Tribhuvan University

Academia International College

Final Year Project Report

On

Books Recommendation System

[CSC 412]

Under the supervision of

"Mr. Ganesh Prasad Bhatta"

Submitted By

Loojah Bajracharya (T.U. Exam Roll No. 26495/077)

Rayan Pokharel (T.U. Exam Roll No. 26504/077)

Reshak Maharjan (T.U. Exam Roll No. 26505/077)

Submitted To

Department of Computer Science and Information Technology
Academia International College
Institute of Science and Technology
Tribhuvan University

January, 2025

Tribhuvan University

Academia International College

Final Year Project Report

On

Books Recommendation System

[CSC 412]

A final year project submitted in partial fulfillment of the requirement for the degree of Bachelor of Science in Computer Science and Information Technology awarded by Tribhuvan University

Submitted by

Loojah Bajracharya (T.U. Exam Roll No. 26495/077)

Rayan Pokharel (T.U. Exam Roll No. 26504/077)

Reshak Maharjan (T.U. Exam Roll No. 26505/077)

Submitted to

Department of Computer Science and Information Technology Academia International College

Institute of Science and Technology

Tribhuvan University

January, 2025

Tribhuvan University

Institute of Science and Technology

Academia International College

Department of Computer Science and Information Technology

Email: mail@academiacollege.edu.np

Supervisor's Recommendation

I hereby recommend that the project work report prepared under my supervision by Mr. Loojah Bajracharya (26495/077), Mr. Rayan Pokharel (26504/077), and Mr. Reshak Maharjan (26505/077) entitled "Books Recommendation System (HamraKitab)" be accepted as fulfilling in partial requirements for the degree of Bachelors of Science in Computer Science and Information Technology. In my best knowledge, this is an original work in Computer Science and Information Technology.

.....

Mr. Ganesh Prasad Bhatta

Project Supervisor

Department of Computer Science and Information Technology

Academia International College

Gwarko, Lalitpur

Tribhuvan University

Department of Computer Science and Information Technology

Academia International College

Certificate of Approval

This is to certify that this project prepared by Mr. Loojah Bajracharya, Mr. Rayan Pokharel and Mr. Reshak Maharjan entitled "Books Recommendation System (HamraKitab)" in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Information Technology has been well studied. In our opinion, it is satisfactory in the scope and quality as a project for the required degree.

…….. Mr. Ganesh Prasad Bhatta Project Supervisor Department of Computer Science and IT Academia International College	…….. Mr. Bishwas Mathema HOD/Program Coordinator Department of Computer Science and IT Academia International College
…….. Internal Examiner Academia International College	…….. External Examiner Central Department of CSIT Tribhuvan University

Acknowledgement

We are extremely grateful to Academia International College for providing us with the opportunity to complete this project as a requirement for our course.

We would especially like to thank **Mr. Ganesh Prasad Bhatta**, our supervisor, for his constant advice, encouragement, and criticism during the preparation of the report. We owe him a great deal for giving us this wonderful chance to learn more. It greatly aided us in understanding the purpose of our studies.

We want to sincerely thank all of the people—friends, family, coworkers, and teachers—who helped us finish this assignment in the allotted time by offering insightful comments and suggestions on the report.

Thanking You,

Loojah Bajracharya (T.U. Exam Roll No. 26495/077)
Rayan Pokharel (T.U. Exam Roll No. 26504/077)
Reshak Maharjan (T.U. Exam Roll No. 26505/077)

Abstract

The “Book Recommendation System (HamraKitab)” is a mobile-based web application software that serves to recommend books to book enthusiasts and allow monitoring between each other's activities that might serve as recommendation for them. It offers details about books, their Reviews from previous users, and the books according to genre, with the main aim of maximizing recommendations for the books that user will like.

The system consists of user and admin interfaces with a security authentication feature and necessitates internet connectivity. The main goal of this system is recommending the best books for users. The app tries to provide the best recommendation to users according to their past activities. This project report describes the creation of a new recommendation that aims to simplify recommendation processes, promote a feeling of community for book Readers. This platform is created with React-Native, .NET API, Python, and SQL Server Management.

Keywords: authentication, HamraKitab, .NET, SQL, Python

Table of Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
DFD	Data Flow Diagram
ER	Entity Relationship
IEEE	Institute of Electrical and Electronics Engineers
MS-SQL	Microsoft Structured Query Language
UI	User Interface
VS	Visual Studio

Chapter 1: Introduction

1.1 Introduction

The book recommendation system, entitled HamraKitab, aims to extract data from datasets containing previous user's reading habits to create a system that can recommend books to users based on their personal preferences and past reading habits. Finding books that suit readers' interests and tastes might be difficult because there are millions of volumes available in various libraries. The project's goal is to design a system that may give people a more interesting and pleasurable reading experience.

The core concept of a recommendation system is to gather information from past users in order to comprehend their traits, preferences, and past choices. Book recommendation systems work on this premise: a suggestion made by one reader is likely to be well received by another. They are taught to use information collected about their interactions to comprehend the preferences, past choices, and traits of individuals and goods. These systems can offer users more interesting and relevant recommendations by employing various algorithms to be applied to subsequent users.

1.2 Problem Statement

Online users are required to enter the names, authors, or ratings of books they have read in order to receive book recommendations. Experienced book lovers might find this simple, but amateurs won't. Besides these other issues are:

- The large number of books in different categories makes it difficult for readers to select a decent book because perusing them takes time.
- Most readers lose out on hidden gems because they tend to limit themselves to well-known writers or genres.
- Finding books that are appropriate for kids or particular age groups can be difficult for parents and teachers because of the subjects or moods of the books.
- It might be challenging to evaluate a book's quality solely by looking at its title or description.

1.3 Objectives

The major objectives of books recommendation system are as follows:

- To ease user's search for books that suits their tastes and interests.
- To introduce new authors, genres or themes by offering a range of options based on their likes and preferences.
- To recommend books based on user activities and ratings in order to help readers find books that suit their interests.

1.4Scope and Limitation

1.4.1 Scopes

The scopes of HamraKitab include:

- User Registration: Allows users to register and create accounts on the platform.
- User Profile Management: Users can add an Introduction about themselves and efficiently add new books in their profile from the books they have interacted with.
- User-Friendly Interface: The system boasts a very simple design, ensuring ease of use for individuals of varying technological backgrounds.
- Friend Requests: Users can send friend requests, and make Friends to monitor new books they have read and new reviews from them.
- Ecommerce: In the future if it is liked by its users, we can take advantage of it to add an ecommerce aspect to it.

1.4.2 Limitations

The limitations of HamraKitab include:

- The system has a limited number of books.
- It has not been implemented on mobile already.

1.5 Development Methodology

The project was developed utilizing the prototyping process, which places an emphasis on building a prototype—an early, simple form of a system—to help in identifying and thinking towards system requirements. The project has been accomplished by working closely with our supervisor and interacting with a functioning mini model of the system from the beginning, which led to in improved understanding, feedback, and more specific requirements. The primary objectives were identifying the requirements and scope of the project and the quantity and length of development period. The implementation of the prototype method promoted a client-first approach to software development and enabled quick updates in response to changing requirements.

A small part of the system's requirements and project goals were found during the requirements identification and initiation phase. The system is rapidly constructed as a working sample (prototype), but with minimum abilities. Feedbacks were received from Supervisor which helped to clarify expectations, point out usability problems, and identify new ideas. The prototype undergoes revision, update, and design change throughout the refining phase accordings to the feedback. Users test new iterations of the prototype regularly while this procedure is repeated.

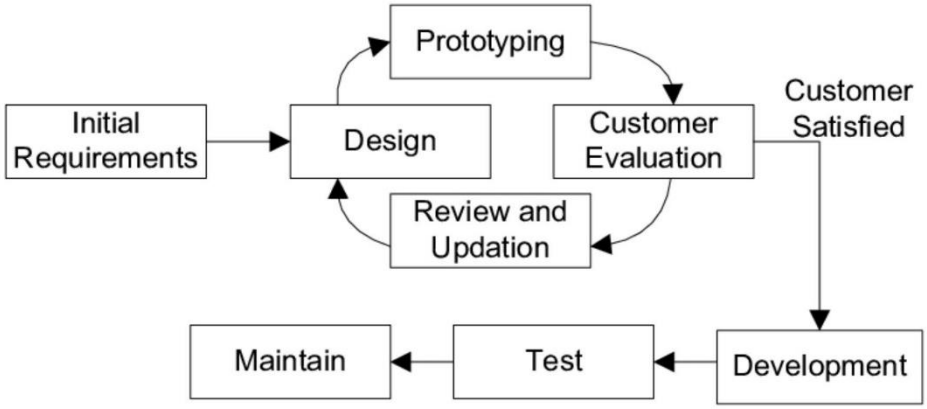


Figure 1.1: Prototyping Methodology of Software Development

1.6 Report Organization

Following the project's successful conclusion, the project documentation had been created. The report's title page, certificate page, acknowledgement, abstract, table of contents, and lists of figures, tables, and abbreviations are all included in the first part. The main report has been organized into six chapters, each of which corresponds to its specific heading and content. These chapters include:

Chapter 1: Introduction

It gives an informative overview of the project by defining elements such as project introduction, problem description, objectives, scope and limitations and methodology.

Chapter 2: Background Study and Literature Review

It covers the project's background research and reviews the body of literature, summarizing related papers, journals, and projects.

Chapter 3: System Analysis

It covers requirements and feasibility analysis with an emphasis on system analysis. A use case diagram is used to define the system's functional needs. The time required for each task in the project is shown graphically using a Gantt chart.

Chapter 4: System Design

It explores deeply into the specifics of the system, emphasizing the implementation procedure and creating the database, interface, forms, and model architecture. Additionally, it offers information about the algorithms that the system uses.

Chapter 5: Implementation and Testing

It goes over the specifics of testing and implementation, including a summary of the dependencies and tools used to put the system into place.

Chapter 6: Conclusion and Recommendations

It brings the project to a close and looks at potential ways to improve it in the future.

References that adhere to IEEE standards and Appendices with system screenshots and pertinent source code snippets are included in the report's last part.

Chapter 2: Background Study and Literature Review

2.1 Background Study

A lot of Book Recommendation Systems are there already on the internet nowadays. Although the store section has not yet been implemented, our HamraKitab is an attempt to offer users recommendations based on the books we have in stock. An integrated bookstores and recommendation system is proposed as a solution to the time-consuming stages of the separate recommendation and purchase processes. Moreover, the existing Book Recommendation system encounters issues like the excessive number of books present, and the unavailability of all those books. These sorts of problems may make it take longer for users to receive their books, which may make book enthusiasts less interested in a certain title. As a result, customers are forced to perform the unpleasant task of waiting for books for a while or physically visiting a bookstore to see whether any are available. Sadly, this can result in book enthusiasts losing their interest or preferring eBooks and hobby of Book Reading. That is why we feel there is a pressing requirement for an interconnected system featuring a centralized database that effectively allows Bookstores to save the books available, guaranteeing sufficient availability when a reader wants, promoting transparency, and ultimately minimizing disorder in the procedure.

2.2 Literature Review

Goodreads, launched in 2007, is the world's largest site for book recommendations. It helps readers discover their favorite books, track their reading, and find personalized recommendations. It uses 20 billion data points and community reviews to provide personalized recommendations. [1]

BookSloth offers personalized recommendations, curated lists, book tracking, review reviews, and in-app achievements. Users can create profiles, connect with book enthusiasts, and engage in discussions or join book clubs to connect with like-minded individuals. [2]

StoryGraph is a reading app that offers charts, graphs, personalized recommendations, and features like buddy reads, read longs, giveaways, and goals. It also allows searching by mood, pace, and custom tags, and uses mood-based recommendations for a unique reading experience. [3]

LibraryThing is an app and website that allows users to add their favorite books to their library, with the Automatic Recommendations feature comparing them to other members' libraries. This feature allows users to receive recommendations based on books with similar

subjects and tags. Users can also turn off the feature if they want to add a book they liked but not receive recommendations. The Unsuggested feature analyzes library statistics to recommend books less likely to occur in users' libraries. [4]

BookCatalogue is an open-source book cataloguing application with over 100,000 users and a 4.3-star rating. It offers features like author sorting, user-defined sort and list styles, data search, thumbnails, borrowing, goodreads synchronization, export and backup, and bookshelves.

Chapter 3: System Analysis

3.1 System Analysis

Prior to the project's commencement, a thorough work list that took into account both functional and non-functional aspects was developed. That was followed by consideration of the desired system's functionality.

3.1.1 Requirement Analysis

The requirements can be functional and non-functional.

i. Functional Requirements

The project considered the subsequent functional requirement:

- Registration and authentication of users
- For registered users, the option to log in
- Distinct user and admin interfaces
- Takes input from user to get recommended books
- Takes rating and reviews from the user

ii. Non-Functional Requirement

The project considered the subsequent non-functional requirements:

- Password should be protected and should not be accessed to unauthentic users.
- The system should provide recommendations within 5-15 seconds of a user query to ensure a responsive experience.
- Consistent, accurate and relevant recommendations based on the user’s preferences.
- Provide personalized recommendations based on user interactions, past behavior, and preferences.
- Support Cross-Platform functioning.

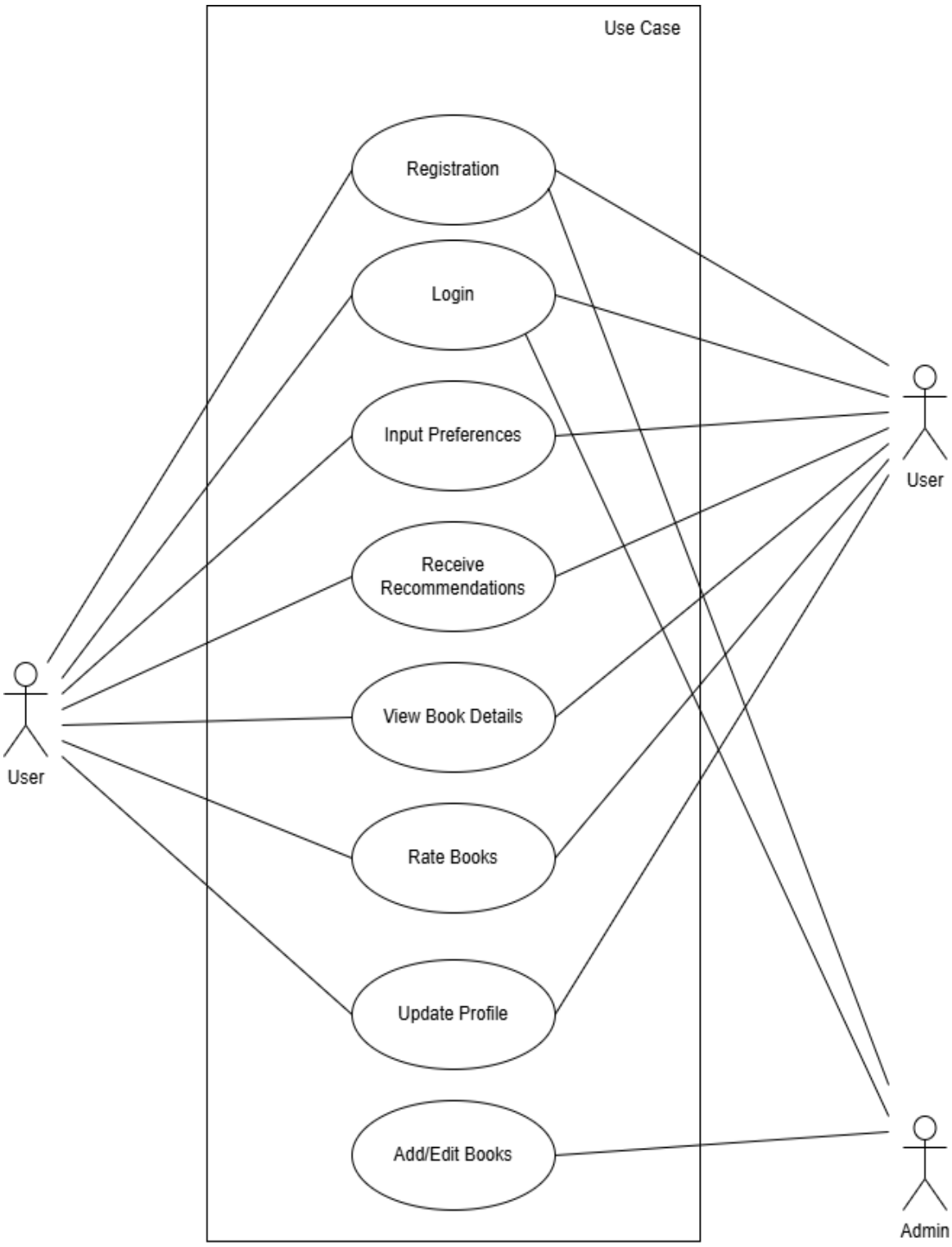


Figure 3.1: Use Case Diagram illustrating Functional Requirements

3.1.2 Feasibility Analysis

The purpose of feasibility study is to ascertain whether a project is technically, financially, operationally, and schedule-wise feasible in order to ensure its successful completion.

i. Technical Feasibility

Since the project makes use of the newest hardware and software, it is technically feasible. The project makes use of the C# and JavaScript programming language and React Native (v0.76) and .NET Framework (v8.0.2). Data is saved in MS-SQL database, which enables seamless real-time data synchronization, and is used for backend services including hosting, cloud storage, authentication, and real-time databases. Express.JS is used to create APIs inside the Node.JS (v20.17.0) runtime environment, and NodeJS is used to execute JavaScript code on the server side.

ii. Operational Feasibility

React Native and MS-SQL database streamline development and operational processes, potentially reducing time-to-market. Three people collaborated as a team to design the program, do research, and write technical articles to ensure the project was successful. The developed technology can also be utilized on a variety of iOS and Android smartphones. Users with limited experience can simply navigate the system thanks to its visually appealing and easy-to-use user interface.

iii. Economic Feasibility

The created project is highly economically viable due to its usage of free cloud-based hosting services, open-source software and frameworks (VS Code and React Native), and low hardware requirements that meet the project's present requirements.

iv. Schedule Feasibility

The estimation of the project's completion time takes into account activities like content production, backend development, UI design, programming, and security precautions. When considering the full timeline, the project was feasible to complete within the allotted time.

The following Gantt chart depicts the schedule established for the development of the system.

Task	Start Date	End Date
Project Initiation	11/10/2024	11/18/2024
System Design	11/19/2024	11/30/2024
FrontEnd Development	12/2/2024	12/19/2024
BackEnd Development	12/16/2024	1/5/2025
System Integration and Testing	1/6/2025	1/16/2025
Documentation	11/15/2024	1/17/2025

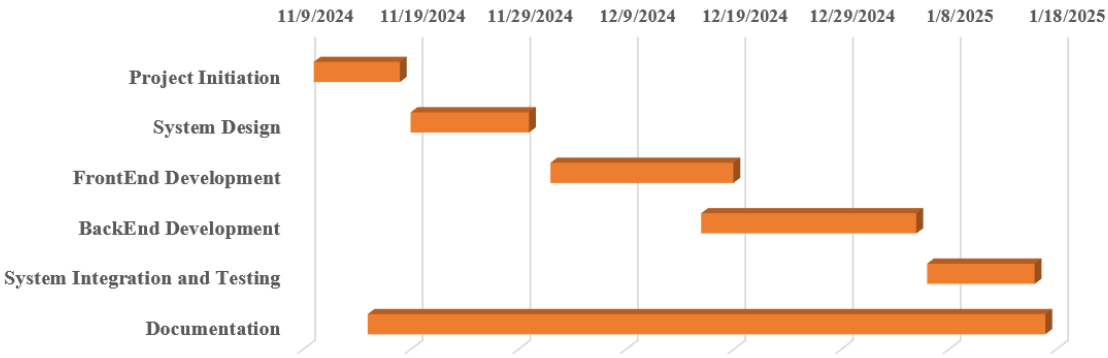


Figure 3.2: Gantt Chart

3.1.3 Data Modeling using ER Diagrams

The below Entity-Relationship (ER) diagram visually represent the relationships between entities or concepts within the developed system, providing a conceptual view of a database.

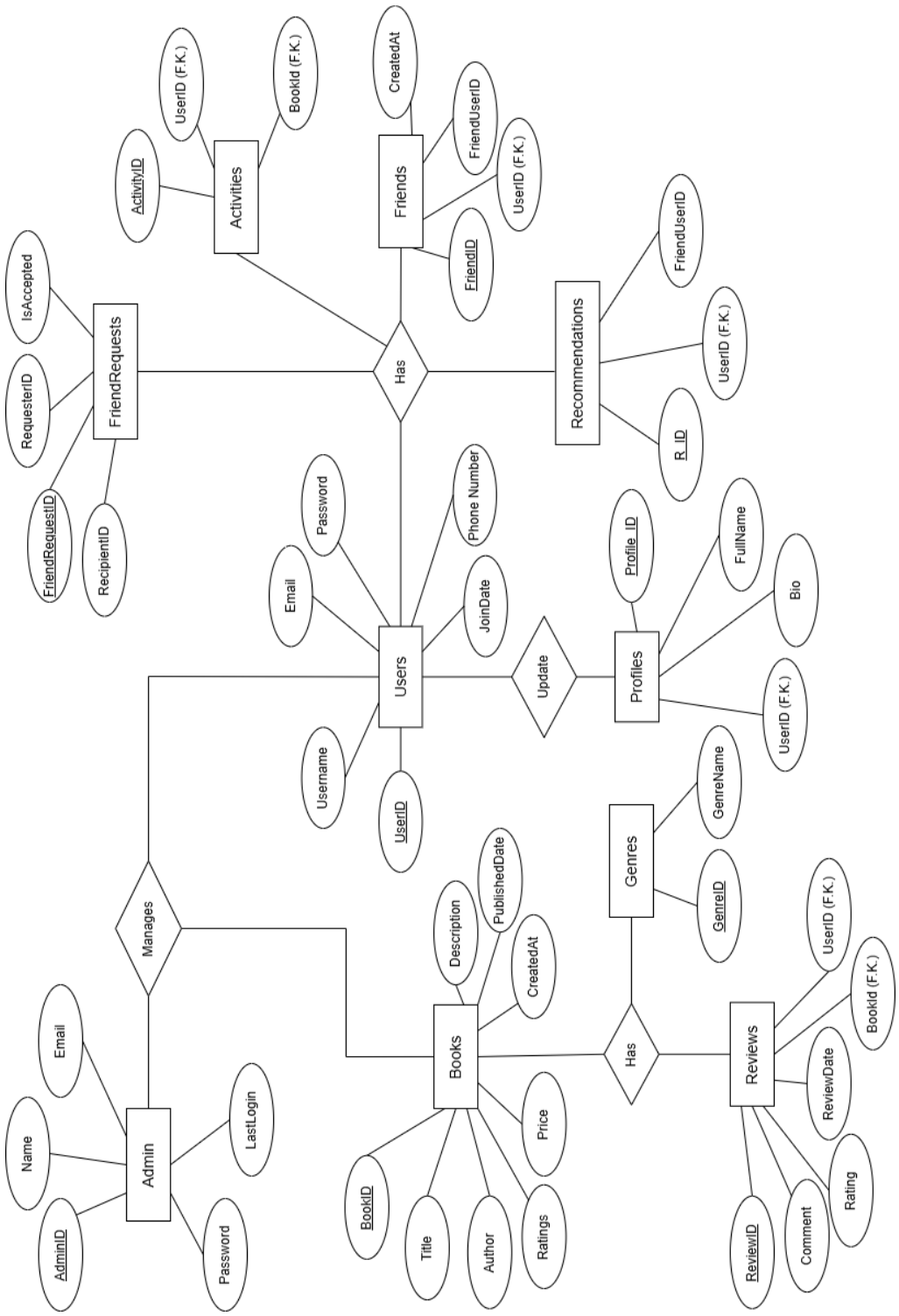


Figure 3.3: ER Diagram

3.1.4 Process Modeling using DFD

The below Data Flow Diagram (DFD) graphically depicts the flow of processes used to capture, manipulate, store, and distribute data between the system and among its components.

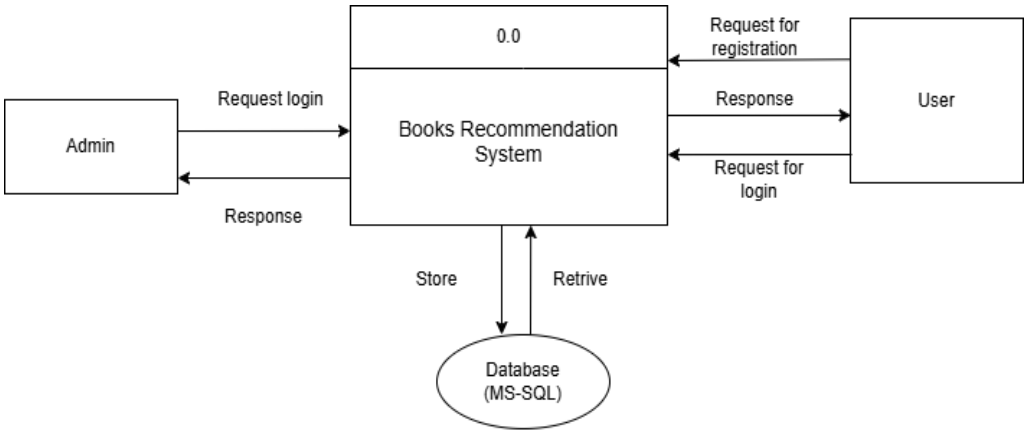


Figure 3.4: Level-0 DFD (Context Diagram)

Basically, there is a Recommendation System, where both users and admin requests to login in the recommendation system. User gets books to view, and gets recommendations, whereas admin edits the users in the recommendation system, and all the changes go through the recommendation system to database through APIs.

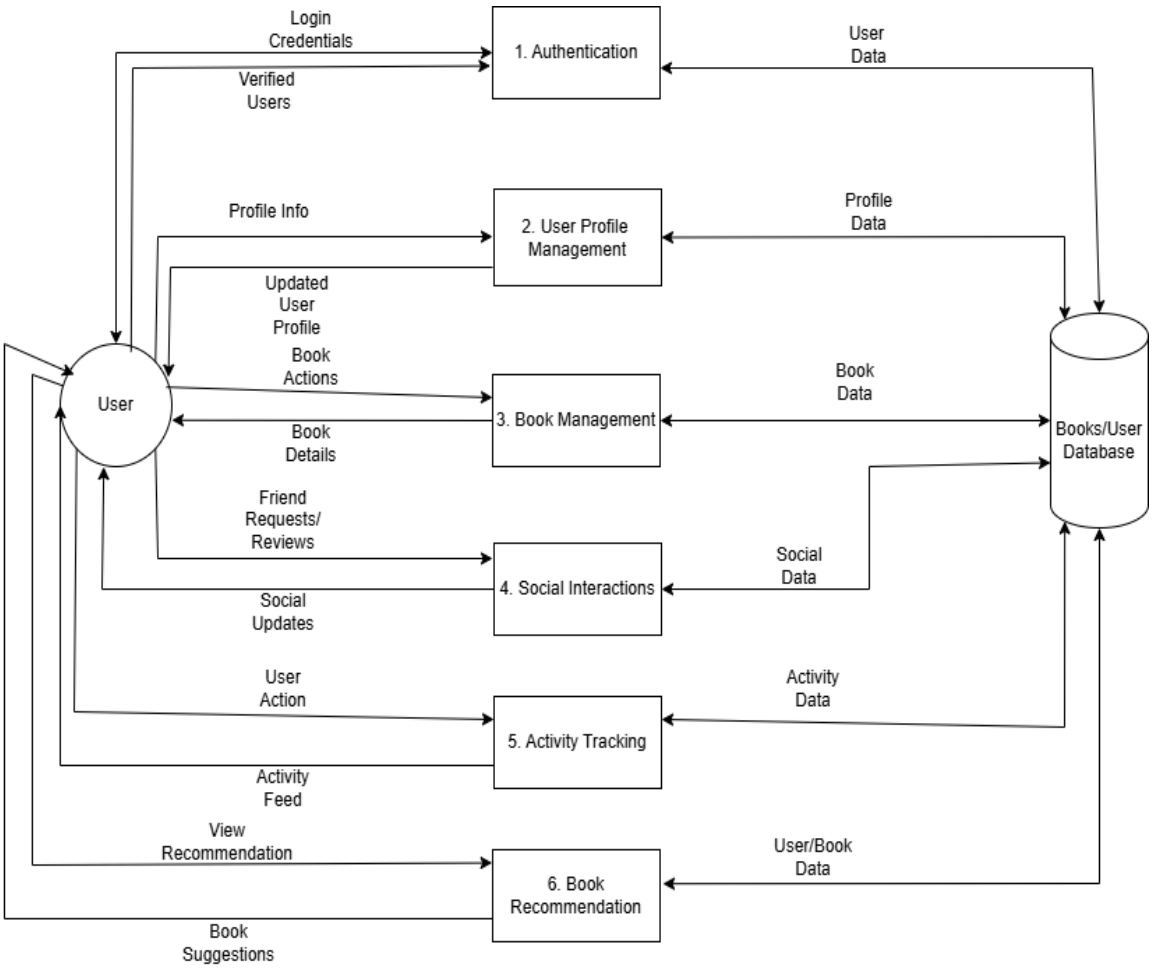


Figure 1.5: Level-1 DFD

At first user requests for login and authentication in backend checks in user table if username and passwords match and if it matches gets users their required data. The users upon login can create a profile and edit their profile from user profile management, which happens through API relating to profiles. Then they can review or add activities to the books the books and the related data goes to the database, and user can check for his reviews and activities as well. Then user can send friend requests to other users and that along with request status is updated in database and if accepted friend's activity and reviews can be seen by the user. The activities of users and their friends are tracked with database and comes in activity feed. The recommendation engine takes user activity and reviews and recommends new books which are updated under recommendation table and new recommendations are suggested to user as book suggestions.

Chapter 4: System Design

4.1 Design

Book Recommendation System's development required describing the system's divisions, parts, and structure as well as how they relate to one another and share information. System architecture, or theoretical framework, explains the structure and operation of the system. A database schema was used to illustrate the system design after the requirements analysis was completed.

4.1.1 Architectural Design

The Book Recommendation System uses the client-server framework and a three-tier architecture. Users can use any Android device to access the system, and they can use the Internet to submit queries to the application server. In response, the application server looks through the database's resources.

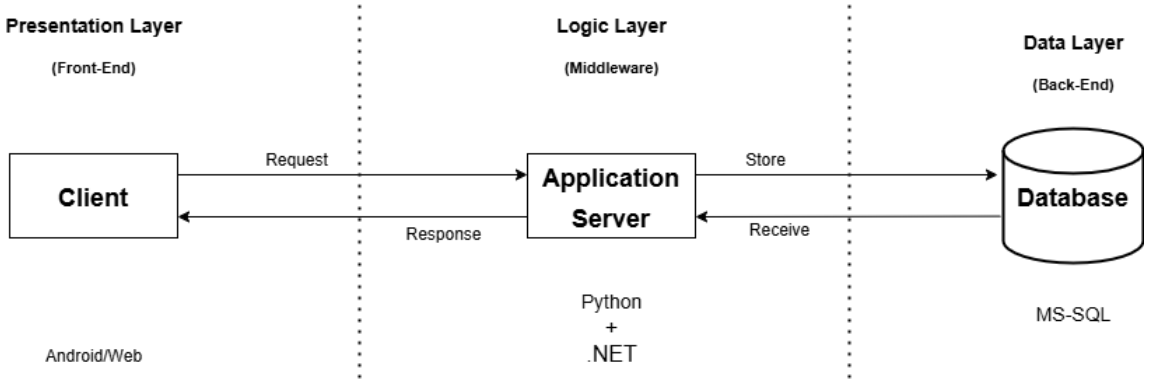


Figure 4.1: Three-Tier Architecture of Book Recommendation System

4.1.2 Database Design

The success of the Book Recommendation System heavily relies on its carefully designed database, which is needed to store user information, book details, ratings, genre and other related data. Firebase's Firestore is employed as a relational database management system.

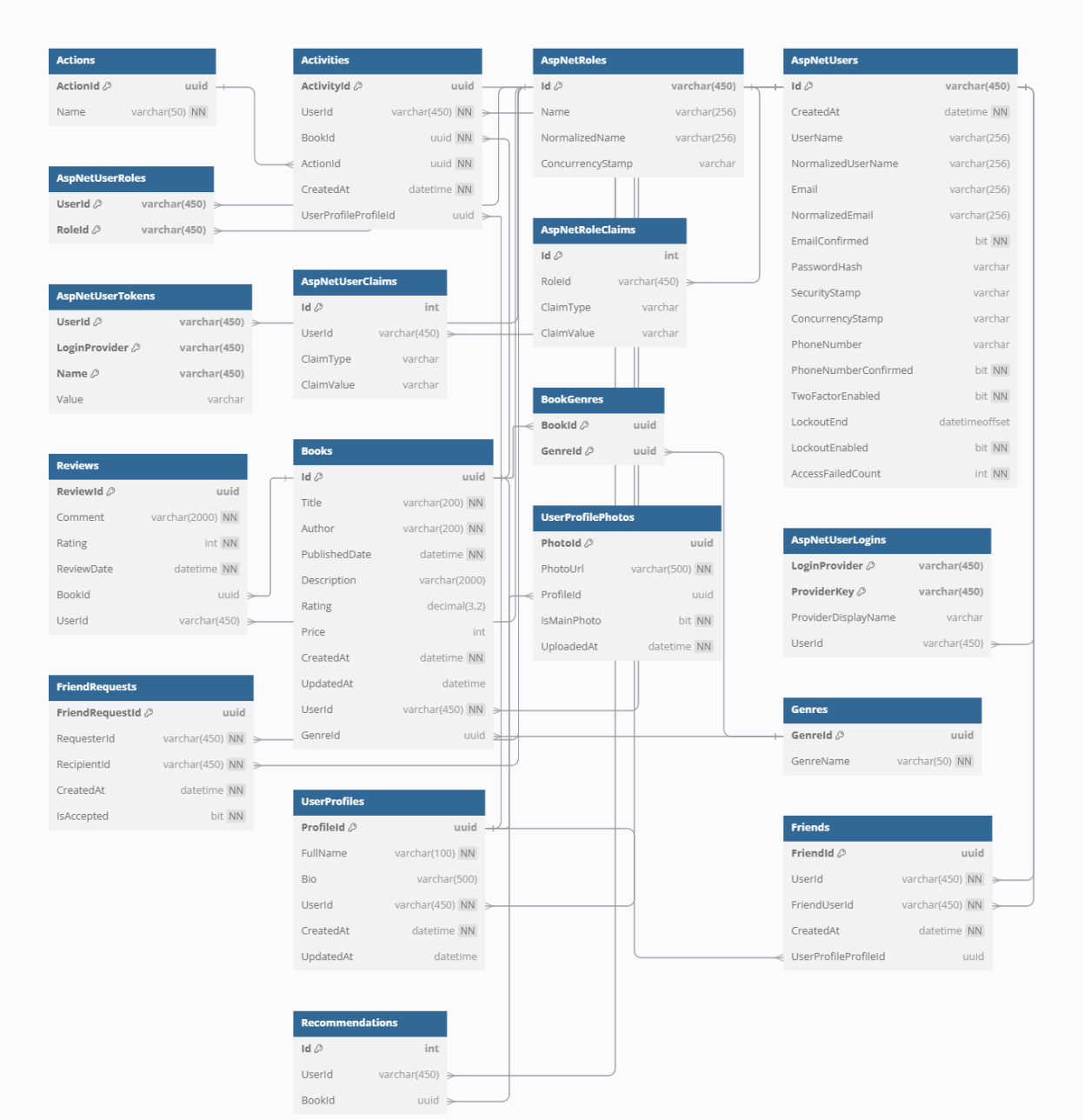


Figure 4.2: Database Schema

4.1.3. Forms and Interface Design

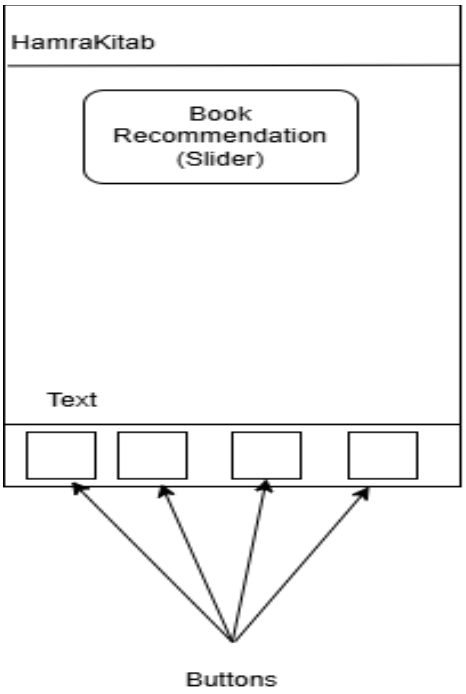


Figure 2.3: Home Page

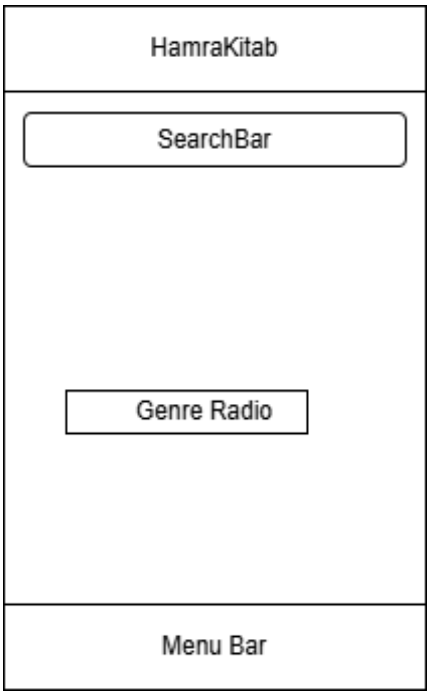


Figure 4.4: Search Result Page

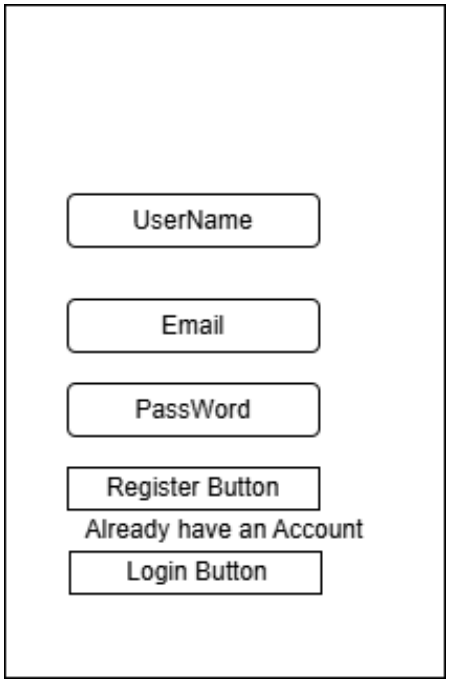


Figure 4.5: Registration Page

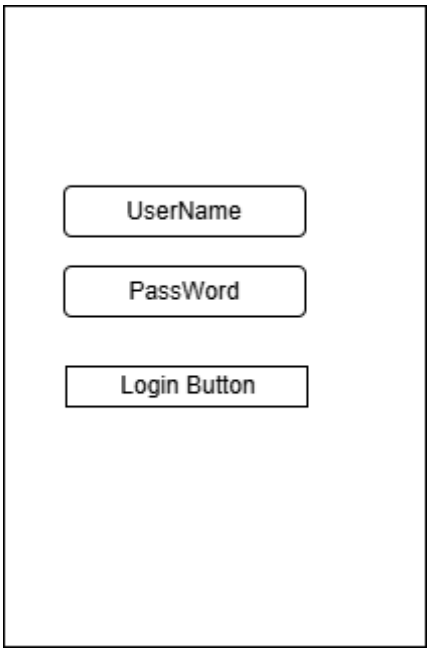


Figure 4.6: Login Page

4.2 Algorithm Details

4.2.1. Algorithms Used

i. Cosine Similarity Algorithm
   - Used for calculating similarity between users based on their rating patterns.
   - Similar to how Haversian calculates physical distance.

ii. Collaborative Filtering
   - User-based collaborative filtering.
   - Matrix Factorization with Stochastic Gradient Descent (SGD)

iii. K-Nearest Neighbors (KNN)
   - Used for finding similar users.
   - Analogous to finding nearby locations in proximity search.

4.2.2. Algorithm Details

I. Cosine Similarity

Basically, cosine similarity calculates the cosine of the angle between two vectors to determine how similar they are. These vectors stand for user ratings for book suggestions. The formula is: $\cos(\theta) = (A \bullet B) / (||A|| \, ||B||)$

Where:

- A, B are rating vectors of user 1 and user 2 respectively.
- $(A \bullet B)$ is the dot product of vector A and vector B respectively.
- $||A||$ and $||B||$ are the magnitude of the vector A and vector B respectively.

II. User-Based Collaborative Filtering

Steps followed for generating recommendations:

Step 1: Create User Rating Profile as,

User: 439

Book: e93d191e-1f2e-43be-4e70-08dcf4ddd1e6

Rating: 5

Step 2: Calculate Similarity Matrix

Find books that both users have rated in common for each pair of users (u1, u2). Calculate cosine similarity and create rating vectors.

Store in similarity matrix.

Step 3: Find Similar Users

Input: Target user ID

Process: Sort users by similarity scores

Output: Top N most similar users

Step 4: Generate Recommendations

For each unrated book:

Calculate weighted average rating

Weight = similar user's rating × similarity score

III. Prediction Formula

The predicted rating for a book is calculated as:

$pred(u,i) = \Sigma(sim(u,v) \times r(v,i)) / \Sigma|sim(u,v)|$

where:

- u is the target user.
- i is the target book to calculate predicted rating unrated by user.
- v represents similar users from top N similar users.
- sim(u,v) is the similarity between users that was calculated before.
- r(v,i) is the rating given by user v to book i.

IV. Matrix Factorization

1. Initialize Matrices

- User matrix U: [users × factors]
- Item matrix I: [items × factors]
- Bias terms: bu (user bias), bi (item bias)

2. Optimization Formula

minimize $\Sigma(r_{ui} - (\mu + b_u + b_i + p_u^T q_i))^2 + \lambda(||p_u||^2 + ||q_i||^2 + b_u^2 + bi^2)$

where:

- rui is the actual rating.
- μ is global mean.
- p_u is the latent factor vector for user u.
- q_i is the latent factor vector for item i.
- λ is regularization parameter.

3. Example Implementation Flow

- Input Data Processing

  User 439's ratings:

  Book1: 5

  Book2: 3

  Book3: 4

- Generate User Vector [5, 3, 4, 0, 0, ...] # zeros for unrated books
- Find Similar Users
    - Calculate cosine similarity with other users.
    - Select top 5 most similar users
- Generate Predictions: For each unrated book,
    - Calculate weighted average from similar users
    - Sort by predicted rating
    - Return top N recommendations

4. Comparison with Proximity Search

- Rating Similarity is calculated
- The highest Similarity are taken for users
- Users Other books Are Taken
- The books with highest similarity are Recommended

5. Performance Considerations

- Similarity Calculation
    - Pre-compute similarity matrix
    - Update periodically or incrementally
    - Store in efficient sparse format
- Scalability
    - Use dimension reduction techniques
    - Implement efficient matrix operations
    - Consider clustering for large user bases
- Cold Start Handling
    - Default recommendations for new users
    - Content-based fallback for new books
    - Hybrid approaches for sparse data

4.3. Explanatory Data Analysis

4.3.1 EDA Datasets

- GoodBooks10K
- Books.csv

  This dataset appears to contain metadata for books, detailing information about various editions of a given book. Here's a brief description of the columns:

- id: A unique id for each record in this dataset.
- book_id: The id for a specific edition of a book.
- best_book_id: Represents the most popular edition of book, mostly matching the book_id but might differ sometimes.
- work_id: A unique id for the work (a book's main content), which may contain multiple editions.
- books_count: The total number of editions available for a book.
- isbn: The International Standard Book Number (ISBN) for the book (usually for a specific edition of a book).
- isbn13: The 13-digit version of the ISBN, used to identify books.
- authors: The author/authors of the book.
- original_publication_year: The year the book was first published.
- original_title: The original title of the book, which could differ among different editions.

- Ratings.csv

  This dataset tracks the ratings given by users to different books. Here's a brief description of the columns:

  - book_id: The id for a specific book.
  - user_id: The id for the user who provided the rating.
  - rating: The rating given by the user to the book.

  This dataset is structured as a table with three columns bookid, userid and rating, where each row contains a user’s rating for a particular book. It's useful to analyze user’s reading habits, recommend books, or calculate average ratings for books.

Book Recommendation Dataset with Genres

- books.csv

  This dataset appears to contain metadata related to book genres. Here's a brief description of the columns:

  - Genre: Genre of the book like “Romance”, “Action”, etc.
  - Name: The name of the book belonging to the Genre.
  - URL: A link related to the book, which is the link to buy book.

  This dataset is likely used for categorizing books into genres, allowing users or systems to filter and search books based on their genre.

4.3.2 Datasets Study and Finding

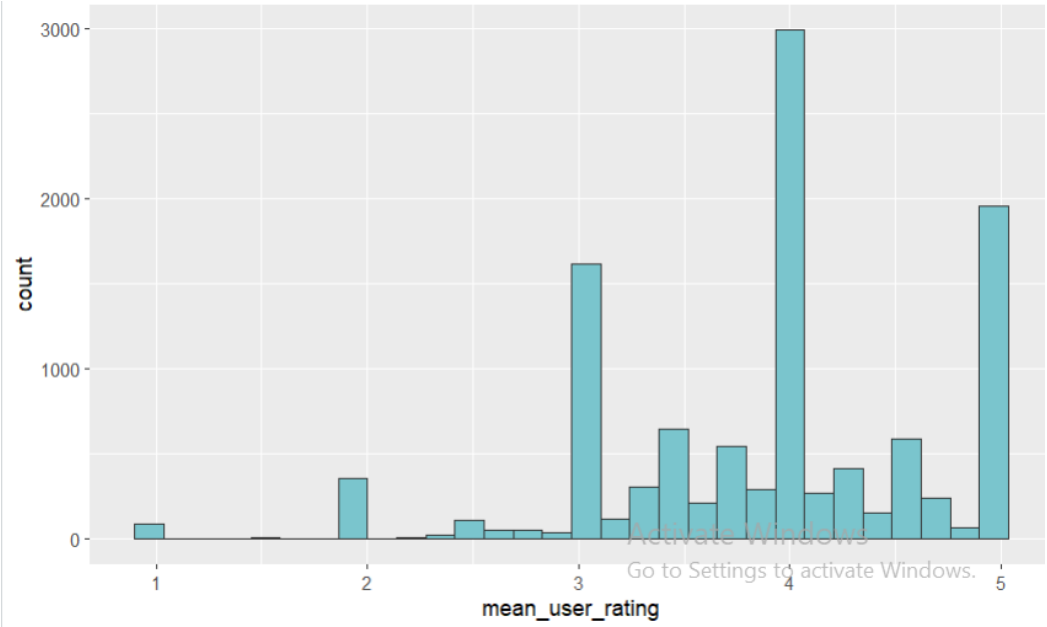


Figure 4.7: Histplot of dataset

People unconditionally acquire a tendency which they have when giving ratings. Some if they are hooked at the start directly give full rating, while some do not give full rating unless it is a masterpiece. Such tendencies can be seen in the figure above. On the end of chart there is a huge spike from users with a mean rating of 5, telling us that they really liked all books (or they only rate books if it's a masterpiece…). We can also observe there that there are very little, black-hearted downvoters rating all books with a 1. Such observation could help us during collaborative filtering later and we used that data to subtract the users mean rating from their ratings. The above figure justifies our Actions.

Chapter 5: Implementation and Testing

5.1 Implementation

5.1.1 Tools Used

- .NET(v8.0.2): Backend programming language for building API and Backend Logic.
- SQL Server Management Studio: For creating database diagram.
- Draw.io: For creating diagrams like DFD and ER diagrams.
- React-Native(v0.76): To make a Fronted UI for a User.
- MS SQL Server Management Studio 2020: A SQL database provided by Microsoft
- Microsoft Word 2013: To prepare documents
- Python(v3.10.5): To make a Recommendation Engine
- MS Excel 2013: To prepare Gantt charts
- VS Code (v20.17.0): IDE for writing and running the code
- Visual Studio 2022: IDE for writing and running the .NET code

5.1.2 Implementation Details

Some of the modules included in the project are:

- Admin Module: It allows administrators to carry out tasks like Book Management, i.e. addition, and editing. Administrators are able to manage Books, price and Their Description.
- User Module: includes important features like user sign up, log in, Book Activities, Reviews and getting Recommendations.
- Registration Module: This function enables users to sign up for an account by entering their registration information, including email, and password. It verifies the information, searches for any matching accounts, and records the new user's details in the user account database.
- Login Module: This function handles user admin and user login by verifying the provided username and password against the stored user account data. If the user’s credentials are valid, they are granted access to their account.

5.1.3 Algorithm Implementations

Cosine Similarity

At first data is taken as:

Book_id | Title | Genre

This data is then converted into genre_matrix.npz where Genre Vectors are like:

Book 1: "Harry Potter"

Genres: Fantasy, Science Fiction

Genre Vector: [0, 0, 1, 1] (0 for Action, 0 for Romance, 1 for Science Fiction, 1 for Fantasy)

Then Books with Activities of Users are taken.

These Books Too have a Genre Vector Calculated for them.

At first we took data from the Datasets which we had already cleaned. We got the data in the form of User/Item matrix which had different Users Ratings for different books.

Then we Calculated $\cos(\theta)$ for calculating Similarity between all the Books Using Cosine Similarity Algorithm, and the highest $\cos(\theta)$ were taken as Recommendation.

Collaborative Algorithm:

In a collaborative recommendation system, Matrix Factorization starts by decomposing a large user-item interaction matrix into smaller matrices that represent hidden factors for users and items, which helps predict missing ratings.

Cosine Similarity is then used to measure how similar users or items are by comparing their rating patterns, helping identify users with similar tastes or items with similar characteristics.

K-Nearest Neighbors (KNN) finds the most similar users or items based on their ratings or behaviors and recommends items that the most similar users have liked.

Together, these algorithms work by uncovering hidden patterns in user preferences, finding connections between users or items, and suggesting items based on the preferences of similar users, resulting in personalized recommendations.

5.2 Testing

A crucial stage in the deployment of any technology is testing. In order to verify books recommendation system's usability, performance, and usefulness, various tests were conducted on it.

5.2.1 Test Cases for Unit Testing

Unit testing involves testing individual testable code units, usually at the function or method level, to make sure they behave correctly.

The following test scenarios were employed for conducting unit testing.

Table 5.1: Test Cases for User Registration

Test ID	Test Scenario	Test Data	Expected Result	Observed Result	Test Status
1	User Registration with valid details	Email:user@example.com Password:abc	Registration Failed Display Error Message	Same as Expected	Success
2	User Registration with invalid details	Email:user@example.com Password:Pass@123	Registered Successfully User Redirects to Login Page	Same as Expected	Success

Table 5.2: Test Cases for User Login

Test ID	Test Scenario	Test Data	Expected Result	Observed Result	Test Status
1	User Login with valid input	Email:user@example.com Password:Pass@123	Login Passed, User Goes to Recommend-ation Page	Same as Expected Result	Success
2	User Login with invalid input	Email:user@example.com Password:abc	Log In Failed Display Error Message	Same as Expected Result	Success

Table 5.3: Test Cases for Searching

Test ID	Test Scenario	Test Data	Expected Result	Observed Result	Test Status
1	Searching a Book with title	Harry	All Books Containing Harry in title are Displayed	All Books Containing Harry in title are Displayed	Success
2	Searching a Book with genre	Genre: Horror	All Books of Horror Genre are Presented	All Books of Horror Genre were Presented	Success

5.2.2. Test Cases for Integration Testing

After successful unit testing, integration testing was conducted to determine the collective functionality of each component unit as a unified project. The interaction between different components within the system were examined.

Table 5.4: Test Cases for integration testing

Test ID	Test Scenario	Test Data	Expected Result	Observed Result	Test Status
1	Book Reviews added	Review message: Nice Book Rating:4	Review is Added successfully	Review was added Successfully	success
2	Book Activities Added	Want to Read button was clicked	Book is there in Watchlist	Book is there in Watchlist	success
3	Friend Request	Friend Request was Sent To User 2	In User2 account, there should be Friend Request	In User2 account, there was Friend Request	success
4	Friend Confirm	Friend Request Accepted	New User in Friend Screen	There was New Activity in Friend Screen	success
5	Recommendation	Recommendation Screen	There are Book Recommendations	There were Book Recommendations	success

5.2.3 Test Cases for System Testing

During system testing, a detailed assessment of the complete operational process of the system was carried out to detect errors and bugs, guaranteeing the proper functioning of the system as a whole. The entire system was developed and tested on a different machine than the development environment to guarantee satisfactory performance and efficiency. Hardware and software components were combined and tested as a whole.

The following test scenarios were employed for system testing.

Table 5.5: Test Case for Device Compatibility

Test ID	Test Scenario	Test Data	Expected Result	Observed Result	Test Status
1	Tried the App in Mobile	Npm start android	It should show UI in mobile	It showed UI in Mobile	Success

5.3 Result Analysis

Analyzing the system's results consisted of going through test results, documenting issues, and making informed decisions. The method included several crucial processes, including assessing results, identifying mistakes, looking into the causes of problems, assessing impacts, and disseminating findings. Test-related problems were brought back to the development phase to make sure the system was prepared for practical application. Testing for perfect compatibility was crucial, even though individual system components were error-free.

```
Performance Metrics:
Accuracy: 0.9862 - Proportion of correct predictions (TP + TN) / Total
Precision: 0.8000 - Proportion of correct recommendations (TP / (TP + FP))
Recall: 0.5714 - Proportion of actual books recommended (TP / (TP + FN))
F1 Score: 0.6667 - Harmonic mean of precision and recall
```

Figure 5.1: F1 Score

Chapter 6: Conclusion and Future Recommendation

6.1 Conclusion

In summary, our book recommendation system, entitled HamraKitab, is designed to help users discover new, interesting books that fit their interests and preferences. With the help of React Native, .NET Framework and Firebase technologies, a vast library of genres and titles, and personalized algorithms such as Cosine similarity, proximity search, etc., the app ensures a personalized experience that encourages discovery and a stronger connection with literature. This app serves as your trustworthy reading companion, providing an extensive range of recommendations and making it easy for you to choose your next favorite book, whether you're looking for an engaging novel, an insightful work of non-fiction, or something quite different. By using iterative development cycles to make sure the system met its objectives, the Agile methodology proved essential in adapting to changing needs. The proper operation was confirmed by extensive testing, which included examining the system as a whole as well as each component independently. The project was successfully completed by making sure it complied with the requirements within the allotted time.

6.2 Future Recommendation

- The Book retailers can be assigned as an admin for books inventory, leading this recommendation system also to function as an e-commerce application.
- Availability of books in online readable format.
- Enhanced interaction between users, allowing them to communicate through the app and enabling to share the books they possesses.
- More user friendly UI design and features according to user needs.

References

[1] "goodreads," [Online]. Available: https://www.goodreads.com. [Accessed August 2024].

[2] "booksloth," [Online]. Available: https://www.booksloth.com/. [Accessed August 2024].

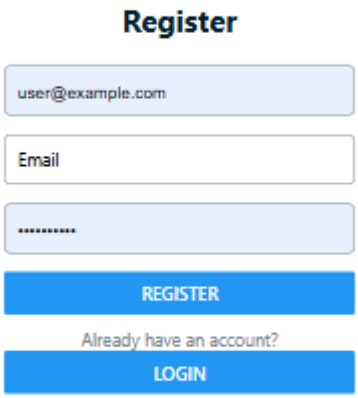
[3] "thestorygraph," The StoryGraph, [Online]. Available: https://www.thestorygraph.com/. [Accessed August 2024].

[4] "LibraryThing," [Online]. Available: https://www.librarything.com/. [Accessed August 2024].

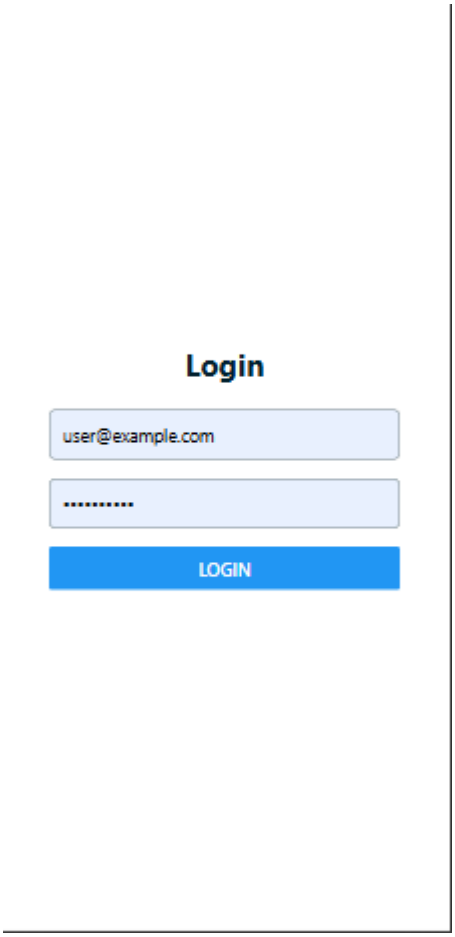
Appendices

Screenshots of the Application

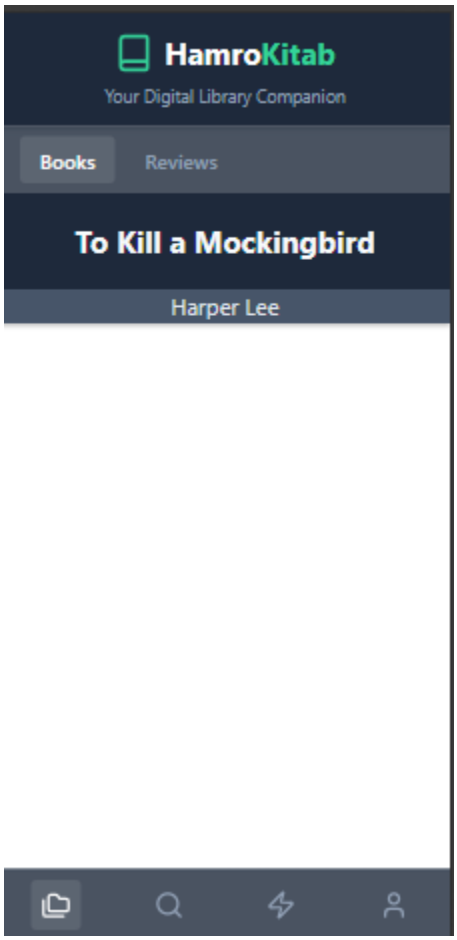
a. Register Page



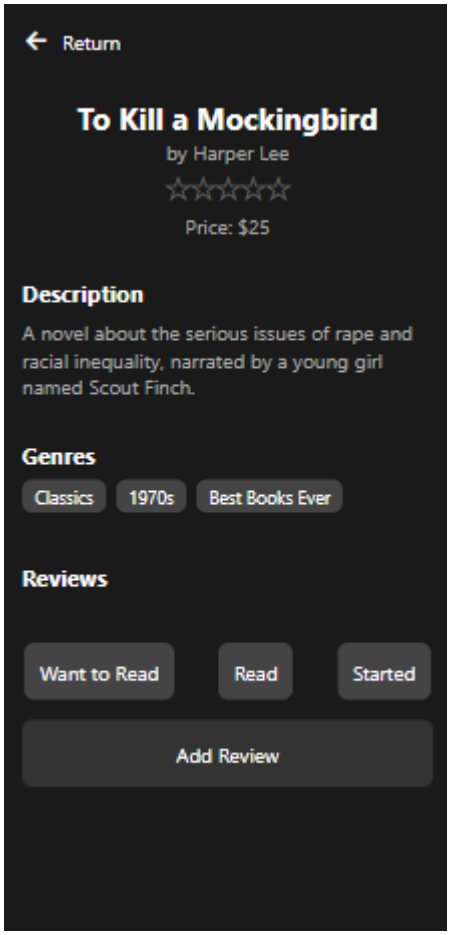
b. Login Page



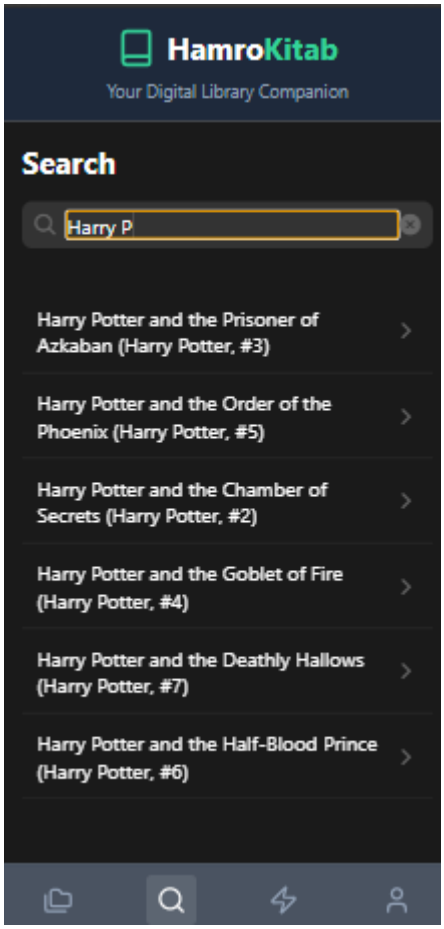
c. Recommendation Page



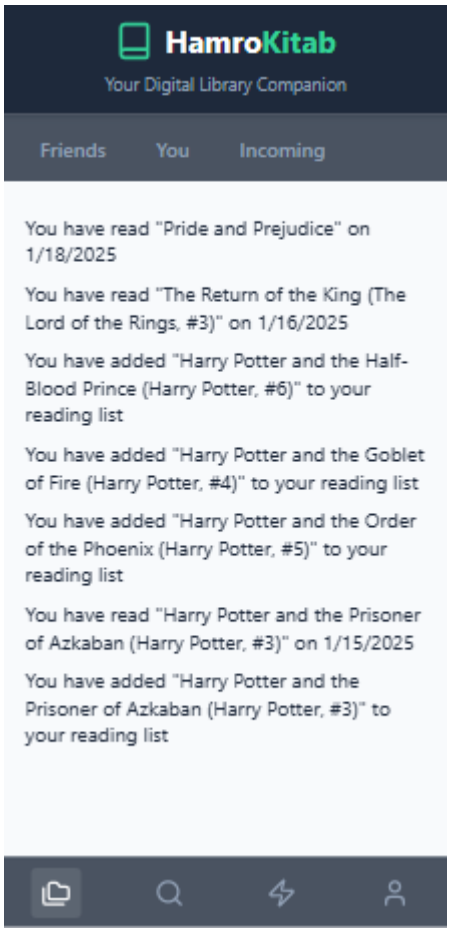
d. Book Details



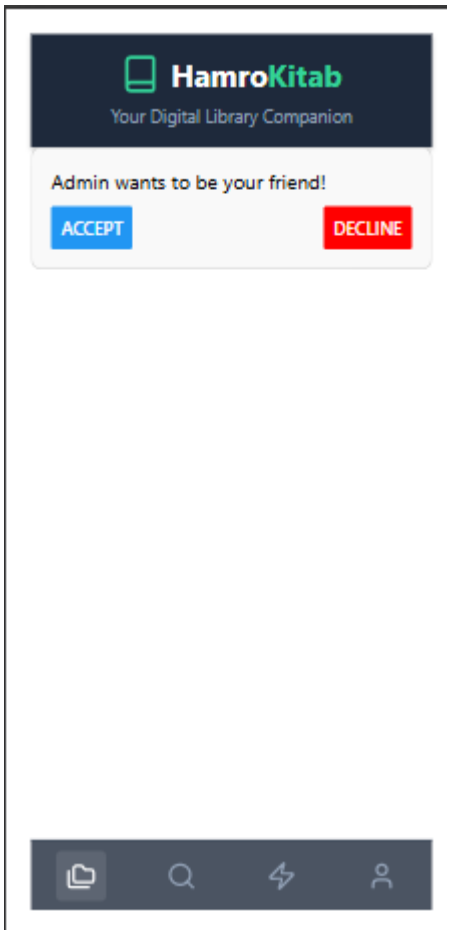
e. Search Page



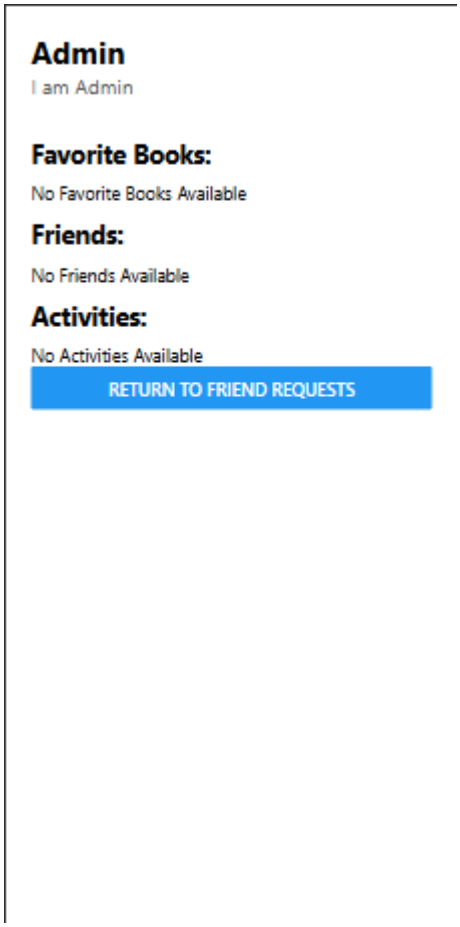
f. Activity List



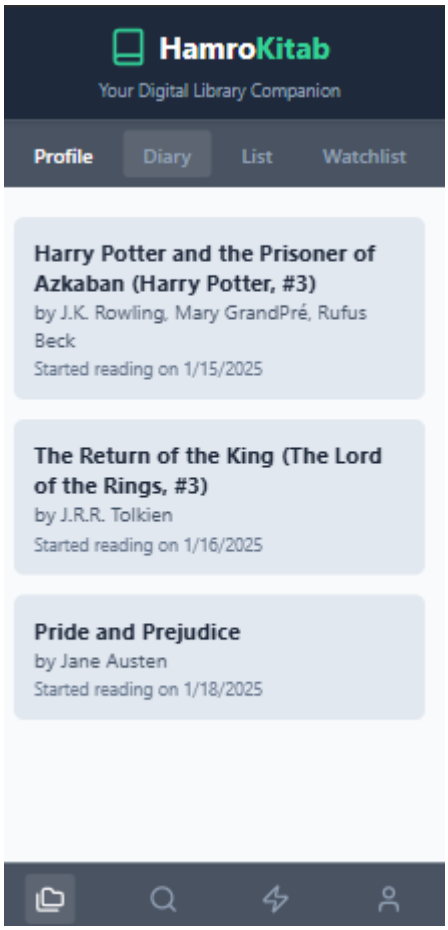
g. Friend Request Page



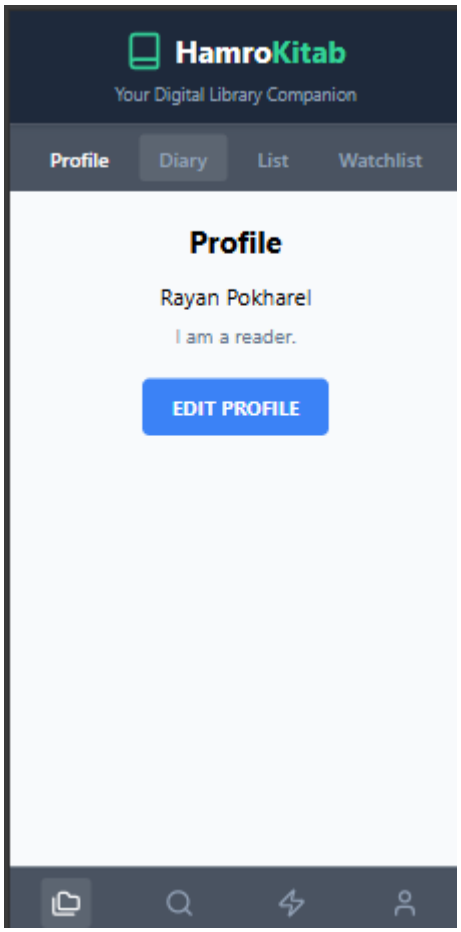
h. Request Sender Profile



i. Diary Page



j. Personal Profile



Algorithm

```
# Main Recommendation Algorithm Structure
 Function calculateUserSimilarity(user1_ratings, user2_ratings):
    # Calculate similarity between two users based on their ratings
    commonBooks = findCommonBooks(user1_ratings, user2_ratings)
    if commonBooks is empty:
        return 0
# Create rating vectors for common books
    vector1 = createRatingVector(user1_ratings, commonBooks)
    vector2 = createRatingVector(user2_ratings, commonBooks)
    # Cosine Similarity is Calculated here
    similarity = calculateCosineSimilarity(vector1, vector2)
    return similarity
Function findSimilarUsers(targetUserId, allRatings, numberOfNeighbors):
    # Find users similar to target user
    targetUserRatings = getUserRatings(targetUserId, allRatings)
    similarityScores = []
    for each userId in allRatings:
        if userId != targetUserId:
            userRatings = getUserRatings(userId, allRatings)
            similarity = calculateUserSimilarity(targetUserRatings, userRatings)
            similarityScores.append((userId, similarity))
    # Sort and return top N similar users
    return sortAndFilterTopUsers(similarityScores, numberOfNeighbors)

Function generateRecommendations(targetUserId, allRatings, numberOfRecommendations):
    # Main recommendation function
        # Step 1: Find similar users
    similarUsers = findSimilarUsers(targetUserId, allRatings, numberOfNeighbors=5)
        # Step 2: Get books rated by target user
    targetUserBooks = getBooksByUser(targetUserId, allRatings)
# Step 3: Calculate predicted ratings for unrated books
    predictions = []
    for each book in allBooks:
        if book not in targetUserBooks:
            predictedRating = calculatePredictedRating(book, similarUsers, allRatings)
            predictions.append((book, predictedRating))
        # Step 4: Sort and filter top recommendations
    return sortAndFilterTopBooks(predictions, numberOfRecommendations)

Function calculatePredictedRating(bookId, similarUsers, allRatings):
    # Calculate predicted rating for a book based on similar users
    weightedSum = 0
    similaritySum = 0

    for each (userId, similarity) in similarUsers:
        userRating = getRatingForBook(userId, bookId, allRatings)
        if userRating exists:
            weightedSum += similarity * userRating
            similaritySum += similarity
        if similaritySum == 0:
             return 0
```

```
    return weightedSum / similaritySum
Function getBooksByUser(userId, allRatings):
    # Get all books rated by a specific user
    return list of books rated by userId

Function getRatingForBook(userId, bookId, allRatings):
    # Get rating given by a user for a specific book
    return rating if exists else null

Function sortAndFilterTopBooks(predictions, numberOfRecommendations):
    # Sort predictions by rating and return top N
    sortedPredictions = sort predictions by rating in descending order
    return first numberOfRecommendations items from sortedPredictions

# Supporting Functions
 Function calculateCosineSimilarity(vector1, vector2):
    # Calculate cosine similarity between vector 1 and vector 2
    dotProduct = calculateDotProduct(vector1, vector2)
    magnitude1 = calculateMagnitude(vector1)
    magnitude2 = calculateMagnitude(vector2)
        if magnitude1 == 0 or magnitude2 == 0:
        return 0
    return dotProduct / (magnitude1 * magnitude2)

Function createRatingVector(ratings, bookList):
    # Create a vector of ratings for given books
    vector = []
    for each book in bookList:
        rating = ratings.get(book, 0)
        vector.append(rating)
    return vector

Function findCommonBooks(ratings1, ratings2):
    # Find books rated by both users
    books1 = set of books in ratings1
    books2 = set of books in ratings2
    return intersection of books1 and books2
# Usage Example:
ratings_data = [
    {
        "book_id": "e93d191e-1f2e-43be-4e70-08dcf4ddd1e6",
        "user_id": "439",
        "rating": 5
    }
]
recommendations = generateRecommendations(
    targetUserId="439",
    allRatings=ratings_data,
    numberOfRecommendations=5
)
```

Code implementation of Recommendation using Cosine Similarity and Collaborative

```
import numpy as np
from typing import Dict, List, Tuple
import pandas as pd
from scipy.sparse import csr_matrix
from sklearn.metrics.pairwise import cosine_similarity

class CosineSimilarityRecommender:
    def __init__(self):
        self.user_item_matrix = None
        self.user_similarity = None
        self.users = None
        self.items = None

    def fit(self, ratings_df: pd.DataFrame):
            Args:
            ratings_df: DataFrame with columns [user_id, item_id, rating]
        # Create user-item matrix
              self.user_item_matrix = pd.pivot_table(
            ratings_df,
            values='rating',
            index='user_id',
            columns='item_id',
            fill_value=0
        )
        # Calculate user similarity matrix using cosine similarity
        self.user_similarity = cosine_similarity(self.user_item_matrix)
        # Store users and items for later use
        self.users = self.user_item_matrix.index
        self.items = self.user_item_matrix.columns
        def get_recommendations(self, user_id: str, n_recommendations: int = 5) -> List[Tuple[str, float]]:

# Get recommendations for a user
            Args:
            user_id: ID of target user
            n_recommendations:          #Number of recommendations to return
            Returns:
            List of (item_id, predicted_rating) tuples
              if user_id not in self.users:
            return []
              # Get user's index in similarity matrix
               user_idx = self.users.get_loc(user_id)
            # Get user's ratings and similarity scores
              user_ratings = self.user_item_matrix.loc[user_id]
              user_similarities = self.user_similarity[user_idx]
               # Calculate predicted ratings
              weighted_sums = np.zeros_like(user_ratings)
              similarity_sums = np.zeros_like(user_ratings)
            for other_idx, similarity in enumerate(user_similarities):
            if other_idx != user_idx:
                other_ratings = self.user_item_matrix.iloc[other_idx]
                weighted_sums += similarity * other_ratings
                similarity_sums += np.abs(similarity) * (other_ratings != 0)
```

```
                # Avoid division by zero
            similarity_sums[similarity_sums == 0] = 1
            predicted_ratings = weighted_sums / similarity_sums
                    # Get unrated items
            unrated_items = self.items[user_ratings == 0]

            # Sort predictions and return top n
            predictions = [
            (item, predicted_ratings[idx])
            for idx, item in enumerate(unrated_items)
                                ]
            predictions.sort(key=lambda x: x[1], reverse=True)

        return predictions[:n_recommendations]
 class CollaborativeFilteringRecommender:
    def __init__(self):
        self.user_factors = None
        self.item_factors = None
        self.global_mean = None
        self.user_biases = None
        self.item_biases = None
        self.user_map = None
        self.item_map = None

    def fit(self, ratings_df: pd.DataFrame, n_factors: int = 20, n_epochs: int = 20,
            learning_rate: float = 0.01, regularization: float = 0.1):
            # Fit the collaborative filtering model using matrix factorization
        Args:
            ratings_df: DataFrame with columns [user_id, item_id, rating]
            n_factors: Number of latent factors
            n_epochs: Number of training epochs
            learning_rate: Learning rate for gradient descent
            regularization: Regularization parameter
             # Create user and item mappings
self.user_map = {user: idx for idx, user in enumerate(ratings_df['user_id'].unique())}
self.item_map = {item: idx for idx, item in enumerate(ratings_df['item_id'].unique())}
              # Convert ratings to matrix format
             user_indices = [self.user_map[user] for user in ratings_df['user_id']]
            item_indices = [self.item_map[item] for item in ratings_df['item_id']]
            ratings = ratings_df['rating'].values
           n_users = len(self.user_map)
            n_items = len(self.item_map)

        # Initialize parameters
        self.global_mean = np.mean(ratings)
        self.user_factors = np.random.normal(0, 0.1, (n_users, n_factors))
        self.item_factors = np.random.normal(0, 0.1, (n_items, n_factors))
        self.user_biases = np.zeros(n_users)
        self.item_biases = np.zeros(n_items)
```

```
# Train model using SGD
        for epoch in range(n_epochs):
            for i in range(len(ratings)):
                user_idx = user_indices[i]
                item_idx = item_indices[i]
                rating = ratings[i]

                # Compute prediction
                pred = (self.global_mean +
                        self.user_biases[user_idx] +
                        self.item_biases[item_idx] +
                        np.dot(self.user_factors[user_idx], self.item_factors[item_idx]))
                # Compute error
                error = rating - pred

                # Update parameters
                self.user_biases[user_idx] += learning_rate * (error - regularization * self.user_biases[user_idx])
                self.item_biases[item_idx] += learning_rate * (error - regularization * self.item_biases[item_idx])

                # Update latent factors
                user_factor = self.user_factors[user_idx]
                item_factor = self.item_factors[item_idx]

self.user_factors[user_idx] += learning_rate * (error * item_factor - regularization * user_factor)
self.item_factors[item_idx] += learning_rate * (error * user_factor - regularization * item_factor)

    def predict(self, user_id: str, item_id: str) -> float:
        if user_id not in self.user_map or item_id not in self.item_map:
            return self.global_mean

        user_idx = self.user_map[user_id]
        item_idx = self.item_map[item_id]

        return (self.global_mean +
                self.user_biases[user_idx] +
                self.item_biases[item_idx] +
                np.dot(self.user_factors[user_idx], self.item_factors[item_idx]))

    def get_recommendations(self, user_id: str, n_recommendations: int = 5) -> List[Tuple[str, float]]:
        #Get recommendations for a user
                Args:
            user_id: ID of target user
            n_recommendations: Number of recommendations to return
                    Returns:
            List of (item_id, predicted_rating) tuples
                if user_id not in self.user_map:
                        return []
        # Predict ratings for all items
        predictions = []
        for item_id in self.item_map:
            predicted_rating = self.predict(user_id, item_id)
            predictions.append((item_id, predicted_rating))
```

```
 # Sort and return top n
        predictions.sort(key=lambda x: x[1], reverse=True)
        return predictions[:n_recommendations]

# Example usage:
def main():
    # Sample data
    ratings_data = [
        {"user_id": "user1", "item_id": "item1", "rating": 5},
        {"user_id": "user1", "item_id": "item2", "rating": 3},
        {"user_id": "user2", "item_id": "item1", "rating": 4},
        # Add more ratings...
    ]
     # Convert to DataFrame
    ratings_df = pd.DataFrame(ratings_data)

    # Cosine Similarity based recommendations
    cosine_recommender = CosineSimilarityRecommender()
    cosine_recommender.fit(ratings_df)
    cosine_recommendations = cosine_recommender.get_recommendations("user1")

    # Collaborative Filtering recommendations
    collab_recommender = CollaborativeFilteringRecommender()
    collab_recommender.fit(ratings_df)
    collab_recommendations = collab_recommender.get_recommendations("user1")
    print("Cosine Similarity Recommendations:", cosine_recommendations)
    print("Collaborative Filtering Recommendations:", collab_recommendations)

if __name__ == "__main__":
main()
```