

HELWAN UNIVERSITY
Faculty of Computers and Artificial Intelligence
Information System Department

كتاب Tabook

A graduation project dissertation by:

- [MAHMOUD BADAWY MOHAMED EWIS (20210857)]
- [MAHMOUD TALAAT MOHAMED MOHAMED (20210865)]
- [MOHAMED AMR ABDELSAYED (20210821)]
- [RANA MOHAMED ZAKARIA (20210319)]
- [SHIMAA NABI ALI (20210475)]
- [FATMA AYMAN RASHAD (20210664)]

Submitted in partial fulfilment of the requirements for the degree of Bachelor of Science
in Computers & Artificial Intelligence, at the Information System Department, the
Faculty of Computers & Artificial Intelligence, Helwan University

Supervised by:

[Dr. Mai El Defrawi]

June 2025

Acknowledgements

First and foremost, we would like to express our deepest appreciation and sincere gratitude to our supervisor,

Dr. Mai El Defrawi, *for her continuous support and guidance throughout the entire project.*

She consistently provided us with the resources, direction, and encouragement we needed from the very beginning, and her contributions were instrumental to the successful completion of our work

We would also like to extend our sincere thanks to our faculty and all the professors who have taught and supported us throughout our academic journey at the faculty.

We truly appreciate their time, effort, and dedication, which have played a significant role in shaping our knowledge and skills

Finally, we would like to express our heartfelt thanks to our families for unwavering support, patience, and encouragement throughout this journey. Their belief in us has been a constant source of motivation

Table of contents

Abstract	6
-----------------------	----------

Chapter 1: Introduction

1.1 Overview	7
1.2 Objectives	8
1.3 Purpose	9
1.4 scope	10
1.5 General constrains	111

Chapter 2: Project “Planning and analysis”

2.1 Project planning

2.1.1 Feasibility Study	12
2.1.2 Estimated Cost	14
2.1.3 Gantt Chart.....	15

2.2 Analysis and Limitation of existing system.....	17
---	----

2.3 Need for the new system	18
-----------------------------------	----

2.4 Analysis of the new system

2.4.1 User requirements	19
-------------------------------	----

2.4.2 System Requirements.....	22
2.4.3 Domain Requirements	23
2.4.4 Functional Requirements.....	24
2.4.5 Non- Functional Requirements.....	25
2.5 Advantages of the new system.....	26
2.6 Risk and Risk Managements.....	27

Chapter 3: Software Design

3.1 Design of database (ERD or Class) Diagram.....	29
3.2 Use case diagram.....	30
3.3 sequence diagram.....	60
3.4 activity diagram.....	67

Chapter 4: Implementation

4.1 Software architecture.....	74
4.2 Pseudocode, Flowchart workflow	87

Chapter 5: Testing

5.1 Unit Testing.....	91
5.2 integration Testing.....	92

5.3 Additional Testing.....	92
-----------------------------	----

Chapter 6: Results and Discussion

6.1 Results.....	99
6.1.1 Expected result.....	102
6.1.2 Actual results.....	103
6.2 Discussion.....	103

Chapter 7: Conclusion..... 105

Chapter 8: Future work..... 106

Abstract

This report presents the development of "**Ktabook**", a web-based eBook reading and social platform designed to enhance the digital reading experience. The system allows users to read books online and interact through posts, comments, and private messaging. It supports various interactive reading features such as bookmarking, highlighting, and note-taking to increase user engagement.

The application was developed using React.js for the frontend and ASP.NET Core for the backend, ensuring a responsive and scalable architecture. Users have access to a personal dashboard that tracks their reading activity and social participation, while administrators can manage users, books, and communities through a dedicated admin panel.

Chapter 1: Introduction

1.1 Overview:

This project aims to design a smart digital library platform using modern web technologies to enhance the reading experience and promote a culture of knowledge and learning. The platform is divided into two main sections: one for regular users (readers) and another for administrators. Users can browse a wide range of books, receive intelligent recommendations based on their reading activity, and interact with the community through posting, commenting, and liking content, similar to social media platforms

Each user has a personal profile and dashboard that tracks their reading habits and engagement over time using graphs and activity summaries. The recommendation system suggests similar books based on the user's interest. On the other side, the admin panel allows administrators to monitor posts, manage the book database, and track registration statistics through a comprehensive dashboard. The platform aims to simulate the library experience digitally, providing a more accessible, interactive, and motivating environment for readers

1.2 Objectives:

- 1. Develop an interactive digital library platform** that allows users to browse, read, and engage with books in an accessible and user-friendly interface.
- 2. Implement a recommendation system** to suggest similar books based on user interests and activity, enhancing personalization.
- 3. Enable social interaction among users** through posting, commenting, and sharing reading experiences, creating a community-driven environment.
- 4. Provide a personalized user dashboard** that tracks individual progress, reading statistics, and activity over time through visual analytics.
- 5. Design an admin panel** that allows administrators to manage book content, monitor user-generated posts, and track platform metrics.
- 6. Promote reading and knowledge** by replicating the traditional library experience digitally in a way that suits the fast-paced, technology-driven lifestyle of modern users.

1.3 purpose

The purpose of this project is to create a comprehensive and engaging digital library platform that promotes reading, knowledge sharing, and community interaction in a modern, accessible way. By simulating the traditional library experience online and integrating social and analytical features, the platform aspires to make reading more interactive, personalized, and motivating—especially for users with limited time or access to physical libraries

1.4 scope:

The scope of this project includes the full lifecycle of developing a digital library platform. The work involved spans several key phases:

- **Planning:** Identifying user needs, defining core features, and setting project goals and timeline.
- **Designing:** Creating the UI/UX for both the user and admin interfaces to ensure a seamless and intuitive experience.
- **Development/Coding:** Building the platform using modern technologies, including user registration, book browsing, recommendation systems, dashboards, and admin control features.
- **Testing:** Performing functional and usability testing, including writing and executing test cases to ensure system reliability and performance.
- **Documentation:** Preparing user manuals, technical documentation, and project reports to support future maintenance and usability

1.5 General constraints:

- *One of the main challenges we faced was finding a suitable and diverse dataset of books that would support meaningful recommendations without making the user feel bored or limited.*
- *We spent a significant amount of time searching, filtering, and preparing the data to ensure that users would always get engaging and relevant suggestions. This process was time-consuming and required a lot of manual work due to the lack of ready-to-use, high-quality datasets especially those that match our platform's goals and user experience standard*
- *Technical challenges in integrating some features like real-time analytics and advanced recommendation logic within the available timeframe*

Chapter 2: Project “Planning and analysis”

2.1 project planning

2.1.1 Feasibility Study:

A feasibility study is a critical process that evaluates the viability of a proposed project or system. It is one of the essential stages in the Software Project Management Process and helps determine whether the project is worth pursuing. The study assesses various aspects of the project, including technical, operational, economic and scheduling feasibility.

Technical feasibility:

Our software 'ktabook' is being developed using **React.js** for the frontend and **.NET (ASP.NET Core)** for the backend. These technologies are modern, reliable, and widely used in the software development industry. React.js provides a responsive and dynamic user interface, while .NET ensures a robust, secure, and scalable server-side infrastructure.

No special hardware is required. The system is designed to run smoothly on both personal computers and mobile devices, ensuring accessibility and ease of use for all users.

Operational feasibility:

The software "**Ktabook**" is designed to be easy to use, with a simple and intuitive user interface that allows users to interact, search it, and read e-books without the need for technical knowledge or training.

Additionally, the system is designed with user privacy in mind. It includes secure login mechanisms and ensures that users' personal data is not shared or exposed.

Economic feasibility:

The development of "Ktabook" system is being carried out entirely by the project team, which eliminates the cost of hiring external developers or designers.

All technologies used in the project, including React.js and .NET, are open-source or available for free, which reduces software licensing expenses

There are no training costs since the system is designed to be intuitive and user-friendly.

Scheduling feasibility:

We devided the system to phases and each phase was assigned a specific duration based on the complexity and team availability. The timeline was created to ensure that the project would be completed within the academic deadline.

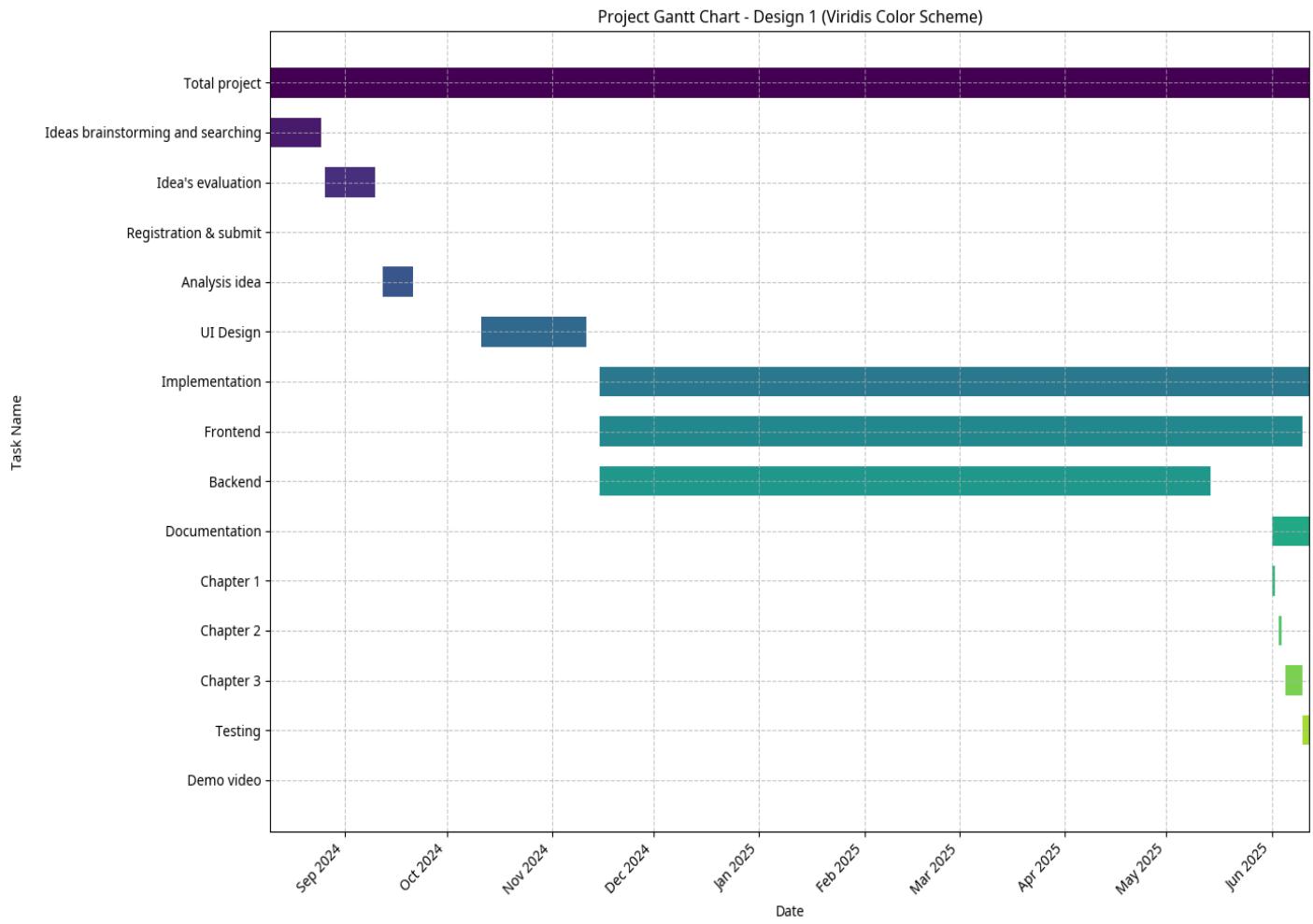
2.1.2 estimated cost:

The system was developed entirely by the student team, using free and open-source technologies, which means that there were no actual financial costs involved.

However, if the system were to be developed by a software company or a professional team, it would require a development team (frontend developers, backend developers, UI/UX designers ,machine learning engineers, and testers), as well as resources for hosting and system maintenance.

2.1.3 Gantt Chart:

	Task Name	Start	End	Duration
1	Total project	10/8/2024	12/6/2025	306
2	Ideas brainstorming and searching	10/8/2024	25/8/2024	15
3	Idea's evaluation	26/8/2024	10/9/2024	15
4	Registration & submit	11/9/2024	11/9/2024	1
5	Analysis idea	12/9/2024	21/9/2024	10
6	UI Design	11/10/2024	11/11/2024	30
7	Implementation	15/11/2024	12/6/2025	209
8	Frontend	15/11/2024	10/6/2025	208
8	Backend	15/11/2024	14/5/2025	181
9	Documentation	1/6/2025	12/6/2025	11
10	Chapter 1	1/6/2025	2/6/2025	2
11	Chapter 2	3/6/2025	4/6/2025	2
12	Chapter 3	5/6/2025	10/6/2025	6
13	Testing	10/6/2025	12/6/2025	3
14	Demo video	12/6/2025	12/6/2025	1



2.2 Analysis & Limitations of existing system

Project Gutenberg (<https://www.gutenberg.org>)

is an online digital library that offers thousands of free eBooks

Limitations:

- It is a static library system that does not support any form of user interaction.
- There is no option for user engagement while reading, such as adding highlights, sticky notes, or bookmarks, which can lead to a less engaging and more passive reading experience.
- There are no social features such as commenting, sharing, or discussing books with other readers

Open Library (<https://openlibrary.org>):

provides a large number of books, many of them are only available for borrowing for a limited time.

Limitations :

- Limited social interaction, with no support for users communication or content sharing.
- The interface is somewhat outdated and may not offer a modern user experience compared to newer platforms

2.3 Need for the new system

System features	Ktabook	Project Gutenberg	Open Library
Available books to read	yes	yes	yes
Interactive reading	yes	no	limited
Recommend similar books	yes	no	no
Social features	yes	no	no
Modern smooth ui	yes	no	no
Discuss and sharing opinions	yes	no	no
User Activity tracking dashboard	yes	no	no

From the comparison above, we can clearly see

- The current eBook platforms focus only on delivering content, but they ignore the reader's experience, interaction, and engagement.
- "**Ktabook**" introduces a new generation of eBook systems that not only let users read, but also communicate, interact, share opinions, track their daily reading progress and discover books based on their

interests. This shift turns reading from a solo passive activity into a social and dynamic experience

2.4 Analysis for the new system

2.4.1 User Requirements:

"Ktabook" aims to provide an engaging, user-friendly, and interactive eBook reading platform. The system targets two main user types: readers and administrators.

Readers (General Users):

- The system should have a clean, intuitive, and easy-to-use interface.
- Users should be able to browse and read books online
- Support for interactive reading tools such as (Adding personal notes ,Highlighting text and Inserting bookmark)
- Ability to communicate with other users (Add new posts ,comments, private messages)

- Option to save books to a favorites list .
- Users should have access to a personal **dashboard** that shows: Their reading activity (time spent on the platform). Graphs that show their interaction (number of posts , comments , and likes they did)
- Users should have their own profiles with the ability to (add their own interests and hobbies , add and update their information including cover and profile pictures)
- Responsive design that works seamlessly on both desktop and mobile devices.

Administrators:

- Admins should have full access to **control, manage, and organize** system content and users.
- Ability to **manage user accounts**, including blocking and deleting accounts
- Ability to **add, edit, or delete books**

- Ability to **create, edit, or delete communities**
- Manage and **moderate user posts**, including the ability to remove inappropriate content and view the responsible use
- System should provide admin dashboard that show summarized aggregated report shows(number of visits or registers done by month in every year , summary of the number of posts ,books and communities exist in the system)
- Ability to **generate and print PDF reports** from the admin dashboard.

2.4.2 System Requirements:

Hardware Requirements:

- No special hardware is required.
- The system is designed to work efficiently on :
 - Personal computers
 - Mobile devices
- Internet connection: Required for real-time communication features.

Software requirements:

- **For Client Side (Frontend):**
 - Technology: React.js
 - Browser Support: Chrome, Edge
 - Responsive UI: custom styles for cross-device compatibility
- **For Server Side (Backend)**
 - Technology: ASP.NET Core
 - Database: SQL Server
 - API Communication: RESTful APIs
 - Authentication: Token-based (e.g., JWT)

2.4.3 Domain Requirements:

- Users should be able to read eBooks directly online.
- The system should support social interaction (posts, comments).
- Admins must be able to manage communities and users.
- Books must support highlighting, bookmarks, and notes.
- System must be able to recommend similar books.

2.4.4 Functional Requirements:

For Readers (general users):

- Register and log into the system.
- View and edit their profile (including interests, profile/cover photos).
- Browse, search, and read available books online.
- Add books to a favorites list.
- Use interactive reading tools (highlighting, notes, bookmarks).
- Track personal reading activity and time spent.
- View personalized dashboard with stats and graphs.
- Create, edit, and delete posts within communities.
- Comment on other users' posts.
- Send and receive private messages with other users.
- Receive notifications
- Receive book recommendations based on interests or reading history

For Admins:

- Log in to the admin dashboard.
- View a summary of system activity (e.g., total users, books, communities).
- Add, edit, or delete books.
- Create and manage communities.
- View and moderate user posts and delete inappropriate content.
- Block or delete user accounts.
- Generate and download system reports as PDF

2.4.5 non-Functional Requirements:

○ Usability:

The system must be easy to use and intuitive. Users should not require any training to navigate or use its features.

○ Scalability:

The system must be designed to handle increased numbers of users and content without performance degradation.

○ Responsiveness:

The platform should adapt seamlessly to various screen sizes (mobile, tablet, desktop) and browsers.

○ Maintainability:

The codebase should be modular and well-documented, making future updates and bug fixes easier to implement.

○ Availability:

The system should be available 24/7 with minimal downtime, especially during peak usage times.

2.5 Advantages of the new system

- Provides an interactive eBook reading experience, including features like highlights, bookmarks, and personal notes.
- Supports social interaction between users through posts, comments, and private messaging, making reading more engaging.
- Includes a personalized dashboard for users to track their reading time and activity in a visual format (graphs, statistics).
- Offers book recommendations based on user interests
- Allows users to create and manage personal profiles, including profile and cover pictures, hobbies, and preferences.
- Provides a smooth and modern user interface that works seamlessly on both desktop and mobile devices.
- Gives administrators full control over the system, including book management, user moderation, and activity reports.
- Encourages community building by allowing the creation and moderation of interest-based communities.

- Enhances user motivation through visible reading progress and interaction tracking.
- Ensures ease of use without the need for any prior training.

2.6 Risk and Risk management

Unauthorized access or security breach:

Use secure authentication, data encryption, and role-based access

Compatibility issues on mobile devices:

Test thoroughly on various devices and screen sizes

Users may not engage with the social features:

Add gamification or motivation (badges, interaction tracking)

Negative or inappropriate content shared by users:

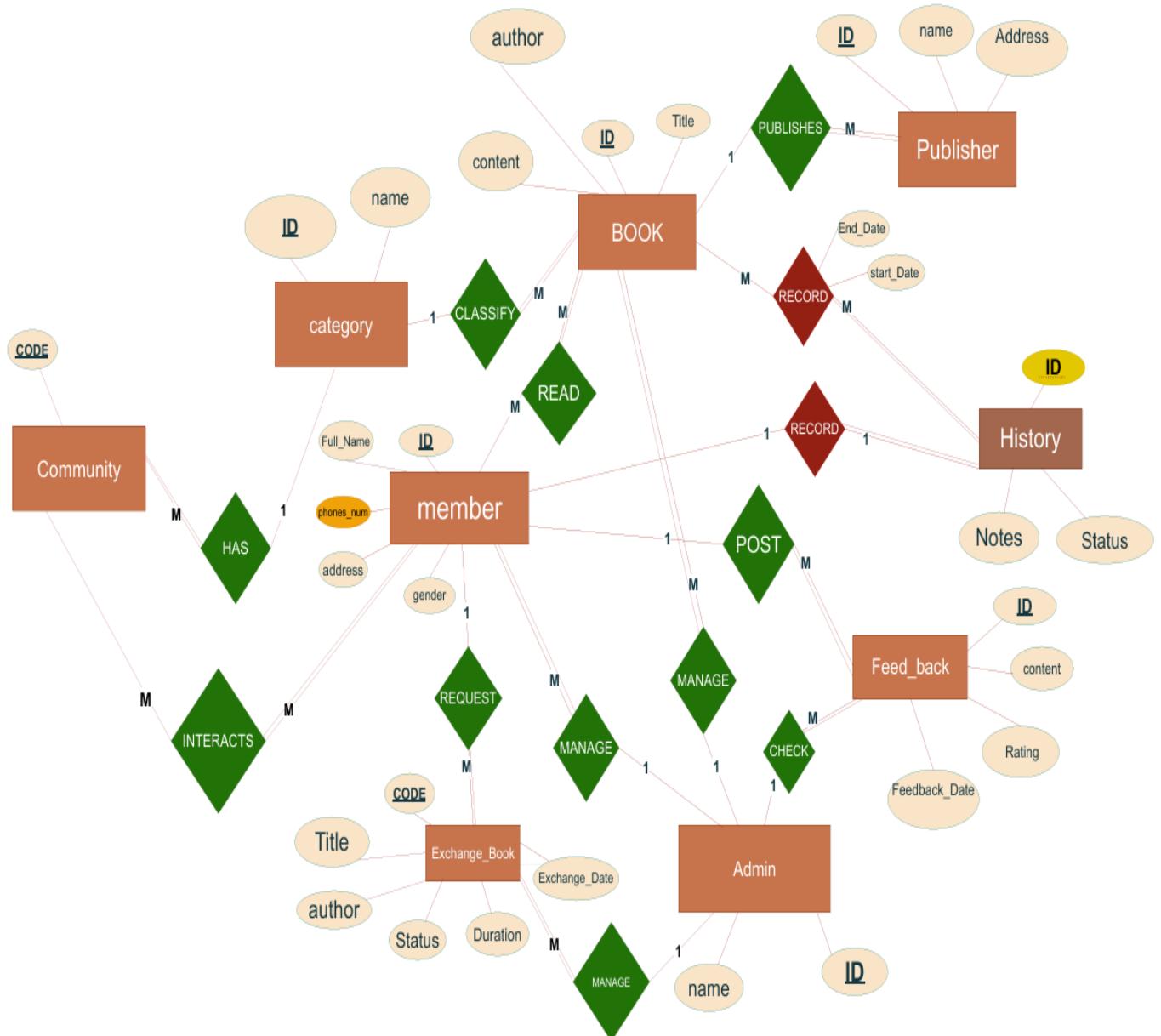
Apply admin moderation tools and reporting mechanisms

Delay in development timeline:

Use task breakdown, clear deadlines, and track team progress

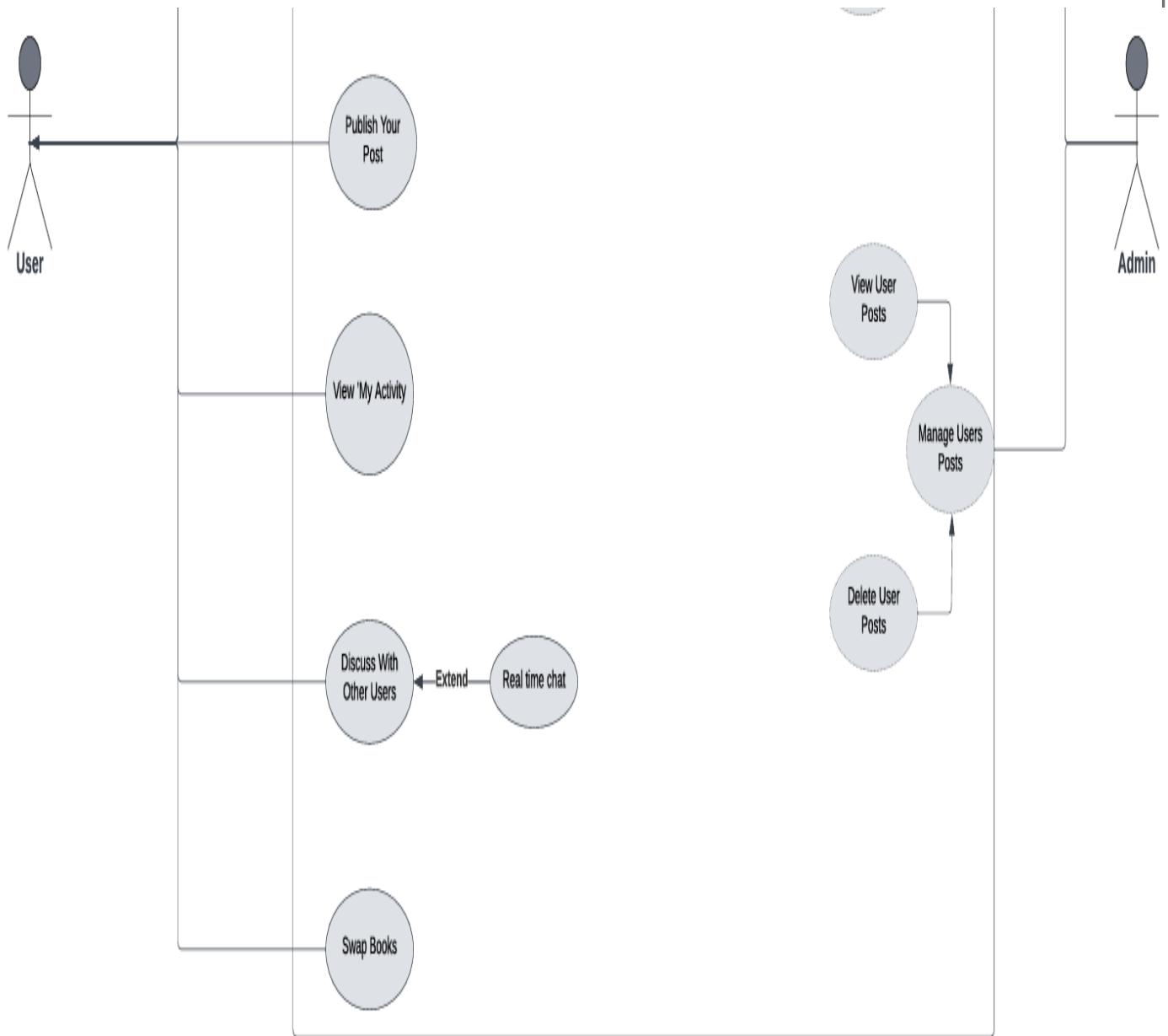
Chapter 3: Software Design

3.1 Design of database (ERD or Class) Diagram



3.2 Use case diagram





Use case scenario:

Use Case ID: 1 (Manage Books).

Actor: Admin.

Preconditions: The admin must be logged into the system.

Main Flow: The system displays a list of available books with details such as title, author, and ISBN.

Delete Book: The admin selects a book they want to delete by clicking on the "Delete" button next to the book. The system prompts the admin to confirm the deletion. Upon

confirmation, the system deletes the book and updates the list of available books.

Add Book: The admin clicks on the "Add Book" button. The system presents a form for the admin to input details of the new book, including title, author, category, and ISBN.

The admin submits the form, and the system adds the new book to the list of available books, displaying a confirmation message.

Edit Book: The admin selects a book they want to edit it by clicking on the "Edit" button next to the book. The system prompts the admin to confirm the edition. Upon

Postconditions: The list of available books is updated to reflect any additions or deletions made by the admin. A confirmation message is displayed to the admin after

successfully adding or deleting a book.

Alternative Flows: If the admin submits incomplete information in the "Add Book" form, the system should prompt the admin to complete the necessary fields.

Use Case Admin Manage Accounts:

Use Case ID: 2 (Manage User Accounts).

Actor: Admin.

Preconditions: The admin must be logged into the system.

Main Flow

1) **View Accounts:** The admin clicks on the "Accounts" button to view a list of all user accounts, including pending account requests and active accounts.

2) **Delete Account:** The admin selects an active account they want to delete by clicking on the "Delete" button next to the account. The system prompts the admin to confirm

the deletion. Upon confirmation, the system deletes the account and updates the list of user accounts accordingly.

Postconditions: The list of user accounts is updated to reflect any approvals or deletions made by the admin. Notifications are sent to users whose accounts have been

approved or deleted. System logs are updated to record the admin's actions for auditing purposes.

Alternative Flows

- 1) **Attempt to Approve an Already Approved Account:** If the admin attempts to approve an account that is already approved, the system displays an error message indicating that the account is already active.
 - 2) **Attempt to Delete a Non-existent Account:** If the admin attempts to delete an account that does not exist, the system displays an error message indicating that the account cannot be found.
 - 3) **System Error During Approval or Deletion:** If a system error occurs while approving or deleting an account, the system displays an appropriate error message and logs the error for further investigation.
-

Use Case Admin Manage User Opinions:

Use Case ID: 3 (Manage Users' Posts).

Actor: Admin.

Preconditions: The admin must be logged into the system.

Main Flow -> 1) View Posts: The admin clicks on the "Posts" button to access a list of all user Posts. The system displays the Posts, including details such as the user's name, date of submission, and the content of the posts.

2) **Delete Posts:** If the admin decides to delete a post, they click on the "Delete" button next to the relevant post. The system prompts the admin to confirm the

deletion. Upon confirmation, the system deletes the post and updates the list of posts accordingly.

Postconditions: The list of user posts is updated to reflect any deletions made by the admin. A confirmation message is displayed to the admin after successfully deleting a

post. The deleted post is no longer visible to any users in the system.

Alternative Flows -> 1) **Attempt to Delete a Non-existent post:** If the admin attempts to delete a post that has already been deleted or does not exist, the system

displays an error message indicating that the post cannot be found.

2) **System Error During Deletion:** If a system error occurs while deleting a post, the system displays an appropriate error message and logs the error for

further investigation.

Use Case User Search For Book:

Use case ID: 4 (Search for book).

Actor: user

Goal: the user found or not found the book he searched for it.

Main Flow: -> 1) The user writes the name or the title or anything about the book he wants to find it.

2) The system displays the book to the user.

Preconditions:

1) The user has access to the book search system (e.g., a library catalog, an online bookstore, etc.).

2) The system contains a database of books with searchable information (titles, authors, genres, etc.).

3) The user has an internet connection if accessing an online system.

4) The user understands how to use the search feature provided by the system.

Postconditions:

1) If the book is found, the system displays the relevant book details (title, author, publication date, summary, availability, etc.).

2) If the book is not found, the system provides a message indicating that no results were found for the search criteria.

3) The user may have options to refine their search or search for related books.

4) The search action does not alter the state of the book database.

Alternative Flow:

1) Invalid Search Input:

If the user enters invalid characters (symbols, excessive whitespace, etc.), the system displays an error message prompting the user to enter a valid search term. The user re-enters a valid search term.

2) Partial Match Found:

- * If the user's search returns results that do not exactly match the search term (e.g., partial matches), the system displays all relevant results and may highlight the matching parts of the title or description.
- * The user can select a result from the list for more details.

3) Network Issues:

- * If the system encounters a network issue while searching, it displays a message indicating the problem and prompts the user to try again later.
- * The user may choose to retry the search after resolving the network issue.

4) Multiple Results Found:

- * If multiple books match the search criteria, the system displays a list of all matching books.
- * The user can browse the list and select a specific book for more detailed information.

5) User Cancels Search:

- * The user may decide to cancel the search operation before completion.
 - * The system returns to the main search interface without performing any actions.
-

Use Case User View Profile:

Use Case ID: 5 (View Profile).

Actor: User

Goal: The user can view and modify their profile.

Preconditions:

- 1) The user must be registered in the system.
- 2) The user must be logged into the system.
- 3) The system must have the user's profile data stored and accessible.

Postconditions:

1) If the user views their profile:

1.1) The system successfully displays the user's profile information (e.g., name, email, contact information, preferences).

2) If the user modifies their profile:

2.1) The system updates the user's profile with the new information provided.

2.2) The system confirms successful changes to the user.

Main Flow -> 1) The user clicks on their profile icon or link.

2) The system retrieves and displays the user's profile information.

3) The user reviews their profile details.

4) Track personal reading activity and time spent.

5) View personalized dashboard with stats and graphs.

6) Receive notifications.

7) The user makes modifications to their profile as needed (e.g., updating personal information, changing password).

8) The user submits the changes.

9) The system validates the changes and updates the user's profile.

10) The system displays a success message to confirm that the profile has been updated.

Alternative Flows:

1) Profile Not Found:

- * If the system cannot find the user's profile data (e.g., due to a data issue), it displays an error message conveying that the profile could not be retrieved.
- * The user may be provided options to create a new profile or contact support.

2) User Makes No Changes:

- * If the user accesses their profile but decides not to make any changes, they can simply navigate away, and the system retains the original profile without changes.
- * The system may prompt for confirmation if unsaved changes were made.

3) Error While Saving Changes:

- * If there is an error during the profile update process (e.g., due to a network issue), the system displays an error message indicating that the changes could not be saved.
- * The user can address the issue and attempt to save again.

4) Validation Errors on Modification:

- * If the user attempts to save invalid data (e.g., incorrect email format), the system highlights the errors and prevents the update.
- * The system displays relevant messages to guide the user to correct the input.

5) User Cancels the Modification:

- * If the user chooses to cancel edits to their profile, the system may prompt for confirmation and then discard any changes made.
- * The user is returned to the profile view without any updates.

6) Session Timeout:

- * If the user's session times out while they are viewing or editing their profile, the system displays a session timeout message.
- * The user is redirected to the login page to re-authenticate before accessing their profile again.

Use Case User Discuss with Other Users:

Use Case ID: 6 (Discuss with Other Users).

Actor: User.

Goal: The user can discuss with other users via real-time chat or online meetings.

Preconditions:

- 1) The user must be registered in the system.
- 2) The user must be logged into the system.
- 3) The user must have access to the discussion features (real-time chat and online meetings).
- 4) The user must have an active internet connection to facilitate real-time communication.

Postconditions:

- 1) If the user engages in a discussion:
 - 1.1) The system successfully establishes a real-time chat or online meeting based on the user's selection.
 - 1.2) The user can communicate with other users effectively.
- 2) If the user exits the discussion:
 - 2.1) The system ends the chat session or meeting and returns the user to the previous interface.
 - 2.2) The discussion history (if applicable) is saved for future reference, depending on system design.

Main Flow -> 1) The user clicks on the book they want to read.

- 2) The system displays the book details along with two options for discussion: Real-Time Chat and Online Meetings.
- 3) The user selects either the Real-Time Chat or Online Meetings option.

4) If the user selects Real-Time Chat:

4.1) The system opens a chat window where the user can send messages to other users currently reading the same book.

4.2) The user can type and send messages, and view incoming messages in real-time.

5) If the user selects Online Meetings:

5.1) The system prompts the user to schedule or join an ongoing meeting.

5.2) The user can set a time for the meeting or join an existing one with other users.

6) The user engages in discussions through the chosen method (chat or meeting).

Alternative Flows:

1) No Other Users Available:

* If there are no other users available for discussion, the system notifies the user that there are currently no participants online.

* The user can choose to leave a message or try again later.

2) User Cancels Discussion:

* If the user decides to cancel the discussion before it starts, they can exit the chat window or meeting prompt.

- * The system returns the user to the book details page without initiating any discussion.

3) Error in Establishing Connection:

- * If there is a technical issue (e.g., network error) while trying to establish a chat or meeting, the system displays an error message.
- * The user can choose to retry connecting or exit the discussion option.

4) User Experiences Technical Issues During Discussion:

- * If the user faces technical issues (e.g., audio/video problems in online meetings), the system provides troubleshooting tips or options to reconnect.

- * The user can report the issue or exit the discussion.

5) User Session Timeout:

- * If the user's session expires while in the discussion, the system displays a session timeout message.
- * The user is redirected to the login page to re-authenticate before accessing the discussion features again.

6) User Chooses to Switch Discussion Methods:

- * If the user wants to switch from real-time chat to an online meeting (or vice versa), they can exit the current discussion and select the new option.
- * The system prompts for confirmation before switching.

Use Case User Exchange Book:

Use Case ID: 7 (Exchange Book).

Actor: User.

Goal: The user can exchange a hard copy of a book for another book or a soft copy in the system.

Preconditions:

- 1) The user cannot find a soft copy of the book they desire in the application.
- 2) The user wants to exchange their hard copy of a book with another user.
- 3) The user must be registered and logged into the system.
- 4) The user must have a hard copy of the book they intend to exchange.

Postconditions:

- 1) The exchange request is processed, and the user receives the book they were seeking, either through an exchange of hard copies or by obtaining the soft copy.
- 2) The system updates the availability of books and records the new ownership of the exchanged books.
- 3) Any exchanged books are marked as "exchanged" in the system to maintain accurate availability for other users.

Main Flow -> 1) Search for Book:

* The user searches for the book they want in the application.

- * If the book is not available in soft copy format, they proceed to the next step.

2) Request Exchange:

- * The user initiates a request to exchange their hard copy of the book.
- * They provide details about the book they have (title, author, condition) and the book they are looking for.

3) Submit Request:

- * The user submits the exchange request.
- * The exchange request is made visible to other users in the system.

4) Other Users View Request:

- * Other users can view the exchange request.
- * If any user has the soft copy of the book that the requesting user is looking for, they can offer to upload the soft copy for the requesting user.

5) Upload Soft Copy:

- * If a user with the soft copy of the desired book is found, they upload the soft copy to the system and offer it to the requesting user.

6) Rate the Request:

- * The requesting user reviews and rates the offers they receive, focusing on the quality and suitability of the exchange offers.

7) Finalize Exchange:

- * The requesting user accepts a suitable offer (either an exchange of hard copies or receiving the soft copy) and finalizes the transaction.
- * The system updates the records to reflect the new ownership of the books.

8) Show Offers:

- * The system displays other available exchange offers to the user, which they can review and consider for future exchanges.

Alternative Flows:

1) No Suitable Offers:

- * If no suitable offers are received, the user may choose to either modify their request or cancel it.

2) Modify or Cancel Request:

- * The user has the option to modify the details of their request (e.g., changing the desired book) or cancel it if they no longer wish to pursue the exchange.

3) Technical Issues:

- * If there is a technical issue while submitting the request or during the upload process, the system displays an error message.

- * The user can retry the action or seek assistance if the problem persists.

4) User's Session Timeout:

- * If the user's session expires during the process, the system displays a session timeout message.

- * The user is redirected to the login page to re-authenticate before they can continue.

5) Insufficient Information:

- * If the user submits an exchange request without providing necessary information, the system prompts the user to fill in all required fields.

- * The user must supply the required information before resubmitting the request.

6) Duplicate Requests:

- * If the user attempts to submit an exchange request for a book they already have an active request for, the system alerts them that a request already exists.

- * The user can choose to modify the existing request instead of creating a new one.

Use Case User publish Your Post:

Use Case ID: 9 (publish Your Post).

Actor: User.

Goal: The user can post their post or review of a book they have read, engage in discussions, and interact with other users.

Preconditions:

1) The user must be registered and logged into the system.

2) The user must have completed reading the book to provide a relevant and informed posts.

3) The book must be available in the system for review.

Postconditions:

1) The user's posts (including their thoughts, rating, and comments) is publicly posted and visible to other users.

2) Other users can view and respond to the user's post, leading to potential discussions about the book.

3) The system records the user's engagement with the book for future reference.

Main Flow -> 1) Complete Reading the Book:

* After finishing the book, the user navigates to the book's page within the system.

2) Access the post Section:

* The user clicks on the "Post" or "Review" link/section of the book's page.

3) Post posts:

* The user writes their post about the book in a text box provided.

* The user selects a rating (e.g., 1 to 5 stars) based on their reading experience.

* The user submits their post.

4) Engage in Discussion:

- * After submitting, the user can view their posted post along with posts from other users.
- * The user has an option to comment on other users' posts or engage in discussions by replying to their comments.

Alternative Flows:

1) Edit or Delete post:

- * If the user wants to modify or remove their post after posting:
 - 1.1) They navigate to their post through their profile or the book's page.
 - 1.2) They select the edit option to modify the existing post or choose the delete option to remove it entirely.

2) User Account Required:

- * If the user is not logged in:
 - 2.1) The user is prompted with a message to log in or register before proceeding to post their post.
 - 2.2) Upon logging in or registering, the user is redirected back to the post posting section.

3) Incomplete post Submission:

- * If the user submits a post without providing a rating or leaving the post text blank:
 - 3.1) The system displays an error message prompting the user to complete all required fields before submitting their post.

4) Technical Issues:

* If there is a technical issue during the post submission (e.g., time-out, server error):

4.1) The system notifies the user of the issue, allowing them to retry submitting their post.

5) Duplicate post Submission:

* If the user attempts to submit an identical post multiple times:

5.1) The system alerts the user that they have already posted the same post and gives them the option to edit the existing post instead.

6) User Feedback on posts:

* Users can provide feedback (e.g., "Helpful" or "Not Helpful") on other users' posts:

6.1) If a user clicks to provide feedback, the system updates the feedback count for that post.

Use Case User View My Activity:

Use Case ID: 10 (View My Activity).

Actor: User.

Goal: The user can view their activity in the application, including visit counts and reading history.

Preconditions:

1) The user must be logged into their account.

2) The system must have recorded data on the user's visits and reading history.

Postconditions:

- 1) The activity data, including the visit count and reading history, is updated with each new visit or reading activity.
- 2) The user has an overview of their engagement with the application, enhancing their interaction experience.

Main Flow -> 1) Navigate to Activity Page:

*The user logs into their account and navigates to the "Activity" page from the main menu or dashboard.

2) Retrieve Visit Count:

* The system retrieves the user's total visit count to the application.

3) Retrieve Reading History:

* The system fetches the list of books the user has read, along with relevant details such as:

Book title

Author

Reading date

Rating (if applicable)

Any submitted reviews

4) Display Activity Data:

- * The system displays the visit count prominently at the top of the Activity page.

- * Below the visit count, the system presents the reading history in a well-organized format, such as a table or list.

5) Review Activity Data:

- * The user reviews their activity data, reflecting on their reading habits and engagement with the platform.

Alternative Flows:

1) No Activity Data Available:

- * If the user has no recorded activity (e.g., they have not read any books or visited the application previously):

#The system displays a message indicating that no activity data is available.

The message may include suggestions for how to begin engaging with the application (e.g., "Start exploring books to see your activity here!").

2) Technical Issues:

- * If there are issues retrieving the data due to server problems or system errors:

The system displays an error message informing the user that their activity data could not be retrieved at that moment and advises them to try again later.

3) Session Timeout:

* If the user's session times out while attempting to view the activity page:

The user is prompted to log in again to access their account. After successfully logging in, the user may need to navigate back to the Activity page.

4) Data Format Issues:

* If the retrieved data contains inconsistencies or errors (e.g., missing book titles):

The system displays a warning message indicating that some data may be inaccurate and prompts the user to refresh the page or contact support.

Use Case User Reading a Book:

Use Case ID: 11 (Reading a Book).

Actor: User.

Goal: The user can search for, read, and manage their reading experience for books available in the system.

Preconditions:

- 1) The user must be registered and logged into the system.
- 2) The user should have a stable internet connection if the books are accessible online.
- 3) The user must have access to the book through purchase, loan, or subscription.

Postconditions:

- 1) The system automatically saves the user's reading progress when the book is closed.
- 2) After finishing the book, the user can rate it, allowing other users to benefit from their review.
- 3) The system may recommend other books based on the user's rating and reading history.

Main Flow ->1) Search for a Book:

* The user uses the search feature to enter the title, author, or keywords related to the desired book.

2) Open the Book:

* After finding the book in the search results, the user clicks on the book title, which redirects them to the book's reading interface.

3) Review Contents:

* The user accesses the table of contents or chapter list within the book to understand its structure before starting to read.

4) Read the Book:

* The user begins reading the book, navigating through the chapters as needed.

5) Mark Important Sections:

* The user identifies significant paragraphs and marks them using sticky notes or highlighting features provided by the system.

6) Bookmark Progress:

- * If the user cannot finish the book in one sitting, they can bookmark their current reading position to easily return later.

7) Customize Reading Experience:

- * The user customizes the reading settings, such as adjusting the font size, background color, or disabling distractions.
- * The user can also enable night mode for a more comfortable reading experience in low light.

8) Add to Favourites:

- * The User can mark the book to add to the user favourite section.

9) Download Book:

- * The User can download the book from website to read it offline without website features.

Alternative Flows:

1) Book Not Found:

- * If the desired book is not available in the system:

1.1) The user can submit a request for the book by providing the title, author, and preferred type of copy (soft copy or hard copy).

1.2) The system confirms the request submission and may contact the user once the book becomes available.

2) Receive Reading Reminders:

- * The system may send notifications reminding the user to continue reading any unfinished book after a certain period.

3) Manage Notes and Highlights:

- * While reading, the user can manage their sticky notes and highlights:

- 3.1) The user can edit existing notes, delete notes they no longer need, or add new ones as they continue to read.

4) Technical Issues:

- * If the user encounters any technical issues (e.g., slow loading, app crashes):

- 4.1) The system notifies the user of the issue and suggests trying again later or checking their internet connection.

Use Case ID: 12

Use Case Name: View Suggested Reading Based on Content

Actor: User

Goal:

The user can view a list of book recommendations generated based on the content similarity between books they've interacted with and other books in the system.

Preconditions:

1. The user must have an active account on the platform.
2. The user must be logged in.

3. The user must have interacted with at least one book (e.g., opened, liked, bookmarked, rated).
 4. The system must have access to book content features (e.g., descriptions, keywords, topics) and a recommendation engine.
-

Postconditions:

1. The user views a list of book recommendations based on textual or thematic content similarity.
 2. The recommended books include metadata: title, author, genre, and description.
 3. The system logs user interaction with the recommendations to refine future results.
-

Main Flow:

1. The user navigates to the “**Recommended Reading**” section.
2. The system identifies books the user interacted with (e.g., read or bookmarked).
3. The system extracts content features (e.g., tags, descriptions, categories) of those books.
4. Using content-based filtering, the system finds books with high similarity scores based on content.
5. The system displays a list of recommended books, including:
 - Title
 - Author
 - Genre
 - Short description or summary
6. The user may:
 - View book details
 - Save the book to a list
 - Mark it as read/interested

7. If the user interacts with a book, the system updates the model to improve future suggestions.
-

Alternative Flows:

1. Not Enough Interactions Yet

- If the user hasn't interacted with any book yet:
 - The system displays a message like: "**We need more info to personalize your reading list.**"
 - The user is prompted to explore or rate a few books to help generate personalized suggestions.

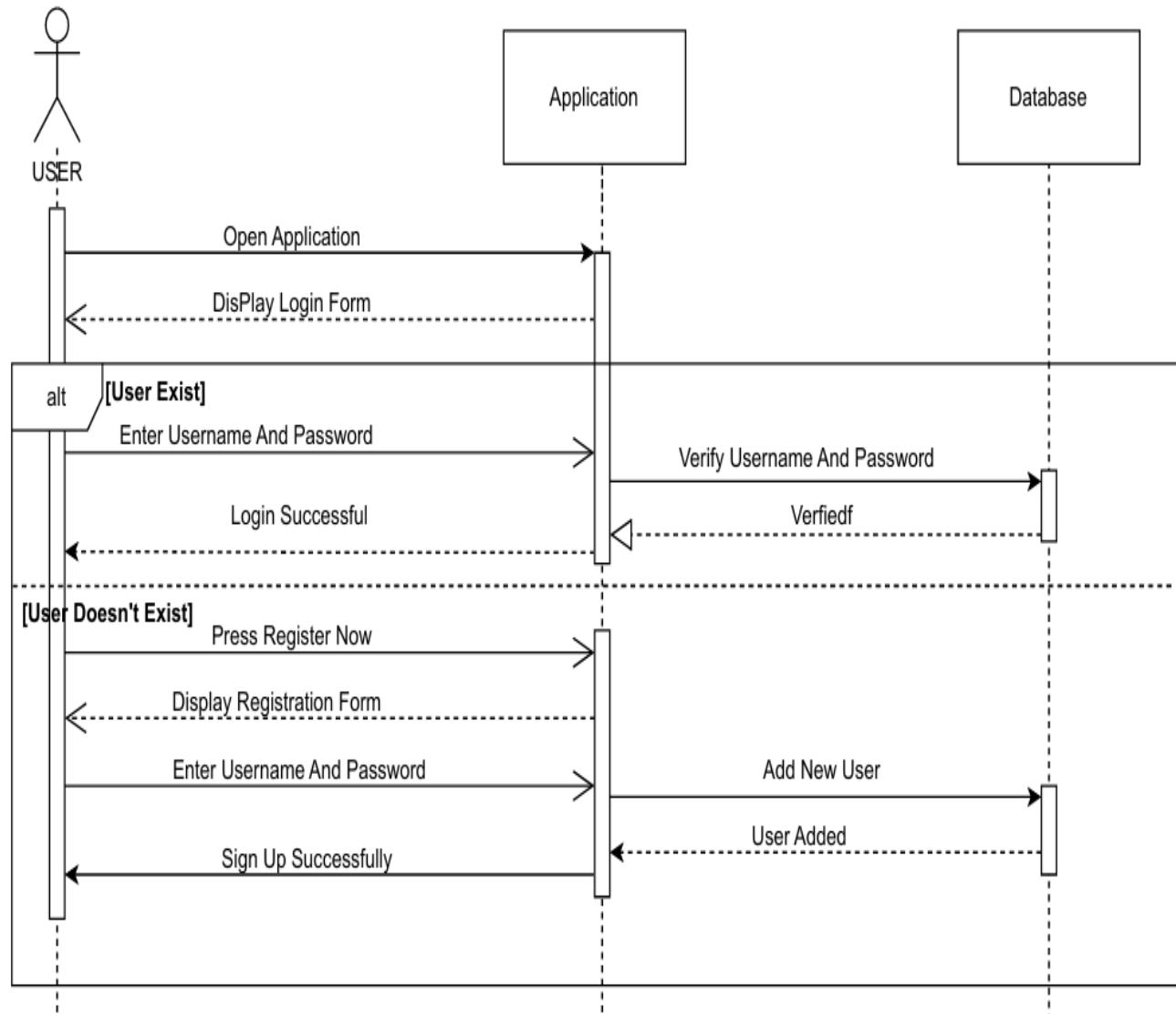
2. User Updates Preferences (Optional Layer)

- The user can optionally modify filter preferences such as:
 - Preferred genre or topic
 - Exclude books already read
 - Reading difficulty level
 - These are used as additional filters post-recommendation, not in place of content similarity.
-

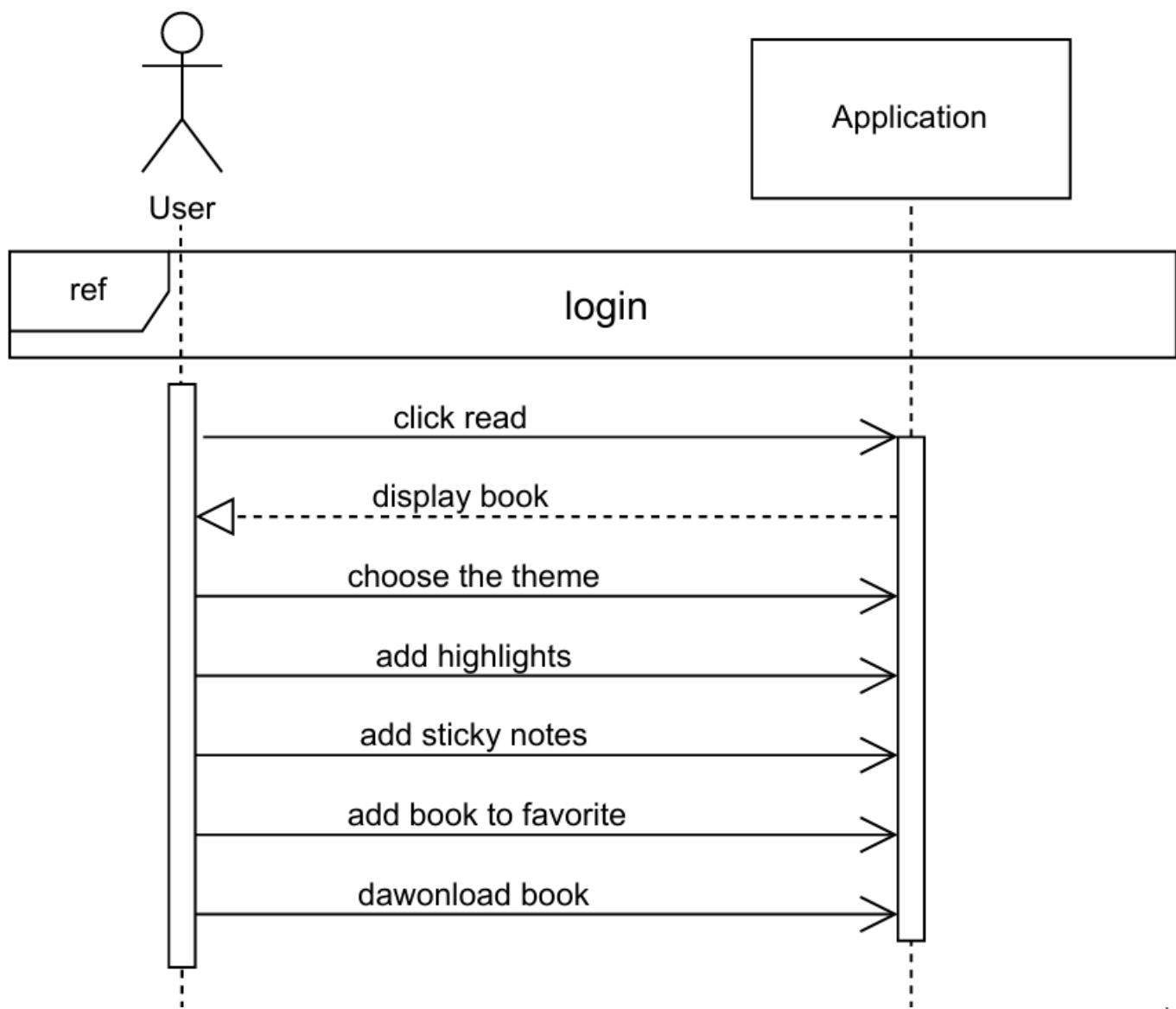
3.3 sequence diagram

Login and register

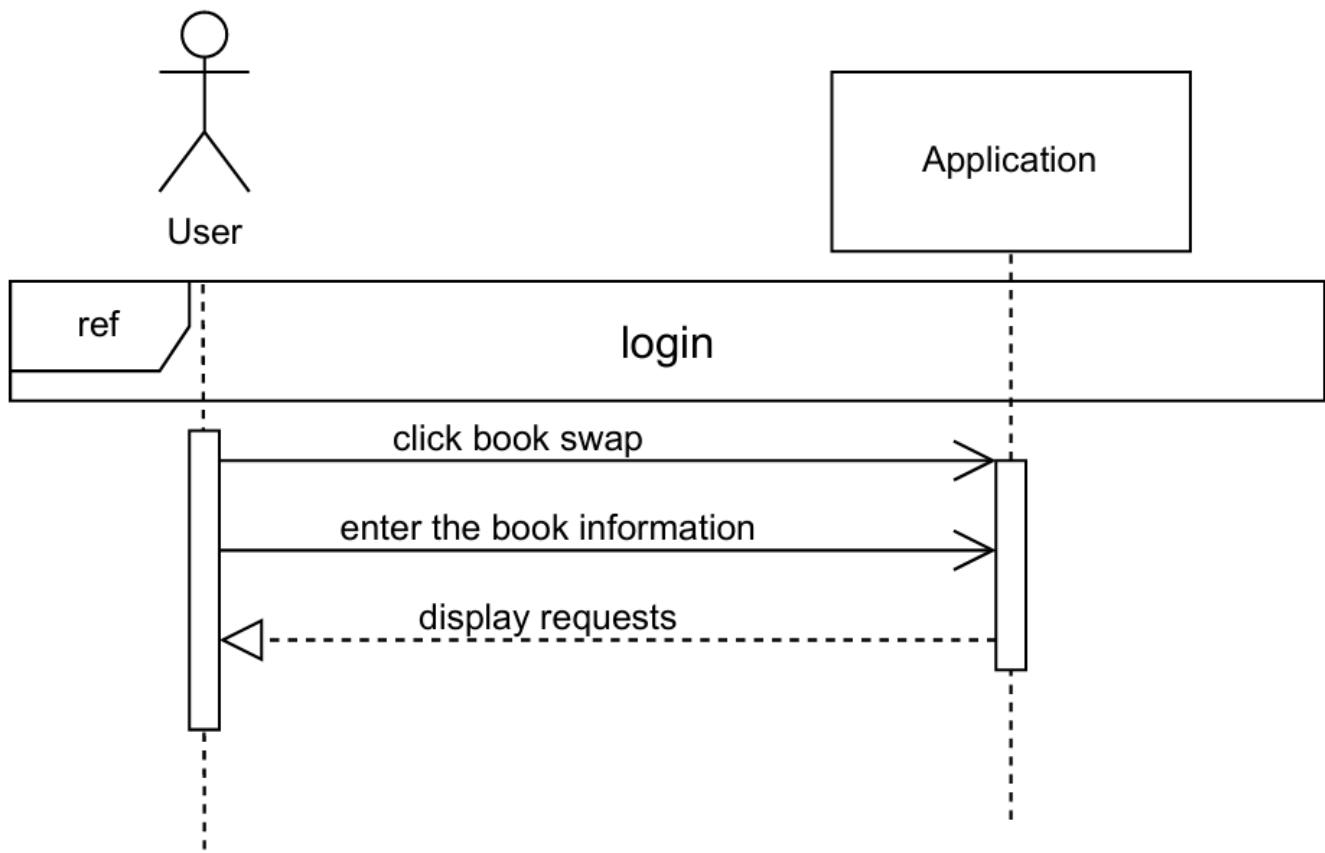
Login / Register



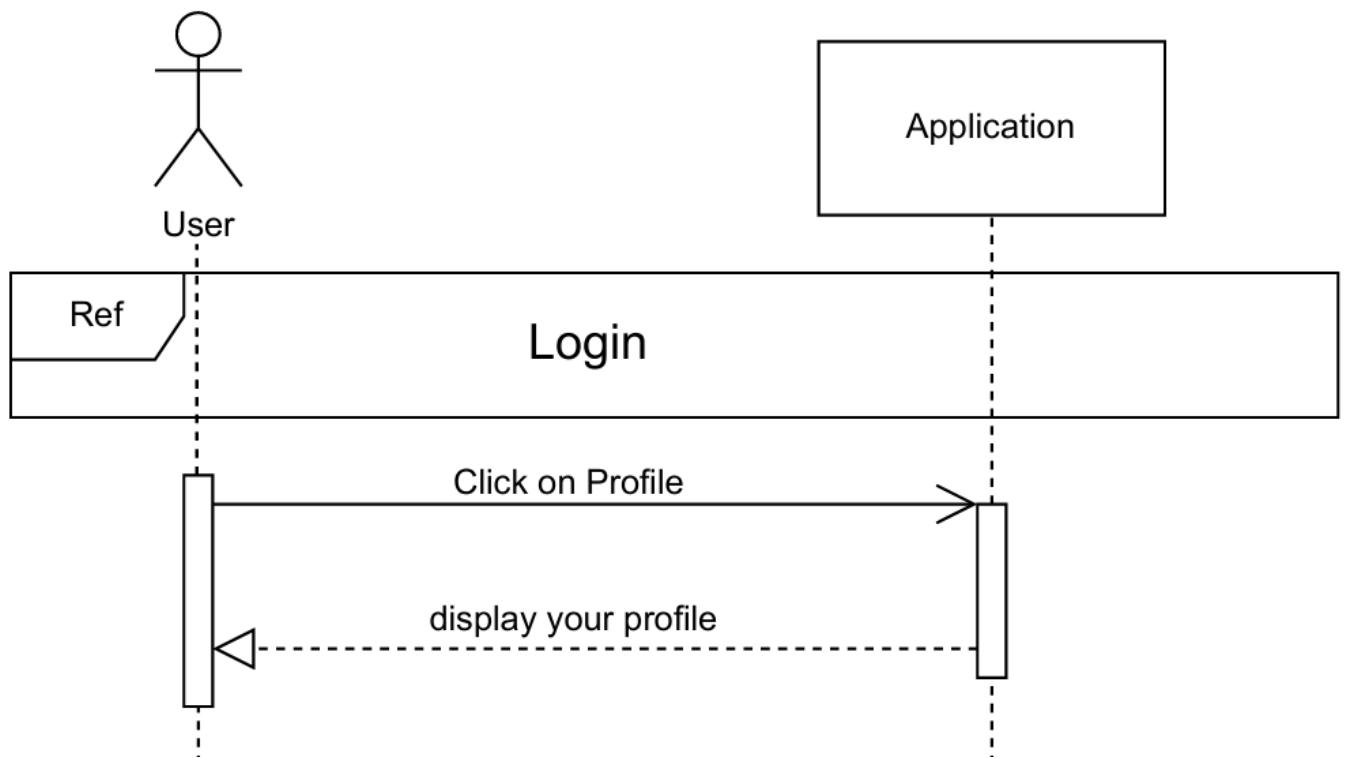
Read book



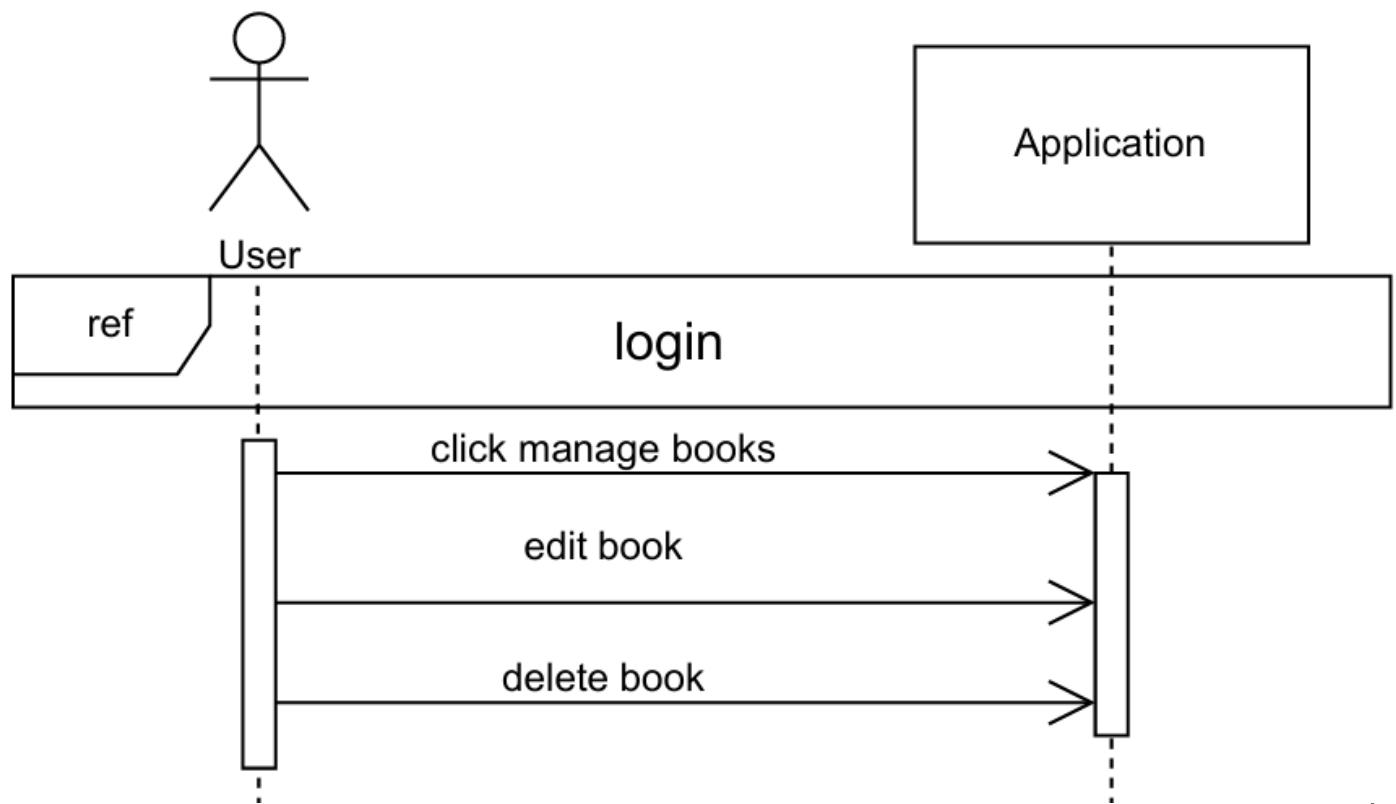
Book swap



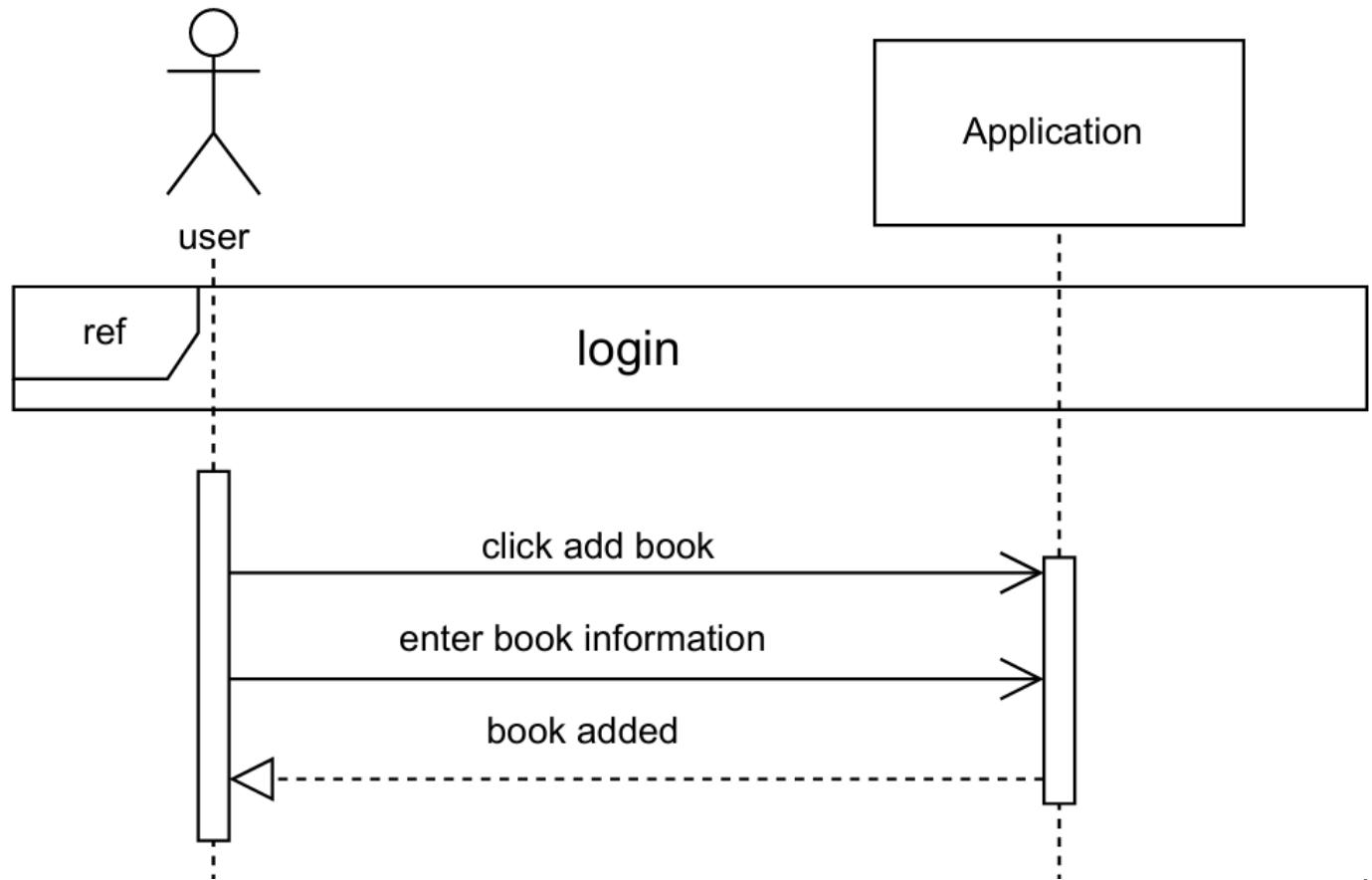
Show user profile



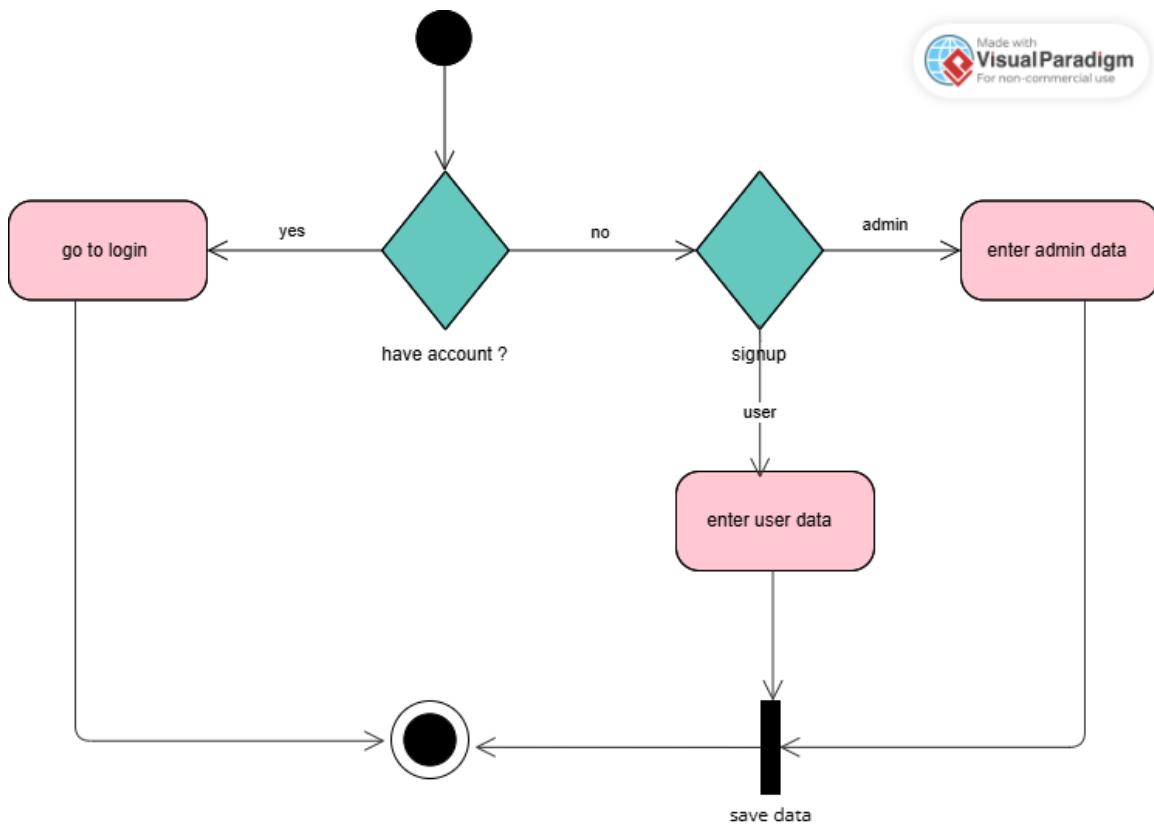
Manage books

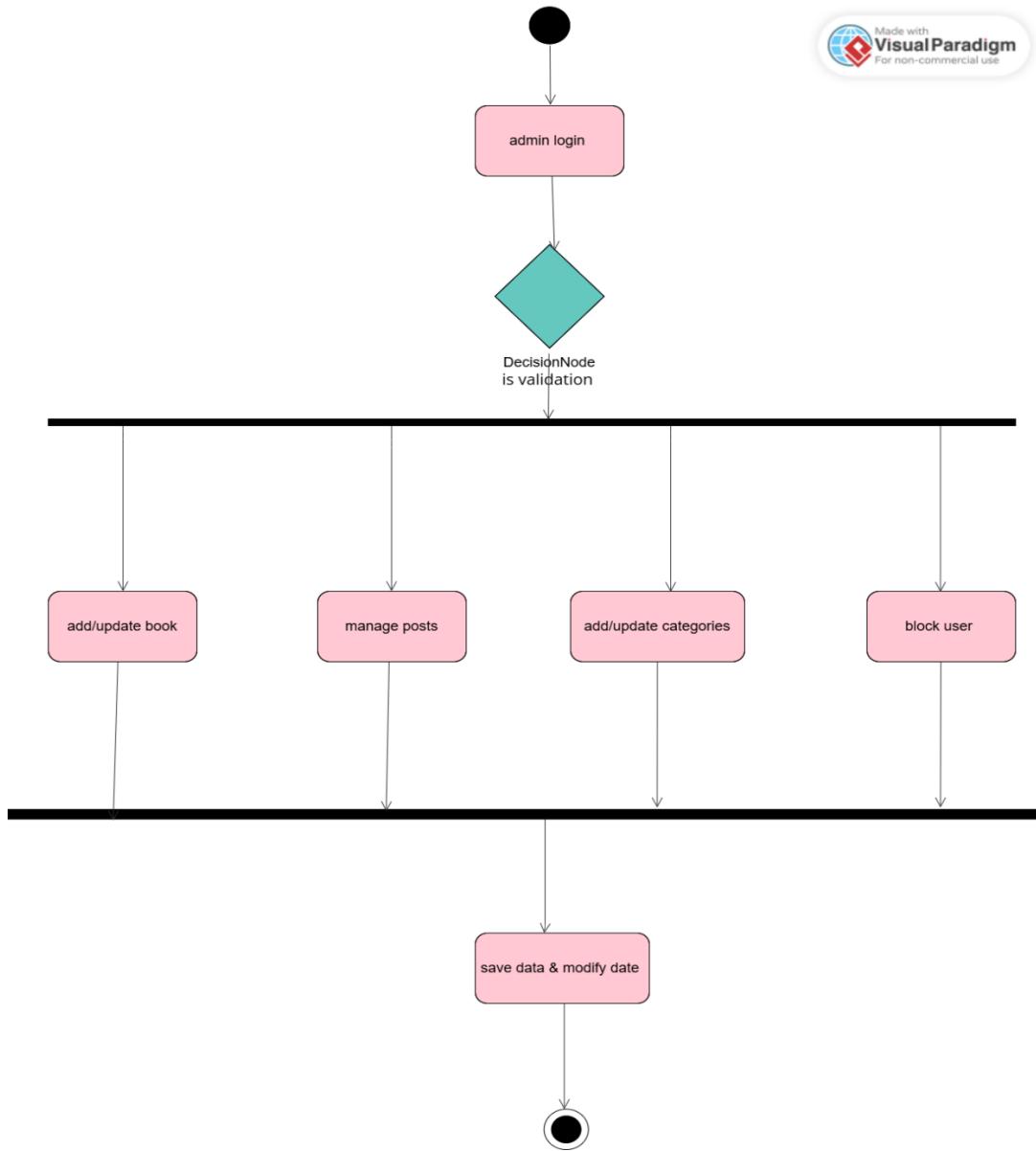


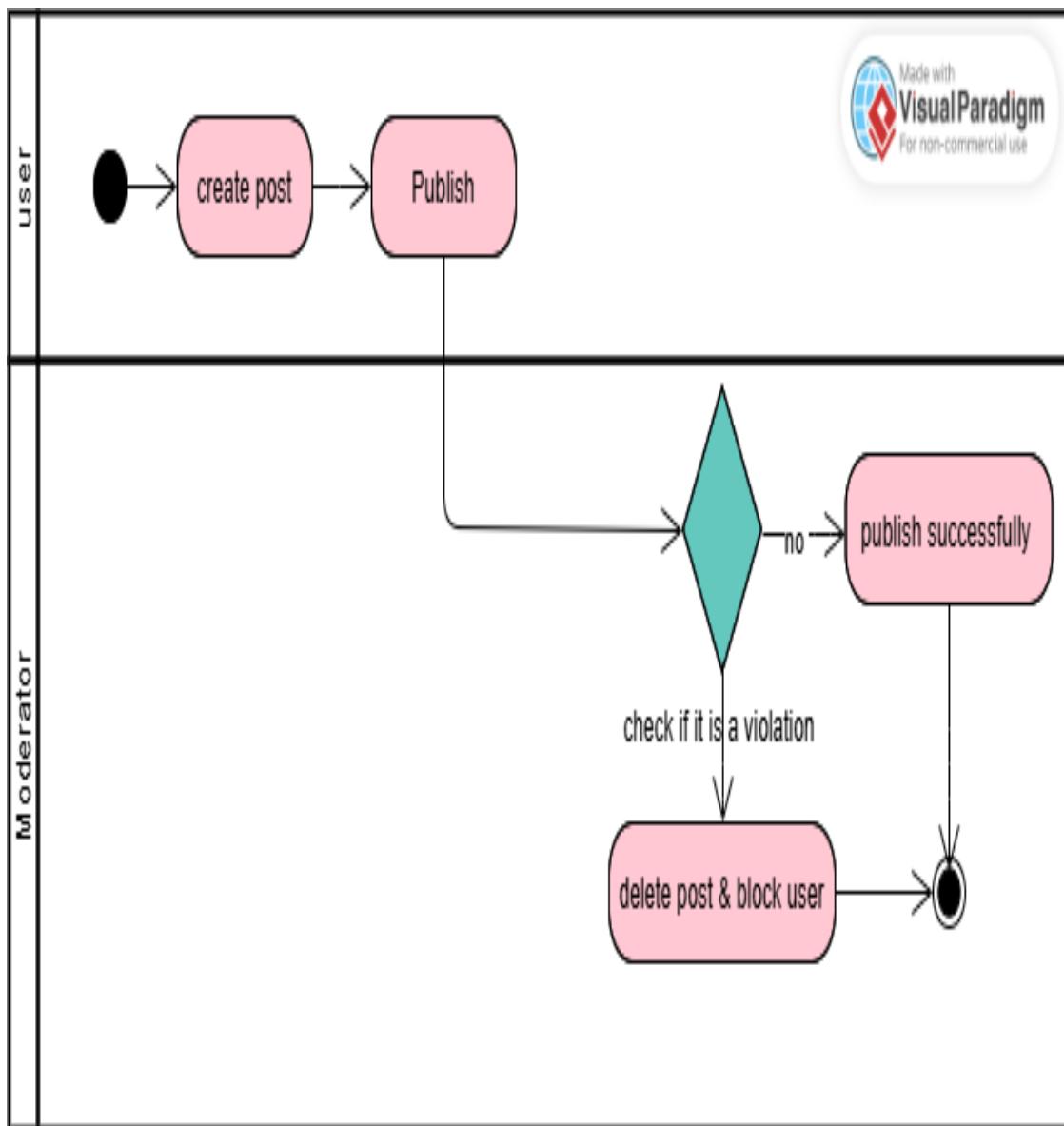
Add book

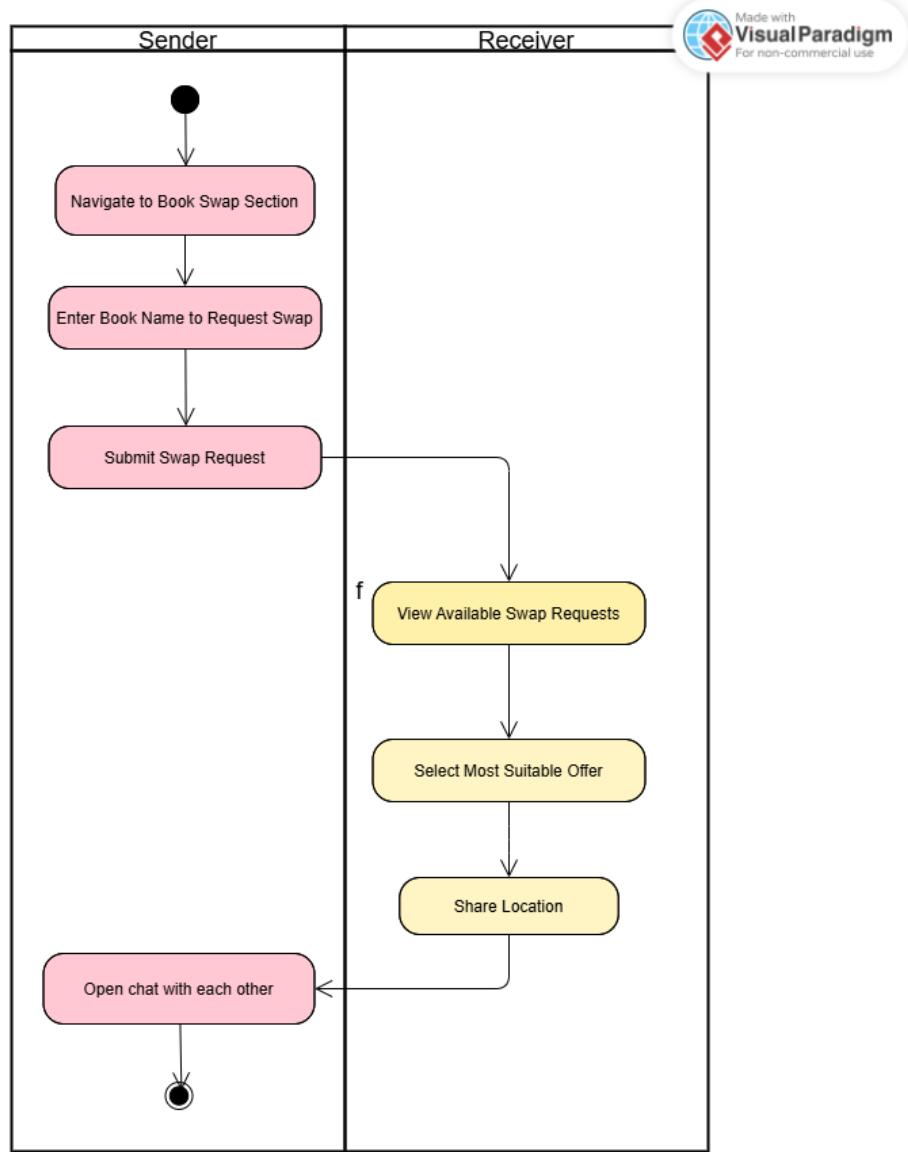


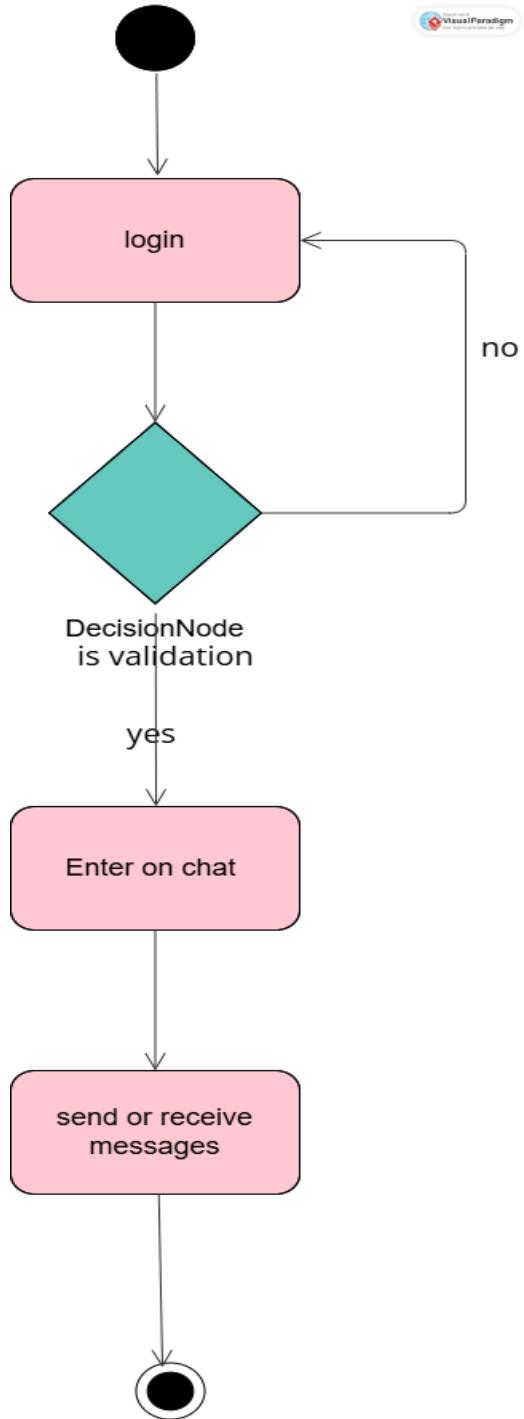
3.5 activity diagram

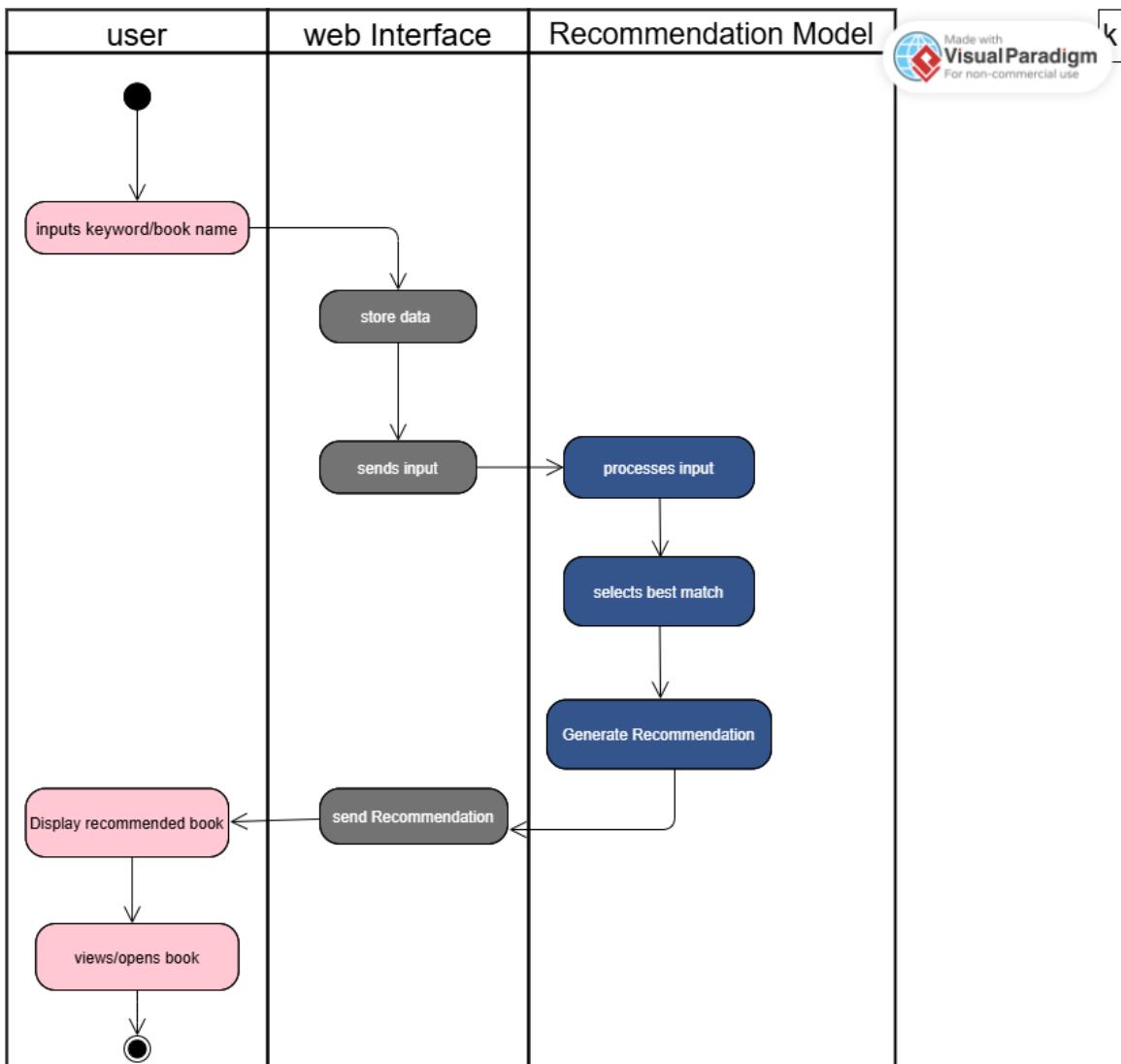










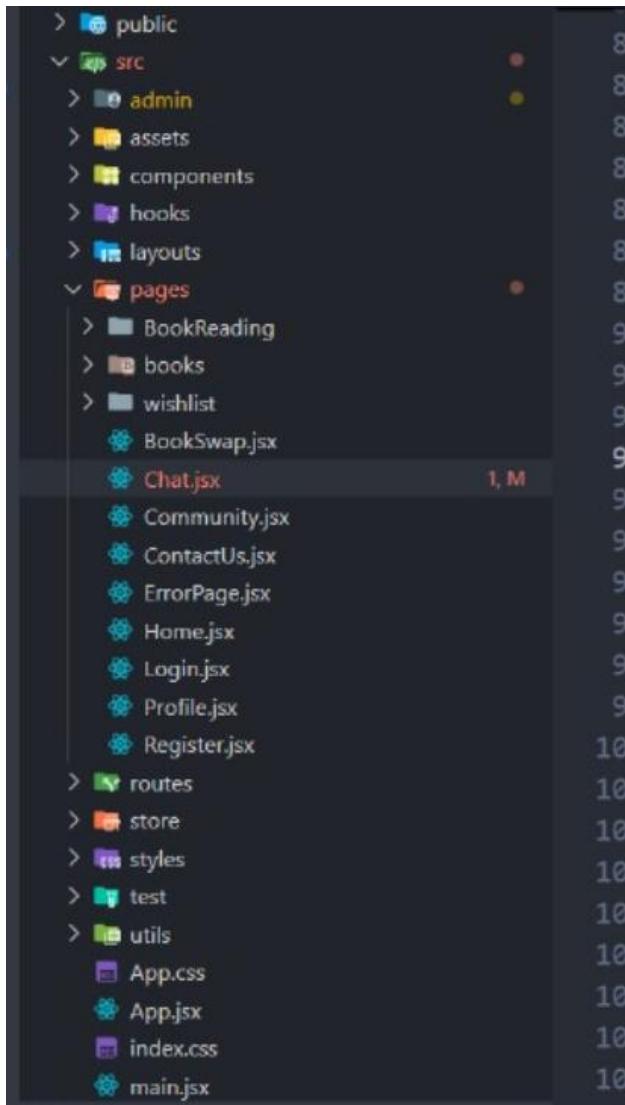


Chapter 4: Implementation

chapter 4 presents the practical implementation of the Online Library system `Stabook`, focusing on both the client and server sides. It explains the software architecture, communication flow, and technologies used to handle user interactions, data processing, and system features.

Client Side (Frontend)

The client application, developed using [React]



A screenshot of a file explorer window showing the directory structure of a React application. The structure is as follows:

- > public
- < src
 - > admin
 - > assets
 - > components
 - > hooks
 - > layouts
 - < pages
 - > BookReading
 - > books
 - > wishlist
 - BookSwap.jsx
 - Chat.jsx 1, M
 - Community.jsx
 - ContactUs.jsx
 - ErrorPage.jsx
 - Home.jsx
 - Login.jsx
 - Profile.jsx
 - Register.jsx
 - > routes
 - > store
 - > styles
 - > test
 - > utils
 - App.css
 - App.jsx
 - index.css
 - main.jsx

allows users to perform the following: login

Chat

```
// Setup SignalR connection
useEffect(() => {
  const newConnection = new signalR.HubConnectionBuilder()
    .withUrl(HUB_URL, {
      accessTokenFactory: () => token,
    })
    .withAutomaticReconnect()
    .build();

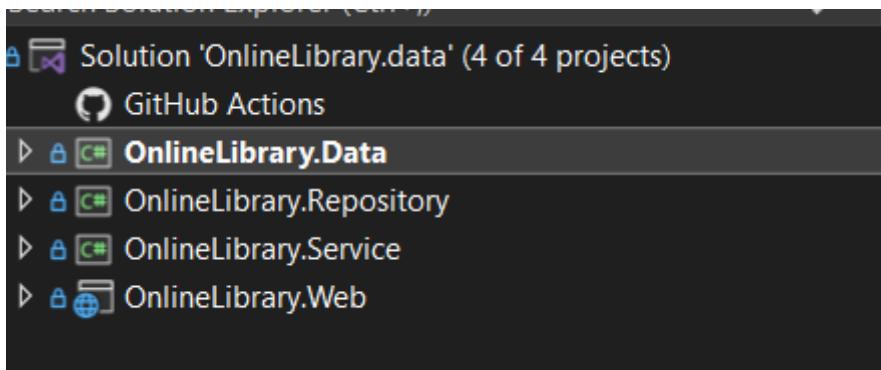
  setConnection(newConnection);
}, [token]);
```

Server Side (Backend)

The backend is built using **ASP.NET Core Web API** and handles:

- User authentication and authorization (JWT tokens).
- Processing requests and sending appropriate responses in **JSON** format.
- Database interactions using **Entity Framework Core** (ORM).
- Business logic such as post creation, likes, comments, chat handling, swap processing, recommendation engine, text-to-speech conversion, etc.
- Admin/moderator functionalities to manage content and users.

We using **ASP.NET Core Web API c#**



Database

- Uses **Entity Framework Core** for object-relational mapping (ORM).
- Core Entities:
 - User, Book, Post, SwapRequest, ChatMessage, Category, WishList, ReadingHistory
 - We use code first handle it with data set

```
28 references
public class OnlineLibraryIdentityDbContext : IdentityDbContext<ApplicationUser>
{
    6 references
    public DbSet<PendingUserChange> PendingUserChanges { get; set; }
    0 references
    public DbSet<BooksDatum> BooksData { get; set; }
    0 references
    public DbSet<History> History { get; set; }
    0 references
    public DbSet<Review> Reviews { get; set; }
    0 references
    public DbSet<FavoriteBook> FavoriteBook { get; set; }
    0 references
    public DbSet<Wishlist> Wishlists { get; set; }
    0 references
    public DbSet<Readlist> Readlists { get; set; }
    0 references
    public DbSet<Community> Communities { get; set; }
    2 references
    public DbSet<CommunityMember> CommunityMembers { get; set; }
    1 reference
    public DbSet<CommunityPost> CommunityPosts { get; set; }
    0 references
    public DbSet<PostLike> PostLikes { get; set; }
    0 references
    public DbSet<PostUnlike> PostUnlikes { get; set; }
    0 references
    public DbSet<PostComment> PostComments { get; set; }
    0 references
```

- Data is transferred between layers using **DTOs** (Data Transfer Objects), ensuring separation of concerns and security.

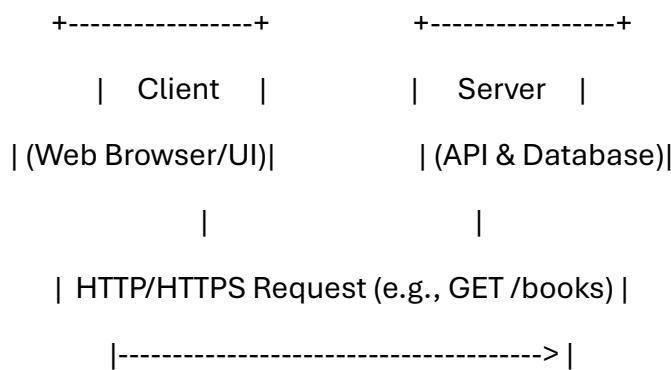
```

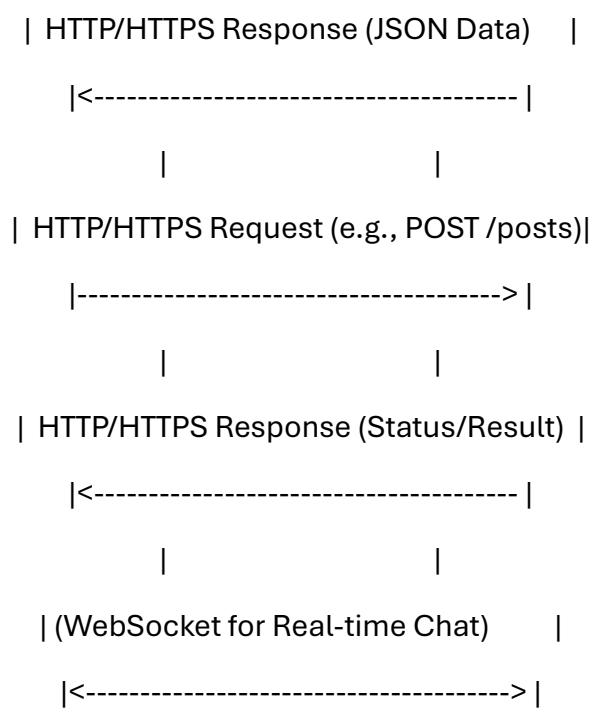
▲ 🔒 Dtos
  ▶ 🔒 C# AddBookDetailsDto.cs
  ▶ 🔒 C# BookDetailsDto.cs
  ▶ 🔒 C# BookProfile.cs
  ▶ 🔒 C# BookSearchResultDto.cs
  ▶ 🔒 C# CategoryDto.cs
  ▶ 🔒 C# GetAllBookDetailsDto.cs
  ▶ 🔒 C# PaginatedBookDto.cs
  ▶ 🔒 C# BookService.cs
  ▶ 🔒 C# IBookService.cs
▲ 🔒 CommunityService
  ▲ 🔒 Dtos
    ▶ 🔒 C# AssignModeratorDto.cs
    ▶ 🔒 C# CommunityDto.cs
    ▶ 🔒 C# CommunityMemberDto.cs
    ▶ 🔒 C# CommunityPostDto.cs
    ▶ 🔒 C# CommunityProfile.cs

```

Client-Server Communication

The client and server will communicate asynchronously via HTTP/HTTPS requests. The client will send requests to specific API endpoints on the server, and the server will respond with data, typically in JSON format.





Login

```

46
47     2 references
48     public async Task<UserDto> Login(LoginDto input)
49     {
50
51         var user = await _userManager.Users
52             .Include(u => u.UserProfile)
53             .FirstOrDefaultAsync(u => u.Email == input.Email);
54         if (user == null)
55             return null;
56
57         var result = await _signInManager.CheckPasswordSignInAsync(user, input.Password, false);
58
59         if (!result.Succeeded)
60             throw new Exception("User Not Found");
61
62         if (user.IsBlocked)
63             throw new Exception("User is blocked and cannot log in.");
64
65         string? profilePicture = null;
66         if (user.UserProfile?.ProfilePhoto != null)
67         {
68             profilePicture = $"{_baseImageUrl}{user.UserProfile.ProfilePhoto}";
69         }
70

```

```

        return new UserDto
    {
        Id = Guid.Parse(user.Id),
        FirstName = user.firstName,
        LastName = user.LastName,
        UserName = user.UserName,
        Email = user.Email,
        Gender = user.Gender,
        Age = user.DateOfBirth.HasValue ? CalculateAge(user.DateOfBirth) : null
        Token = await _tokenService.GenerateJwtToken(user),
        ProfilePicture = profilePicture
    };
}

```

Create post

```

2 references
public async Task<CommunityPostDto> CreatePostAsync(CreatePostDto dto, string userId)
{
    var community = await _unitOfWork.Repository<Community>().GetByIdAsync(dto.CommunityId);
    if (community == null)
    {
        throw new Exception("Community not found.");
    }

    var user = await _userManager.FindByIdAsync(userId);
    if (user == null)
    {
        throw new Exception("User not found.");
    }

    var isMember = (await _unitOfWork.Repository<CommunityMember>().GetAllAsync())
        .Any(m => m.CommunityId == dto.CommunityId && m.UserId == userId);

    if (!isMember)
    {
        throw new Exception("Only community members can create posts.");
    }

    var post = new CommunityPost
    {
        Content = dto.Content,
        UserId = userId,
        CommunityId = dto.CommunityId,
        CreatedAt = DateTime.UtcNow,
        CommentCount = 0
    };
}

```

```

};

if (dto.ImageFile != null && dto.ImageFile.Length > 0)
{
    var imagesFolder = Path.Combine(Directory.GetCurrentDirectory(), "wwwroot", "post-images");
    if (!Directory.Exists(imagesFolder))
    {
        Directory.CreateDirectory(imagesFolder);
    }

    var uniqueFileName = $"{Guid.NewGuid()}-{dto.ImageFile.FileName}";
    var filePath = Path.Combine(imagesFolder, uniqueFileName);

    using (var stream = new FileStream(filePath, FileMode.Create))
    {
        await dto.ImageFile.CopyToAsync(stream);
    }

    post.ImageUrl = $"/post-images/{uniqueFileName}";
}

await _unitOfWork.Repository<CommunityPost>().AddAsync(post);

community.PostCount = (community.PostCount ?? 0) + 1;
UnitOfWork.Repository<Community>().Update(community);

await _unitOfWork.CountAsync();

var postDto = _mapper.Map<CommunityPostDto>(post);
postDto.UserName = user != null ? $"{user.firstName} {user.lastName}" : "Unknown";
postDto.CommunityName = community.Name;
return postDto;
}

```

2 references

Exchange Books

```

4 references
public interface IExchangeBooks
{
    2 references
    Task CreateExchangeRequestAsync(string userId, CreateExchangeRequestDto requestDto);
    2 references
    Task<List<ExchangeRequestDto>> GetAllPendingExchangeRequestsAsync(string currentUserId);
    2 references
    Task<bool> AcceptExchangeRequestAsync(string acceptingUserId, AcceptExchangeRequestDto acceptDto);
    2 references
    Task<ExchangeRequestDto> GetExchangeRequestByIdAsync(long requestId);
}

```

```

2 references
public async Task<bool> AcceptExchangeRequestAsync(string acceptingUserId, AcceptExchangeRequestDto acceptDto)
{
    var request = await _unitOfWork.Repository<ExchangeBookRequestx>().GetByIdAsync(acceptDto.RequestId);
    if (request == null || request.IsAccepted == true)
    {
        return false;
    }

    request.IsAccepted = true;
    request.ReceiverUserId = acceptingUserId;
    var receiver = await _userManager.FindByIdAsync(acceptingUserId);
    request.ReceiverName = receiver != null ? $"{receiver.FirstName} {receiver.LastName}" : null;

    _unitOfWork.Repository<ExchangeBookRequestx>().Update(request);
    await _unitOfWork.CountAsync();
    return true;
}

2 references
public async Task CreateExchangeRequestAsync(string userId, CreateExchangeRequestDto requestDto)
{
    var exchangeRequest = _mapper.Map<ExchangeBookRequestx>(requestDto);
    exchangeRequest.SenderUserId = userId;
    var sender = await _userManager.FindByIdAsync(userId);
    exchangeRequest.SenderName = sender != null ? $"{sender.FirstName} {sender.LastName}" : null;

    await _unitOfWork.Repository<ExchangeBookRequestx>().AddAsync(exchangeRequest);
    await _unitOfWork.CountAsync();
}

```

Book Service (*using paginated for improve loading speed*)

```

10   4 references
11   public interface IBookService
12   {
13       2 references
14       Task AddBookAsync(AddBookDetailsDto addBookDetailsDto);
15       2 references
16       Task DeleteBookAsync(long id);
17       2 references
18       Task RemoveBookCoverAsync(long id);
19       2 references
20       Task<IReadOnlyList<GetAllBookDetailsDto>> SearchBooksAsync(string term);
21
22       2 references
23       Task<IReadOnlyList<GetAllBookDetailsDto>> GetAllBooksAsync();
24       2 references
25       Task<BookDetailsDto> GetBookByIdAsync(long id);
26       2 references
27       Task UpdateBookAsync(BookDetailsDto bookDetailsDto);
28       2 references
29       Task<PaginatedBookDto> GetAllBooksAsyncUsingPaginated(int pageIndex, int pageSize);
30       2 references
31       Task<IReadOnlyList<CategoryDto>> GetAllCategoriesAsync();
32   }

```

Chat

```
[HttpPost("send")]
0 references
public async Task<IActionResult> SendMessage([FromBody] SendMessageDto messageDto)
{
    var senderId = User.FindFirst(System.Security.Claims.ClaimTypes.NameIdentifier)?.Value;
    if (string.IsNullOrEmpty(senderId) || string.IsNullOrEmpty(messageDto.ReceiverId))
        return BadRequest("Sender or Receiver ID is missing.");

    var message = new ChatMessage
    {
        SenderId = senderId,
        ReceiverId = messageDto.ReceiverId,
        Message = messageDto.Message
    };

    await _unitOfWork.Repository<ChatMessage>().AddAsync(message);
    await _unitOfWork.CountAsync();

    await _chatHub.Clients.User(messageDto.ReceiverId).SendAsync("ReceiveMessage", senderId, message.Message);

    return Ok(new { MessageId = message.Id });
}
```

Notification

```
public class NotificationHub : Hub
{
    private readonly OnlineLibraryIdentityDbContext _dbContext;

    0 references
    public NotificationHub(OnlineLibraryIdentityDbContext dbContext)
    {
        _dbContext = dbContext;
    }

    0 references
    public override async Task OnConnectedAsync()
    {
        var userId = Context.User?.Identity?.Name;
        if (string.IsNullOrEmpty(userId))
        {
            throw new HubException("User not authenticated");
        }

        await Groups.AddToGroupAsync(Context.ConnectionId, "AllUsers");

        var communityMemberships = _dbContext.CommunityMembers
            .Where(cm => cm.UserId == userId)
            .Select(cm => cm.CommunityId)
            .ToList();

        foreach (var communityId in communityMemberships)
        {
            await Groups.AddToGroupAsync(Context.ConnectionId, $"Community_{communityId}");
        }
    }

    await base.OnConnectedAsync();
}
```

Text-to-Speech Translation “Machine Learning Integration”

```
# Convert translated text to speech
tts = gTTS(text=translated_text, lang=selected_target_lang, slow=False)
# Save to static/audio directory with proper path handling
audio_filename = f"output_{int(time.time())}.mp3"
audio_save_path = os.path.join(audio_dir, audio_filename)
tts.save(audio_save_path)
# Use forward slashes for URL path
audio_path_relative = f"audio/{audio_filename}"
conversion_time = round(time.time() - start_time, 2)

except Exception as e:
    error_message = f"حدث خطأ أثناء الترجمة أو تحويل النص إلى كلام: {e}"
    print(f"Error: {e}") # Log error to console
else:
    # Handle cases where the form might be submitted without text_input (e.g., initial load)
    pass

return render_template("index.html",
                      # transcript=transcript, # Removed STT variable
                      audio_path_relative=audio_path_relative,
                      conversion_time=conversion_time,
                      selected_target_lang=selected_target_lang,
                      input_text=input_text, # Pass input text back to template
                      error_message=error_message)
```

```
# دالة لتحويل الوصوت إلى ملف الصوت (تحويل المسار)
@app.route("/static/audio/<path:filename>")
def get_audio(filename):
    return send_from_directory(audio_dir, filename)

# API endpoint for text-to-speech
@app.route("/api/tts", methods=["POST"])
def text_to_speech_api():
    try:
        data = request.get_json()
        if not data or 'text' not in data:
            return jsonify({"error": "No text provided"}), 400

        input_text = data['text']
        selected_target_lang = data.get('lang', 'en') # Default to English if not specified

        # Translate text
        translator = Translator()
        translated = translator.translate(input_text, dest=selected_target_lang)
        translated_text = translated.text

        # Convert translated text to speech
        tts = gTTS(text=translated_text, lang=selected_target_lang, slow=False)

        # Save to static/audio directory
        audio_filename = f"output_{int(time.time())}.mp3"
        audio_save_path = os.path.join(audio_dir, audio_filename)
        tts.save(audio_save_path)

        # Return the audio file URL
        audio_url = f"/static/audio/{audio_filename}"

        return jsonify({
            "success": True,
            "audio_url": audio_url,
            "translated_text": translated_text
        })
    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

Book Recommendation

```
97     if best_match is None:
98         البحث في المحتوى إذا لم تجد نتائج في العنوان
99         # جاري البحث في محتوى الكتاب عن "book_title"
100        print(f"جارى البحث في محتوى الكتاب عن '{book_title}'")
101        filtered_books = data[
102            (data['text'].str.contains(book_title, case=False, na=False))
103        ]
104
105        recommendations = []
106        for _, row in filtered_books.iterrows():
107            # كتاب بدرجة التشابه مع نفس المدخل
108            text_similarity = cosine_similarity(
109                vectorizer.transform([book_title]),
110                vectorizer.transform([row['text']]))
111            )[[0]][0]
112
113            # تعرض فقط الكتب التي لديها درجة تشابه مطلقة #
114            if text_similarity > 0.1: # يمكنك تحويل هذه القيمة حسب الحاجة
115                recommendations.append({
116                    'ID': row.get('id', None),
117                    'title': row['title'],
118                    'similarity': round(text_similarity * 100, 2),
119                    'rating': float(row.get('Rating', 0)),
120                    'cover': row.get('Cover', '') if pd.notna(row.get('Cover')) else ''
121                })
122
123            # الترتيب حسب درجة التشابه ثم التقييم #
124            recommendations = sorted(recommendations, key=lambda x: (x['similarity'],
125                x['rating']), reverse=True)
126
127            # تأخذ أفضل 10 توصيات فقط
128            recommendations = recommendations[:10]
129
130        # إذا وجدنا نتائج في العنوان، نستخدم مصفوفة التشابه
131        book_index = index
132        similar_books = list(enumerate(similarity_matrix[book_index]))
133        similar_books = sorted(similar_books, key=lambda x: x[1], reverse=True)[1:11] # اخذ
134        # أفضل 10 توصيات
```

```

134
135     recommendations = []
136     for idx, sim_score in similar_books:
137         row = data.iloc[idx]
138         recommendations.append({
139             'ID': row.get('id', None),
140             'title': row['title'],
141             'similarity': round(sim_score * 100, 2),
142             'rating': float(row.get('Rating', 0)),
143             'cover': row.get('Cover', '') if pd.notna(row.get('Cover')) else ''
144         })
145
146     return best_match, recommendations, []
147
148 # الصفحة الرئيسية وعرض النتائج
149 @app.route('/')
150 def index():
151     return render_template('index.html')
152
153 # صفحة البحث عن الكتب
154 @app.route('/recommend', methods=['POST'])
155 def recommend():
156     book_title = request.form['book_title']
157     best_match, recommendations, suggestions = recommend_books(book_title, df,
158     similarity_matrix)
159
160     return render_template(
161         'recommendations.html',
162         best_match=best_match,
163         recommendations=recommendations,
164         suggestions=suggestions,
165         original_title=book_title # تحرير لاما هنا
166     )
167
168 # صفحة التفاصيل
169 @app.route('/book/<int:book_id>')
170 def book_details(book_id):
171     try:
172         book = df.loc[df['id'] == book_id].iloc[0]
173         print(f"Debug - Book data: {book.to_dict()}") # لقطة سطر التحكم
174
175         # التشكيل من وجود الموردة
176         cover_url = book.get('Cover', '')
177         if pd.isna(cover_url) or cover_url == '':
178             cover_url = 'https://via.placeholder.com/300x450?text=No+Cover'
179
180         book_data = {
181             'title': book['title'] if pd.notna(book['title']) else '',
182             'author': book['author'].strip("") if pd.notna(book['author']) else '',
183             'category': book['Category'] if pd.notna(book['Category']) else 'غير معروض', # معرف
184             'language': 'English',
185             'summary': book['Summary'] if pd.notna(book['Summary']) else ''
186         }
187
188     except Exception as e:
189         return f"An error occurred: {str(e)}"
190
191     return render_template('book_detail.html', book=book_data)

```

4.2 Pseudocode, Flowchart or workflow

4.2.1.1 Pseudocode

- **Pseudocode for Admin Moderation**

```
IF user.role == 'admin':  
    ENABLE CRUD_Operations(Books, Categories)  
    MONITOR UserPosts:  
        IF post.violates_guidelines:  
            DELETE_POST  
            NOTIFY_USER  
            OPTIONALLY BLOCK_USER
```

- **Pseudocode for Book Swap Feature**

// CLIENT SIDE

User A selects a book and clicks "Swap Request"

```
→ Client sends POST /api/Exchange/createrequest  
{  
    senderId: userA,  
    receiverId: userB,  
    bookId: ___,  
    message: Request body  
}
```

→ Server receives and validates request

→ Server saves swap request in database with status = "Pending"
→ Server sends notification to User B

User B checks swap requests

→ Client sends GET /api/Exchange/GetPendingRequests
→ Server returns list of pending requests

User B accepts/rejects a request

```
→ Client sends POST /api/ Exchange /GetRequestById  
{  
    requestId: _____,
```

- **Pseudocode: Book Recommendation**

// CLIENT SIDE

User enters a keyword, book title, or selects a book
→ Client sends GET /api/recommendations?

// SERVER SIDE

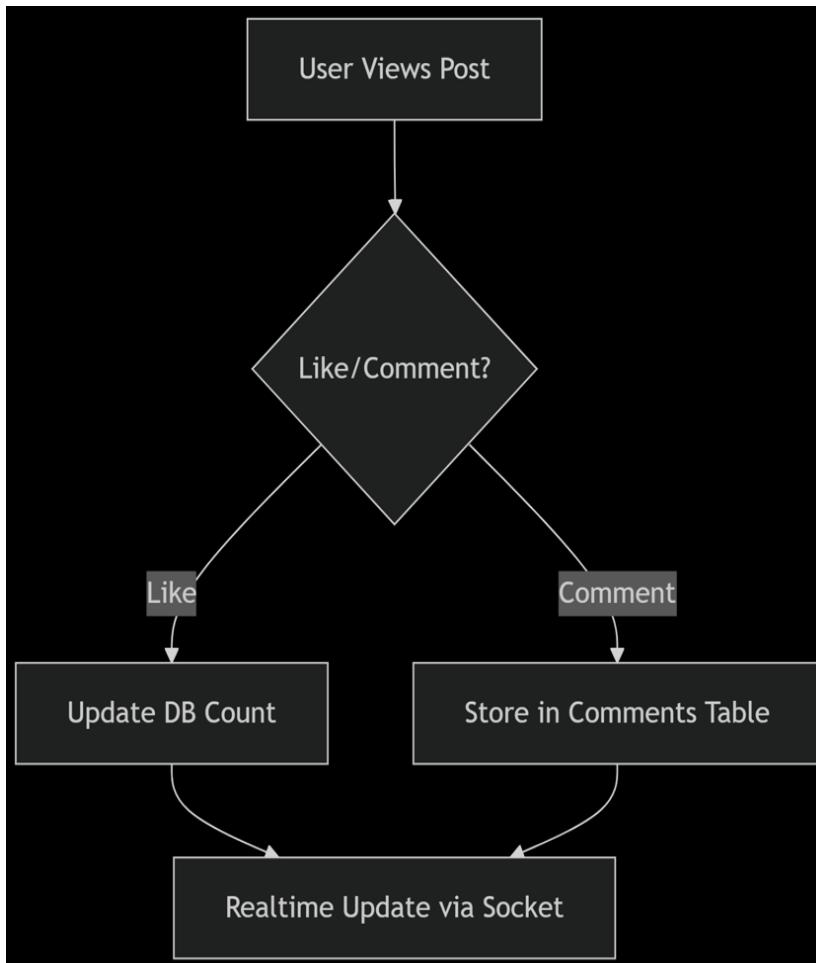
1. Receive input from client
2. Preprocess the input:
 - Tokenize text
 - Lowercase, remove stop words, stemming, etc.
3. Vectorize input using the same method used to train the model (e.g., TF-IDF or word embeddings)
4. Pass the vector to the trained ML model (e.g., cosine similarity or neural network)
 - ML model returns a list of book IDs that are most similar to the input
5. Fetch book details (title, author, cover, etc.) from the database using the recommended book IDs
6. Send the recommended book list back to the client as a JSON response

// CLIENT SIDE

7. Display the recommended books in a visually appealing layout

4.2.1 .2 workflow

Post service



4.2.2 Communication Steps Between Client and Server

1 User Action:

The user performs an action on the client side (e.g., logs in, opens a book, sends a post).

2 HTTP Request Sent:

The client sends an HTTP request (e.g., GET, POST, PUT, DELETE) to the server's API endpoint with necessary data (e.g., credentials, book ID, post content).

3 Server Receives Request:

The backend receives the request through a controller and processes it using business logic.

4 Data Validation & Processing:

The server validates the request data, interacts with the database using ORM (like Entity Framework), and executes the required operations.

5 Response Creation:

After processing, the server creates an appropriate HTTP response (e.g., success message, data object, error info).

6 HTTP Response Returned:

The response is sent back to the client in JSON format.

7 Client Receives & Renders Data:

The client interprets the response and updates the UI accordingly (e.g., shows a book, confirms post creation, displays error).

Chapter 5 "Testing"

Is a method to check whether the actual software product matches expected requirements and to ensure that software product is defect free. It involves execution of software/system components using manual or automated tools to evaluate one or more properties of interest.

The purpose of software testing is to identify errors, gaps, or missing requirements in contrast to actual requirements.

5.1 Unit Testing:

Is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application by the developers. Unit Tests isolate a section of code and verify its correctness. A unit may be an individual function, method, procedure, module, or object. Unit testing is the first level of testing done before integration testing

5.2 Integrated Testing:

Is defined as a type of testing where software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated.

5.3 Additional Testing:

- **Responsive Testing:**

We tested the layout on various screen sizes (mobile, tablet, desktop) to ensure that the UI adjusts correctly.

- **Regression Testing**

We also performed regression testing to ensure that new features or updates did not negatively affect existing functionalities

Test cases:

Login & Register test cases:

id	Description	Expected	Actual
TC1	User Register with valid Data	User registers successfully and a new account created	As expected
TC2	User registers with data of already existing account	Error message that say the account is already exist	As expected
TC3	User registers with invalid email format	Error message “email is not valid”	As expected
TC4	User register with any blank field	Error message “please fill this field”	As expected
TC5	User login with valid data	User login successfully	As expected
TC6	User login with valid email and wrong password	Error message “wrong password”	As expected
Tc7	User login with invalid email and right password	Error message “invalid email”	As expected

posts test cases:

id	Description	Expected	Actual
TC8	User select a community and try to create new post by clicking on new enter the post content and click on post	Message “post created successfully” And post added in the user profile and in the community in a stack manner	As expected
TC9	User try to create post with a photo by creating new post ,uploading photo and click post	Photo uploaded Message “post created successfully “	As expected
TC10	User try to like a post	Likes counter incremented by one	As expected
TC11	User click on like button of a post he liked before	Likes counter decremented by one	As expected
TC12	User click on comment to write a comment on a specific post	Comment added to the post comments and appear in a stack manner	As expected
TC13	User try to delete a post he created	The post deleted from the community and from his profile	As expected

TC14	User try to edit a post he already posted	The post is updated with the new content	As expected
TC15	User delete his comment	Comment delete from the post	As expected

User profile test cases:

ID	Description	expected	actual
TC16	User login , open his profile and try to change the profile picture	Profile picture updated with the new photo he uploaded	As expected
TC17	User try to change the cover profile picture	Cover profile picture is updated with the new photo the user uploaded	As expected
TC18	User open profile , posts to show all his posts	all the posts the user created are shown in a stack manner from the newest to oldest	As expected
TC19	User open profile ,about to edit the personal information	User information upadated successfully with the new data	As expected
TC20	User open profile , dashboard to show his activity	User show his daily activity in minutes time stamp and show	As expected

Reading book test cases:

ID	Description	expected	Actual
TC21	User open specific book try to add a note on a page	Note added and saved in the user saved notes	As expected
TC22	User try to highlight a text from the book with a chosen color	Text highlighted with the color the user chose and saved in the highlights	As expected
TC23	User logout then login again	The notes and highlights he saved still exist without any lost	As expected

Machine learning models Test cases:

ID	Description	expected	Actual
TC24	User try to read a book the same as the content of a book he select	The top most similar ten books appear below the book	As expected
TC25	User select a specific text from the book and select a language to listen it with	The ML model detect the input text language translate it to the language the user select then repeat it in a loud voice	As expected

Admin Test cases:

ID	Description	Expected	Actual
TC26	Admin open manage books tab and try to edit an existing book	The book is updated with the new data at the client and server sides	As expected
TC27	Admin try to delete specific book	The book is deleted from both the client and server side	As expected
TC28	Admin open manage users and block specific user account	User cannot login to the system with an error message “this account is blocked try again later”	As expected
TC29	Admin try to delete a user account	Error message appear to the user that try to login “invalid data”	As expected
TC30	Admin open manage posts and delete an existing post	Post is deleted from the community it posted in and from the user's profile	As expected
TC31	Admin add community ,enter it's valid data	Message “community added successfully”	As expected

TC32	Admin add community with empty field	Error message “please fill this field is requireda”	As expected
TC33	Admin delete an existing community	Community is deleted from both client and server side	As expected
TC34	Admin edit a community information	Community is updated with the new information	As expected
TC35	Admin install the insights report	Report copy installed in pdf format	As expected

Chapter 6: Results and Discussion

This chapter presents the system's results and analyzes how closely the actual outcomes align with the expected objectives.

6.1 Results

6.1.1 Expected Results

The online library system **ktabook** was designed to offer a comprehensive digital reading and community platform. The theoretical outcomes of the system are as follows, assuming the system functions at 100% efficiency:

User Authentication:

Users are expected to be able to register and log in using a valid email and password.

Book Browsing and Reading:

Users are expected to browse all available books and read any book online.

Post Creation and Interaction:

Users are expected to create posts that may include text and images, and assign them to a specific category.

Users should also be able to like or comment on any post.

Book Swap Feature:

Users are expected to:

- Send book swap requests (as senders).
- Receive swap requests (as receivers) and communicate with senders directly.
-

Chat Feature:

Users are expected to chat with any other user in the system.

User Activity Tracking:

The system is expected to calculate:

- The total time spent on the platform.
- The total number of pages read by each user.

Book Recommendations:

When a user selects a keyword or book title, the system should recommend similar books.

Text-to-Speech with Language Detection:

When a user submits text, the system automatically detects the language and converts it to speech in any language selected by the user

Wish List:

Users are expected to add any book to their personal wish list.

Admin and Moderator Capabilities:

- Admin can add, delete, or edit any book or category.
- Admin or moderator can delete any user or post, and block users if necessary.

6.1.2 Actual Results

The actual results achieved in the project are as follows:

- All major expected features were successfully implemented and are working correctly:
 1. User authentication via email and password.
 2. Viewing and reading books online.
 3. Post creation with images and category selection.
 4. Liking and commenting on posts.
 5. Sending and receiving book swap requests, with direct communication between users.
 6. Real-time chatting functionality.
 7. Tracking user activity (time spent and pages read).
 8. Book recommendations based on keyword or title.
 9. Automatic language detection and text-to-speech conversion.
 10. Wish list feature.

11. Full admin and moderator controls as expected.

All expected functionalities were achieved

6.2 Discussion

When comparing the expected and actual results, it is clear that the project successfully achieved all its functional goals. The system operated correctly in terms of user registration, content management, interaction features, and administrative controls.

The only deviation from expectations was related to **system performance**. Some pages experienced slower load times, especially those involving large datasets or complex queries.

This performance limitation is likely due to several factors:

- The volume of data being processed and displayed on certain pages.
- Absence of performance enhancement techniques such as caching or lazy loading.
- Uploading and serving uncompressed images and files.

Therefore, it is recommended to conduct further optimization at the backend level, including:

- Optimize database queries using indexing and caching.
- Implement background services for heavy operations (e.g., book recommendation).
- Use a more scalable infrastructure (e.g., cloud hosting or microservices).

These improvements are expected to significantly improve system responsiveness and provide a smoother user experience, and we will work on that.

Chapter 7: Conclusion

In conclusion, this project aimed to create a digital library platform that not only provides access to books but also encourages reading, interaction, and personalization through modern features like recommendation systems, dashboards, and social engagement.

Despite facing some challenges, especially in dataset preparation and time management, we were able to deliver a functional and user-friendly platform that simulates the library experience in a more accessible and engaging way.

We believe that this system can contribute to promoting a reading culture, especially in today's fast-paced world where traditional libraries are becoming harder to reach.

Future improvements may include expanding the book database, enhancing recommendation algorithms, and developing a mobile version of the platform.

Chapter 8: Future work

As part of the Ktabook project,

we successfully integrated core features such as AI-powered text-to-speech, a recommendation system based on content, a personalized user dashboard, and a community interaction space. These features provide a rich and accessible reading experience for users.

- In future iterations of the project, we aim to expand the text-to-speech module by adding voice tone customization (e.g., male/female voices, emotional tone), and allowing users to control the playback speed for a more personalized audio experience.
- Regarding the recommendation system, we plan to enhance it by incorporating advanced machine learning techniques such as deep learning and collaborative filtering to offer more

accurate and real-time suggestions based on user behaviour .

- We also recommend developing a dedicated **mobile application for Ktabook**, integrating **real-time voice search**, and deploying the system on a **cloud platform** to ensure high scalability, low latency, and continuous availability.
- Additionally, we plan to integrate a **group video meeting feature** to allow users and book clubs to host live discussions, enabling real-time communication, collaborative reading and a stronger sense of community
- Finally we aim to **improve the platform's experience for children** by designing a dedicated **storytelling section** with **visual and interactive elements** to make reading more engaging and enjoyable for younger users

Stabook

Thank you