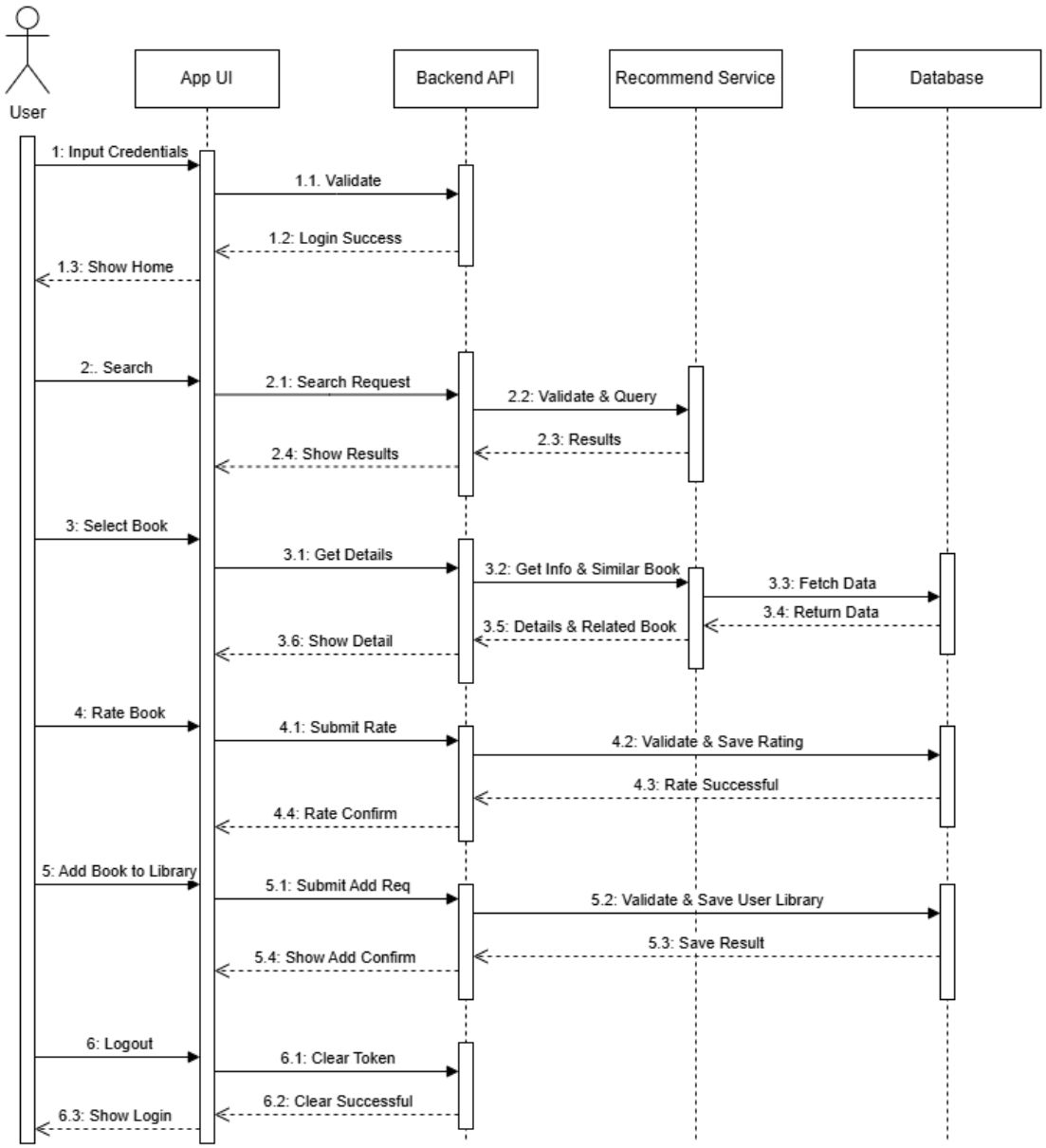


Figure 3.2.4 Sequence Diagram for User

This diagram represents the flow of interactions that occur when a user uses the book recommender application. First, there is the login flow. The user interacts with the app's user interface by providing credentials to initiate the login process. The app UI forwards this login request to the backend API, which is in charge of authenticating the user. After successful authentication, the app's UI receives and stores an authentication token. It then displays a successful login message before redirecting the user to the main page.

When a user searches for books, the App UI forwards the request to the backend. The backend validates the token, processes the search request, and searches the database for appropriate books. The search results are returned from the database to the backend, which then sends them back to the App UI for display to user. Similarly, when a user selects a book to view its details, the App UI requests information from the backend. The backend retrieves the book's main information from the database and requests related books from the recommendation service. The recommendation service fetches data and returns book information or a list of relevant books to the backend. The backend then sends details and related books to the App UI for display.

User activities, such as rating a book or adding it to a library, follow a similar pattern. The App UI delivers the user's actions and related data, such as the rating value or book ID, to the backend API. The backend validates the submitted information and saves the user's rating or library list to the database. When the database is successfully saved, the backend sends a confirmation response to the app UI. The app UI updates the user interface and notifies that the interaction was successful.

Finally, when the user logs out, the app UI clears the locally stored authentication token and ends the authenticated session. The app UI returns to the first login screen.

CHAPTER 4

System Design

4.1 System Block Diagram

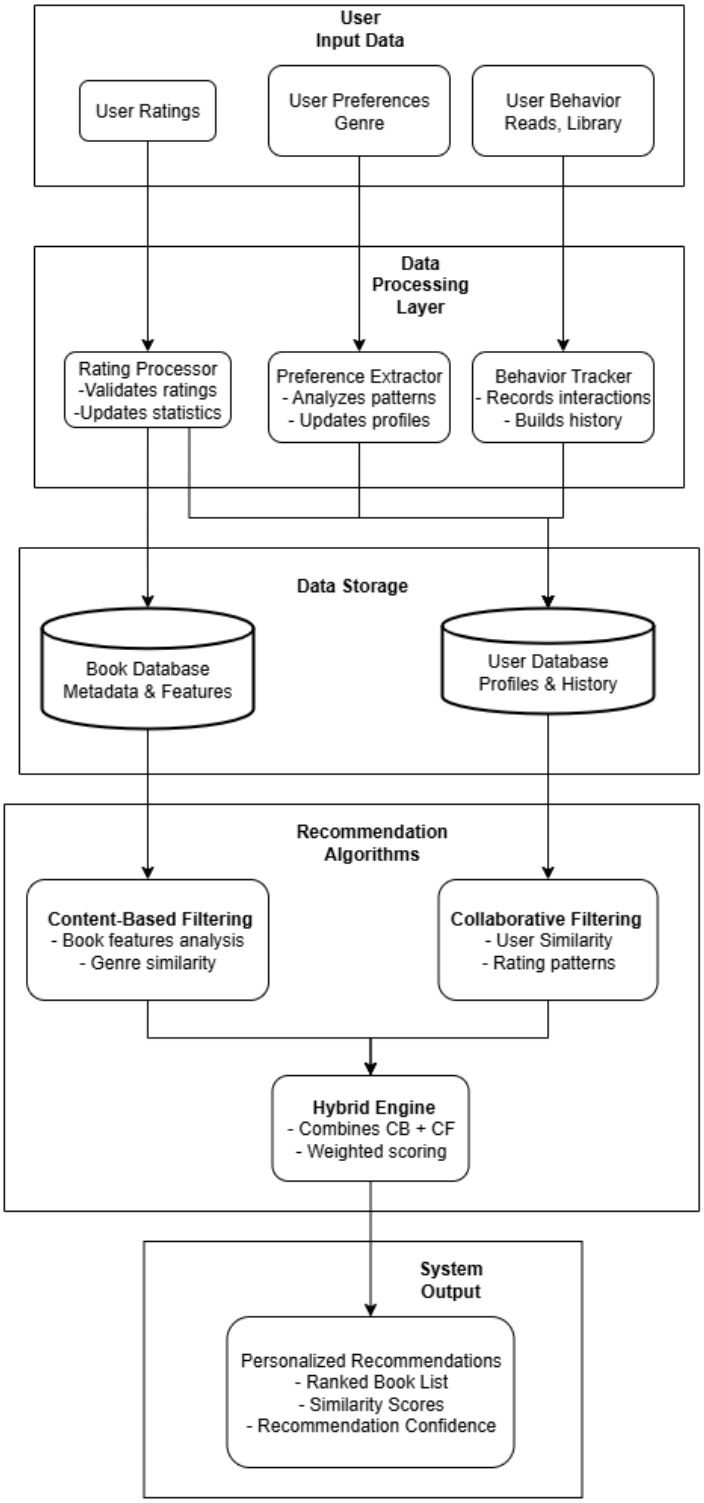


Figure 4.1.1 Block Diagram of Book Recommender System

The system block diagram shows the entire data process for the recommendation system, including how user actions are managed through to book recommendations. This diagram allows for clarity of the data processing pipeline and the role and functions of each of the recommendation's algorithms are outlined.

User Input Data:

The system begins by obtaining three categories of user input information: User Ratings reflects explicit feedback of their preferences, **User Preferences** captures explicit interests of the user, and **User Behaviour** like reads, library add reflects implicit interest patterns in the user's interaction with the app.

Data Processing:

The user input data subsequently flows into the connection to the Data Processing layer, where the three types of data follow through with a processor for each. The **Rating Processor** validates their rating to update the user book statistics. The **Preference Extractor** identifies developments in the user and potentially adjust their profile. The Behaviour Tracker records their interaction with the system and develop their overall interaction history.

Data Storage:

The combined data is then stored within specialized databases. The **User Database** secures the user's profiles, user rating history, and user preferences. The **Book Database** secures the standard metadata, features, and characteristics of the book. Each database serves as a type of process related to each specific algorithm.

Recommendation Algorithms:

The algorithm implementation consists of three active approaches, each with specialized access to such data.

- **Content-Based Filtering:** Utilizes the book database to assess book attributes such as genre, author, tags, descriptions and produces a list of books that have similarities to books that the user liked based on item-to-item similarity of content attributes.

- **Collaborative Filtering:** Utilizes the user database to find users that have similar preferences as this user and provide recommendations that similar users liked based on user-to-user comparison and rating patterns rather than book content.
- **Hybrid Engine:** Accepts results from both Content-Based Filtering and Collaborative Filtering and synthesizes the results using a weighted scoring methodology, also allowing the engine to exercise the benefits of both strategies.

System Output:

The hybrid engine generates personalized recommendations, which contains a ranked book list along with similarity score and weighted rating scores for each suggestion. The engine combines the weighted scores of content-based similarity and collaborative filtering predictions to produce a prioritized list of books for the user based on their reading preferences or history.

4.2 System Components Specifications

The system components specifications section includes technical details for each main component of the book recommender system. It offers a thorough understanding of frontend technology, backend technology, database, external services, and data sources, providing full insight into the technical architecture and implementation of the system.

The book recommendation system employs a modern client-server architecture that clearly distinguishes the presentation, business logic, and data layer. The frontend uses Flutter as a cross-platform mobile technology that emulates a native appearance and experience for Android users. The backend utilizes FastAPI to create high-performance APIs, automatic documentation, and type-safe services. The data layer uses PostgreSQL for managing relational data, while the SQLite database provides local caching of previously retrieved data for offline use. External service integration is provided through Firebase to create the system's authentication, data storage, and messaging capabilities. The Goodreads dataset will be the primary source for book metadata and community ratings.

4.2.1 Frontend Components

Flutter Mobile Applications

- Framework: Flutter 3.x with Dart programming language
- Package Name: book_recommend (Flutter project name)
- Platform Support: Android
- State Management: Provider pattern for reactive UI updates
- Key Features: Cross-platform compatibility, hot reload for development, native performance
- API Base URL: https://api.booknests.site

User Interface Components

- Screens: Login, Home, Book Detail, User Profile, Library, Settings, Onboarding, Personal Center, Explore
- Widgets: Custom book cards, rating components, search bars, and navigation elements
- Navigation: Bottom navigation bar following Material Design
- Themes: Light and dark theme support with Google Fonts
- Localization: Multi-language support

Local Data Management

- SQLite Database: Stores book data locally so users can browse without internet
- SharedPreferences: Saves user settings and login information on the phone
- Image Caching: Keeps book covers and profile pictures stored for faster loading

4.2.2 Backend Components

FastAPI Server

- Framework: FastAPI with Python

- API Design: RESTful endpoints with automatic documentation
- Authentication: Firebase JWT token validation
- Performance: Asynchronous request handling for better performance

Database Layer

- Primary Database: PostgreSQL for storing user data, books, and ratings
- ORM: SQLAlchemy to handle database operations easily
- Migrations: Alembic for updating database structure when needed
- Connection Pooling: Manages database connections efficiently

Recommendation Engine

- Algorithm: Hybrid approach combining content-based and collaborative filtering
- Data Processing: User preference analysis and similarity calculations
- Performance: Cached recommendations with real-time updates
- Accuracy: Machine learning models based on user rating patterns

4.2.3 External Service Integrations

Firebase Services

- Authentication: User login with email/password and Google Sign-In
- Cloud Messaging: Push notifications for book recommendations and updates
- Firestore: Real-time user profile synchronization

Goodreads Dataset

- Data Source: Historical Goodreads dataset for book metadata
- Integration: Local CSV data processing for book information
- Caching: SQLite storage of book data for offline access

- Data Processing: Converting CSV data to database format

4.3 Database Design and Data Flow

4.3.1 Database Schema Design

The book recommender system uses a relational database design with the following core tables:

Users Table:

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    hashed_password VARCHAR(255),
    full_name VARCHAR(255),
    avatar_url VARCHAR(500),
    bio TEXT,
    is_anonymous BOOLEAN DEFAULT FALSE,
    fcm_token VARCHAR(500),
    is_active BOOLEAN DEFAULT TRUE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Books Table:

```
CREATE TABLE books (
    id SERIAL PRIMARY KEY,
    title VARCHAR(500) NOT NULL,
```

```
    author VARCHAR(255),
    isbn VARCHAR(20) UNIQUE,
    description TEXT,
    cover_url VARCHAR(500),
    published_date TIMESTAMP,
    average_rating DECIMAL(3,2) DEFAULT 0.0,
    ratings_count INTEGER DEFAULT 0,
    language_code VARCHAR(10)
);
```

Ratings Table:

```
CREATE TABLE ratings (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    book_id INTEGER REFERENCES books(id),
    rating DECIMAL(3,2) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Reviews Table:

```
CREATE TABLE reviews (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    book_id INTEGER REFERENCES books(id),
```

```
    rating DECIMAL(3,2) NOT NULL,
    content TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Library Table:

```
CREATE TABLE library (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    book_id INTEGER REFERENCES books(id),
    status VARCHAR(50) DEFAULT 'want_to_read', -- reading/finished/want_to_read
    added_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    UNIQUE(user_id, book_id)
);
```

User Preferences Table:

```
CREATE TABLE user_preferences (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id) UNIQUE,
    favorite_genres VARCHAR(255)
);
```

Tags and Book-Tags Association Tables:

```
CREATE TABLE tags (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) UNIQUE NOT NULL
);

CREATE TABLE book_tags (
    book_id INTEGER REFERENCES books(id),
    tag_id INTEGER REFERENCES tags(id),
    PRIMARY KEY (book_id, tag_id)
);
```

Likes Table:

```
CREATE TABLE likes (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    review_id INTEGER REFERENCES reviews(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Notifications Table:

```
CREATE TABLE notifications (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
```

```
    title VARCHAR(255),
    body TEXT,
    type VARCHAR(50),
    is_read BOOLEAN DEFAULT FALSE,
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    data JSON
);
```

User Achievements Table:

```
CREATE TABLE user_achievements (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    achievement_id VARCHAR(100),
    unlocked_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

4.3.2 Entity Relationship Diagram

The following Entity Relationship Diagram illustrates the relationships between all database tables in the book recommender system:

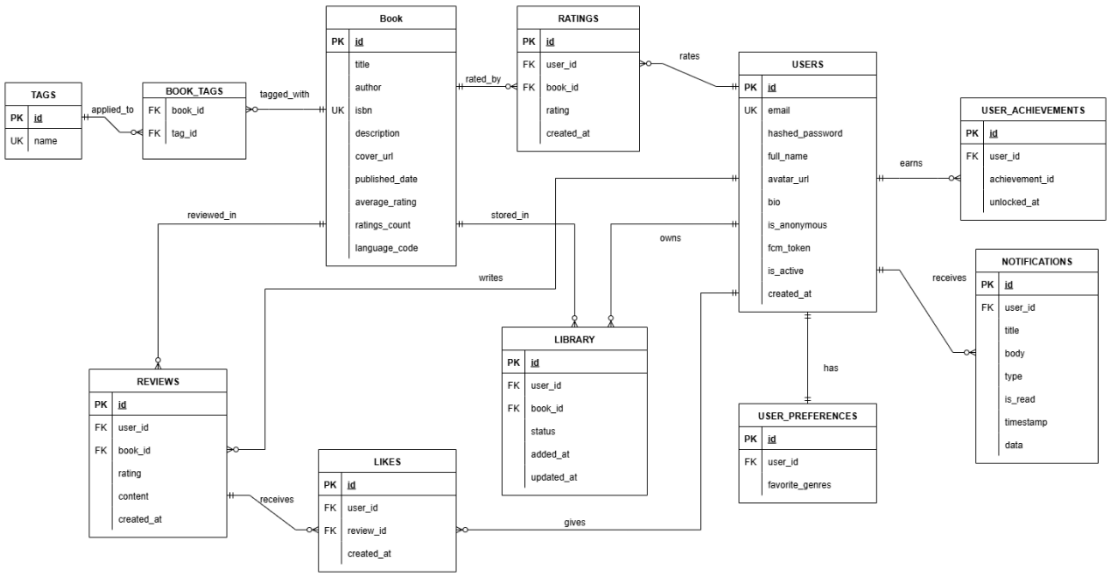


Figure 4.3.2 Entity Relationship Diagram

This diagram shows how all the data in book recommender system is connected. Each user has their own preferences, can rate many books, write many reviews, add many books to their library, like many reviews, receive many notifications, and earn many achievements. Books can have multiple tags such as "fantasy", "romance" and "mystery" through a connection table, be rated by many users, have many reviews and be in many users' libraries. This setup makes it easy for the recommender system to find books that match what users like based on their preferences, ratings, and reading history.

4.3.3 Data Flow Architecture

Here is what happens when users interact with the book recommender application: the application sends requests to the back-end server. Before the server processes them, it checks whether the user is authenticated user through Firebase authentication. Once the user is validated, the server queries the database for any user preferences, and books information as appropriate, and executes the recommendation algorithms to provide user recommendations. The recommendation result is returned to the application in a standard combination so the application can then update the screen with the recommendations. This is done quickly for the user to receive feedback once they interact with the application.

The data flow for this process is facilitated through multiple components. The front-end uses the Dio HTTP client to send requests to the backend FastAPI, which validates the user's Firebase token before processing the request. The backend uses SQLAlchemy ORM to query user data, book information, and ratings from the PostgreSQL database. The recommendation algorithm processes the first data and returns suggestions.

```dart
import 'package:dio/dio.dart';
import 'package:firebase_auth/firebase_auth.dart';

class ApiService {
  final Dio _dio;

  ApiService(this._dio);

  // Get Firebase authentication token
  Future<String?> getToken() async {
    try {
      final user = FirebaseAuth.instance.currentUser;
      if (user != null) {
        return await user.getIdToken();
      }
      return null;
    } catch (e) {
      print('Error getting Firebase token: $e');
      return null;
    }
  }

  // Get current user ID
  Future<String?> getCurrentUserId() async {
    try {
      final user = FirebaseAuth.instance.currentUser;
      return user?.uid;
    } catch (e) {
      print('Error getting current user ID: $e');
      return null;
    }
  }

  // Get current user email
  Future<String?> getCurrentUserEmail() async {
    try {
      final user = FirebaseAuth.instance.currentUser;
      return user?.email;
    } catch (e) {
      print('Error getting current user email: $e');
      return null;
    }
  }
}
```

Figure 4.3.3.1 API Request Flow

```python
async def get_firebase_user_optional(authorization: Optional[str] = Header(None)):
    """
    Optional Firebase authentication - returns None if no valid token provided
    """
    if not authorization or not authorization.startswith("Bearer "):
        return None

    id_token = authorization.split(" ")[1]
    if not id_token:
        return None

    user_info = await get_firebase_user(id_token)
    return user_info

@router.post("/firebase-login", response_model=FirebaseLoginResponse)
async def firebase_login(authorization: Optional[str] = Header(None)):
    """
    Verify Firebase ID Token from frontend and return user info.
    Frontend should send the ID token in Authorization header as: Bearer <ID_TOKEN>
    """
    if not authorization or not authorization.startswith("Bearer "):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Authorization header missing or invalid. Use format: Bearer <ID_TOKEN>"
        )

    id_token = authorization.split(" ")[1]
    if not id_token:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="ID token is missing"
        )

    user_info = await get_firebase_user(id_token)
    if not user_info:
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Invalid Firebase ID token"
        )

    return FirebaseLoginResponse(
        firebase_user=user_info,
        message="Firebase authentication successful"
    )

@router.get("/verify-token")
async def verify_token(authorization: Optional[str] = Header(None)):
    """
    Simple endpoint to verify if a Firebase token is valid
    """
    if not authorization or not authorization.startswith("Bearer "):
        raise HTTPException(
            status_code=status.HTTP_401_UNAUTHORIZED,
            detail="Authorization header missing or invalid"
        )

    id_token = authorization.split(" ")[1]
    user_info = await FirebaseAuthService.verify_firebase_token(id_token)

    if user_info:
        return {"valid": True, "user": user_info}
    else:
        return {"valid": False, "user": None}
```

Figure 4.3.3.2 Backend Token Validation

```python
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    email = Column(String, unique=True, index=True)
    hashed_password = Column(String)
    full_name = Column(String, nullable=True)
    avatar_url = Column(String, nullable=True)  # New avatar URL field
    bio = Column(String, nullable=True)  # New bio field
    is_anonymous = Column(Boolean, default=False)  # New anonymous mode field
    fcm_token = Column(String, nullable=True)  # New FCM token field
    is_active = Column(Boolean, default=True)
    created_at = Column(DateTime, default=datetime.utcnow)

    ratings = relationship("Rating", back_populates="user")
    preferences = relationship("UserPreference", back_populates="user", uselist=False)
    reviews = relationship("Review", back_populates="user")
    library = relationship("Library", back_populates="user")
    likes = relationship("Like", back_populates="user")
    achievements = relationship("UserAchievement", back_populates="user")
    notifications = relationship("Notification", back_populates="owner")
```

Figure 4.3.3.3 Database Query Example

4.3.4 Recommendation Algorithm Flow

To recommend good books to users, the recommendation engine takes a hybrid method, combining content-based filtering with collaborative filtering. Initially, it checks whether the user has provided a rating for any books in the past. If they have, the system examines the books the user liked and creates a list of similar books. Next, It compares the user's preferences to those of other users and generates a list of book recommendations based on similar tastes. Finally, the two approaches are combined with weighted scoring, to create a recommended book list for the user, based on both approaches. In the case that the user has not rated any books previously, the recommendation system will simply display the most popular books in the database.

The primary recommendation logic employed by the recommendation system is initiated in the improved_hybrid function as shown in Figure 4.3.5. This function utilizes cosine similarity for the content-based filtering and SVD for the collaborative filtering.

```python
def improved_hybrid(user_id, title, books_df, cosine_sim, svd, indices, n=10):
    """Hybrid recommendation algorithm combining content-based and collaborative filtering"""
    # Find book index by title
    if title not in indices:
        raise HTTPException(status_code=404, detail="Book not found")

    idx = indices[title]
    if isinstance(idx, (list, np.ndarray)):  # Handle multiple indices case
        idx = idx[0]

    # Get similar books based on content
    sim_scores = list(enumerate(cosine_sim[idx].tolist()))
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
    sim_scores = sim_scores[1:51]  # Exclude itself, take top 50
    book_indices = [i[0] for i in sim_scores]

    # Get candidate books and calculate weighted rating
    df = books_df.iloc[book_indices][['book_id', 'title', 'ratings_count', 'average_rating']]
    v = df['ratings_count']
    m = df['ratings_count'].quantile(0.60)
    R = df['average_rating']
    C = df['average_rating'].mean()
    df['weighted_rating'] = (R*v + C*m) / (v + m)

    # Add collaborative filtering estimated ratings (skip for anonymous users)
    if user_id:
        if svd is not None and HAS_SURPRISE:
            df['est'] = df['book_id'].apply(lambda x: svd.predict(user_id, x).est)
        else:
            df['est'] = 0.0
        # Combine scores for hybrid recommendation
        df['score'] = (df['est'] + df['weighted_rating']) / 2
    else:
        # For anonymous users, use only content-based scoring
        df['score'] = df['weighted_rating']

    # Sort and return top N recommendations
    df = df.sort_values('score', ascending=False)
    return df[['book_id', 'title', 'score']].head(n)
```

Figure 4.3.4 improved_hybrid Function

4.4 System Implementation Guide

4.4.1 Development Environment Setup

Prerequisites and Tools:

To rebuild the system, developers need Python 3.8+ with pip package manager, Flutter SDK 3.0+, PostgreSQL database server, Firebase project setup, and a code editor like VS Code. The development environment requires separate directories for backend (book_recommend_backend) and frontend (book_recommend_frontend) components to maintain clean separation of concerns.

Backend Environment:

Figure 4.4.1.1 shows how to create a Python virtual environment and install dependencies using command pip install -r requirements.txt. The backend requires FastAPI framework, SQLAlchemy ORM, PostgreSQL adapter, Firebase SDK, and machine learning libraries including scikit-learn and pandas. Environment variables must be configured for database connections, Firebase credentials, and API endpoints as shown in Figure 4.4.1.2.

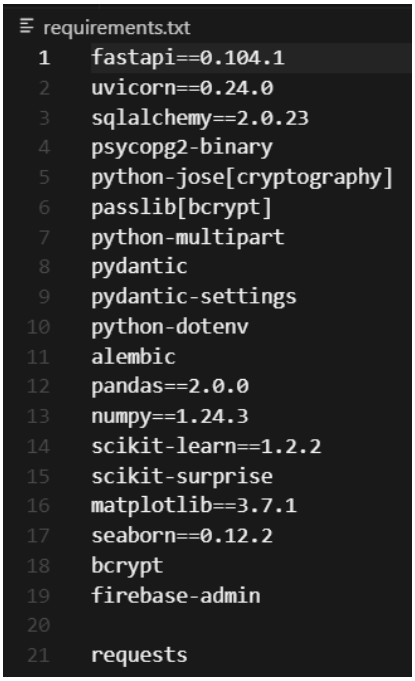


Figure 4.4.1.1 requirements.txt File

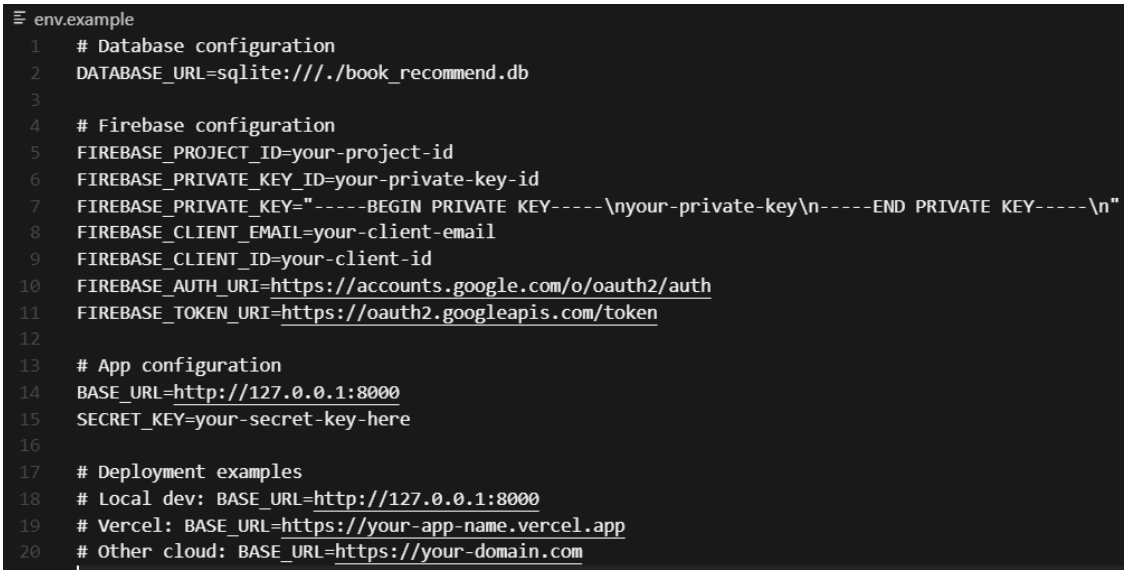


Figure 4.4.1.2 Environment Configuration Example

Frontend Environment:

Install Flutter SDK and set up Android development environment. Run flutter pub get to install project dependencies as shown in Figure 4.4.1.3 and configure Firebase integration by placing google-services.json in the android/app directory. The frontend requires Provider for state management, Dio for making HTTP requests, and Firebase Auth for user authentication.

```yaml
dependencies:
  flutter:
    sdk: flutter

  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.6
  sqflite: ^2.3.0
  path: ^1.8.3
  logger: ^2.0.2+1
  google_fonts: ^6.1.0
  provider: ^6.1.1
  shared_preferences: ^2.2.2
  cached_network_image: ^3.3.0
  flutter_svg: ^2.0.9
  carousel_slider: ^4.2.1
  intl: ^0.19.0
  flutter_localizations:
    sdk: flutter
  # Notification functionality
  firebase_messaging: ^14.7.10
  flutter_local_notifications: ^19.4.1

  # Firebase
  firebase_core: ^2.24.2
  firebase_auth: ^4.15.3
  google_sign_in: ^6.2.1
  cloud_firestore: ^4.13.6
  firebase_storage: ^11.5.6
  # HTTP client
  dio: ^5.4.0
  android_intent_plus: ^5.1.0
  # Local storage
  hive: ^2.2.3
  hive_flutter: ^1.1.0
  flutter_launcher_icons: ^0.14.3
  ml_linalg: ^13.12.6
  csv: ^6.0.0
  # Share functionality
  share_plus: ^7.2.1
  image_picker: ^1.2.0
  url_launcher: ^6.3.0
  path_provider: ^2.1.4

dev_dependencies:
  flutter_test:
    sdk: flutter
```

Figure 4.4.1.3 pubspec.yaml Dependencies

4.4.2 Database Setup and Data Import

Database Schema Implementation:

Set up PostgreSQL database named booknest_db and execute Alembic migration scripts as shown in Figure 4.4.2.1 to create all required tables including users, books, reviews, ratings, and user preferences. As shown in Figure 4.4.2.2, the database schema supports the complete recommendation system with proper relationships between entities.

alembic > versions > abbcc1924d71_init_tables.py

```python
from typing import Sequence, Union

from alembic import op
import sqlalchemy as sa


# revision identifiers, used by Alembic.
revision: str = 'abbcc1924d71'
down_revision: Union[str, None] = None
branch_labels: Union[str, Sequence[str], None] = None
depends_on: Union[str, Sequence[str], None] = None


def upgrade() -> None:
    """Upgrade schema."""
    # ### commands auto generated by Alembic - please adjust! ###
    op.create_table('books',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.Column('title', sa.String(), nullable=True),
    sa.Column('author', sa.String(), nullable=True),
    sa.Column('isbn', sa.String(), nullable=True),
    sa.Column('description', sa.String(), nullable=True),
    sa.Column('cover_url', sa.String(), nullable=True),
    sa.Column('published_date', sa.DateTime(), nullable=True),
    sa.Column('average_rating', sa.Float(), nullable=True),
    sa.Column('ratings_count', sa.Integer(), nullable=True),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_index(op.f('ix_books_author'), 'books', ['author'], unique=False)
    op.create_index(op.f('ix_books_id'), 'books', ['id'], unique=False)
    op.create_index(op.f('ix_books_isbn'), 'books', ['isbn'], unique=True)
    op.create_index(op.f('ix_books_title'), 'books', ['title'], unique=False)
    op.create_table('tags',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.Column('name', sa.String(), nullable=True),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_index(op.f('ix_tags_id'), 'tags', ['id'], unique=False)
    op.create_index(op.f('ix_tags_name'), 'tags', ['name'], unique=True)
    op.create_table('users',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.Column('email', sa.String(), nullable=True),
    sa.Column('hashed_password', sa.String(), nullable=True),
    sa.Column('full_name', sa.String(), nullable=True),
    sa.Column('is_active', sa.Boolean(), nullable=True),
    sa.Column('created_at', sa.DateTime(), nullable=True),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_index(op.f('ix_users_email'), 'users', ['email'], unique=True)
    op.create_index(op.f('ix_users_id'), 'users', ['id'], unique=False)
    op.create_table('book_tags',
    sa.Column('book_id', sa.Integer(), nullable=False),
    sa.Column('tag_id', sa.Integer(), nullable=False),
    sa.ForeignKeyConstraint(['book_id'], ['books.id'], ),
    sa.ForeignKeyConstraint(['tag_id'], ['tags.id'], ),
    sa.PrimaryKeyConstraint('book_id', 'tag_id')
    )
    op.create_table('ratings',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.Column('user_id', sa.Integer(), nullable=True),
    sa.Column('book_id', sa.Integer(), nullable=True),
    sa.Column('rating', sa.Float(), nullable=True),
    sa.Column('created_at', sa.DateTime(), nullable=True),
    sa.ForeignKeyConstraint(['book_id'], ['books.id'], ),
    sa.ForeignKeyConstraint(['user_id'], ['users.id'], ),
    sa.PrimaryKeyConstraint('id')
    )
    op.create_index(op.f('ix_ratings_id'), 'ratings', ['id'], unique=False)
    op.create_table('user_preferences',
    sa.Column('id', sa.Integer(), nullable=False),
    sa.Column('user_id', sa.Integer(), nullable=True),
    sa.Column('favorite_genres', sa.String(), nullable=True),
    sa.Column('reading_goal', sa.Integer(), nullable=True),
    sa.Column('notification_enabled', sa.Boolean(), nullable=True),
    sa.Column('theme_preference', sa.String(), nullable=True),
    sa.Column('language_preference', sa.String(), nullable=True),
    sa.ForeignKeyConstraint(['user_id'], ['users.id'], ),
    sa.PrimaryKeyConstraint('id'),
    sa.UniqueConstraint('user_id')
    )
```

Figure 4.4.2.1 Alembic Migration Script File

app > database > session.py

```python
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from ..config import get_settings

settings = get_settings()

engine = create_engine(
    settings.DATABASE_URL,
    echo=False,
    pool_size=5,  # Further reduce connection pool size
    max_overflow=10,  # Further reduce overflow connections
    pool_timeout=20,  # Reduce connection timeout
    pool_pre_ping=True,  # Enable connection pre-check
    pool_recycle=1800,  # Connection recycle time (30 minutes)
    pool_reset_on_return='commit',  # Reset when returning connection
)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)

# Create Base class for model inheritance
Base = declarative_base()

# Dependency
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

Figure 4.4.2.2 Database Connection

Data Import Process:

As shown in Figure 4.4.2.3 and Figure 4.4.2.4, import the Goodreads dataset following a specific sequence: tags first, then books with ISBN validation, book-tag associations, user creation from rating data, and finally rating import. The system uses chunked processing for large datasets and includes error handling for data validation and duplicate prevention.

```python
# 1. import tags
print("Importing tags...")
tags_df = pd.read_csv('data/tags.csv', chunksize=10000)
for chunk in tags_df:
    for _, row in chunk.iterrows():
        tag = Tag(id=int(row['tag_id']), name=row['tag_name'])
        session.merge(tag)
    session.commit()
print("Tags imported.")

# 2. import books
print("Importing books...")
books_df = pd.read_csv('data/books.csv', chunksize=10000)
for chunk in books_df:
    for _, row in chunk.iterrows():
        isbn = row['isbn']
        # Handle NaN and empty string for ISBN
        if pd.isna(isbn) or (isinstance(isbn, float) and math.isnan(isbn)) or str(isbn).strip() == "":
            isbn = None

        # Use original_publication_year as published_date (set to Jan 1st of that year)
        pub_year = row.get('original_publication_year', None)
        published_date = None
        if not pd.isna(pub_year):
            try:
                year = int(float(pub_year))
                if year > 0:
                    published_date = pd.to_datetime(f"{year}-01-01")
            except Exception:
                published_date = None

        # Skip if ISBN already exists (if you want to keep ISBN unique)
        if isbn is not None:
            exists = session.query(Book).filter(Book.isbn == isbn).first()
            if exists:
                continue

        book = Book(
            id=int(row['book_id']),
            title=row['title'],
            author=row['authors'],
            isbn=isbn,
            description=row.get('description', None),
            cover_url=row.get('image_url', None),
            published_date=published_date,
            average_rating=row['average_rating'],
            ratings_count=row['ratings_count']
        )
        session.merge(book)
    session.commit()
print("Books imported.")
```

Figure 4.4.2.3 Data Import Script Part 1

```python
# 3. import book_tags
print("Importing book_tags...")
book_tags_df = pd.read_csv('data/book_tags.csv', chunksize=10000)
conn = engine.connect()

for chunk in book_tags_df:
    trans = conn.begin()
    try:
        for _, row in chunk.iterrows():
            stmt = book_tags.insert().values(
                book_id=int(row['goodreads_book_id']),
                tag_id=int(row['tag_id'])
            )
            try:
                conn.execute(stmt)
            except Exception as e:
                if "duplicate key" not in str(e).lower():
                    print(f"Import Error: {e}")
        trans.commit()
    except Exception as e:
        trans.rollback()
        print(f"Transaction Error: {e}")

conn.close()
print("Book_tags imported.")

# 4. import users
print("Importing users...")
ratings_df = pd.read_csv('data/ratings.csv', usecols=['user_id'])
unique_user_ids = ratings_df['user_id'].unique()
for user_id in unique_user_ids:
    user = User(id=int(user_id), email=f"user{user_id}@example.com", hashed_password="dummy", full_name=None)
    session.merge(user)
session.commit()
print("Users imported.")

# 5. import ratings
print("Importing ratings...")
# First, get all existing book IDs
existing_book_ids = set()
for book in session.query(Book.id).all():
    existing_book_ids.add(book[0])

print(f"Found {len(existing_book_ids)} existing books")

ratings_df = pd.read_csv('data/ratings.csv', chunksize=10000)
skipped_count = 0
imported_count = 0

for chunk in ratings_df:
    for _, row in chunk.iterrows():
        book_id = int(row['book_id'])

        # Skip if book doesn't exist
        if book_id not in existing_book_ids:
            skipped_count += 1
            continue

        rating = Rating(
            user_id=int(row['user_id']),
            book_id=book_id,
            rating=float(row['rating']),
            created_at=None
        )
        session.add(rating)
        imported_count += 1
    session.commit()

print(f"Ratings imported: {imported_count}, skipped: {skipped_count}")

session.close()
print("All data import completed!")
```

Figure 4.4.2.4 Data Import Script Part 2

4.4.3 Core System Components

Backend API Implementation:

As shown in Figure 4.4.3.1, backend of FastAPI includes user, book, review, and rating class models, along with API endpoints for user operations, book operations, and recommendation generation. Figure 4.4.3.2 shows the structure of the user model. The backend handles Firebase authentication, database operations, and serves as the central processing unit for all recommendation requests.

```
app = FastAPI(
    title=settings.PROJECT_NAME,
    openapi_url=f"{settings.API_V1_STR}/openapi.json"
)

# Configure CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],  # In production, specify specific origins
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Mount static files directory
app.mount("/static", StaticFiles(directory="static"), name="static")

# Import routes
from .api import books, users, auth, ratings, recommendations, preferences, firebase_auth
from .api import share
from .api import notifications, fcm_notifications
from .api import reviews
from .api import library
from .api import likes
from .api import reading_stats

# Register routes
app.include_router(
    auth.router,
    prefix=f"{settings.API_V1_STR}/auth",
    tags=["authentication"]
)

app.include_router(
    users.router,
    prefix=f"{settings.API_V1_STR}/users",
    tags=["users"]
)

app.include_router(
    books.router,
    prefix=f"{settings.API_V1_STR}/books",
    tags=["books"]
)
```

Figure 4.4.3.1 FastAPI App Setup

```python
class User(Base):
    __tablename__ = "users"

    id = Column(Integer, primary_key=True, index=True)
    email = Column(String, unique=True, index=True)
    hashed_password = Column(String)
    full_name = Column(String, nullable=True)
    avatar_url = Column(String, nullable=True)  # New avatar URL field
    bio = Column(String, nullable=True)  # New bio field
    is_anonymous = Column(Boolean, default=False)  # New anonymous mode field
    fcm_token = Column(String, nullable=True)  # New FCM token field
    is_active = Column(Boolean, default=True)
    created_at = Column(DateTime, default=datetime.utcnow)

    ratings = relationship("Rating", back_populates="user")
    preferences = relationship("UserPreference", back_populates="user", uselist=False)
    reviews = relationship("Review", back_populates="user")
    library = relationship("Library", back_populates="user")
    likes = relationship("Like", back_populates="user")
    achievements = relationship("UserAchievement", back_populates="user")
    notifications = relationship("Notification", back_populates="owner")
```

Figure 4.4.3.2 User Model

Frontend Application Structure:

The figures below show that Flutter application uses AuthProvider to manage state, BookService for local storage of data, ApiService for communication with backend, and BookCard widget for rendering of multiple UI components. The frontend provides user authentication, book browsing, rating functionality, and personalized recommendation display.

```
AuthProvider() {
  _isLoading = true;
  notifyListeners();

  _authService.authStateChanges.listen((user) {
    // Only update _user if login is not in progress
    // This prevents the auth state listener from overriding our manual _user = null setting
    if (!_isLoginInProgress) {
      _user = user;
      _isLoading = false; // Set loading to false after getting auth state
      notifyListeners();
    }
  });
}

User? get user => _user;
bool get isLoading => _isLoading;
String? get error => _error;
bool get isAuthenticated => _user != null && _user!.emailVerified;

Future<void> signUpWithEmail(String email, String password) async {
  _isLoading = true;
  _error = null;
  notifyListeners();

  try {
    await _authService.signUpWithEmail(email, password);
    // Set user ID after successful registration
    if (_user != null) {
      await _setCurrentUserId(_user!.uid);
    }
  } catch (e) {
    _error = e.toString();
  } finally {
    _isLoading = false;
    notifyListeners();
  }
}
```

Figure 4.4.3.3 AuthProvider Class

```dart
import 'package:dio/dio.dart';
import 'package:firebase_auth/firebase_auth.dart';

class ApiService {
  final Dio _dio;

  ApiService(this._dio);

  // Get Firebase authentication token
  Future<String?> getToken() async {
    try {
      final user = FirebaseAuth.instance.currentUser;
      if (user != null) {
        return await user.getIdToken();
      }
      return null;
    } catch (e) {
      print('Error getting Firebase token: $e');
      return null;
    }
  }

  // Get current user ID
  Future<String?> getCurrentUserId() async {
    try {
      final user = FirebaseAuth.instance.currentUser;
      return user?.uid;
    } catch (e) {
      print('Error getting current user ID: $e');
      return null;
    }
  }

  // Get current user email
  Future<String?> getCurrentUserEmail() async {
    try {
      final user = FirebaseAuth.instance.currentUser;
      return user?.email;
    } catch (e) {
      print('Error getting current user email: $e');
      return null;
    }
  }
}
```

Figure 4.4.3.4 API Service Class

```dart
class BookCard extends StatelessWidget {
  final Book book;
  final VoidCallback? onTap;
  final double width;
  final double height;
  final bool isHorizontal;

  const BookCard({
    super.key,
    required this.book,
    this.onTap,
    this.width = 140,
    this.height = 200,
    this.isHorizontal = true,
  });

  Widget _buildBookCover(Book book) {
    // Get image URL, prioritize coverUrl, then imageUrl
    String? imageUrl = book.coverUrl ?? book.imageUrl ?? book.smallImageUrl;

    // Check if it is invalid URL (empty, null, or contains placeholder)
    if (imageUrl == null ||
        imageUrl.isEmpty ||
        imageUrl.contains('nophoto') ||
        imageUrl.contains('via.placeholder.com') ||
        imageUrl == 'placeholder') {
      // Show custom placeholder
      return Container(
        color: Colors.grey[200],
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(
              Icons.book_rounded,
              size: 40,
              color: Colors.grey[600],
            ),  // Icon
            const SizedBox(height: 8),
            Text(
              'No Cover',
              style: TextStyle(
                color: Colors.grey[600],
                fontSize: 12,
                fontWeight: FontWeight.w500,
              ),  // TextStyle
              textAlign: TextAlign.center,
            ),  // Text
          ],
        ),  // Column
      );  // Container
    }
```

Figure 4.4.3.5 BookCard Widget

4.4.4 Machine Learning Algorithms

Content-Based Filtering:

As shown in Figure 4.4.4.1, this technique analyses the various features of each book and finds books that are similar to ones the user has previously liked. It looks at features like genre, author, tags, and descriptions. The algorithm uses a CountVectorizer algorithm to mathematically transform book descriptions into numerical features, which are then used to calculate similarity between books. Books that have higher similarity scores are recommended first.

```
# Get similar books based on content
sim_scores = list(enumerate(cosine_sim[idx].tolist()))
sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)
sim_scores = sim_scores[1:51]  # Exclude itself, take top 50
book_indices = [i[0] for i in sim_scores]

# Get candidate books and calculate weighted rating
df = books_df.iloc[book_indices][['book_id', 'title', 'ratings_count', 'average_rating']]
v = df['ratings_count']
m = df['ratings_count'].quantile(0.60)
R = df['average_rating']
C = df['average_rating'].mean()
df['weighted_rating'] = (R*v + C*m) / (v + m)
```

Figure 4.4.4.1 Content-based Similarity Calculation

Collaborative Filtering:

As shown in Figure 4.4.4.2, this technique finds similar users based on their ratings and identifies a group of users with similar preferences. SVD is the mathematical technique used to find latent patterns in user rating behaviour. If group users rated a book highly, it is recommended to other users.

```
# Add collaborative filtering estimated ratings (skip for anonymous users)
if user_id:
    if svd is not None and HAS_SURPRISE:
        df['est'] = df['book_id'].apply(lambda x: svd.predict(user_id, x).est)
    else:
        df['est'] = 0.0
```

Figure 4.4.4.2 SVD Calculation

Hybrid Approach:

As shown in Figure 4.4.4.3, this technique is simply an equal combination of the two methods. This method takes advantage of the content-based filtering which looks in books characteristics, and the collaborative filtering which looks at patterns of behaviour. The final score is calculated simply as: final_score = (content_score + collaborative_score) / 2.

```
# Combine scores for hybrid recommendation
df['score'] = (df['est'] + df['weighted_rating']) / 2
```

Figure 4.4.4.3 Hybrid Scoring Calculation

4.4.5 System Deployment and Testing

Backend Deployment:

Start the FastAPI server by executing uvicorn app.main:app --host 0.0.0.0 --port 8000 as shown in Figure 4.4.5.1 and configuring Cloudflare Tunnel as shown in Figure 4.4.5.2. Then the backend can be publicly accessed. All backend endpoints should be tested to ensure they work properly, paying specific attention to the Firebase authentication flow and recommendation algorithm working as expected.

start_backend_task.ps1 > ...

```
# Task Scheduler
Set-Location "C:\Users\lings\book_recommend_backend"
$env:PYTHONPATH = "C:\Users\lings\book_recommend_backend"

& "C:\Users\lings\book_recommend_backend\venv\Scripts\python.exe" -m uvicorn app.main:app --host 0.0.0.0 --port 8000
```

Figure 4.4.5.1 Server Startup Script

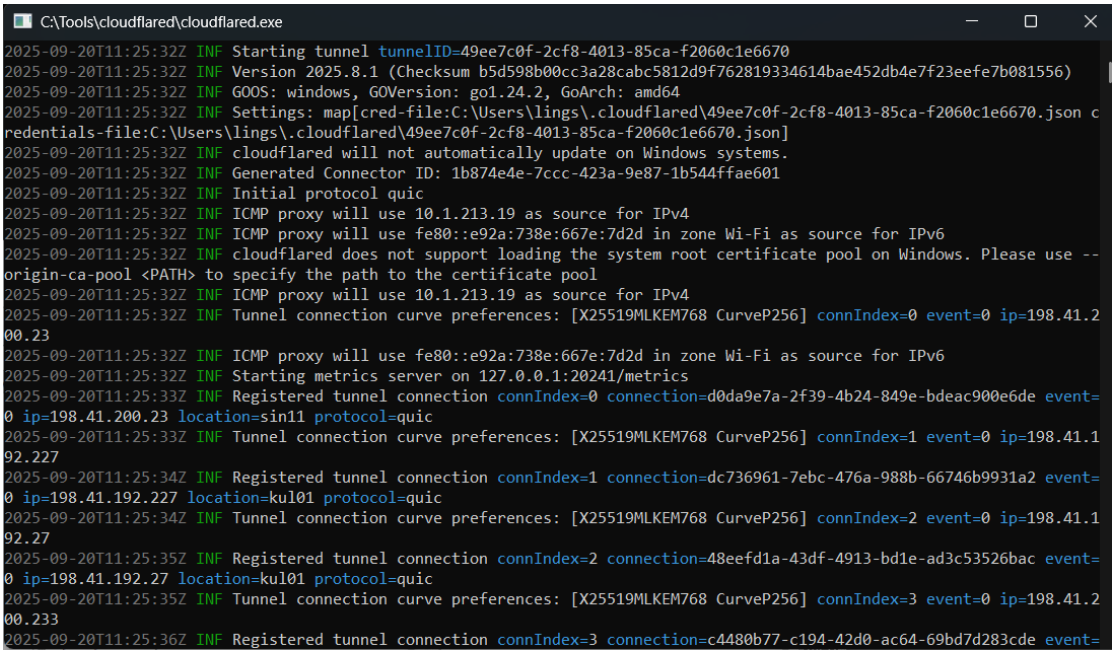


Figure 4.4.5.2 Cloudflare Tunnel

Frontend Deployment:

Create the APK using flutter build apk --release and deploy it to Android devices. Update the API endpoint in this application to point to the backend server, while also developing a way to enable developer options so everything works properly.

System Integration Testing:

Execute end-to-end tests for the full user journey from log in to receiving recommendations. Check Firebase authentication, check recommendations are being created and ensure everything is working in the end-to-end scenario.

CHAPTER 5

System Implementation

5.1 Hardware Setup

This project uses a laptop as its hardware. The laptop will be used throughout the development of the book recommender system as well as during testing and deployment phases. The specification of the laptop is stated in Table 5.1.

Table 5.1 Specifications of laptop

Description	Specifications
Model	LAPTOP-Q720F3GG
Processor	12th Gen Intel(R) Core(TM) i5-12450H
Operating System	Windows 11 Home Single Language
Graphic	Intel® UHD Graphics
Memory	16GB DDR4 RAM
Storage	512 GB NVMe PCIe SSD

5.2 Software Setup

Before starting the development of the book recommender application, various foundational software components are required to be installed and set up on the development laptop. The principal development environment is Android Studio for Flutter mobile application development, along with Python 3.9 with FastAPI framework for back-end development, and PostgreSQL database server for application data. Other tools include Jupyter Notebook for data analysis and machine learning development purposes, as well as Firebase Console for authentication and project management.

Android Studio

Android Studio is the official integrated development environment (IDE) for Android devices. The Android emulator allows users to test programs on a variety of virtual

devices with varying screen sizes, resolutions, and versions of Android. It is also a fast and stable Android mobile app development platform.

Firebase Console

Firestore Database easily stores and synchronizes data across all connected clients in real time. It provides a user-friendly interface for project management and connecting apps. It connects iOS, Android, Web, Unity, or Flutter apps to Firebase projects and provides authentication features such as email and gmail.

PostgreSQL

PostgreSQL is an open source, free, high performance, reliable, processing and data security database management system. It can handle queries and vast amounts of data. In addition, it adheres strictly to the SQL standard and provides a high degree of compatibility with many applications and technologies.

5.3 Setting and Configuration

After completing development stage, the application is ready to be tested, deployed, and used on a regular basis. The system operation begins with initializing backend servers where PostgreSQL is initialized, and migration scripts are executed to create all the tables that are needed. Environment variables for database and Firebase connections are set, then the FastAPI server is executed using uvicorn app.main:app --host 0.0.0.0 --port 8000 as shown in Figure 5.3.

```
PS C:\WINDOWS\system32> cd C:\Users\lings\book_recommend_backend
PS C:\Users\lings\book_recommend_backend> .\venv\Scripts\activate
(venv) PS C:\Users\lings\book_recommend_backend> python -m uvicorn app.main:app --host 0.0.0.0 --port 8000
←[32mINFO←[0m:     Started server process [←[36m26240←[0m]
←[32mINFO←[0m:     Waiting for application startup.
←[32mINFO←[0m:     Application startup complete.
←[32mINFO←[0m:     Uvicorn running on ←[1mhttp://0.0.0.0:8000←[0m (Press CTRL+C to quit)
```

Figure 5.3 Code of Initiating Uvicorn Server

The Flutter application is compiled into an APK file using flutter build apk --release and installed on Android devices through USB debugging or direct APK installation. After a successful build, developers can simply click the "Run" button in Android Studio and launch the app in their chosen environment.

5.4 System Operation

5.4.1 Splash Screen

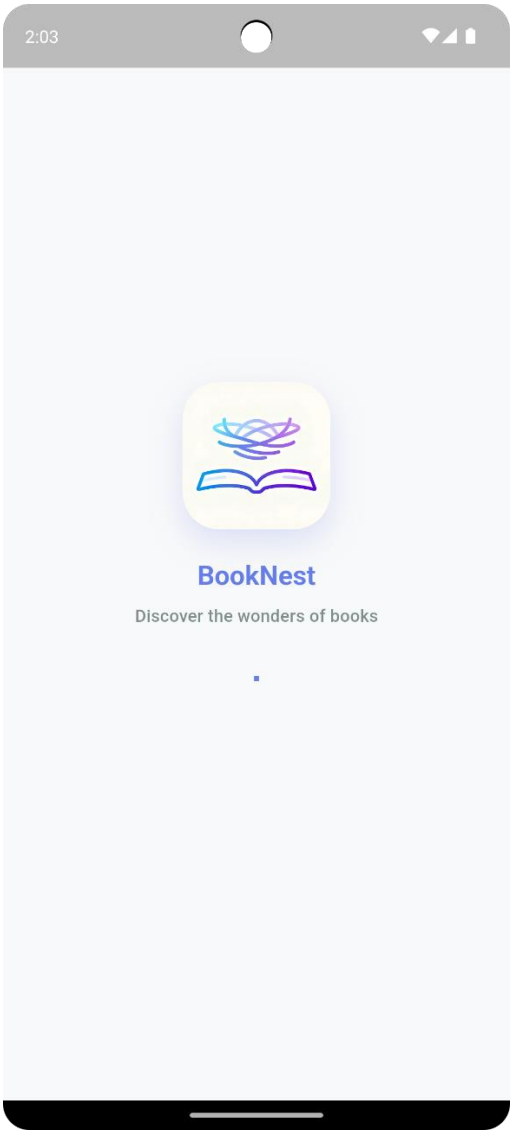


Figure 5.4.1 Splash Screen

This figure displays the splash screen that appears when a user first enters the app. This screen will last for about two seconds.

5.4.2 Login and Registration Page

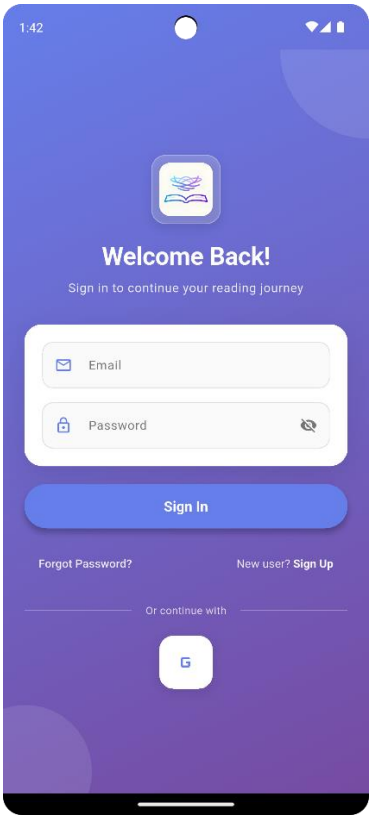


Figure 5.4.2.1 Login Page

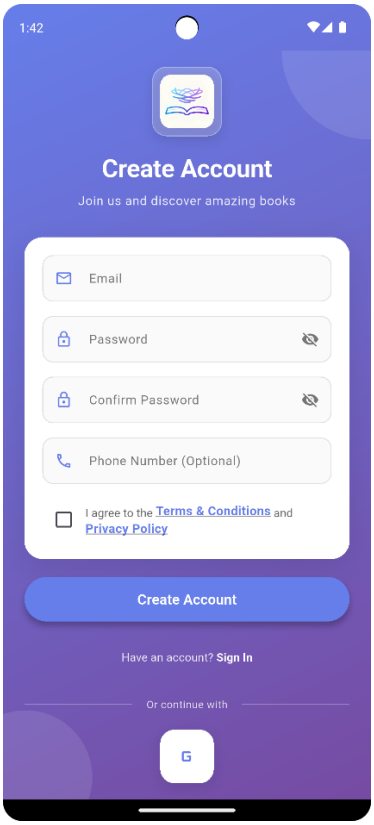


Figure 5.4.2.2 Registration Page

As shown in Figure 5.4.2.1, the user will first enter the login page. The user can choose to enter the email address and password or use the gmail verification method to log in. When the user input does not conform to the form format, the system will also prompt an error reminder. If the user forgets their password, they can utilize the Forgot Password option to accept the password change email. If the user is a first-time user, he or she can go to the registration page by clicking the sign-up prompt, as illustrated in Figure 5.4.2.2. On this page, the user can choose to enter the email address, password and confirm password or use the gmail method to register. Upon successful authentication, JWT tokens are generated and stored locally for API calls.

5.4.3 Onboarding Page

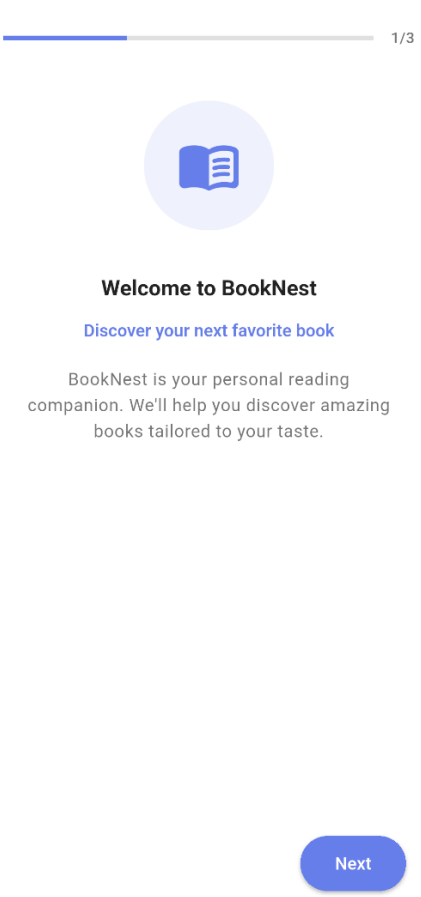


Figure 5.4.3.1 Onboarding Page

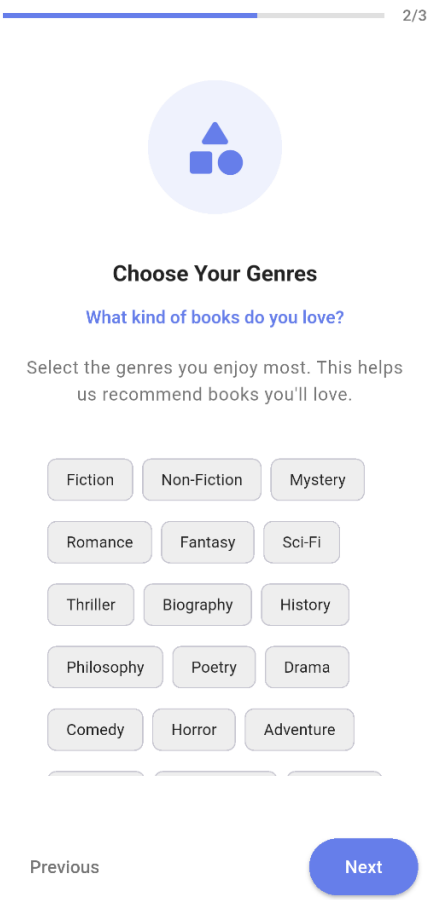


Figure 5.4.3.2 Onboarding Page

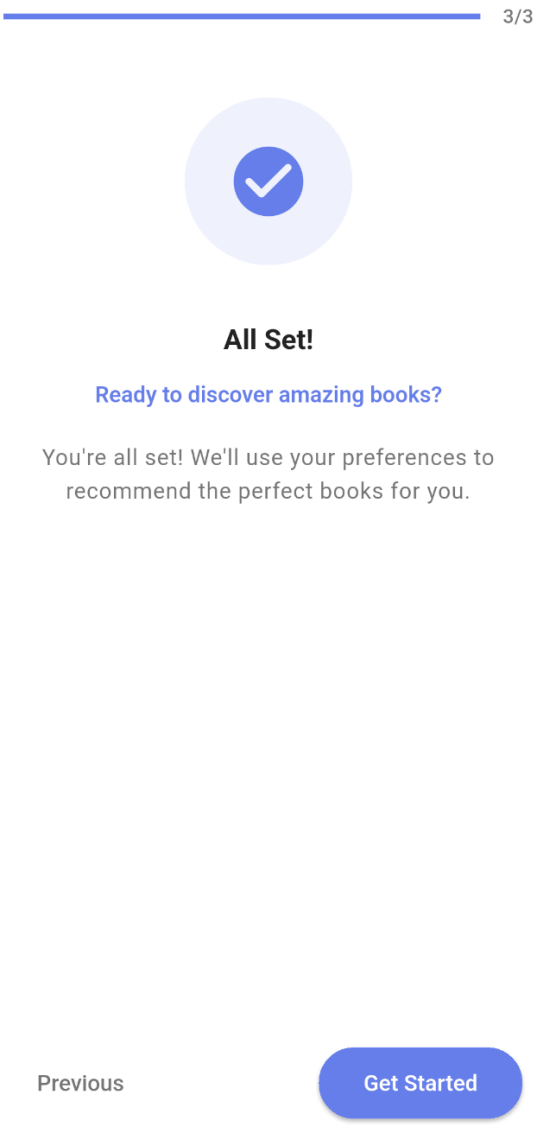


Figure 5.4.3.3 Onboarding Page

As figures shown above, the onboarding screen guides new users through the initial setup, including a welcome flow and type preference selection to personalize their experience.

5.4.4 Home Page

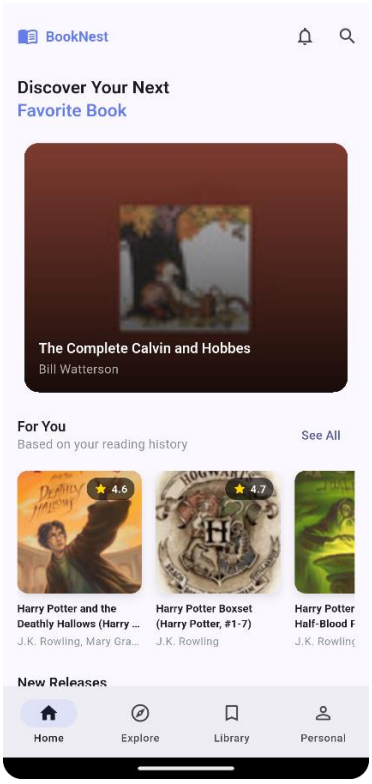


Figure 5.4.4.1 Home Page

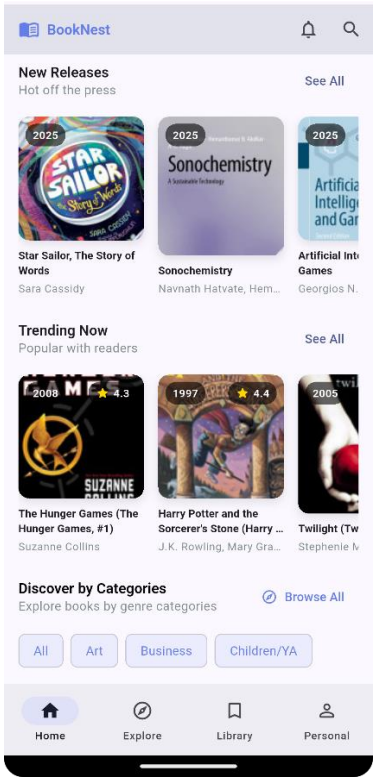


Figure 5.4.4.2 Home Page

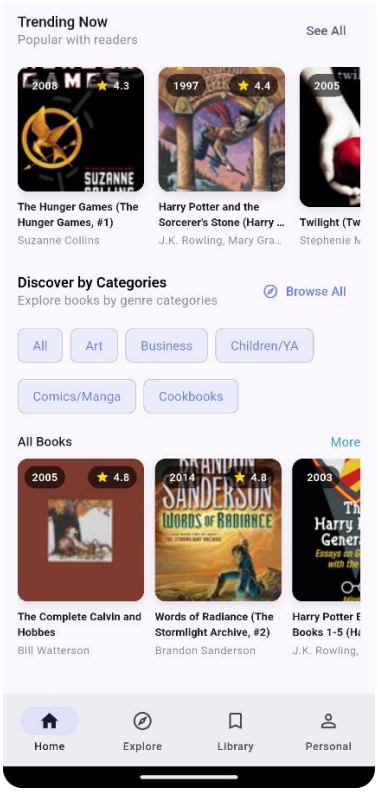


Figure 5.4.4.3 Home Page

The above figures show the home page that users enter after logging in. In this home page, user can see the top navigation bar showing search and profile icons. The page also has a welcome greeting showing "Discover Your Next Favorite Book" . The main content area displays several key sections including featured books in a horizontal scrolling carousel, personalized "For You" recommendations based on reading history, new releases section showcasing latest published books, and trending books that are currently popular among users. Each section includes a "See All" button for accessing complete lists. When users request book recommendations, the system uses a combination of content-based and collaborative filtering approaches to deliver tailored recommendations based on user and book features. Users can interact with an tag selector to filter books by specific categories. The entire page supports pull-to-refresh functionality to update all content, and users can tap on any book to navigate directly to book detail screens. The bottom navigation bar provides four main areas, including Home, Explore, Library and Personal.

5.4.5 Discover Category Page

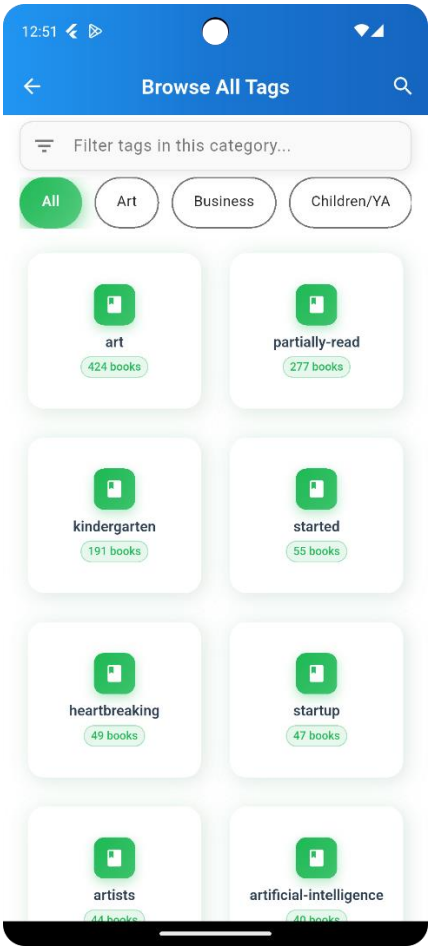


Figure 5.4.5.1 Browse All Tages

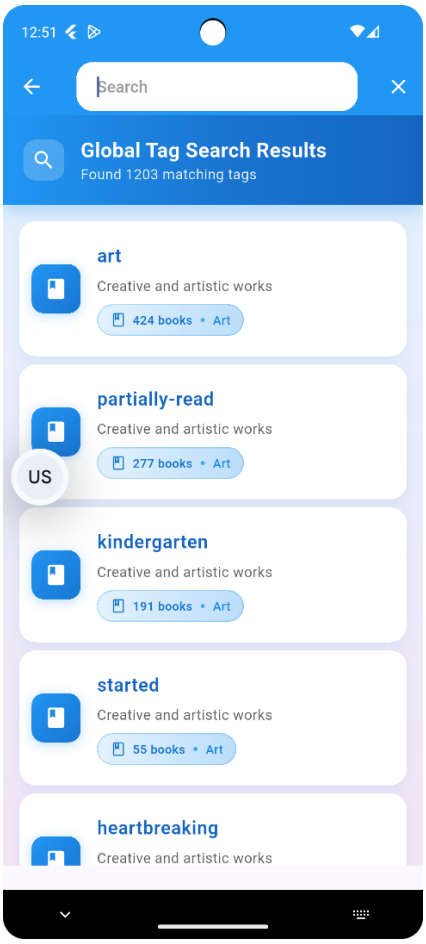


Figure 5.4.5.2 Global Tag Search Page

Clicking the "Browse All" button in the upper right corner of the "Discover by Category" section on the homepage takes users to a tag browsing page as shown in Figure 5.4.5.1, where they can browse all available book categories and tags. Clicking the search button in the upper right corner of this page allows users to search across all tags globally in the system as shown in Figure 5.4.5.2.

5.4.6 Explore Page

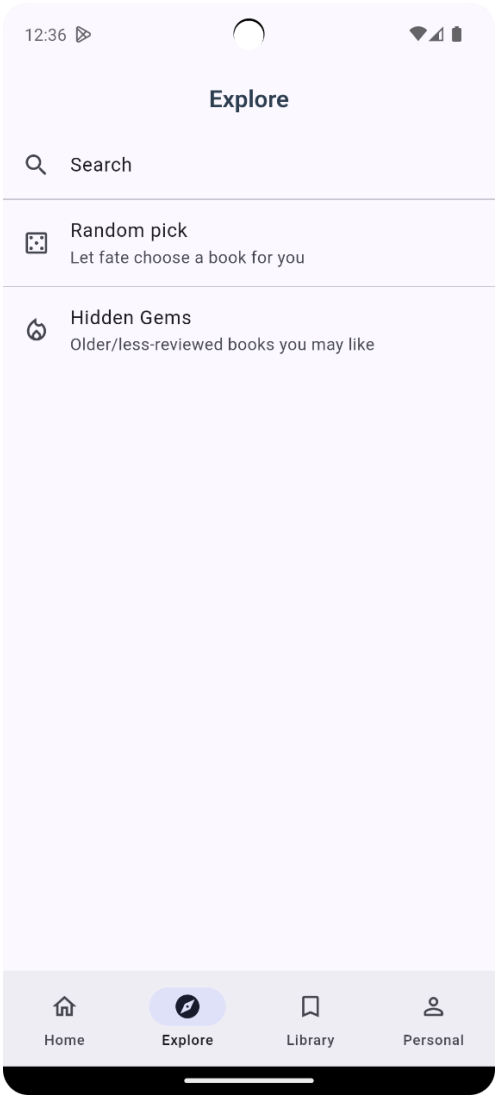


Figure 5.4.6.1 Explore Page

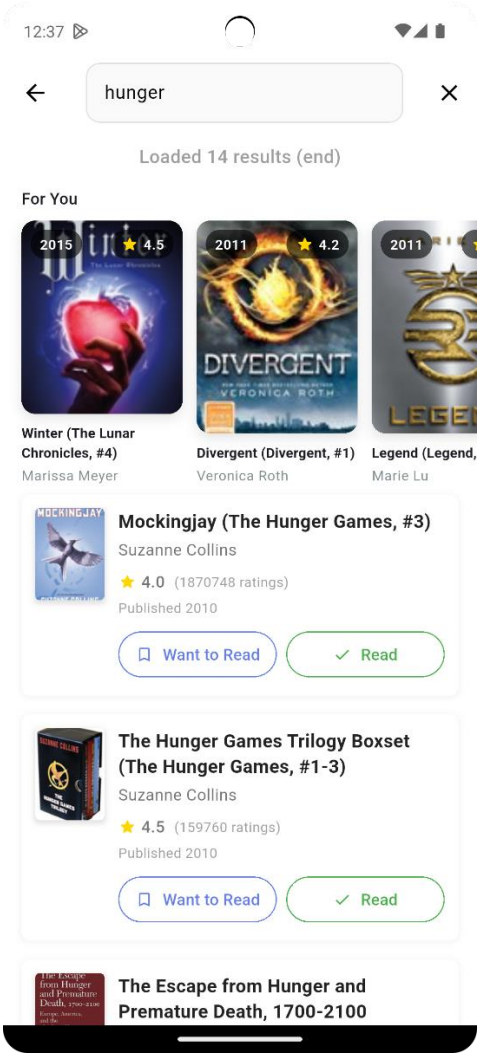


Figure 5.4.6.2 Search Page

When the users selects the explore button from the navigation bar below, the page will be redirected to the explore page. This page features three functionalities: search, random selection, and hidden gems. Upon clicking the search button, a prominent search bar appears at the top of search page. Users can perform full-text searches across book titles, authors, and book descriptions, with results displayed accordingly. The system will compute and recommend the most relevant books to the user. The “For You” section on the search page extracts the genre of the first book in the results and provides personalized recommendations within that genre based on user preferences and reading history.

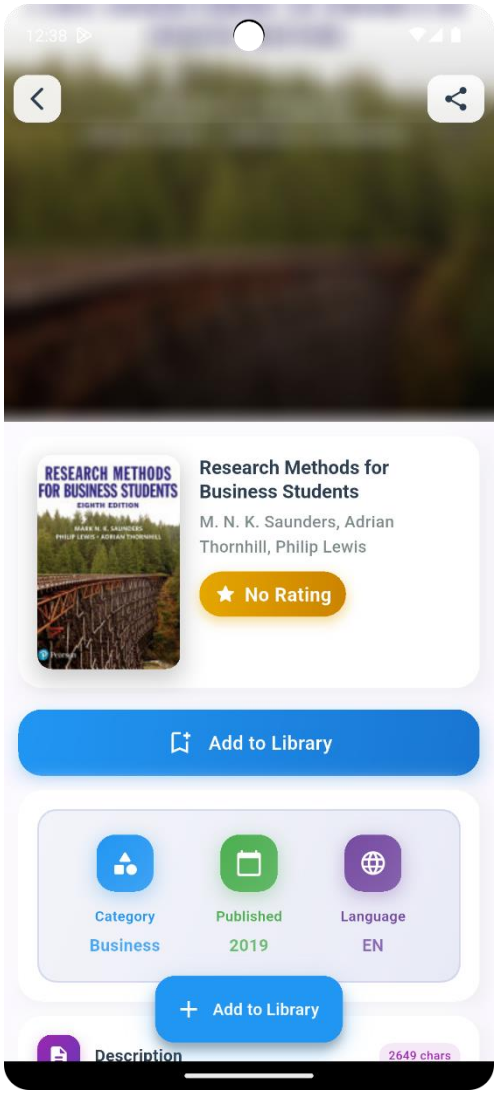


Figure 5.4.6.3 Random Pick Result

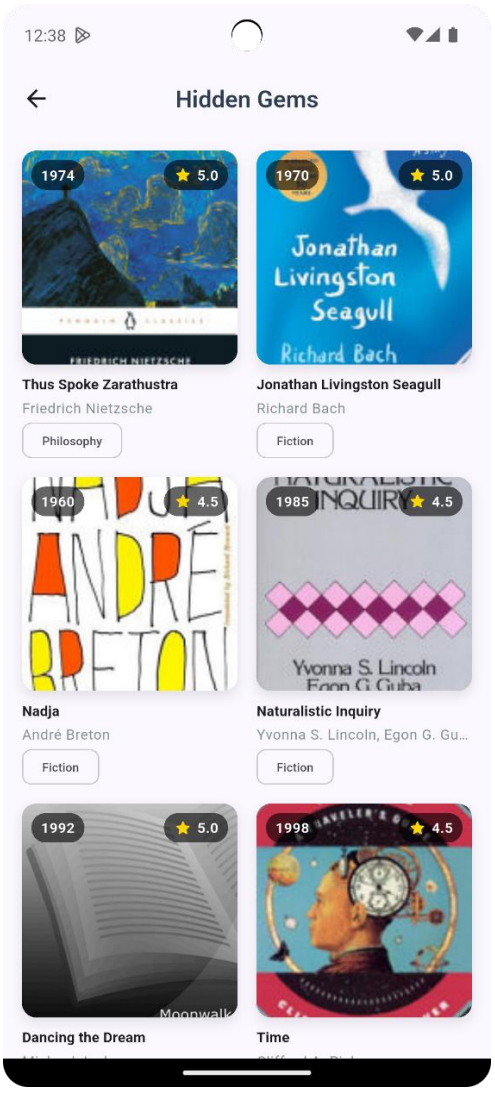


Figure 5.4.6.4 Hidden Gems Page

Users can use the "Random Pick" feature, which instantly selects a random book from the database as shown in Figure 5.4.6.3, perfect for discovering unexpected gems or breaking out of a reading rut. As shown in 5.4.6.4, the "Hidden Gems" section showcases lesser-known but highly-rated books with fewer than five reviews, helping users discover unique and underrated titles that may not appear in the main recommendations.

5.4.7 Book Details Page

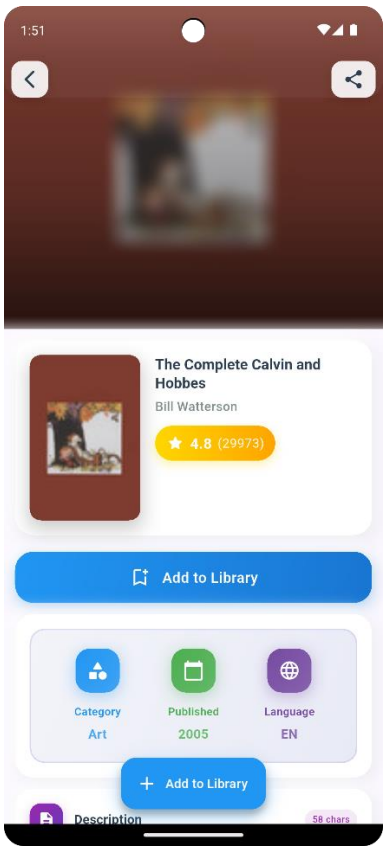


Figure 5.4.7.1 Book Details Page

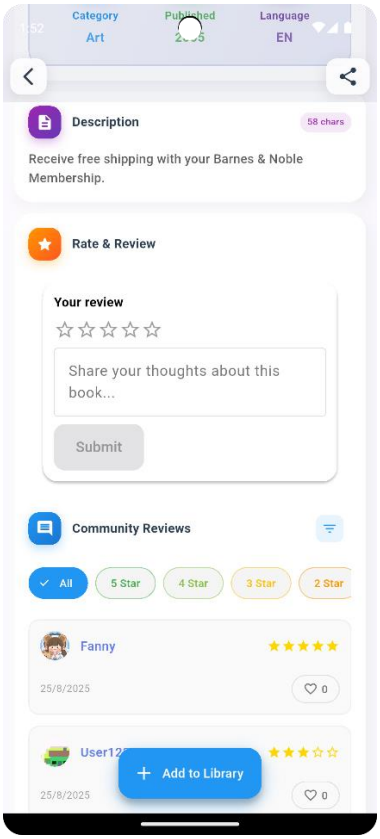


Figure 5.4.7.2 Book Details Page

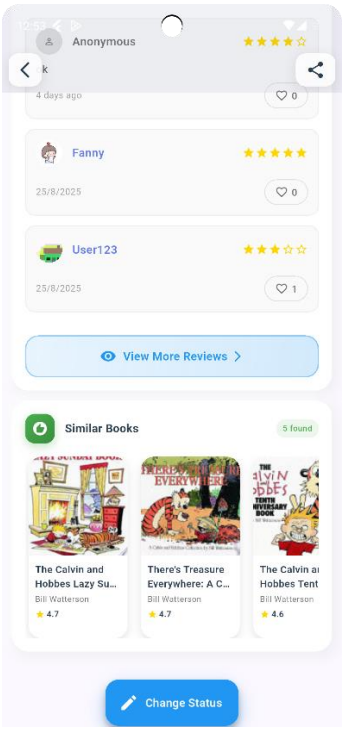


Figure 5.4.7.3 Book Details Page

The above figures show the book details page that appears after users selects a book. This page shows the book's title, author, rating, genre, publish date, and description. The users can also choose whether to read the book immediately or add it to a reading list. Users can rate and review books using a five-star rating system in the "Rate & Review" section. The "Community Reviews" section displays all users' ratings, review times, and the number of likes for the book. Scrolling down the page, the bottom section displays a list of similar books to this book.

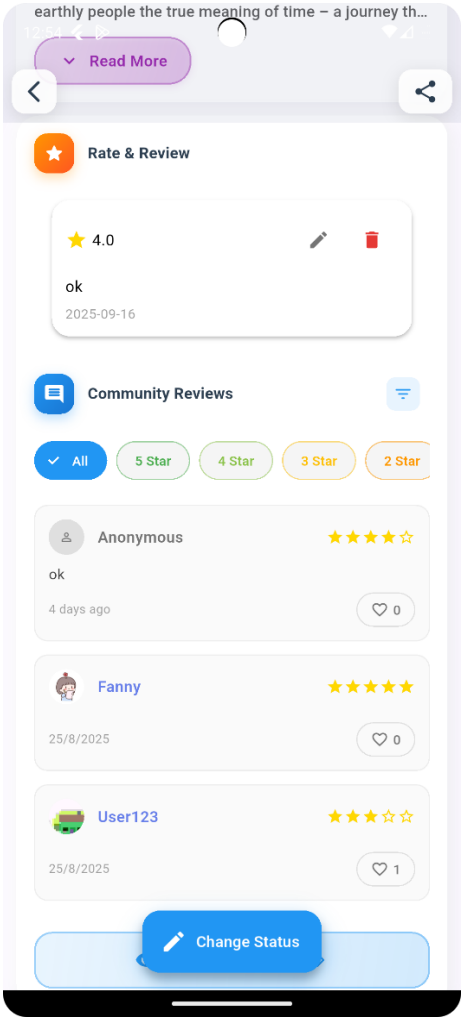


Figure 5.4.7.4 Rate & Review Page

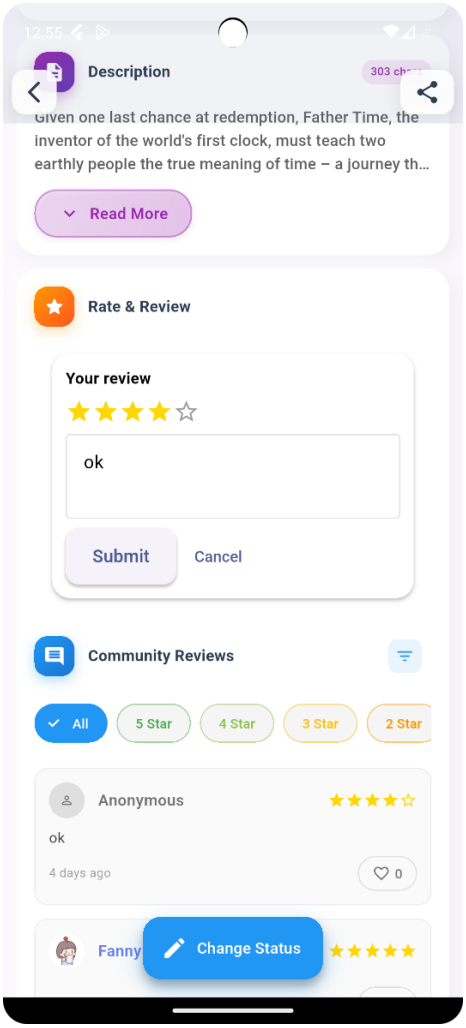


Figure 5.4.7.5 Edit Review

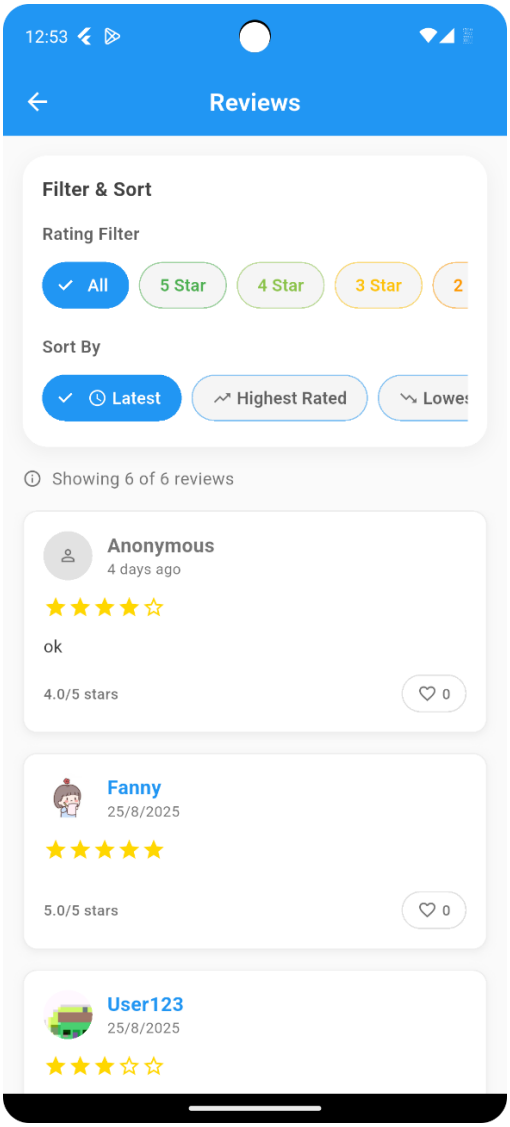


Figure 5.4.7.6 View More Reviews

As shown in Figures 5.4.7.4 and 5.4.7.5, users can click the Edit button on a review to modify their comments and submit the changes to the backend database. Users can also click the View More Reviews button under the "Community Reviews" section to be redirected to the All Reviews page, as shown in Figure 5.4.7.6. Users can read other users' reviews, filter them by rating, and sort them by date or usefulness.

5.4.8 Share Screen

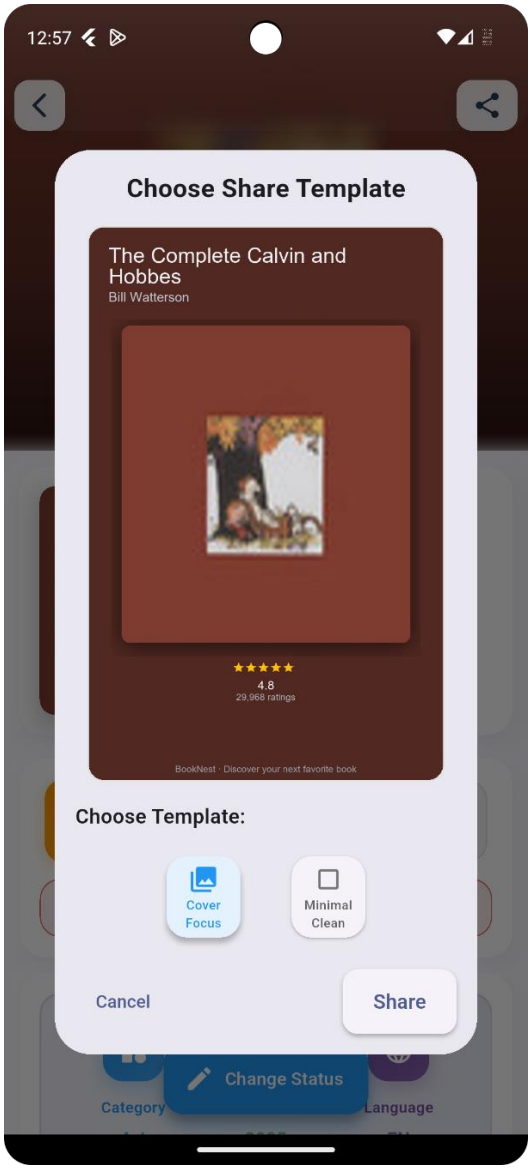


Figure 5.4.8.1 Share Template 1

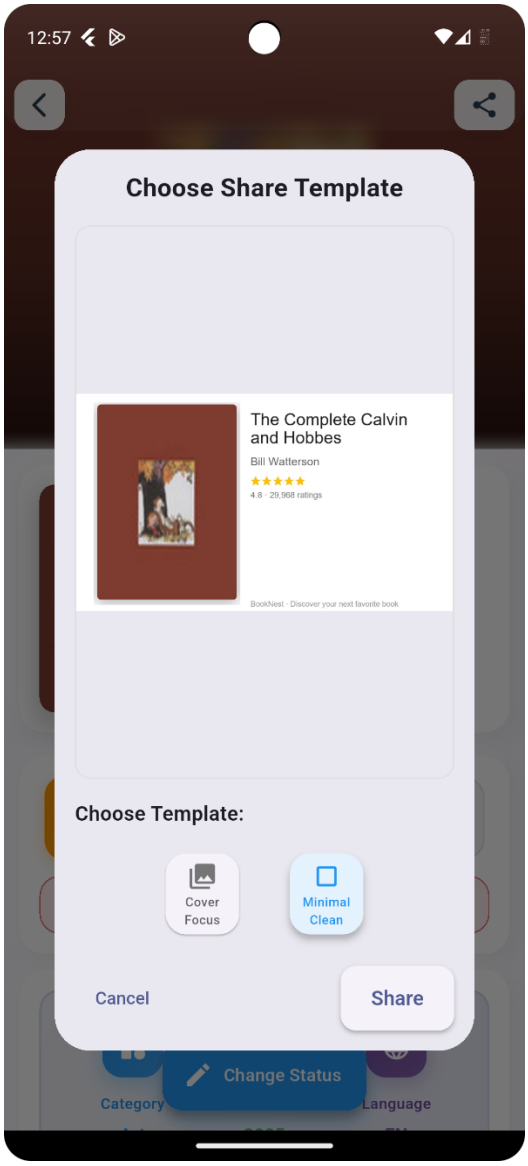


Figure 5.4.8.2 Share Template 2

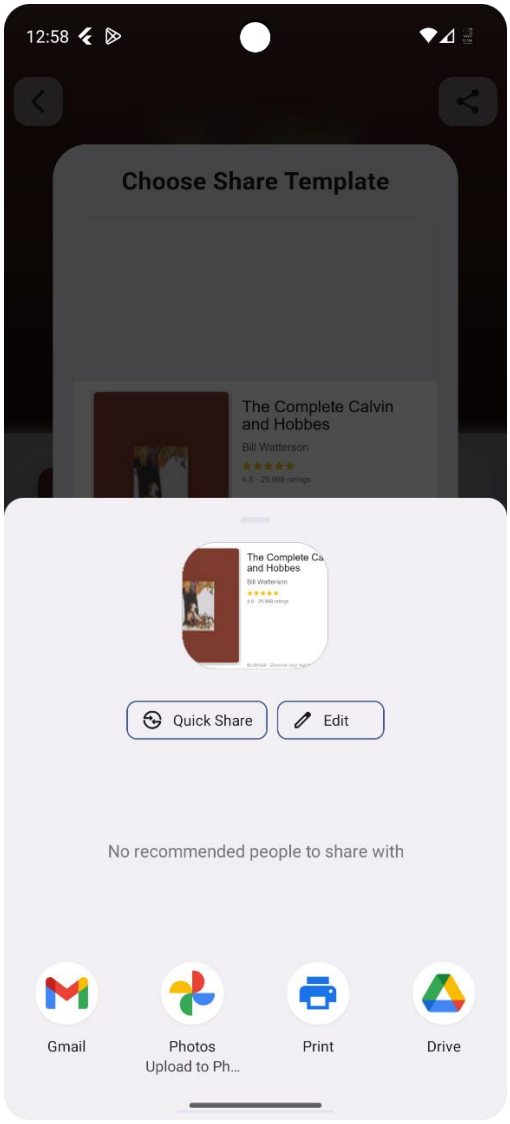


Figure 5.4.8.3 Share Platform Choice

The figures above show the sharing feature on a book details page. There are two share card templates: Cover Focus, which focuses on the cover, and Minimal Clean, which minimizes space and presents a concise card. Users can freely select and preview a share card template and share it to other platforms.

5.4.9 Library Page

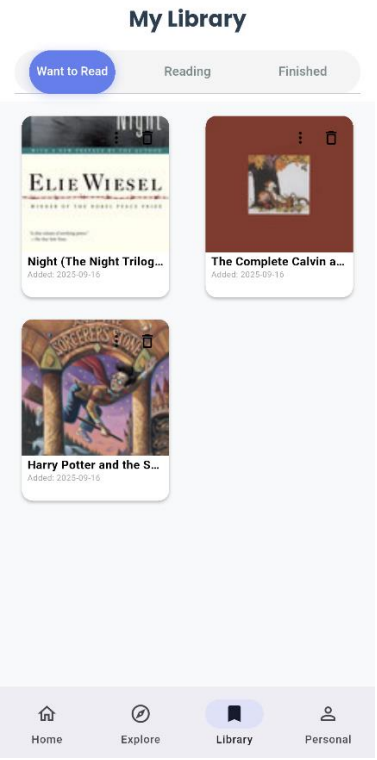


Figure 5.4.9.1 Want to Read Page

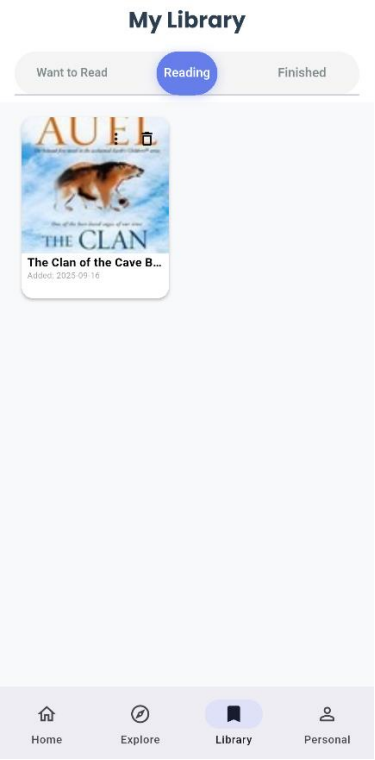


Figure 5.4.9.2 Reading Page

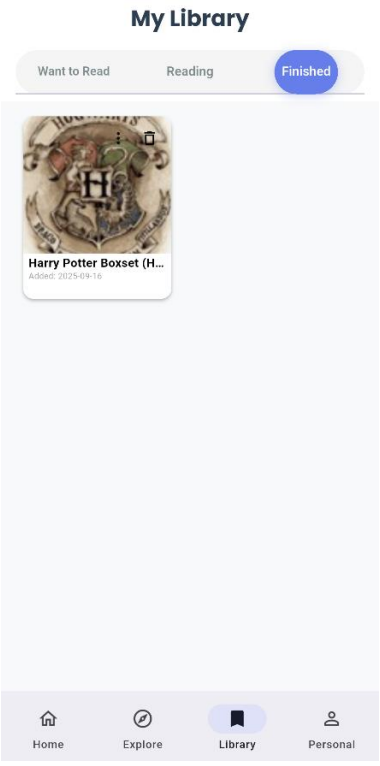
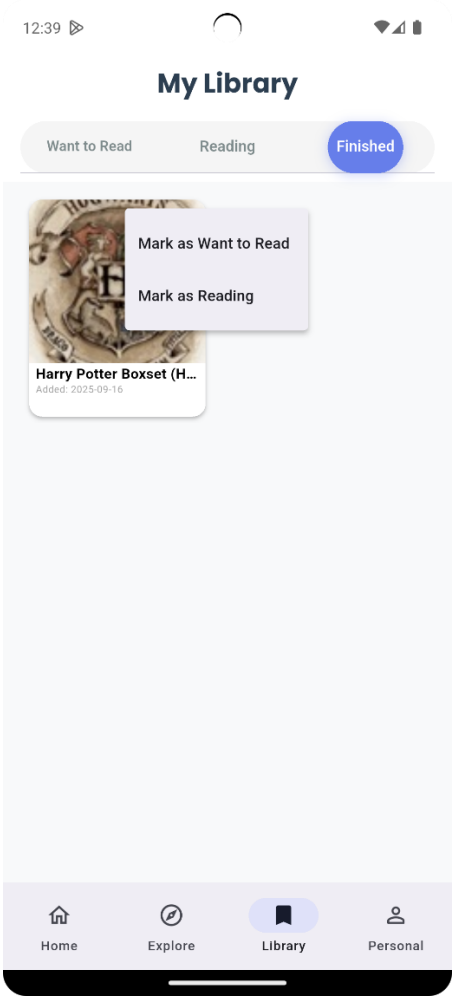


Figure 5.4.9.3 Finished Page

These figures show the pages that users visit after pressing the library button in the navigation bar. This pages are divided into three categories according to the reading status, including reading, want to read, and finished. The system allows users to switch between different reading states. The reading page displays the books that the uses are reading. The Want to Read page displays the books the users want to read in the future. The Finished page shows the books the users have finished reading. When a category is empty, users see a prompt encouraging them to add the book to their library. Users can click on any book in the library to access detailed information.



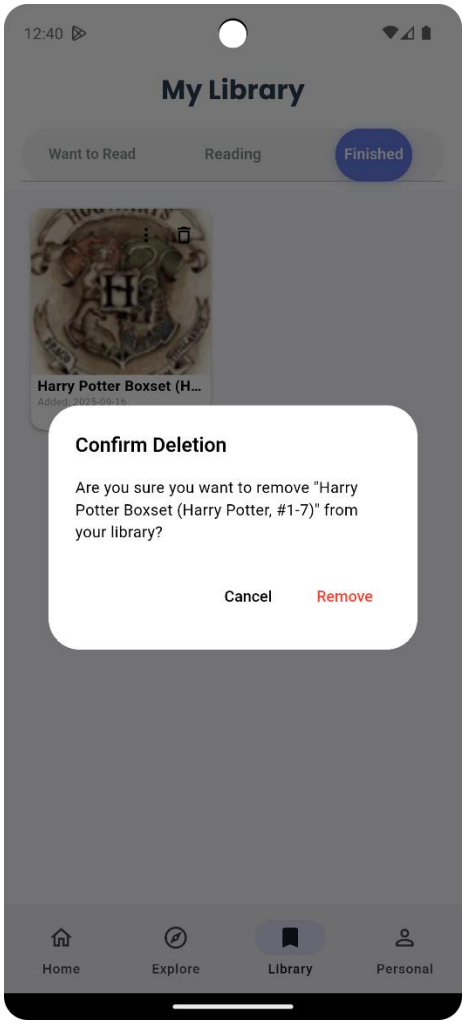
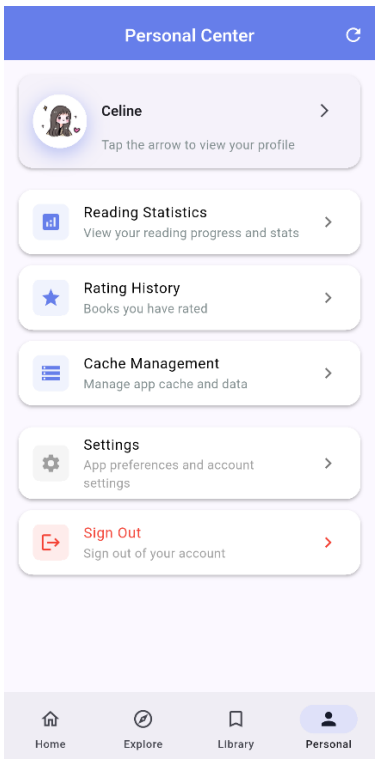


Figure 5.4.9.4 Change Library Status Figure 5.4.9.5 Delete Book

As shown in the figure above, users can also click the three-dot icon in the upper right corner of the book card to switch the current reading status of the book and click the delete icon to delete the book from the library.

5.4.10 Personal Center Page



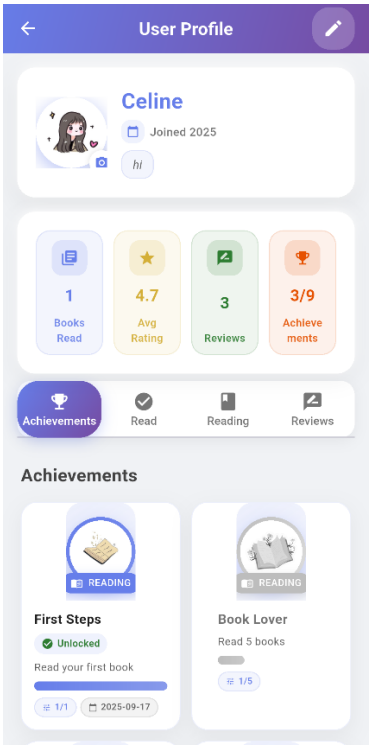


Figure 5.4.10.1 Personal Center Figure 5.4.10.2 User Profile

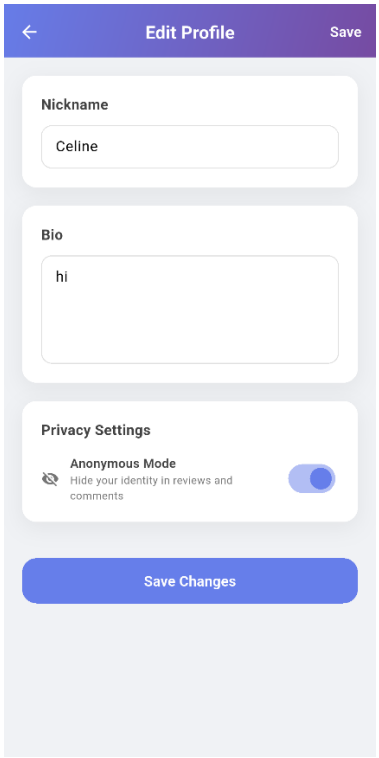


Figure 5.4.10.3 Edit Profile

Figure 5.4.10.1 shows the personal center page that displays once the users clicks the personal navigator. The Personal Center is the users' account management hub. The user information card at the top leads to the user's profile page for more personal information. The profile page includes the user's photo, nickname, joining date, biography, and reading statistics. Users can also alter their personal information, such as nicknames, biographies, and anonymous mode.

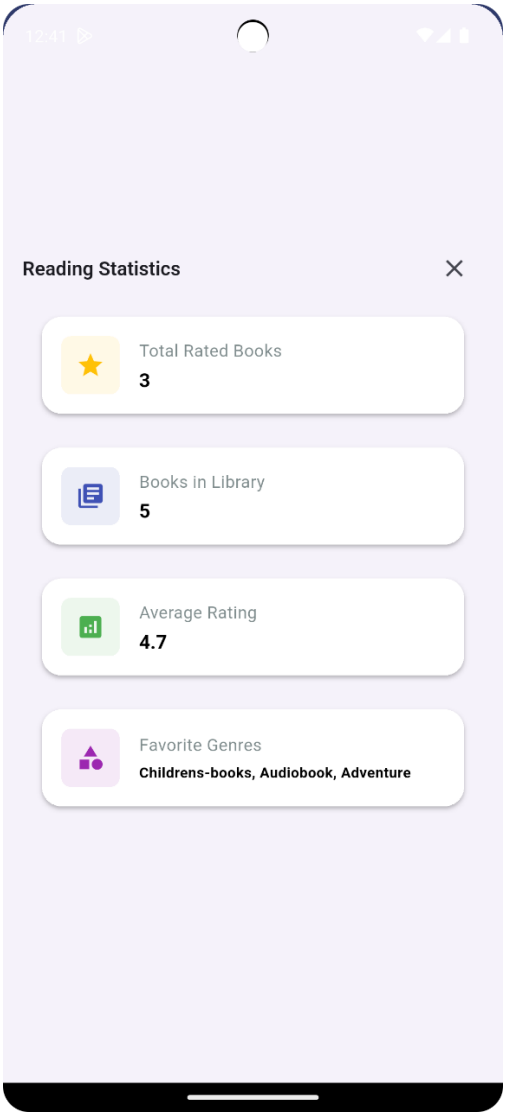


Figure 5.4.10.4 Reading Statistics

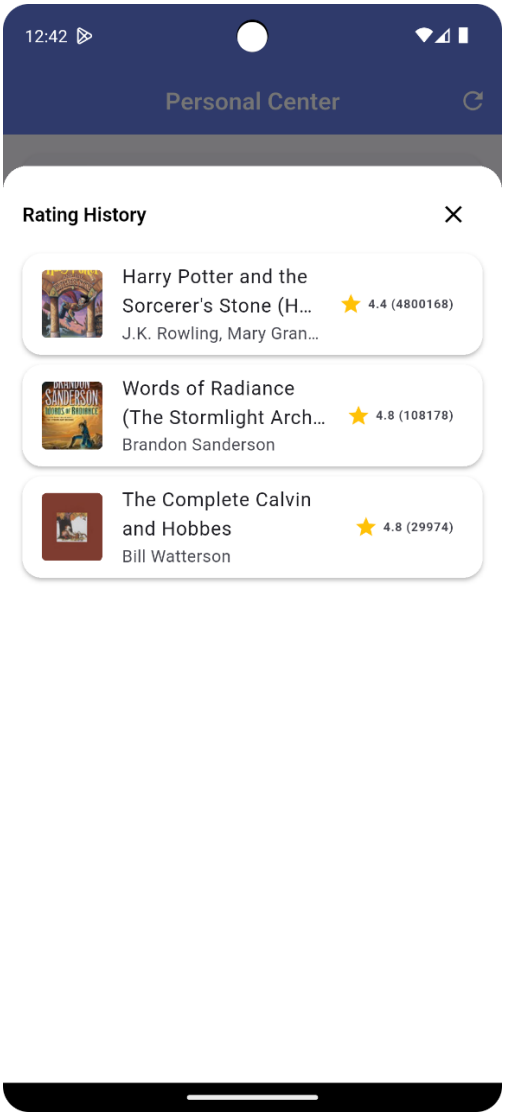
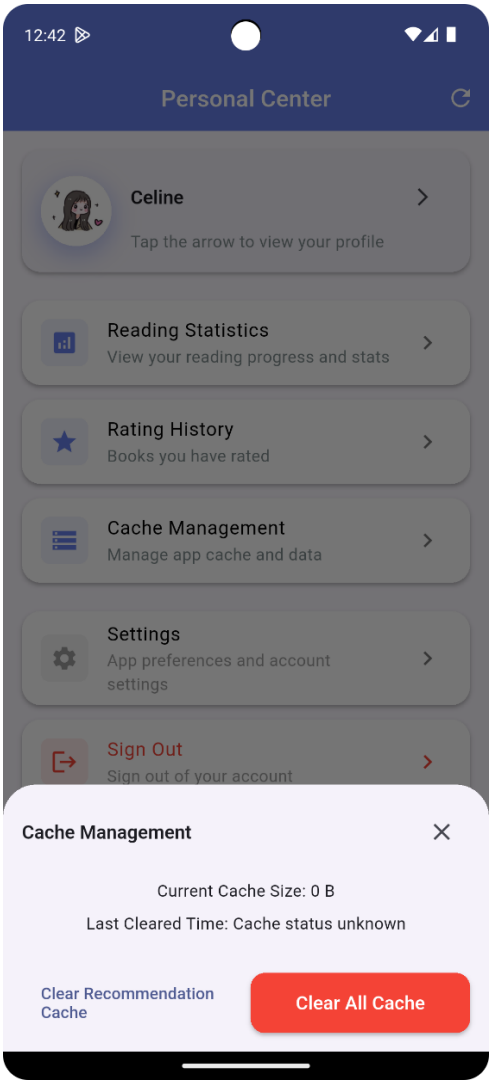


Figure 5.4.10.5 Rating History



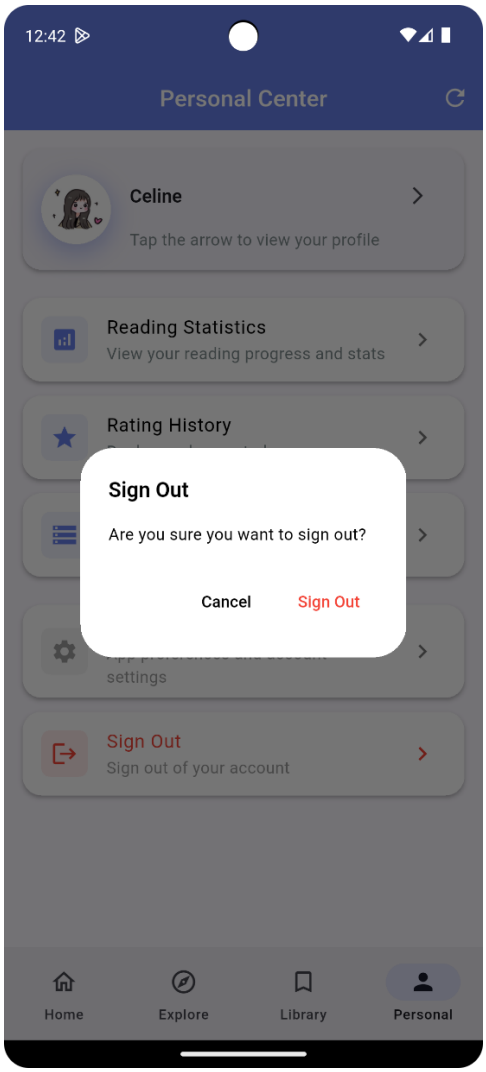


Figure 5.4.10.6 Cache Management Figure 5.4.10.7 Sign Out Prompt

As shown in the figures above, below the profile section, users will find several features, including reading statistics for detailed reading progress and analysis, a rating history for viewing all rated books, cache management for app data and storage, settings management, and a secure sign-out feature. For users signing in with Google, they can optionally set a password for their email login. Figure 5.4.10.7 shows that when the users select sign out, the page will pop up a prompt to confirm the logout. If user confirms the logout, user will be redirected to the login page.

5.4.11 Settings Screen

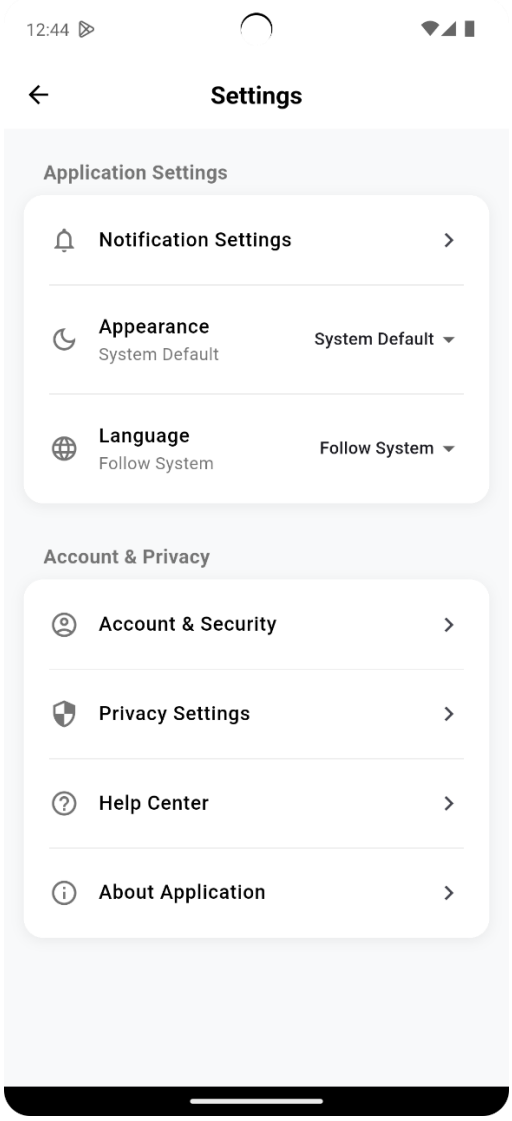


Figure 5.4.11.1 Settings Page

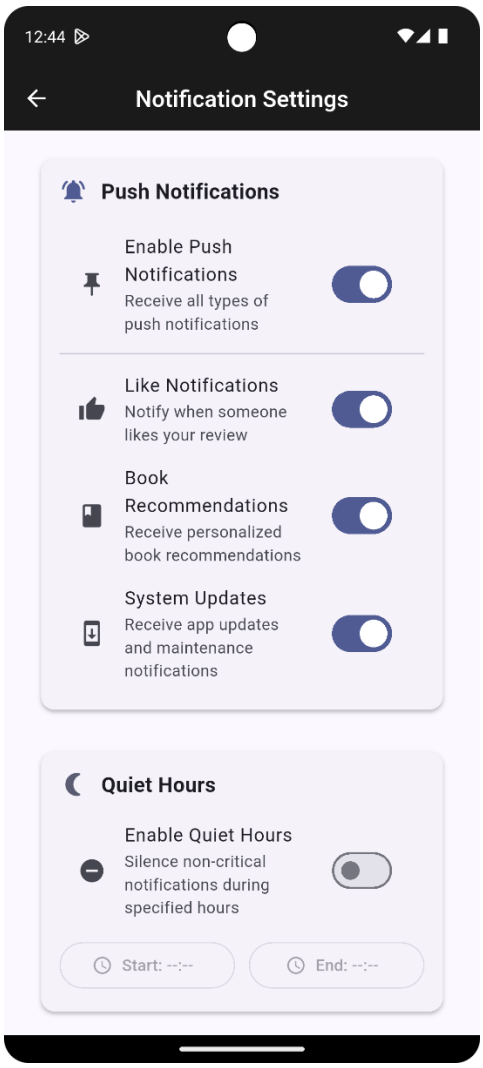


Figure 5.4.11.2 Notification Settings

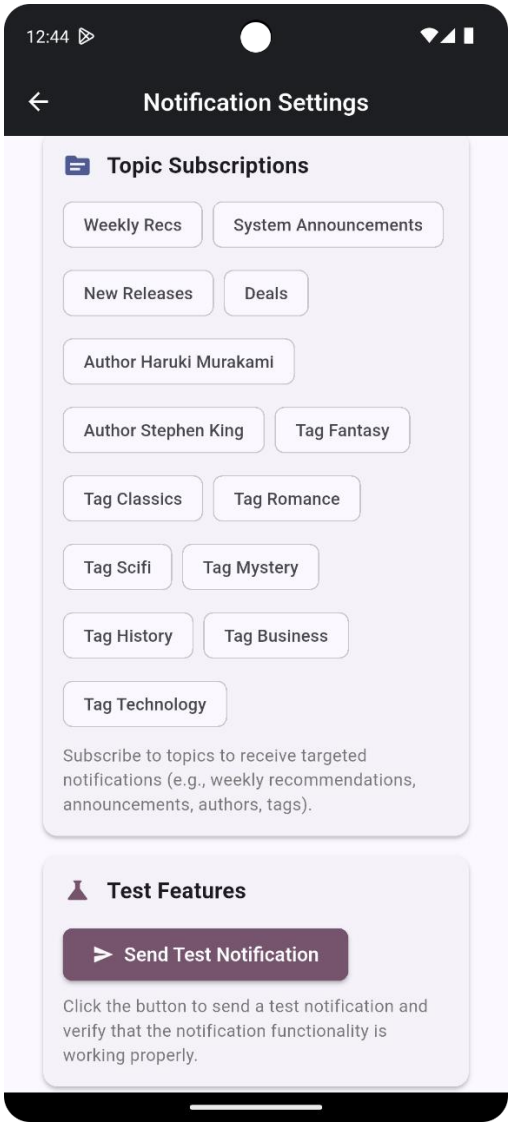


Figure 5.4.11.3 Notification Settings

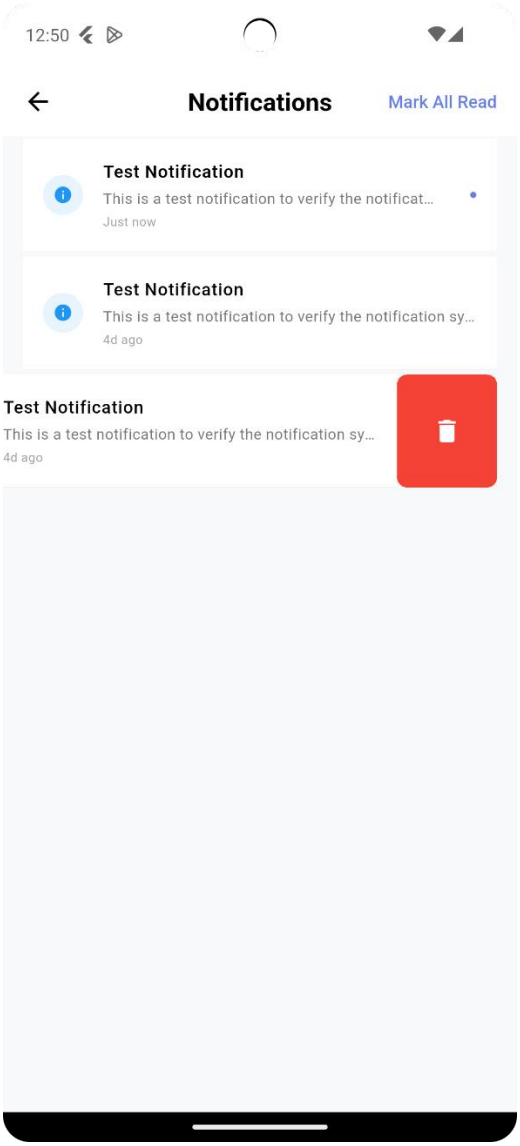


Figure 5.4.11.4 Notification Center

Figure 5.4.11.1 shows the "Settings" screen, which provides app settings such as notification controls, appearance, and language, as well as account and privacy management, including security, privacy, the help center, and about application. As shown in Figures 5.4.11.2 and 5.4.11.3, the Notification Settings section allows users to control the on/off status of various notification types, set up silent notifications, and subscribe to topics. Figure 5.4.11.4 shows the "Notification Center" on the homepage, which manages all types of app notifications, such as system updates and book recommendations. Users can view recent notifications and delete read notifications.

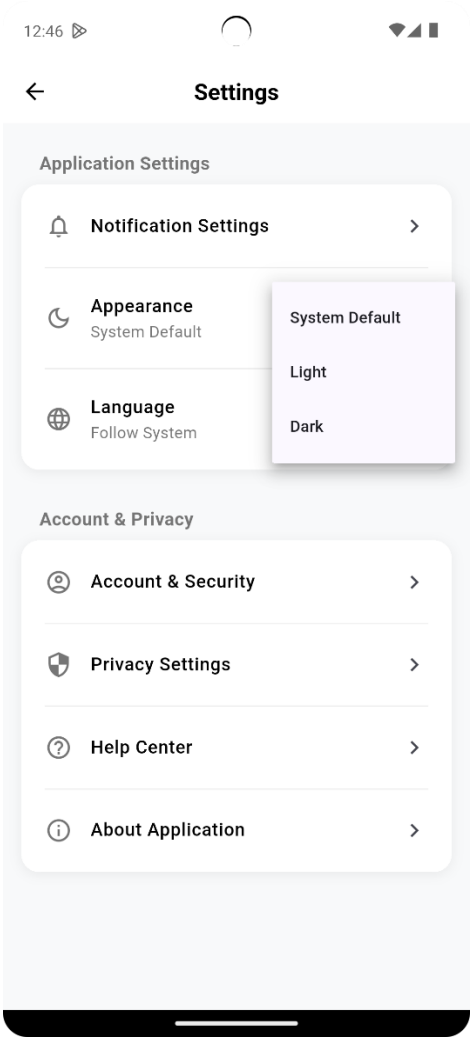


Figure 5.4.11.5 Appearance

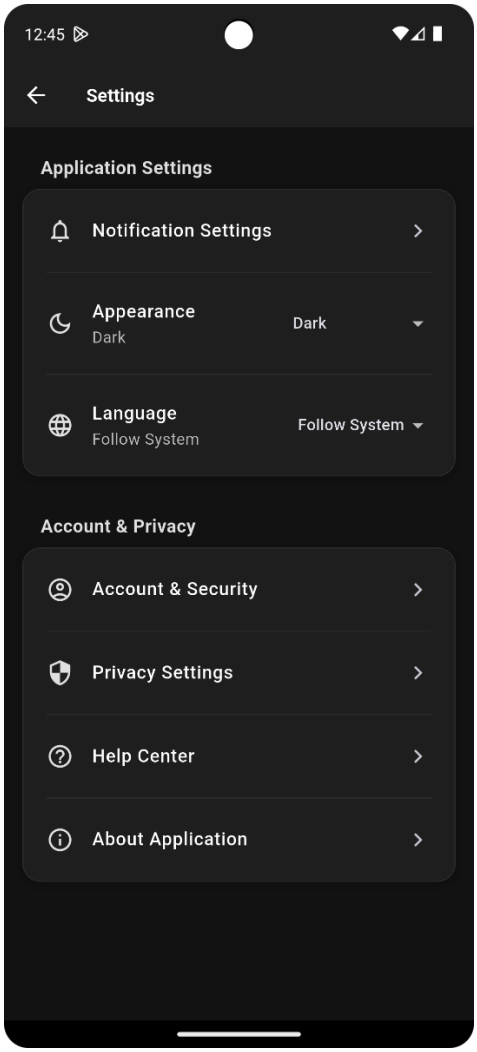


Figure 5.4.11.6 Appearance-dark

The images above show the appearance options available to users. “System Default” will automatically adjusts the app's display settings based on the user's phone's system display settings. Users can also manually select either light or dark mode to customize the app's visual appearance.

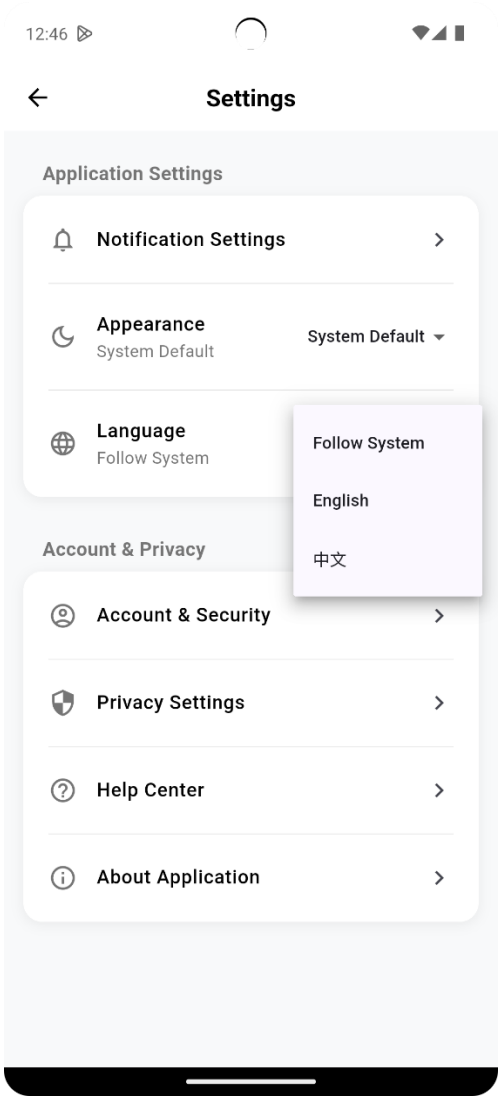


Figure 5.4.11.7 Language

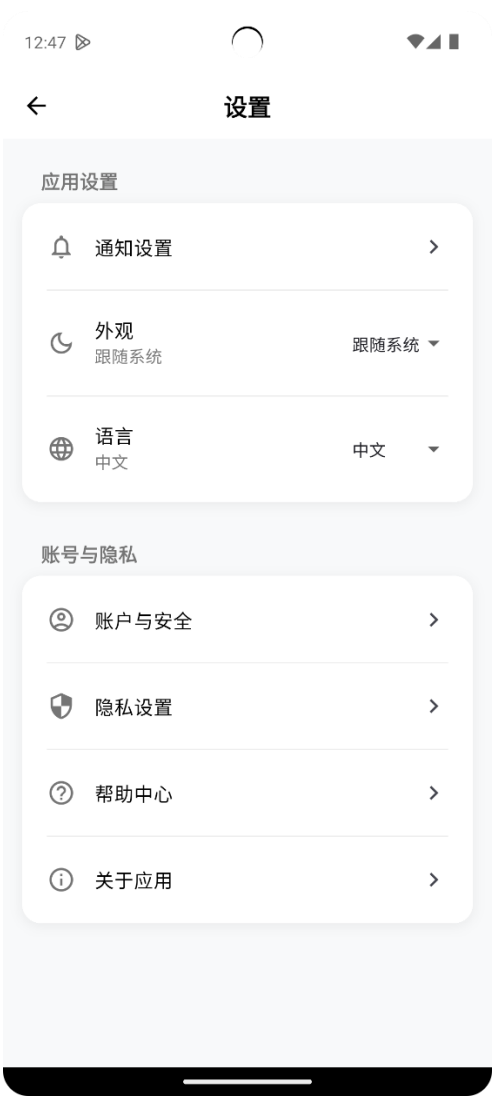


Figure 5.4.11.8 Language-Chinese

The figures above show the languages available for user selection. The system currently offers two display languages: English and Chinese. Once the users select one of these language modes, the system will apply global localization settings to adjust the display of content within the application.

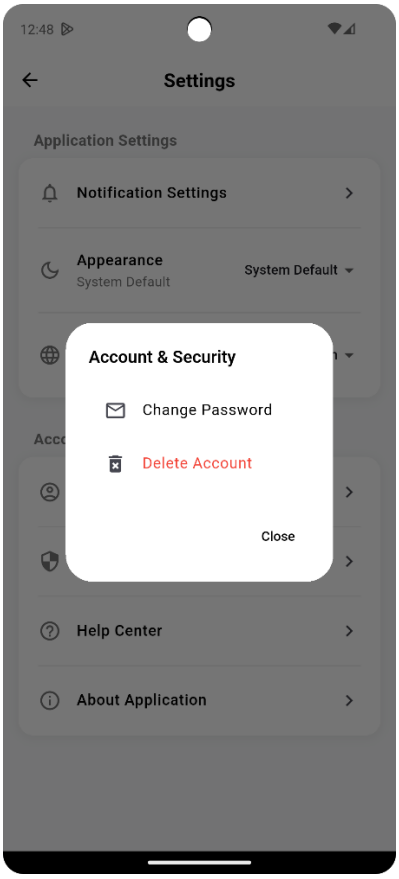


Figure 5.4.11.9 Account & Privacy

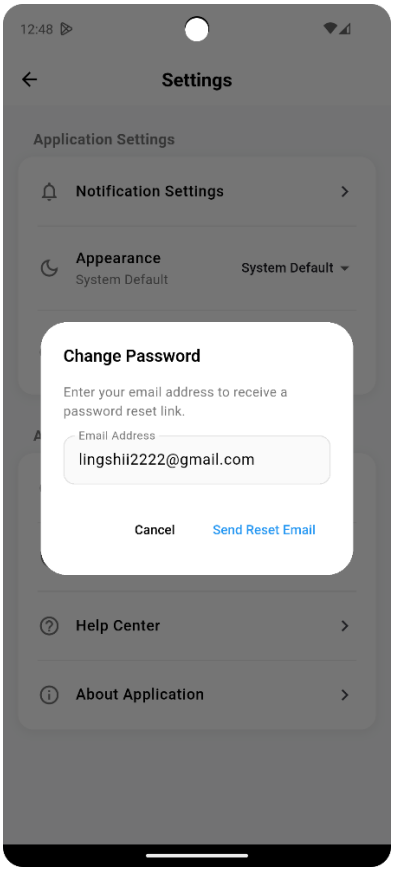


Figure 5.4.11.10 Change Password

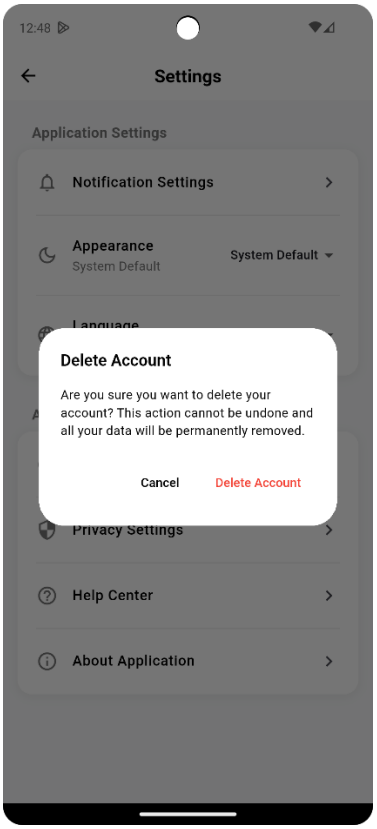
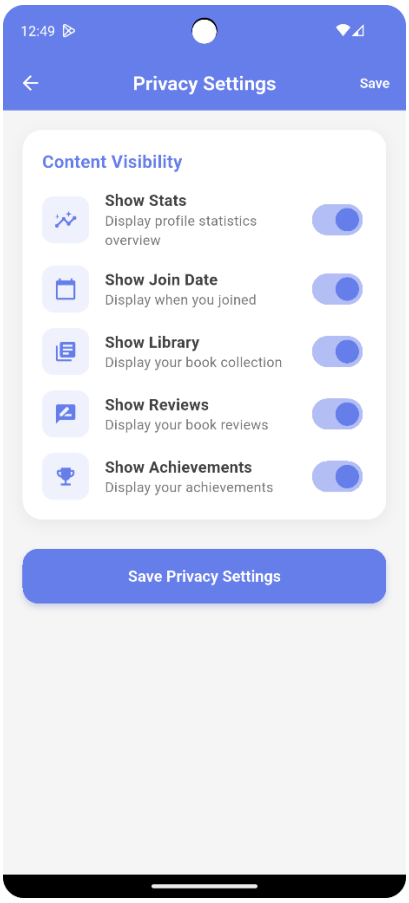


Figure 5.4.11.11 Delete Account

The image above shows the Account & Security section within the settings page, which manages password reset and account deletion processes. The password reset feature automatically uses the email registered during user signup and sends a password reset email containing a link to that user's email address. The account deletion feature comprehensively removes all data associated with the user from backend databases, Firebase, etc., including profiles, ratings, reviews, and review likes, ensuring no residual data remains within the application.



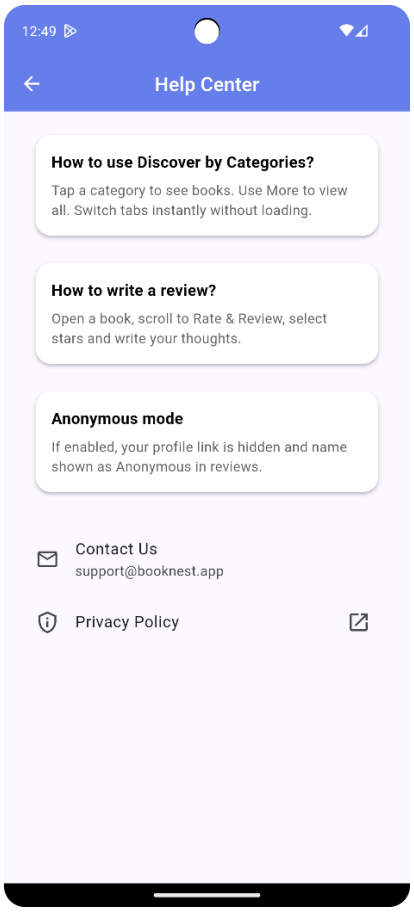


Figure 5.4.11.12 Privacy Settings Figure 5.4.11.13 Help Center

Figure 5.4.11.12 illustrates users' privacy management. Users can independently configure which data elements within their personal profiles are displayed or hidden to other users, such as statistical status, join date, book collection, reviews, and achievements. Meanwhile, Figure 5.4.11.13 shows that the “Help Center” provides user support, frequently asked questions, and privacy policies.

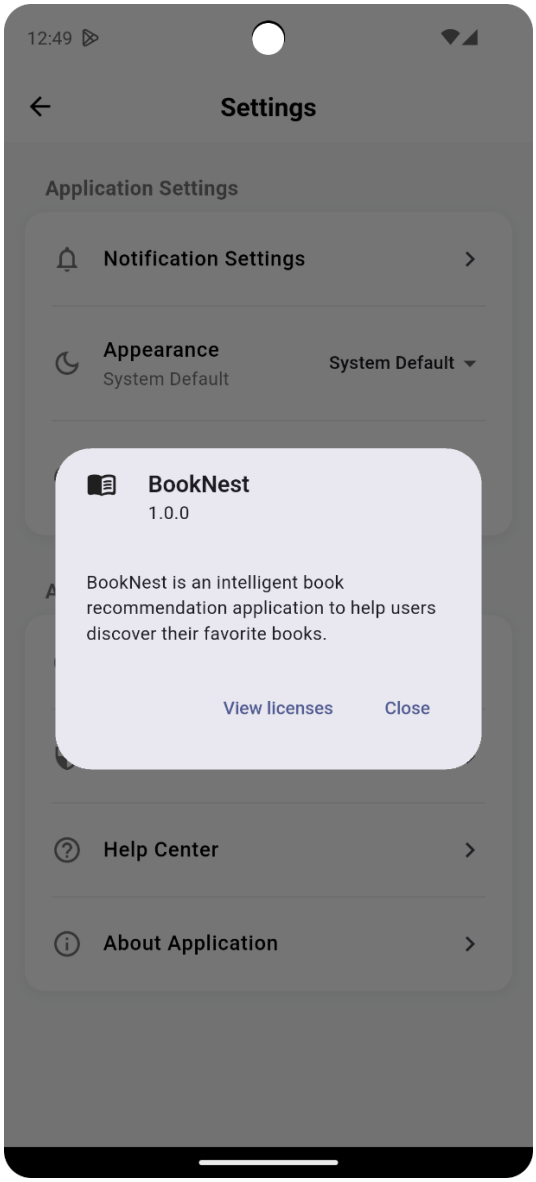


Figure 5.4.11.14 About Application

The figure above shows the "About Application" section, which contains basic application information and the licenses.

5.5 Implementation Issues and Challenges

The development of the hybrid book recommendation system faced several practical issues and challenges. One issue consisted of how to combine content-based recommendations along with collaborative filtering technologies to support balanced and individualistic recommendations. Content-based filtering recognized characteristics such as type of book, author, and description, while collaborative filtering consisted of user ratings and behaviour. Therefore, designing a system which combines the strengths of both approaches without duplicating or biasing the results takes time and effort to test and tune.

Another major challenge was maximizing the performance of the system, which placed a focus on rebuilding recommendation models and processing frontend requests with minimal latency. A full set of caching strategies were put in place to assist in this effort, including a dedicated CacheService to manage using SharedPreferences locally, caching user data for requests to avoid multiple calls to the API, and guarding against initializing more than once at the same time. The frontend included cache management features, such as monitoring cache size and cache clearing, which the user can select and clear caches for their own benefit, as well as prompts when the cache has exceeded 7 days. The backend recommendation system included basic model caching mechanisms to prevent unnecessary rebuilding of models. When deploying the backend, making it publicly accessible without exposing the server technology infrastructure can also be challenging. To accomplish this, Cloudflare Tunnel was put in place to provide secure public access without revealing anything about the server.

5.6 Concluding Remark

The development and deployment of book recommender application showed the intersection of mobile development with backend API services and machine learning algorithms. The hybrid recommendation approach combined content-based and collaborative filtering to provide personalized book suggestions while ensuring secure user identity through Firebase authentication across different platforms.

The implementation process involved setting up a comprehensive development environment with Android Studio, Python backend, and PostgreSQL database. The system architecture was achieved successfully with appropriate configuration of Firebase authentication, Cloudflare Tunnel to expose it to the public, and caching mechanisms. Algorithm integration, performance optimization, and deployment security were challenges encountered. The backend was successfully implemented with functional recommendation features, user authentication, and a mobile app deployment.

Chapter 6 : System Evaluation And Discussion

6.1 System Testing and Performance Metrics

6.1.1 Experimental Setup and Evaluation Process

To better understand the efficiency of various recommendation techniques, this project used Jupyter Notebook to perform preliminary tests and evaluations of three recommendation methods, including content-based filtering, collaborative filtering, and hybrid methods. The goal is to evaluate which model produces the best accurate recommendations for the book recommender system. The dataset used includes interactions between users, books, book tags, and ratings. The data is divided into training and test sets. The root mean square error (RMSE) was chosen as the main evaluation metric because it reflects the difference between the expected and actual ratings. The three models are implemented independently, and their prediction accuracy is evaluated. RMSE can provide an intuitive numerical representation of prediction accuracy. The smaller the RMSE value, the more accurately the model predicts a user's preferences.

The following are the steps of evaluation process:

i. Import the dataset containing book metadata, ratings and tags into Jupyter Notebook.
ii. Preprocess the data to ensure consistency, including handling missing values and splitting the data as needed.
iii. Implement three recommendation methods, including Content-Based Filtering, Collaborative Filtering, and a Hybrid approach that combines both.
iv. Train each model on the training set, then evaluate performance on the test set.
v. Calculate the RMSE for each method to evaluate how accurately the models predicted user ratings.

6.1.2 Results and Interpretation

Content-Based Filtering

```python
from sklearn.metrics import mean_squared_error

# Compute the weighted average rating based on similarity
def predict_rating(user_id, book_id, ratings_df, cosine_sim, books_df):
    # If the book ID is not in the books dataset, return NaN
    if book_id not in books_df['book_id'].values:
        return np.nan

    # Get the index of the book in books_df
    book_idx = books_df[books_df['book_id'] == book_id].index[0]

    # Sort by similarity and select the top 5 most similar books (excluding itself)
    similar_books_indices = np.argsort(cosine_sim[book_idx])[::-1][1:6]
    similar_books = books_df.iloc[similar_books_indices]['book_id'].values

    # Get user rating data
    user_ratings = ratings_df[ratings_df['user_id'] == user_id]
    ratings = user_ratings[user_ratings['book_id'].isin(similar_books)]['rating'].values
    weights = cosine_sim[book_idx][similar_books_indices]

    # Ensure the shapes of weights and ratings match
    min_length = min(len(weights), len(ratings))
    weights = weights[:min_length]
    ratings = ratings[:min_length]

    # If there are no corresponding ratings or weights, return NaN to avoid division by zero
    if len(weights) == 0 or len(ratings) == 0:
        return np.nan

    # Compute the weighted average rating
    return np.dot(weights, ratings) / np.sum(weights)


# Compute RMSE (Root Mean Squared Error)
def compute_rmse(ratings_df, cosine_sim, books_df):
    actual_ratings = []  # Actual ratings
    predicted_ratings = []  # Predicted ratings

    # Iterate through all rating data
    for _, row in ratings_df.iterrows():
        user_id = row['user_id']
        book_id = row['book_id']
        actual = row['rating']  # Actual rating

        # Compute predicted rating
        predicted = predict_rating(user_id, book_id, ratings_df, cosine_sim, books_df)

        # Skip if the predicted rating is NaN to avoid affecting RMSE calculation
        if np.isnan(predicted):
            continue

        # Store actual and predicted ratings
        actual_ratings.append(actual)
        predicted_ratings.append(predicted)

    # Compute Mean Squared Error (MSE) and take the square root to get RMSE
    mse = mean_squared_error(actual_ratings, predicted_ratings)
    rmse = np.sqrt(mse)  # Compute the square root

    return rmse
```

Figure 6.1.2.1 Functions for Calculating Prediction Rating and RMSE

```python
# calculate RMSE
rmse = compute_rmse(ratings, cosine_sim, books)
print("Content-Based Filtering RMSE:", rmse)
```

```
Content-Based Filtering RMSE: 1.2351910473727645
```

Figure 6.1.2.2 RMSE Result of Content Based Filtering

The content-based filtering model generates recommendations by comparing book characteristics such as title, genre, and other metadata. The purpose is to recommend books similar to those that the user has already liked it. The RMSE for this approach is 1.2351, which is quite high. This shows that, while the approach can produce some

good predictions, its overall accuracy is low. The restricted expressive power of book metadata may be the most significant factor influencing its accuracy. It only focuses on user history and recommends books that are very similar to each other, typically resulting in less diverse recommendations and difficulty responding to changing preferences.

Collaborative Filtering

```
from surprise import Reader, Dataset, SVD
from surprise.model_selection import cross_validate
```

```
reader = Reader()
data = Dataset.load_from_df(new_ratings[['user_id', 'book_id', 'rating']], reader)
```

```
svd = SVD()
cross_validate(svd, data, measures=['RMSE', 'MAE'])
```

```
{'test_rmse': array([0.84163669, 0.84102717, 0.84252892, 0.84133186, 0.84310189]),
 'test_mae': array([0.65898975, 0.65736011, 0.65865862, 0.65763058, 0.65926848]),
 'fit_time': (22.405536651611328,
  22.138097047805786,
  21.96229839324951,
  22.679483890533447,
  22.93682312965393),
 'test_time': (3.404756546020508,
  3.2318239212036133,
  2.9770936965942383,
  3.805542469024658,
  2.899312973022461)}
```

Figure 6.1.2.3 Functions and Results for Calculating Collaborative Filtering RMSE

The collaborative filtering model based on user similarity can use user-item interaction data, such as ratings, to identify users with similar reading behaviours and recommend books that these users might enjoy. This technique has a substantially lower root mean square error (RMSE) than the content-based filtering method, reaching 0.8419, as well as a significantly higher accuracy. This shows how utilizing user behaviour patterns, such as rating history, can help generate more relevant and personalized recommendations. However, collaborative filtering has some limitations, particularly the "cold start" problem. When a user or book is new and has not yet been rated, the system has difficulty to make good recommendations.

Hybrid Approach

```python
from sklearn.metrics import mean_squared_error

# Define function: Get Hybrid score
def hybrid_predict(user_id, book_id):
    # Find the title corresponding to book_id
    title_row = books[books["book_id"] == book_id]
    if title_row.empty:
        return svd.predict(user_id, book_id).est  # Fallback to SVD

    title = title_row["title"].values[0]

    # Call improved_hybrid to get recommendation list
    hybrid_results = improved_hybrid(user_id, title, n=10)

    # If book_id is in the recommendation list, return its Hybrid score; otherwise, return the SVD score
    if book_id in hybrid_results["book_id"].values:
        return hybrid_results[hybrid_results["book_id"] == book_id]["score"].values[0]
    else:
        return svd.predict(user_id, book_id).est  # Fallback to SVD
```

```python
# calculate RMSE
true_ratings = [r[2] for r in testset]
hybrid_predictions = [hybrid_predict(user_id, book_id) for user_id, book_id, _ in testset]
```

```python
hybrid_rmse = np.sqrt(mean_squared_error(true_ratings, hybrid_predictions))
print(f"Hybrid RMSE: {hybrid_rmse}")
```

```
Hybrid RMSE: 0.6360126576558909
```

Figure 6.1.2.4 Functions and Results for Calculating Hybrid Approach RMSE

To integrate the strengths of both content-based and collaborative filtering, a hybrid recommendation approach was implemented. This method predicts book evaluations by combining both approaches in a weighted manner. The hybrid model achieved the best RMSE score (0.6360), indicating high prediction performance. This result shows that combining content metadata with user behaviour can result in a more well-rounded and accurate recommendation engine. It also addresses issues such as data sparsity and cold starts, while providing both personalization and diversity in book recommendations.

Model Performance Summary

Model	RMSE Score	Performance Level	Key Characteristics
Content-based Filtering	1.2351	Low	High similarity bias, limited diversity
Collaborative Filtering	0.8419	Medium	Good accuracy, cold start issues
Hybrid Approach	0.6360	High	Best accuracy, addresses limitations

Table 6.1.2.1 Comparison between Three Recommendation Models

The results indicate that the hybrid method provided the greatest accuracy of prediction, as it was 48.5% more accurate than the content-based filtering and 24.5% more accurate than the collaborative filtering. This supports the conclusion that combining different recommendation strategies provides better outcomes than each strategy would independently provide.

6.2 Testing Setup and Result

The BookNest application was tested thoroughly to ensure that each of the main features was running properly. The tests covered user login, book browsing, recommendation features, profile management, and system interactions. The table includes test case, expected result, actual result (passed / failed) for each test case. All test cases passed, and results are reflected in the table below, indicating that the overall BookNest system has functionality as designed.

6.2.1 Authentication and Onboarding Testing

No.	Test Case	Expected Outcome	Actual Outcome	Pass / Fail
1.	Input fields are empty on login	Show error message	Display error message	Pass
2.	Email not verified	Show email verification dialog	Show email verification dialog	Pass
3.	Send verification email	Verification email sent successfully	Verification email sent successfully	Pass
4.	Wrong password entered	Show invalid password message	Show invalid password message	Pass
5.	Input field with wrong format	Show error message	Show error message	Pass

6.	Reset password	Reset password link sent to user email	Reset password link sent to user email	Pass
7.	Complete onboarding flow	Navigate to home screen after genre selection	Navigate to home screen after genre selection	Pass
8.	Successful login	Navigate to home screen	Navigate to home screen	Pass
9.	Logout	System navigates to login page	System navigates to login page	Pass

Table 6.2.1 Test Case - Authentication and Onboarding Testing

6.2.2 Home Screen and Navigation Testing

No.	Test Case	Expected Outcome	Actual Outcome	Pass / Fail
1.	Access home screen	Display recommendation information and book categories	Display recommendation information and book categories	Pass
2.	Navigate between bottom tabs	Switch between Home, Explore, Library, Profile	Switch between Home, Explore, Library, Profile	Pass
3.	Access recommendations page	Personalized book recommendations displayed	Personalized book recommendations displayed	Pass
4.	Access trending books	List of trending books shown	List of trending books shown	Pass

5.	Access new releases	List of new book releases shown	List of new book releases shown	Pass
6.	Browse books by categories	Books filtered by selected category	Books filtered by selected category	Pass

Table 6.2.2 Test Case - Home Screen and Navigation Testing

6.2.3 Book Browsing and Search Testing

No.	Test Case	Expected Outcome	Actual Outcome	Pass / Fail
1.	View book catalog	List of all books shown	List of all books shown	Pass
2.	Search for a book	Showing list of books matching input string	Showing list of books matching input string	Pass
3.	Browse books by tag	Books filtered by selected tag	Books filtered by selected tag	Pass
4.	Click on a book	Showing correct book detail	Showing correct book detail	Pass
5.	View long tail books	Long tail recommendation books displayed	Long tail recommendation books displayed	Pass
6.	Explore random pick feature	Random book recommendations displayed	Random book recommendations displayed	Pass

Table 6.2.3 Test Case - Book Browsing and Search Testing

6.2.4 Library Management Testing

No.	Test Case	Expected Outcome	Actual Outcome	Pass / Fail

1.	Add book to want-to-read	Book added to want-to-read list	Book added to want-to-read list	Pass
2.	Mark book as reading	Book moved to reading list	Book moved to reading list	Pass
3.	Mark book as finished	Book moved to finished list	Book moved to finished list	Pass
4.	Remove book from library	Book removed from library	Book removed from library	Pass
5.	Switch between library tabs	Display correct books in each tab	Display correct books in each tab	Pass

Table 6.2.4 Test Case - Library Management Testing

6.2.5 Book Interaction Testing

No.	Test Case	Expected Outcome	Actual Outcome	Pass / Fail
1.	Rate a book	Rating data stored to database	Rating data stored to database	Pass
2.	Edit book rating	Rating updated in database	Rating updated in database	Pass
3.	Write book review	Review data stored to database	Review data stored to database	Pass
4.	Edit book review	Review updated in database	Review updated in database	Pass
5.	Delete book rating and review	Rating and review removed from database	Rating and review removed from database	Pass
6.	Like a review	Review like count increased	Review like count increased	Pass
7.	View book ratings and reviews	Correct ratings and reviews displayed	Correct ratings and reviews displayed	Pass

Table 6.2.5 Test Case - Book Interaction Testing

6.2.6 Profile and Settings Testing

No.	Test Case	Expected Outcome	Actual Outcome	Pass / Fail
1.	View own profile	Personal profile information displayed	Personal profile information displayed	Pass
2.	Edit personal details	Profile data updated in database	Profile data updated in database	Pass
3.	View other user profile	Other user profile information displayed	Other user profile information displayed	Pass
4.	View anonymous profile	Anonymous profile cannot be viewed	Anonymous profile cannot be viewed	Pass
5.	Access settings page	Settings page displayed	Settings page displayed	Pass
6.	Switch to light mode	Interface changed to light theme	Interface changed to light theme	Pass
7.	Switch to dark mode	Interface changed to dark theme	Interface changed to dark theme	Pass
8.	Clear cache	Cache cleared successfully	Cache cleared successfully	Pass
9.	Delete account	Navigate to login page, account data removed	Navigate to login page, account data removed	Pass
10.	Login using deleted account	Show user does not exist message	Show user does not exist message	Pass

Table 6.2.6 Test Case - Profile and Settings Testing

6.2.7 Reading Status and Notifications Testing

No.	Test Case	Expected Outcome	Actual Outcome	Pass / Fail
1.	View reading statistics	Reading stats displayed correctly	Reading stats displayed correctly	Pass
2.	Enable notifications	Notification settings enabled	Notification settings enabled	Pass
3.	Disable notifications	Notification settings disabled	Notification settings disabled	Pass
4.	Receive notification	Notification received	Notification received	Pass
5.	Access notification center	Notification list displayed	Notification list displayed	Pass

Table 6.2.7 Test Case - Reading Status and Notifications Testing

6.3 Project Challenges

While developing the book recommender system, multiple challenges arose that affected the application design, performance, and user experience. These issues arosed from technical limitations and practical implementation issues that needed to be addressed to ensure smooth user experience.

One of the biggest development challenges was managing asynchronous operations throughout the application. The Flutter frontend needed to manage API calls to the backend carefully, particularly while loading book recommendation data and user data. Implementing proper state management with Provider pattern while ensuring the responsiveness of the UI during loading operations proved to be complicated and challenging. Users would experience loading states while the recommendation system was retrieving data. Paying attention to these loading states and trying not cause memory leaks or freeze the UI took many time for testing and optimization.

Another challenge presented was optimizing recommendation loading time for new users. New users that had no previous ratings, would take time to get their personalized recommendations. This involved the recommendation system not being able to use a previous rating history to recommend items and the system fell back to using content-based filtering or recommendation popular books. This will significantly degrade the user experience when first using the app, particularly when using slower internet connections. Implementing effective caching and data pre-loading strategies became crucial to foster user engagement while they were being exposed to the application.

6.4 Objectives Evaluation

1. To prepare and preprocess datasets on books and ratings from available datasets and organize data prepared for recommendations:

This objective was achieved successfully by performing careful data gathering and preparation. This project obtained book information from the Goodreads dataset and the Google Book API that included title, author, genre, tags, and user ratings. The raw data had many issues, including duplication and unknown values, which presented a huge challenge to the project. Data cleaning was done to resolve these issues by removing duplicates, filling unknown values, and normalizing text using Python. In the end, a clean dataset was obtained. This dataset was well-structured, with complete book features such as genre, author, and descriptive tags. This quality dataset allows the recommendation system to understand user preference information for making sound recommendations.

2. To develop and evaluate different recommendation models including content-based filtering, collaborative filtering, and hybrid approaches:

This objective was accomplished by developing and testing three distinct recommendation approaches. The first content-based recommendation algorithm uses features such as book genre and author to recommend similar books likely to users who enjoy certain book characteristics. The second collaborative filtering recommendation algorithm will look for others that have some similar preferences as the user and then

analysed the other user's reading history to provide book recommendations based on books favoured by similar user. Finally, the third approach is a hybrid recommendation algorithm that combines both content and collaborative filtering algorithms, like having multiple individuals providing recommendations which represent various perspectives. Extensive testing proved that the hybrid recommendation algorithm provided the best prediction rate, with an overall error rate of 0.6360 means that this approach provides more reliable recommendations. In addition, the user generally has higher satisfaction in these recommendations.

3. To design and develop a mobile application using Flutter for book recommendations with personalized user experience:

To achieve this objective, a book recommender mobile application was successfully developed. The phase of development centres on the navigation interface, which is user-friendly and attractive to the user's eye, making it very easy and comfortable to navigate. The application has relatively complete functionality, enabling secure login and registration, book browsing, the interface to rate books as well as manage the user's library. The most important feature of the application is the recommendation feature itself. Once the user logs in, the application acts like on knowledgeable librarian and continuously recommends books for the user to read not only based on their own personal preferences but also by analysing similar users accumulated reading histories. The application was written using the Flutter framework which provides smooth operation and user experience. Backend was using Python to process the recommendation algorithms, and front end was using Flutter to display results, making it easy to integrate both components together.

6.5 Concluding Remark

The chapter provides a general evaluation of the book recommendation system, from its machine learning model's performance to functionality of its mobile application. Test results reveal that a hybrid recommendation approach performs best, successfully combining content-based and collaborative filtering techniques in a bid to create

personalized and diverse book suggestions. Functional testing confirms the mobile application works fine through various user interactions.

Despite numerous challenges faced in the evaluation process, such as the fine-tuning of algorithm parameters and development of a robust test framework, the project did satisfy its goals. The book recommendation system demonstrates the seamless integration of mobile development, backend API services, and machine learning algorithms and presents a valuable platform for discovering books. The outcome of this evaluation will serve to guide future enhancements to the system.

Chapter 7: Conclusion and Recommendation

7.1 Conclusion

This project aimed to build a personalized book recommendation system that would help users find books that matched their interests. When faced with a big number of books, users are sometimes overwhelmed and unclear what to read next. The objective of this project was to address this problem by providing users with useful and reliable book recommendations based on their reading preferences and habits.

To reach this goal, a hybrid recommendation approach that combines content-based and collaborative filtering was utilized. Content-based filtering suggests books that match the user's preferences based on characteristics such as book type or author. Collaborative filtering recommends popular books chosen by other users with similar reading tastes. Combining both of these approaches allows the system to deliver more accurate and balanced recommendations, compensating for the shortcomings of a single method.

The rapid application development method was used to gradually build the system while collecting feedback and making adjustments along the process. The project uses tools such as Flutter to develop mobile applications and plans to use Python to develop the recommendation engine. Although this project did not generate a new algorithm, it did cleverly combine existing approaches to build a useful system. The hybrid engine provides access to reading suggestions that are more personal and improves user experience in reading.

In summary, the project developed a complete book recommendation system that integrates data processing, machine learning, and mobile development into one system. The hybrid recommendation algorithm was successfully developed and integrated into a simple mobile application to receive book recommendations based on user behaviour and preferences. This system demonstrates that technology can help meet real-world needs and improve reading experience by improving the process of discovering new titles for book lovers.

7.2 Recommendation

On the basis of the project result and problems encountered during development, some recommendations can be given for further expansion and improvement of the book recommender system. The system should be improved with more advanced machine learning techniques, including deep learning models that can understand complex patterns in user behavior and make even more accurate predictions. Real-time learning capability needs to be incorporated so that the system can update suggestions in real time according to new user activity, ratings, and browsing. Social features can also be incorporated to enhance user experience further by allowing users to follow friends, share book recommendations, and see what books their community is reading. It can add a social touch to the recommendation system to make it more collaborative and interactive.

Technical improvements will focus on performance optimization to handle greater users and larger sets of data, quicker database access, and more efficient recommendation engine processing. The mobile app can leverage additional capabilities such as offline reading lists, preview for books, easy integration with external book services, and better user interface design with more polished visual elements and fluid animations. The data set must be expanded with more books, genres, and user information to improve the quality of recommendations and make recommendations more diverse. Security must be improved to protect user information, and privacy settings must be provided that allow users to regulate how their data is used in recommendation generation to them.

REFERENCES

[1] N. Idrissi and A. Zellou, "A systematic literature review of sparsity issues in recommender systems," *Social Network Analysis and Mining*, vol. 10, no. 1, Feb. 2020. doi:10.1007/s13278-020-0626-2

[2] A. S. Tewari, A. Kumar, and A. G. Barman, "Book recommendation system based on combine features of content based filtering, collaborative filtering and Association Rule Mining," *2014 IEEE International Advance Computing Conference (IACC)*, Feb. 2014. doi:10.1109/iadcc.2014.6779375

[3] S. Sharma, V. Rana, and M. Malhotra, "Automatic recommendation system based on hybrid filtering algorithm," *Education and Information Technologies*, vol. 27, no. 2, pp. 1523–1538, Jul. 2021. doi:10.1007/s10639-021-10643-8

[4] B. Walek and V. Fojtik, "A hybrid recommender system for recommending relevant movies using an expert system," *Expert Systems with Applications*, vol. 158, p. 113452, Nov. 2020. doi:10.1016/j.eswa.2020.113452

[5] S. Jayalakshmi, N. Ganesh, R. Čep, and J. Senthil Murugan, "Movie Recommender Systems: Concepts, methods, challenges, and future directions," *Sensors*, vol. 22, no. 13, p. 4904, Jun. 2022. doi:10.3390/s22134904

[6] E. Ahmed and A. Letta, "Book recommendation using collaborative filtering algorithm," *Applied Computational Intelligence and Soft Computing*, vol. 2023, pp. 1–12, Mar. 2023. doi:10.1155/2023/1514801

[7] A. Rana and K. Deeba, "Online book recommendation system using collaborative filtering (with jaccard similarity)," *Journal of Physics: Conference Series*, vol. 1362, no. 1, p. 012130, Nov. 2019. doi:10.1088/1742-6596/1362/1/012130

[8] M. H. Mohamed, M. H. Khafagy, and M. H. Ibrahim, "Recommender Systems Challenges and Solutions Survey," *2019 International Conference on Innovative Trends in Computer Engineering (ITCE)*, Feb. 2019. doi:10.1109/itce.2019.8646645

[9] M. Elahi, F. Ricci, and N. Rubens, "A survey of active learning in collaborative filtering recommender systems," *Computer Science Review*, vol. 20, pp. 29–50, May 2016. doi:10.1016/j.cosrev.2016.05.002

[10] P. Mathew, B. Kuriakose and V. Hegde, "Book Recommendation System through content based and collaborative filtering method," 2016 International Conference on Data Mining and Advanced Computing (SAPIENCE), Ernakulam, India, 2016, pp. 47-52, doi: 10.1109/SAPIENCE.2016.7684166.

[11] M. Chandak, S. Girase, and D. Mukhopadhyay, "Introducing hybrid technique for optimization of book Recommender System," *Procedia Computer Science*, vol. 45, pp. 23–31, 2015. doi:10.1016/j.procs.2015.03.075

[12] Y. Tian, B. Zheng, Y. Wang, Y. Zhang, and Q. Wu, "College Library personalized recommendation system based on hybrid recommendation algorithm," *Procedia CIRP*, vol. 83, pp. 490–494, 2019. doi:10.1016/j.procir.2019.04.126

[13] Z. Ali, S. Khusro, and I. Ullah, "A hybrid book recommender system based on table of contents (TOC) and association rule mining," *Proceedings of the 10th International Conference on Informatics and Systems*, May 2016. doi:10.1145/2908446.2908481

Poster


HYBRID BOOK RECOMMENDATION SYSTEM
INTRODUCTION
Finding the appropriate book to read can be difficult due to the large amount of options available. This project aims to develop a personalized book recommendation system that helps users in discovering books based on their interests and preferences. To make more accurate recommendations, the system uses an approach that blends Content-Based Filtering and Collaborative Filtering.
METHODS
• User Input
Users interact with the app by rating books, searching, or browsing.
• Preprocessing
User ratings are cleaned, and book information like genre and author is extracted.
• Databases
Book Database stores book titles, genres, authors. User Database stores reading history and ratings.
• Filtering
Content-Based Filtering: Recommends books with similar content to user preferences.
Collaborative Filtering: Recommends books liked by similar users.
• Hybrid Recommender
Combines both filtering results to provide more accurate and diverse suggestions.
DISCUSSION
• The hybrid method balances the strengths and weaknesses of both content-based and collaborative approaches.
• Rapid Application Development methodology was used, with continuous improvement based on feedback.
• The system was built using Flutter for the mobile interface and Python for recommendation logic.
• Initial testing shows more relevant and diverse suggestions compared to using one method alone.
TOOLS USED
• Flutter (Mobile App UI)
• Python (Recommendation Engine)
• Firebase (Optional backend)
• Jupyter Notebook (Data Preprocessing & Evaluation)
jupyter
CONCLUSION
This project developed a personalized book recommendation system using a hybrid filtering method. It makes it easier for users to find books they may enjoy by combining user behavior and book content. The system provides a good foundation for further development, such as adding user reviews or improving recommendation accuracy.
Bachelor of Computer Science (Hons)
By: Ling Shi Shi
Supervisor: Ts Dr Phan Koo Yuen