



Book Recommendations System

Graduation Project



Nour Elmergawy

Amr Ashraf

Sarah Hesham

Maryam Ebrahim

Youssef Maged

Moaaz Ismail

Ahmed Essam

Mohamed Nagi

Under Supervision
of Dr. Nora Shoip

Abstract

This abstract describes a mobile app that recommends books to users based on their reading preferences. The app is designed to provide a personalized reading experience, with a user-friendly interface that makes it easy to find and discover new books.

The app uses a sophisticated algorithm that takes into account the user's reading history, genre preferences, and ratings of previous books to generate recommendations. Users can also browse through curated lists, such as "New Releases" or "Bestsellers," or search for specific titles or authors.

The app includes a social aspect, allowing users to connect with friends and share book recommendations. Users can also create their own reading lists and track their progress as they read. The app includes features such as book summaries, reviews, and ratings from other users, as well as links to purchase books directly from the app. It also offers personalized recommendations for audiobooks and e-books, making it a comprehensive reading companion for users on the go.

Overall, this book recommendation mobile app is an essential tool for book lovers who want to discover new titles and connect with other readers. With its intuitive design and personalized recommendations, it is sure to become a go-to resource for anyone looking to expand their reading horizons.

In the era of digital content consumption, finding the right book to read can be overwhelming due to the vast amount of available options. Our project aims to develop a book recommendation system using unsupervised machine learning techniques. This system will be applied to a mobile app to enhance the user experience by providing personalized book recommendations for both new and existing users.

Contents

Graduation Project	2
Abstract	3
Chapter I Introduction	8
1.4 Overall descriptions.....	10
1.4.1 Product perspective.....	10
1.4.2 Product functions	11
1.4.3 User characteristics	12
1.4.4 Constraints	12
1.4.5 Assumptions and Dependencies	13
System requirement and analysis	15
2.1. Functional Requirements.....	16
2.1.1. User account creation and management	16
2.1.2. User preferences and reading history collection.....	16
2.1.3. Book data source integration	16
2.1.4. Machine learning-based recommendation engine	16
2.1.5. Book browsing and recommendation interface	16
2.1.6. Book search functionality	17
2.1.7. External book purchasing links	17
2.1.8. User feedback and rating system	17
2.2. Non-functional requirements.....	17
2.2.1. Security	17
2.2.2. Storage	18
2.2.3. Easy to handle.....	18
2.3. Domain and other requirements	18
2.3.1. Domain requirements.....	18
2.3.2. Legal requirements	18
2.3.3. Integration with other systems.....	18
2.3.4. System structure.....	19
2.3.5. Reliability	19
3.1 Introduction	21
3.2 UML Diagram	21
3.3 Use-Case Diagram.....	22
3.4 Use-case scenario	22
3.5 Class Diagram	24

Activity Diagram	26
Sequence Diagram	27
Sequence Diagram	27
3.6 Data flow diagrams.....	28
3.6.1 DFD Symbols	29
3.6.2 Constructing a DFD	30
Several rules of thumb are used in drawings of DFD:	30
3.6.3 Salient features of DFD	30
3.6.4 Types of data flow Diagram	31
3.6.5 Current Physical.....	31
3.6.6 Current Logical.....	31
3.6.7 New Logical.....	31
3.6.8 New Physical	31
3.6.9 Rules governing the DFD	32
3.6.9.1 Process.....	32
3.6.9.2 Data store	32
3.6.9.3 Source or sink.....	32
3.6.9.4 Data flow	32
3.7 Context Diagram	33
3.8 Unified Modeling Language Diagrams	34
The primary goals in the design of the UML were as follows:	35
UML is a language used to:.....	36
3.8.1 Architectural views and diagrams of the UML	36
3.8.1.1 The user model view	36
3.8.1.2 The Structural model view	37
3.8.1.3 The behavioral model view	37
3.8.2 UML Diagrams	38
4.1 Node.js.....	40
4.2 Express.js.....	41
4.3 MongoDB	42
4.4 Mongoose	44
4.5 RESTful APIs.....	47
4.6 Android	48
4.7 Scikit-learn	50
4.8 Flask	53
4.9 GitHub	55

System Implementation	57
5.1 Mobile Application.....	58
6.1 Introduction	70
6.2 Strategic approach to software testing.....	70
6.3 Levels of testing	71
6.3.1 Unit Testing	74
6.3.1.1 White box testing	74
6.3.1.2 Conditional testing	74
6.3.1.3 Data flow testing	74
6.3.1.4 Loop testing.....	75
6.4 Methodology.....	75
6.4.1 Pros of using the methodology	76
6.4.2 Cons of using the methodology	76
6.5 Interface Testing.....	77
6.5.1 Functional requirements	77
Test Case 1:	77
Test Case 2:	77
Test Case 3:	78
Test Case 4:	79
Test Case 5:	79
Test Case 6:	80
7.1 Introduction	82
7.2 Threat Assessment.....	82
7.3 Security Objectives.....	83
7.4 Security Controls	84
7.5 Data Protection	86
7.6 Access Control.....	87
7.7 Network Security	88
Conclusion and Future Work.....	90
8.1 Conclusion.....	91
8.2 Benefits.....	93
8.3 Future work	94
Chapter IX	95
References	95

Chapter I

Introduction

1.1 Purpose

The purpose of this document is to provide a detailed description of the requirements for the Book Recommendation System Mobile App. This app aims to suggest books to users based on various factors, including their preferences, reading history, and book popularity. By leveraging unsupervised machine learning techniques, the app refines recommendations over time, offering personalized suggestions that enhance the reading experience and help users discover new books aligned with their interests.

1.2 Scope

This document covers the requirements for the Book Recommendation System Mobile App, including the overall description, specific requirements, and supporting information. The app is designed to serve readers looking for book suggestions across various genres, formats, and categories, as well as authors and publishers seeking to promote their books to a broader audience.

1.3 Background

In today's digital age, readers have access to a vast assortment of books, spanning various genres, formats, and categories. While this abundance of choices is generally positive, it can also make it challenging for readers to find books that align with their preferences and interests. Furthermore, authors and publishers face increased competition to capture the attention of readers, leading to a need for effective book promotion and discovery tools.

According to recent statistics, the literacy rate in Egypt has shown a decline. In 2017, the literacy rate was 71.17%, a 1.27% decrease from 2013. This

decline highlights the need for effective, flexible, and versatile tools to encourage people to read and recommend valuable books for those who are already readers. The goal is to upgrade the readers' community, which will contribute to improving and enlightening society.

Machine learning-based recommendation systems have gained popularity in recent years due to their ability to analyze large datasets and identify patterns to provide personalized suggestions. These systems have been successfully implemented by major online retailers and content platforms to improve user engagement and satisfaction. The

Book Recommendation System Mobile App seeks to address these challenges by providing a tailored reading experience for users, leveraging unsupervised machine learning techniques to suggest books based on users' preferences, reading history, and book popularity.

1.4 Overall descriptions

1.4.1 Product perspective

The Book Recommendation System Mobile App is an independent application designed to provide personalized book suggestions to users, enhancing their reading experience and promoting a diverse range of books to a larger audience. The app integrates with various book data sources, such as online bookstores, public APIs, and publisher databases, to gather information about books, including genres, authors, publication dates, and user ratings.

The app's recommendation engine uses unsupervised machine learning algorithms to analyze user preferences, reading history, and book popularity to generate tailored book suggestions. By offering personalized recommendations, the app can help users discover new books that match their interests, while also promoting a diverse range of books to a broader audience.

Instead of allowing users to purchase books directly within the app, the Book Recommendation System Mobile App facilitates the process by linking to external sources where users can buy the recommended books. This functionality helps users easily acquire books that match their interests, while also supporting authors and publishers by promoting their books to a broader audience.

By leveraging advanced machine learning techniques and a user-friendly interface, the Book Recommendation System Mobile App aims to create an engaging and enjoyable reading experience for users while contributing to the growth and enrichment of the readers' community in Egypt.

1.4.2 Product functions

The Book Recommendation system will provide the following functions:

- **User profiling:** The system should analyze user profiles to understand their interests and preferences in books
- **Book recommendations:** The system should provide personalized book recommendations based on user profiling, and can use various techniques such as collaborative filtering, content-based filtering, and hybrid models to generate recommendations.”
- **Book information display:** The system should display comprehensive book information, including author, summary, and previous reader ratings.
- **Favorites list:** Users can add their favorite books to their own favorites list. The system can use this information to improve its recommendations for users and provide a better reading experience for users to read their favorite books.
- **Purchase support:** The mobile application can support the reader in purchasing the book they like from websites that provide online purchasing

services. This can be done by providing direct links to websites that sell the book, or the application can integrate different e-bookstores within the app itself to facilitate the purchasing process.

1.4.3 User characteristics

The Book Recommendation System Mobile App targets a broad user base, including:

- **Casual readers:** Individuals who read occasionally and are looking for book suggestions to match their interests and preferences.
- **Avid readers:** Frequent readers who want to discover new books and authors that align with their reading habits and tastes.
- **Book clubs:** Groups of readers who want to find books to read and discuss together.
- **Authors and publishers:** Professionals in the publishing industry who want to promote their books to a wider audience and get insights into reader preferences.

1.4.4 Constraints

The development and deployment of the Book Recommendation System Mobile App may be subject to several constraints that can impact its functionality, performance, and overall success. These constraints include:

- **Scalability:** The system should be scalable and able to handle an increasing volume of user data and traffic over time.
- **Usability:** The system should be user-friendly and easy to navigate, with clear instructions and a minimal learning curve for users with varying levels of technical expertise.
- **Security:** The system should have robust security features to protect user data from unauthorized access or breaches, and ensure secure transmission of data over the internet.
- **Technology:** The app relies on machine learning algorithms and data

processing capabilities to provide personalized book recommendations. The performance of these algorithms may be limited by available computing resources or the need to support a wide range of mobile devices with varying hardware specifications.

- **Data quality and availability:** The quality and accuracy of the book recommendations depend on the availability and reliability of book data sources. Constraints in accessing comprehensive, up-to-date, and accurate data may impact the app's ability to generate meaningful recommendations for users.

1.4.5 Assumptions and Dependencies

The Book Recommendation System Mobile App relies on several assumptions and dependencies that may impact its development, functionality, and performance. These factors include:

- **Data availability:** The app's recommendation engine depends on the availability of comprehensive and up-to-date book data from various sources, such as online bookstores, public APIs, and publisher databases. Any changes or disruptions in these data sources may affect the quality and accuracy of the recommendations provided by the app.
- **User data privacy and security:** The app collects and analyzes user preferences, reading history, and other personal information to generate personalized recommendations. It is assumed that proper measures will be implemented to ensure the privacy and security of this user data, in compliance with relevant data protection regulations.
- **Machine learning model performance:** The effectiveness of the app's unsupervised machine learning algorithms in providing accurate and relevant book recommendations depends on the quality of the input data and the fine-tuning of the model. It is assumed that the development team will have the necessary expertise and resources to optimize the model's performance

during the development process.

- **External bookstore and retailer integration:** The app facilitates the process of purchasing recommended books by linking to external sources where users can buy them. It is assumed that these external sources (e.g., online bookstores and retailers) will provide stable and reliable integration options to ensure a seamless user experience.
- **Platform compatibility:** The app is designed to be compatible with multiple mobile platforms and devices. This requires the development team to be familiar with the specific requirements and constraints of each platform to ensure optimal performance and user experience across all supported devices.
- **User adoption and engagement:** The success of the Book Recommendation System Mobile App depends on user adoption and engagement. It is assumed that the app's features, user interface, and overall experience will be appealing to the target audience, leading to increased adoption and usage over time.

These assumptions and dependencies should be carefully considered and monitored throughout the development process to minimize potential risks and ensure the successful implementation and deployment of the Book Recommendation System Mobile App.

Chapter II

System requirement and analysis

2.1. Functional Requirements

2.1.1. User account creation and management

- Sign up with email
- Edit profile information and preferences
- Manage reading history and favorite books

2.1.2. User preferences and reading history collection

- Collect user preferences through a questionnaire or in-app interactions
- Track user reading history based on the books added to their account

2.1.3. Book data source integration

- Access and retrieve book information from various data sources
- Update book data regularly to ensure accuracy and completeness

2.1.4. Machine learning-based recommendation engine

- Analyze user preferences, reading history, and book popularity
- Generate personalized book recommendations using unsupervised ML algorithms
- Continuously refine recommendations based on user feedback and engagement

2.1.5. Book browsing and recommendation interface

- Display book recommendations in an intuitive and visually appealing format
- Provide filtering and sorting options to customize the recommendation

view

- Offer detailed book information, including descriptions, reviews, and author information

2.1.6. Book search functionality

- Allow users to search for specific books within the app
- Display search results with relevant book information and options to add to reading history or favorites

2.1.7. External book purchasing links

- Provide direct links to external sources where users can purchase recommended books
- Ensure seamless integration with various online bookstores and retailers

2.1.8. User feedback and rating system

- Enable users to rate and provide feedback on the recommended books
- Use user feedback to refine the recommendation engine and improve the quality of suggestions

2.2. Non-functional requirements

2.2.1. Security

- The system must be secure so no one can have access to Users information
- Ensure that password rules are clear (length, special characters, . . .)
- Ensure that all data inside the database is encrypted (confidentiality and data integrity)
- Ability to configure specific allowed list of IPs' to access system

2.2.2. Storage

- The system must be able to handle large volumes of Books and users data. so the system is connected to a server in a secret place.

2.2.3. Easy to handle

- The User should find it easy to deal with the application, whether searching or Purchasing. also, the application notifies Admin if any action happens as the Author or the user add books.

2.3. Domain and other requirements

2.3.1. Domain requirements

- Admin will have access to the list of all books

2.3.2. Legal requirements

- The application follows the laws and regulations of The country

2.3.3. Integration with other systems

- Ability to integrate with other systems such as (kindle, audioable) When the user logout of the system the database must clear all data that concerns the previous user. while keeping other available times for reservation data.

2.3.4. System structure

- Ensure that system consist of more than one level (app – database), app for all user interactions and database to store all these data and interactions. System interfaces for example: User interface Author interface Admin interface

2.3.5. Reliability

- The average time to failure shall be minimum of 30 days. In the event that a server does crash, a backup server will be up and running within an hour..

Chapter III

System Design

3.1 Introduction

Software design sits at the technical kernel of the software engineering process and is applied regardless of the development paradigm and area of application. Design is the first step in the development phase for any engineered product or system. The designer's goal is to produce a model or representation of an entity that will later be built. Beginning, once system requirement has been specified and analyzed, system design is the first of the three technical activities design, code and test that is required to build and verify software.

The importance can be stated with a single word "Quality". Design is the place where quality is fostered in software development. Design provides us with representations of software that can assess for quality. Design is the only way that we can accurately translate a customer's view into a finished software product or system. Software design serves as a foundation for all the software engineering steps that follow. Without a strong design we risk building an unstable system one that will be difficult to test, one whose quality cannot be assessed until the last stage.

During design, progressive refinement of data structure, program structure, and procedural details are developed reviewed and documented. System design can be viewed from either technical or project management perspective. From the technical point of view, design is comprised of four activities architectural design, data structure design, interface design and procedural design.

3.2 UML Diagram

This Project will be represented in a Use-Case diagram, Action Diagram, Sequence diagram, and class diagram. The architecture that will be implemented is MVVM (Model-View-ViewModel), which will appear in sequence and class diagrams.

3.3 Use-Case Diagram

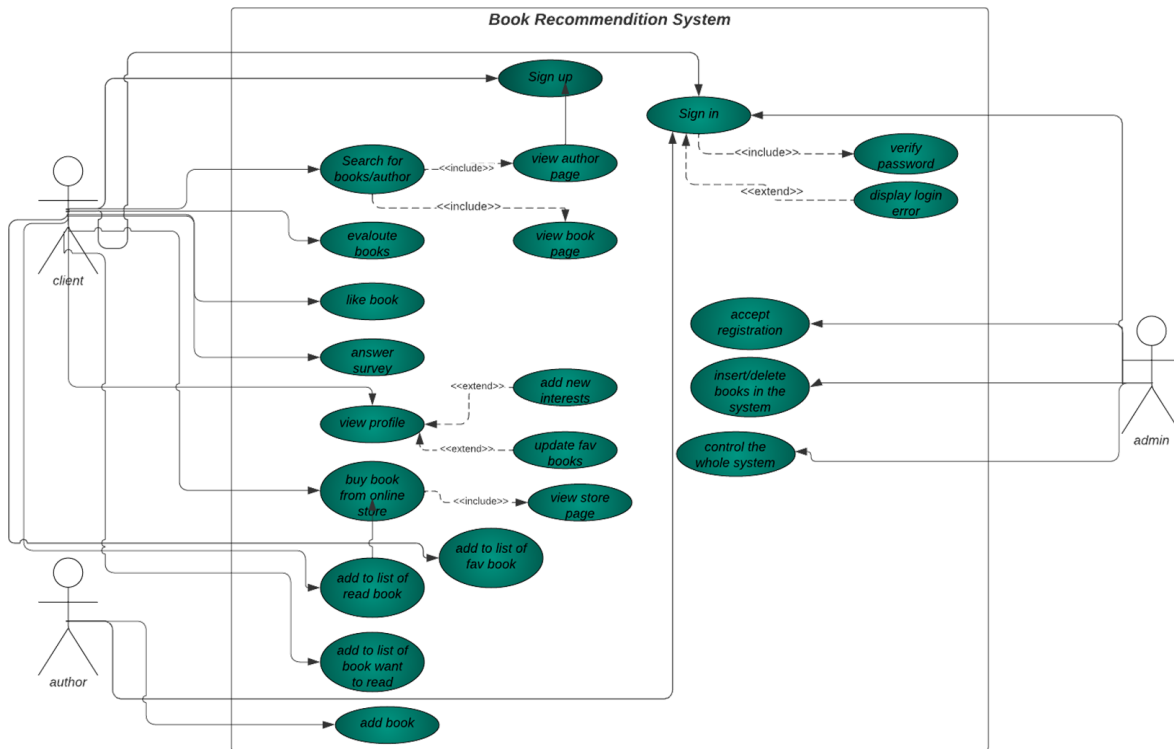


Figure 4.1: Use-Case Diagram

3.4 Use-case scenario

Name:	Registration
Actors:	clients
Pre-conditions:	client chooses to sign in or to register if he doesn't have an account
Main scenario:	Users can register to the system and profile will be created.
Alternative scenario:	1.If entered the password incorrect he will verified password. 2.If forgot password he will displayed login error and he will reset password. 3.the user has been sent a message that this data has been incorrect.
Post-conditions:	1.The user profile has been registered to the system. 2.the user profile wasn't created due to incorrect data.

Name:	Sign in
Actors:	Admin – client – Author
Pre-conditions:	Being registered in the system
Main scenario:	<ol style="list-style-type: none"> 1.user enter his data. 2.the system checks if his data is correct 3.User can sign into the system.
Alternative scenario:	<ol style="list-style-type: none"> 1.If the sign in data in not correct. 2.password is incorrect so the user can make a request to change his password.
Post-conditions:	<ol style="list-style-type: none"> 1.User can open his profile and scroll in system. 2.User can't open his profile and scroll in system.

Name:	Author add his book
Actors:	1.Author 2.Book recommendation system
Pre-conditions:	Being registered as a Author in the system
Main scenario:	<ol style="list-style-type: none"> 1.Author add book. 2.Write book information.
Alternative scenario:	1.If the Author is not have account in the system, he will not access to add any book .
Post-conditions:	<ol style="list-style-type: none"> 1.The book was sent to admin and accepted it. 2.View the book page. 3.Refused to add any book.

Name:	<i>Insert books in the system</i>
Actors:	1-Admin 2-Book recommendation system
Pre-conditions:	New book is added to the system.
Main scenario:	<ol style="list-style-type: none"> 1.Book's information is sent to the admin. 2.The admin cerates the book's page .
Alternative scenario:	<ol style="list-style-type: none"> 1.The System already has the book page. 2.The System refuses to create the book page. 3.The System sends message to the admin that the book is already created.
Post-conditions:	<ol style="list-style-type: none"> 1. book page has been inserted successfullu to the system 2.the System refuses to create the book page.

Name:	Interactive with system
Actors:	1.client 2.Book recommendation system
Pre-conditions:	Client being registered in the system
Main scenario:	<ol style="list-style-type: none"> 1. do the questionnaire 2.Search for book or author 3. Rate books 4. view his profile 5. add to list of read book 6. add to list of fav book 7. add to list of book want to read 8. buy book from online store
Alternative scenario:	User can open book page or interact with it without searching for book
Post-conditions:	<ol style="list-style-type: none"> 1. User can interact with system 2. User can't interact with system

Name:	Admin delete book
Actors:	1-Admin 2-Book recommendation system
Pre-conditions:	An Book page has been made
Main scenario:	<ol style="list-style-type: none"> 1.Admin can delete any book. 3.the book deleted by admin.
Alternative scenario:	<ol style="list-style-type: none"> 1.If there is no accesses for admin to delete books . 2.The admin can't delete books
Post-conditions:	<ol style="list-style-type: none"> 1.The book has been deleted successfully. 2. Couldn't delete books.

3.5 Class Diagram

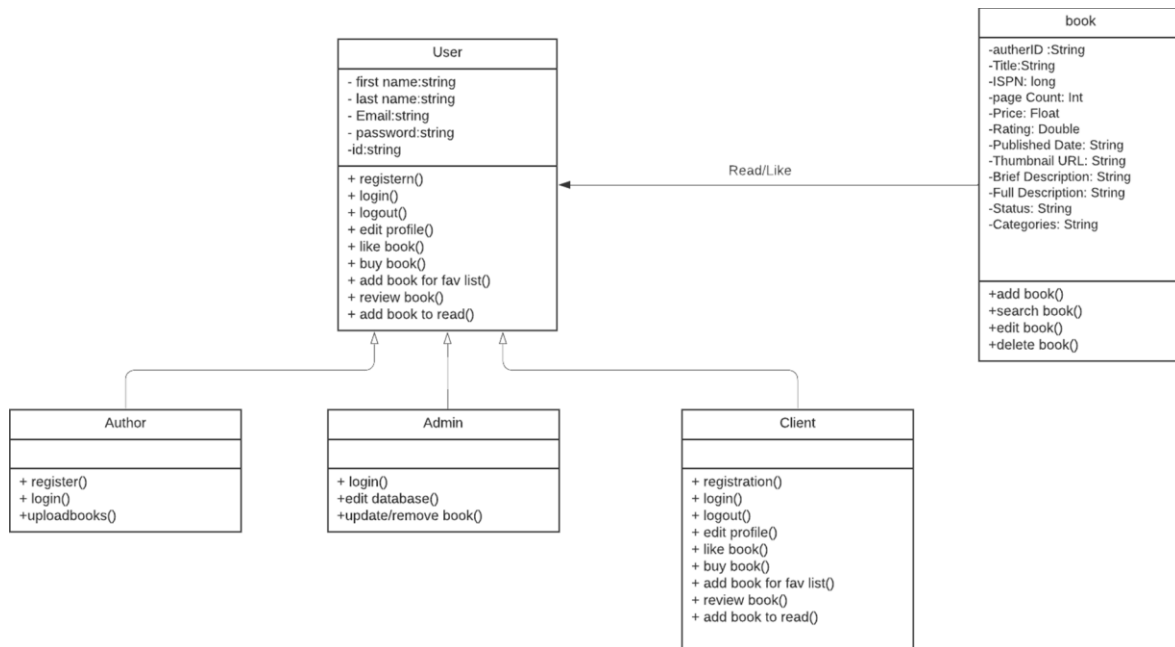


Figure 4.2: Class Diagram

Activity Diagram

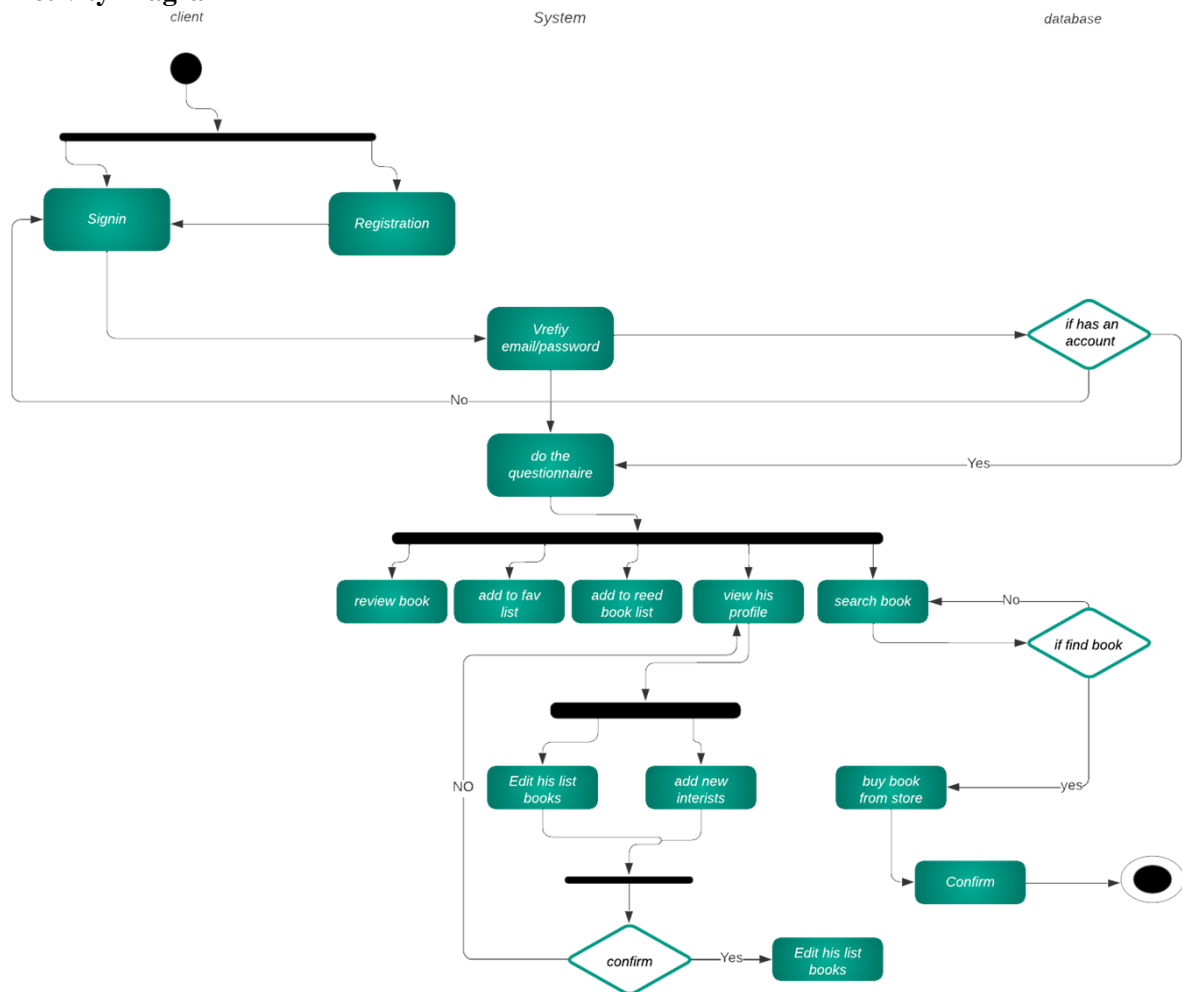


Figure 4.5: Activity Diagram

Sequence Diagram

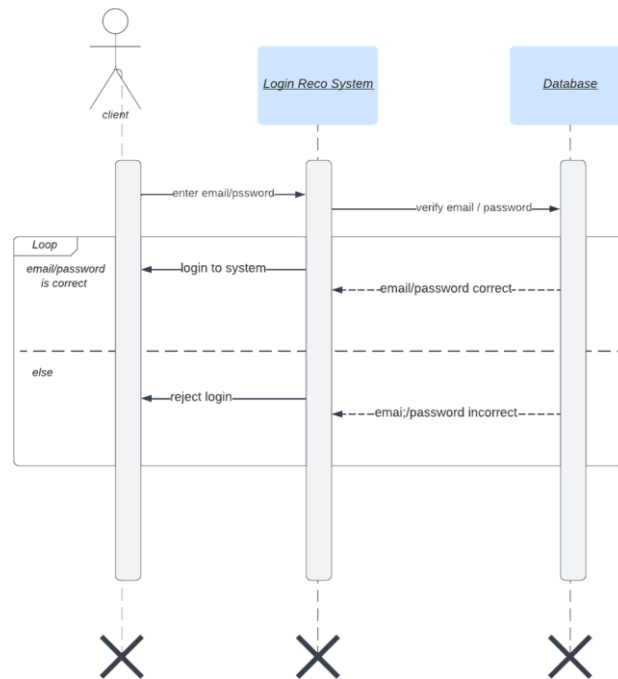


Figure 4.6: Sequence Diagram for the Login Process

Sequence Diagram

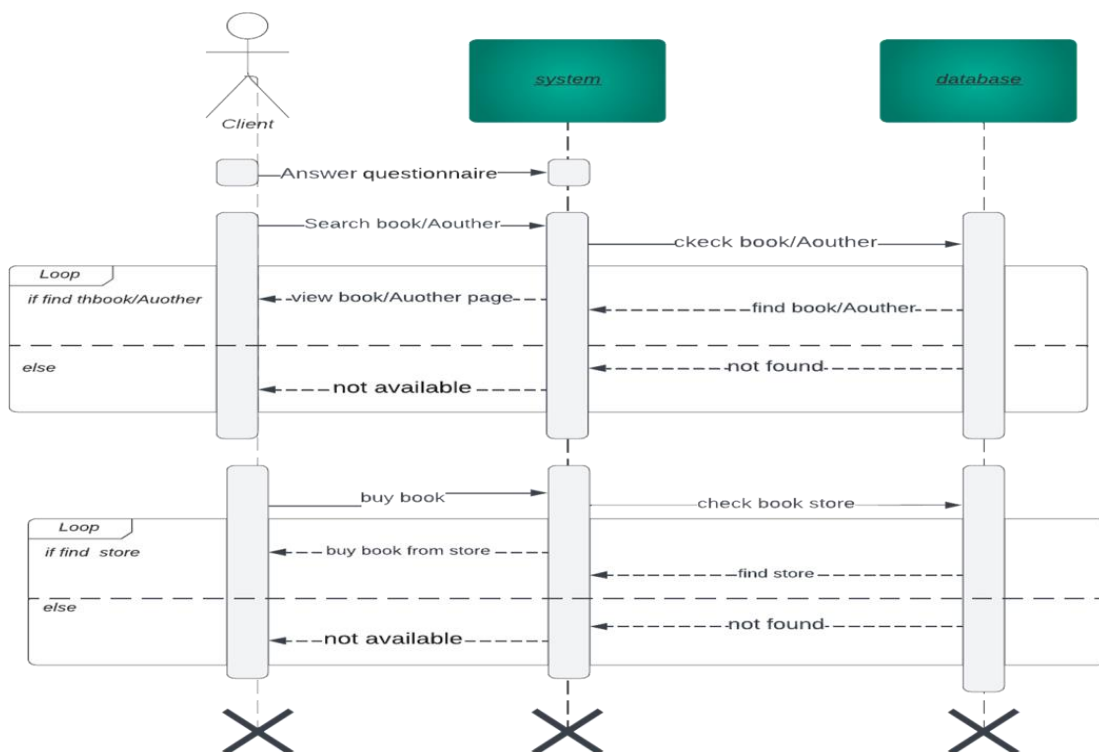


Figure 4.8: Activity Diagram for the Profile Management

3.6 Data flow diagrams

A data flow diagram is graphical tool used to describe and analyze movement of data through a system. These are the central tool and the basis from which the other components are developed. The transformation of data from input to output, through processed, may be described logically and independently of physical components associated with the system. These are known as the logical data flow diagrams. The physical data flow diagrams show the actual implements and movement of data between people, departments and workstations. A full description of a system actually consists of a set of data flow diagrams. Using two familiar notations Yourdon, Gane and Sarson notation develops the data flow diagrams. Each component in a DFD is labeled with a descriptive name. Process is further identified with a number that will be used for identification purpose. The development of DFD'S is done in several levels. Each process in lower- level diagrams can be broken down into a more detailed DFD in the next level. The top-level diagram is often called context diagram. It consists a single process bit, which plays vital role in studying the current system. The process in the context level diagram is exploded into other process at the first level DFD.

The idea behind the explosion of a process into more process is that understanding at one level of detail is exploded into greater detail at the next level. This is done until further explosion is necessary and an adequate amount of detail is described for analyst to understand the process.

Larry Constantine first developed the DFD as a way of expressing system requirements in a graphical from, this leads to the modular design.

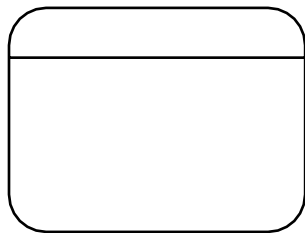
A DFD is also known as a "bubble Chart" has the purpose of clarifying system requirements and identifying major transformations that will become programs in system design. So, it is the starting point of the design to the

lowest level of detail. A DFD consists of a series of bubbles joined by data flows in the system.

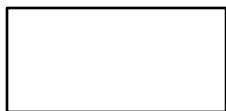
3.6.1 DFD Symbols

In the DFD, there are four symbols:

1. A square defines a source(originator) or destination of system data
2. An arrow identifies data flow. It is the pipeline through which the information flows
3. A circle or a bubble represents a process that transforms incoming data flow into outgoing data flows.
4. An open rectangle is a data store, data at rest or a temporary repository of data



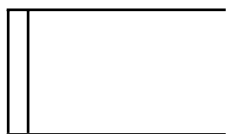
Process that transforms data flow.



Source or Destination of data



Data flow



Data Store

3.6.2 Constructing a DFD

Several rules of thumb are used in drawings of DFD:

1. Process should be named and numbered for an easy reference. Each name should be representative of the process.
2. The direction of flow is from top to bottom and from left to right. Data traditionally flow from source to the destination although they may flow back to the source. One way to indicate this is to draw long flow line back to a source. An alternative way is to repeat the source symbol as a destination. Since it is used more than once in the DFD it is marked with a short diagonal.
3. When a process is exploded into lower-level details, they are numbered.
4. The names of data stores and destinations are written in capital letters. Process and dataflow names have the first letter of each word capitalized

A DFD typically shows the minimum contents of data store. Each data store should contain all the data elements that flow in and out.

Questionnaires should contain all the data elements that flow in and out.

Missing interfaces redundancies and like is then accounted for often through interviews.

3.6.3 Salient features of DFD

1. The DFD shows flow of data, not of control loops and decision are controlled considerations do not appear on a DFD.
2. The DFD does not indicate the time factor involved in any process whether the dataflow take place daily, weekly, monthly or yearly.
3. The sequence of events is not brought out on the DFD.

3.6.4 Types of data flow Diagram

1. Current physical
2. Current Logical
3. New Logical
4. New Physical

3.6.5 Current Physical

In Current Physical DFD processes label include the name of people or their positions or the names of computer systems that might provide some of the overall system-processing label includes an identification of the technology used to process the data. Similarly, data flows and data stores are often labels with the names of the actual physical media on which data are stored such as file folders, computer files, business forms or computer tapes.

3.6.6 Current Logical

The physical aspects at the system are removed as such as possible so that the current system is reduced to its essence to the data and the processors that transform them regardless of actual physical form.

3.6.7 New Logical

This is exactly like a current logical model if the user were completely happy with the user were completely happy with the functionality of the current system but had problems with how it was implemented typically through the new logical model will differ from current logical model while having additional functions, absolute function removal and inefficient flows recognized.

3.6.8 New Physical

The new Physical represents only the physical implementation of the new system.

3.6.9 Rules governing the DFD

3.6.9.1 Process

- No process can have only outputs.
- No process can have only inputs. If an object has only inputs than it must be a sink.
- A process has a verb phrase label.

3.6.9.2 Data store

- Data cannot move directly from one data store to another data store, a process must move data.
- Data cannot move directly from an outside source to a data store, a process, which receives, must move data from the source and place the data into data store.
- A data store has a noun phrase label.

3.6.9.3 Source or sink

The origin and /or destination of data

- Data cannot move directly from a source to sink to must be moved by a process
- A source and /or sink has a noun phrase land

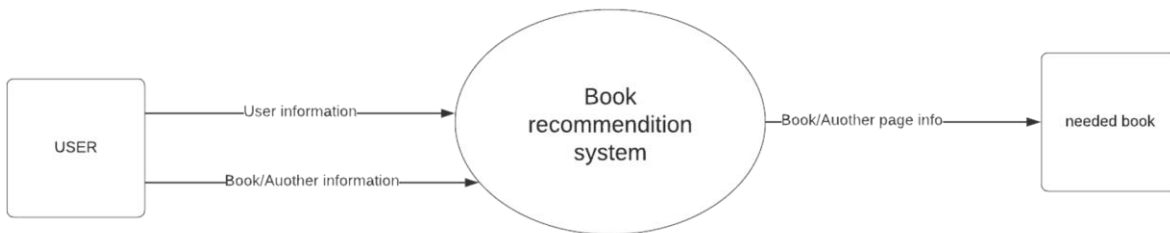
3.6.9.4 Data flow

1. A Data Flow has only one direction of flow between symbols. It may flow in both directions between a process and a data store to show a read before an update. The later is usually indicated however by two separate arrows since these happen at different type.
2. A join in DFD means that exactly the same data comes from any of two or more different processes data store or sink to a common location.

3. A data flow cannot go directly back to the same process it leads.
There must be at least one other process that handles the data flow produce some other data flow returns the original data into the beginning process.
4. A Data flow to a data store means update (delete or change).
5. A data Flow from a data store means retrieve or use.

A data flow has a noun phrase label more than one data flow noun phrase can appear on a single arrow as long as all of the flows on the same arrow move together as one package.

3.7 Context Diagram



3.8 Unified Modeling Language Diagrams

Building a model for a software system prior to its construction is as essential as having a blueprint for building a large building. Good models are essential for communication among project teams. As the complexity of the systems increases, so does the importance of good modeling techniques.

The use of visual notation to represent or model a problem can provide us several benefits relating to clarity, familiarity, maintenance, and simplification. The main reason for modeling is the reduction of complexity.

The Unified Modeling Language (UML) is a set of notations and conventions used to describe and model an application. The UML is intended to be a universal language for modeling systems, meaning that it can express models of many different kinds and purposes, just as a programming language or a natural language can be used in different ways.

A "model" is an abstract representation of a system, constructed to understand the system prior to building or modifying it. The term "system"

is used here in a broad sense to include any process or structure. For example, the organizational structure of a corporation, health services, computer software, instruction of any sort (including computers), the national economy, and so forth all would be termed "systems".

The unified modeling language is a language for specifying, constructing, visualizing, and documenting the software system and its components. The UML is a graphical language with sets of rules and semantics. The rules and semantics of a model are expressed in English, in a form known as "object constraint language"(OCL). OCL is a specification language that uses simple logic for specifying the properties of a system.

The UML is not intended to be a visual programming language in the sense of having all the necessary visual and semantic support to replace programming languages. However, the UML does have a tight mapping to a family of object- oriented languages, so that you can get the best of both worlds.

The primary goals in the design of the UML were as follows:

1. Provide users ready-to-use, expensive visual modeling languages so they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of the OO tools market.
6. Support higher level development concepts.
7. Integrate best practices and methodologies.

UML is a language used to:

- Visualize the software system well-defined symbols. Thus, a developer or tool can unambiguously interpret a model written by another developer, using UML
- Specify the software system and help building precise, unambiguous and complete models.
- Construct the models of the software system that can directly communicate with a variety of programming languages.
- Document models of the software system during its development stages.

3.8.1 Architectural views and diagrams of the UML

The UML Meta model elements are organized into diagrams. Different diagrams are used for different purposes depending on the angle from which you are viewing the system. The different views are called architectural views. Architectural views facilitate the organization of knowledge, and diagrams enable the communication of knowledge. Then knowledge itself is within the model or set of models that focuses on the problem and solution. The architectural views and their diagrams are summarized below:

3.8.1.1 The user model view

Encompasses a problem and solution from the preservative of those individuals whose problem the solution addresses. The view presents the goals and objectives of the problem owners and their requirements of the solution. This view is composed of use case diagrams. These diagrams describe the functionality provided by a system to external interactors. These diagrams contain actors, use cases, and their relationships.

3.8.1.2 The Structural model view

Encompasses the static, or structural, aspects of a problem and solution. This view is also known as the static or logical view. This view is composed of the following diagrams:

- **Class diagrams:** describe the static structure of a system, or how it is declared rather than how it behaves. These diagrams contain classes and associations.
- **object diagrams:** describe the static structure of a system at a particular time during its life. These diagrams contain objects and links.

3.8.1.3 The behavioral model view

Encompasses the dynamic or behavioral aspects of a problem and solution. The view is also known as the dynamic, process, concurrent or collaborative view. This view is composed of the following diagrams:

- **Sequence diagrams:** render the specification of behavior. These diagrams describe the behavior provided by a system to interactions. These diagrams contain classes that exchange messages with in an interaction arranged in time sequence. In generic form, these diagrams describe a set of message exchange sequences among a set of classes. In instance form(scenarios), these diagrams describe one actual message exchange sequence among objects of those classes.
- **Collaboration diagrams:** render how behavior is realized by components with in a system. These diagrams contain classes, associations, and their message exchanges with in a collaboration to accomplish a purpose. In generic form, these diagrams describe a set of classes and associations involved in message exchange sequences. In instance form(scenarios), these diagrams describe a set of objects of those classes links confirming to the associations, and one actual

essage exchange sequence that inconsistent with the generic form and uses those objects and links.

- **State chart diagrams:** render the states and responses of a class participating in behavior, and the life cycle of an object. These diagrams describe the behavior of a class in response to external stimuli.
- **Activity diagrams:** render the activities of a class participating in behavior. These diagrams describe the behavior of a class in response to internal processing rather than external events. Activity diagrams describe the processing activities with in a class.
- **The Implementation model view:** Encompasses the structural and behavioral aspects of the solution's realization. This view is also known as the component or development view and is composed of component diagrams. These diagrams describe the organization of and dependencies among software implementation components. These diagrams contain components and their relationships.
- **The Environment model view:** Encompasses the structural and behavioral aspects of the domain in which a solution must be realized. This view is also known as the deployment or physical view. This view is composed of deployment diagrams. These diagrams describe the configuration of processing resources elements and the mapping of software implementation components onto them. These diagrams contain nodes, components and their relationships.

3.8.2 UML Diagrams

Every complex system is best approached through a small set of nearly independent views of a model; no single viewer is sufficient. Every model may be expressed at different levels of fidelity. The best models are connected to reality. The UML defines nine graphical diagrams.

Chapter IV

Used Software

4.1 Node.js

We choose Node.js to be our backend technology for the mobile application, Node.js is an open-source, cross-platform JavaScript runtime environment that allowed us to build the application's server-side. Node.js uses an event-driven, non-blocking I/O model, making it highly efficient and scalable for handling concurrent requests.

Benefits of Node.js as a backend technology:

1. **Asynchronous and Non-blocking:** One of the key advantages of Node.js is its asynchronous, non-blocking I/O model. This means that Node.js can handle multiple requests concurrently without getting blocked, making it highly efficient for handling real-time applications with high traffic or I/O operations. It utilizes callbacks, promises, or async/await syntax to manage asynchronous operations effectively.
2. **Fast and Scalable:** Node.js is built on the V8 JavaScript engine, which compiles JavaScript into machine code. This results in high-performance execution, enabling Node.js applications to handle a large number of concurrent connections efficiently. Additionally, Node.js supports clustering, allowing you to scale your application across multiple cores, maximizing the utilization of hardware resources.
3. **Rich Ecosystem:** Node.js has a vast and active ecosystem of libraries and packages available through npm (Node Package Manager). This extensive collection of modules makes it easy to integrate third-party libraries, frameworks, and tools into your application, reducing development time and effort.
4. **Microservices Architecture:** Node.js is well-suited for building microservices-based architectures. Its lightweight and modular nature

allows developers to create independent, small-scale services that can communicate with each other via APIs. This approach enables scalability, maintainability, and flexibility in complex applications.

5. **Code Reusability:** Node.js enables code reusability by using JavaScript on both the frontend and backend. This means that functions, modules, and utilities created for the frontend can be reused on the server side, promoting code sharing and reducing duplication.
6. **Real-time Applications:** Node.js is an excellent choice for building real-time applications that require instant data updates, such as chat applications, collaborative tools, or gaming platforms. Its event-driven architecture and WebSocket support facilitate bidirectional communication between the server and client, enabling real-time data exchange.

4.2 Express.js

Express.js was chosen as the backend framework, Express.js is a minimalist and flexible web application framework for Node.js. It provides a set of robust features and utilities that simplify the process of building web applications and APIs.

Benefits of Express.js as a backend framework:

1. **Minimalistic and Unopinionated:** Express.js follows a "less is more" philosophy, providing a lightweight framework that allows developers to have more control over their application architecture. It doesn't impose any strict conventions or structure, giving you the freedom to design your application the way you prefer.

2. **Easy Routing:** Express.js offers a simple and intuitive routing system. You can define routes for different HTTP methods (GET, POST, PUT, DELETE, etc.) and URLs, and associate them with specific callback functions. This makes it easy to handle various requests and perform specific actions based on the URL and method.
3. **Middleware Support:** Express.js has excellent support for middleware, which are functions that can be executed before or after the request reaches its final handler. Middleware allows you to add custom logic, such as authentication, request parsing, error handling, logging, and more, to your application in a modular and reusable manner.
4. **Built-in Error Handling:** Express.js provides built-in error handling mechanisms, making it easier to handle and manage errors throughout your application. You can define error handling middleware to catch and process errors, improving the robustness and reliability of your backend.
5. **Scalability and Performance:** Express.js is designed to be highly scalable and performant. Its minimalist nature and asynchronous I/O capabilities make it suitable for building high-performance applications that can handle a large number of concurrent requests. Additionally, it works well with Node.js clustering to scale applications across multiple cores, maximizing performance.

4.3 MongoDB

For the database we chose MongoDB, MongoDB is an open-source, document-oriented NoSQL database that provides a flexible and scalable solution for storing and managing data. Unlike traditional relational

databases, MongoDB does not use tables and rows to store data but instead uses a flexible and schema-less document model.

Benefits of using MongoDB as a database:

1. **Flexible Data Model:** MongoDB is a document-oriented NoSQL database, which means it stores data in flexible and self-describing documents. Each document can have its own structure and fields, allowing for easy and dynamic schema changes. This flexibility is particularly useful when dealing with evolving data requirements and handling unstructured or semi-structured data.
2. **Scalability and High Performance:** MongoDB is designed to scale horizontally by distributing data across multiple servers or clusters. It supports automatic sharding, which partitions data and distributes it across multiple machines. This allows for seamless scalability as your data grows, ensuring high performance and efficient handling of large volumes of data.
3. **Rich Querying and Indexing Capabilities:** MongoDB offers a powerful and flexible query language that allows you to perform complex queries, filtering, sorting, and aggregations. It supports a wide range of query operators and provides indexing support to optimize query performance. This makes it easier to retrieve and manipulate data in a flexible and efficient manner.
4. **High Availability and Fault Tolerance:** MongoDB provides built-in replication, which creates multiple copies of data across different servers or clusters. This ensures high availability and fault tolerance by automatically maintaining synchronized copies of data. If one server goes down, the system can seamlessly switch to a replica, minimizing downtime and ensuring data reliability.

5. **Horizontal Scalability:** MongoDB's architecture allows it to scale horizontally by adding more machines or clusters to handle increased data loads and user traffic. This makes it well-suited for applications that require scalability and the ability to handle concurrent operations across multiple servers.
6. **Automatic Load Balancing:** MongoDB's sharding feature automatically balances the distribution of data across different shards or partitions. This ensures that data is evenly distributed and evenly accessed, optimizing performance and resource utilization.
7. **Integration with Modern Technologies:** MongoDB integrates well with modern technologies and frameworks. It has official drivers for various programming languages and provides connectors for popular frameworks like Node.js, Express.js, and Django. This makes it easy to incorporate MongoDB into your existing tech stack and leverage its capabilities within your application.

4.4 Mongoose

Mongoose was chosen as our Object-Document Mapping library for Node.js and MongoDB, Mongoose provides a higher-level abstraction for interacting with MongoDB by adding additional features and functionalities on top of the MongoDB driver.

Benefits of using Mongoose as an ODM:

1. **Simplified Data Modeling:** Mongoose simplifies the process of defining and working with data models in MongoDB. It allows you to define the structure, validation rules, relationships, and other aspects of your data using a straightforward and intuitive schema-based approach. This helps maintain consistency and organization in your data models.

2. **Schema Validation:** Mongoose enables you to define schema validation rules for your data. You can specify required fields, data types, enumerations, default values, and more. This helps ensure that the data stored in the database meets the specified criteria, reducing the chances of data inconsistencies and errors.
3. **Middleware and Hooks:** Mongoose provides middleware and hooks that allow you to define pre and post-save, update, delete, and query hooks. This gives you fine-grained control over the data flow and enables you to perform additional actions or validations before or after certain operations. Middleware functions can be used to modify data, execute business logic, or trigger external processes.
4. **Query Building and Population:** Mongoose offers a powerful query API that simplifies the process of constructing complex queries for MongoDB. It provides chainable methods for filtering, sorting, limiting, and projecting data. Additionally, Mongoose supports population, which allows you to populate references between different collections, avoiding manual fetching and joining of related data.
5. **Integration with Express.js:** Mongoose integrates seamlessly with Express.js, a popular web application framework for Node.js. This combination allows you to build robust and scalable web applications by leveraging the capabilities of both libraries. Express.js provides a middleware system for handling HTTP requests and responses, while Mongoose handles the interaction with the database layer.
6. **Data Validation and Type Casting:** Mongoose automatically performs data validation based on the defined schema before saving or updating data in MongoDB. It ensures that data is in the expected format and meets the validation rules specified in the schema.

Mongoose also performs type casting, converting data to the expected types defined in the schema.

7. **Population and Referencing:** Mongoose provides a feature called population, which allows you to reference documents from other collections. This avoids the need for embedding data and enables efficient querying and retrieval of related data. Population simplifies handling relationships between documents and allows for better organization and scalability

4.5 RESTful APIs

For the API we used RESTful APIs, RESTful APIs, or Representational State Transfer APIs, are a set of architectural principles and guidelines used for designing networked applications. RESTful APIs are built on top of the HTTP protocol, which is the foundation of communication on the World Wide Web.

Benefits of using RESTful APIs:

1. **Statelessness:** Each request from a client to a server must contain all the information necessary for the server to understand and process the request. The server should not rely on any information stored from previous requests.
2. **Client-Server Architecture:** The client and server are separate entities that communicate over a network. The client is responsible for the user interface and user experience, while the server is responsible for processing requests and managing resources.
3. **Uniform Interface:** The API should have a consistent and standardized set of methods and conventions that are used for communication between the client and server. This includes the use of HTTP verbs (GET, POST, PUT, DELETE) to perform operations on resources.
4. **Resource-Based:** Resources are the fundamental entities that the API exposes. Each resource should have a unique identifier (URI) and can be accessed and manipulated using standard HTTP methods.
5. **Representation-Oriented:** Resources can have multiple representations, such as JSON, XML, or HTML. The client can specify the desired representation format using the HTTP Accept header.

6. **State Transfer:** When a client requests a resource, the server transfers the state of the resource to the client in the form of a representation. The client can then manipulate or update the state of the resource through subsequent requests.

RESTful APIs are widely used for building web services, web applications, and mobile applications. They provide a scalable and flexible way to expose and consume data and functionality over the internet. The use of standard HTTP methods and the statelessness of RESTful APIs make them highly interoperable and easy to integrate into various platforms and programming languages.

4.6 Android

Android was chosen as our platform for the mobile application development. Android is an open-source operating system developed by Google specifically for mobile devices. It offers a powerful and flexible framework for building high-quality, feature-rich applications.

Benefits of using Android as the mobile platform:

1. **Vast user base:** Android has the largest market share in the mobile industry, with millions of active users worldwide. Developing for Android allows us to reach a broad audience and maximize the potential user base for our application.
2. **Wide range of devices:** Android runs on a diverse range of devices, including smartphones, tablets, wearables, and even smart TVs. This versatility gives us the opportunity to target different form factors and ensure our application is accessible to a wide range of users.
3. **Rich development ecosystem:** Android provides a comprehensive development ecosystem with a wide range of tools, libraries, and resources. Android Studio, the official integrated development environment (IDE) for Android, offers a robust set of features that streamline the development process and enhance productivity.

4. **Native development capabilities:** Android allows developers to write applications using the Java or Kotlin programming languages, which are optimized for the platform. This enables us to leverage the full power of the Android framework and access platform-specific features to create a seamless user experience.
5. **Access to native features and APIs:** Android provides access to a vast array of native features and APIs that allow developers to integrate their applications deeply with the device's capabilities. This includes features such as camera, GPS, sensors, contacts, storage, and much more.
6. **Google Play Store distribution:** By developing for Android, we can distribute our application through the Google Play Store, which is the primary app marketplace for Android devices. This gives us access to a large and established distribution platform with built-in monetization options.
7. **Continuous platform advancements:** Google regularly updates and enhances the Android platform, introducing new features, optimizations, and security improvements. By developing for Android, we can take advantage of these advancements and ensure our application stays up to date with the latest industry standards.

Overall, choosing Android as our platform provides us with a robust and versatile environment to develop our mobile application, with access to a vast user base, native capabilities, and a rich development ecosystem.

4.7 Scikit-learn

(Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistent interface in Python. As a high-level library, it lets you define a predictive data model in just a few lines of code, and then use that model to fit your data. It's versatile and integrates well with other Python libraries, such as matplotlib for plotting, numpy for array vectorization, and pandas for dataframes.

Some of the big key elements of Scikit-learn useful for machine learning include classification, regression and clustering algorithms. For example, Scikit-learn supports work on random forests, where individual digital trees hold node information that is combined in multiple tree architectures to achieve a forest approach. Another way of talking about this is that each tree involves clustered nodes in a tree topology, and analysis from various trees is added together to get a global approach that more accurately crunches data to show outcomes.

In addition to random forest, Scikit-learn helps with gradient boosting, vector machines and other elements of machine learning that are key to achieving results. As the overarching resource, Scikit-learn works with tools like SciPy and matplotlib that provide visualization and much more.

Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space based on the cosine of the angle between them, resulting in a value between -1 and 1. The value -1 means that the vectors are opposite, 0 represents orthogonal vectors, and value 1 signifies similar vectors.

To compute the cosine similarity between vectors A and B, you can use the

following formula:

$$\text{similarity}(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|},$$

Exploring Python libraries for cosine similarity

As explained in the previous chapter, cosine similarity can be easily calculated using the mathematical formula. But what if the data you have becomes too large and you want to calculate the similarities fast? The most popular programming language used for such tasks is definitely Python and its flexibility is partly due to its extensive range of libraries. For calculating cosine similarity, the most popular ones are:

- **NumPy**: the fundamental package for scientific computing in Python, which has functions for dot product and vector magnitude, both necessary for the cosine similarity formula.
- **SciPy**: a library used for scientific and technical computing. It has a function that can calculate the cosine distance, which equals 1 minus the cosine similarity.
- **Scikit-learn**: offers simple and efficient tools for predictive data analysis and has a function to directly and efficiently compute cosine similarity.

From the above-mentioned libraries, only scikit-learn directly calculates the cosine similarity between two vectors or matrices, making it an excellent tool for data analysts and machine learning enthusiasts. It provides `sklearn.metrics.pairwise.cosine_similarity` function to do that, and we will show how it works on an example

Practical use cases of cosine similarity

Cosine similarity is used in various applications, mostly by data scientists, to perform tasks for machine learning, natural language processing, or similar

projects. Their applications include:

- Text analysis, which is applied to measure the similarity between documents and offers crucial functionality for search engines and information retrieval systems, as shown in the example.
- Recommendation systems, to recommend similar items based on user preferences or to suggest similar users in social network applications. An example is to recommend the next page on product documentation based on the text similarity found.
- Data clustering, which in machine learning acts as a metric to classify or cluster similar data points, and in that way, it helps make data-driven decisions.
- Semantic similarity, which, when paired with word embedding techniques like Word2Vec, is used to determine the semantic similarity between words or documents

4.8 Flask

Flask was chosen as our framework for web application development. Flask is a lightweight and flexible web framework written in Python. It offers a simple yet powerful toolkit for building web applications.

Benefits of using Flask as the web framework:

1. **Simplicity and ease of use:** Flask follows a minimalist design philosophy, making it easy to understand and work with. Its simplicity allows developers to get started quickly and build applications without unnecessary complexity.
2. **Python ecosystem:** Flask is built on top of Python, which is a popular and widely-used programming language. Leveraging the Python ecosystem provides access to a vast collection of libraries, tools, and resources, enabling developers to extend Flask's functionality and enhance their applications.
3. **Extensibility and flexibility:** Flask is designed to be highly extensible, allowing developers to customize and tailor the framework to their specific requirements. It provides a modular architecture that supports the integration of various extensions, such as SQLAlchemy for database management or Flask-WTF for form handling.
4. **Lightweight and scalable:** Flask is a lightweight framework that does not impose rigid structures or dependencies. This lightweight nature allows for efficient resource utilization and enables Flask applications to scale effectively, handling increased traffic and user demands.

5. **RESTful API development:** Flask provides excellent support for building RESTful APIs. It offers features like request routing, request parsing, and response formatting, making it easy to develop APIs that follow the principles of REST architecture.
6. **Testing and debugging support:** Flask comes with built-in support for testing and debugging. It provides a testing client for simulating requests and responses, making it convenient to write and execute test cases. Additionally, Flask's interactive debugger helps in identifying and resolving issues during development.
7. **Active community and documentation:** Flask has a vibrant and active community of developers. This community actively contributes to the framework by creating extensions, sharing knowledge, and providing support. The extensive documentation and numerous tutorials available make it easier for developers to learn and utilize Flask effectively.
8. **Deployment options:** Flask applications can be deployed using a variety of options, such as running them on standalone servers like Gunicorn or integrating them with popular web servers like Nginx. Flask also supports deployment on cloud platforms like Heroku and AWS, making it flexible and adaptable to different hosting environments.

Choosing Flask as our web framework provides us with a lightweight, flexible, and extensible environment to develop our application. Its simplicity, Python integration, and extensive ecosystem make Flask an excellent choice for building high-quality web applications that can scale and adapt to evolving needs.

4.9 GitHub

GitHub was chosen as our collaboration tool, GitHub is a web-based platform that provides version control and collaboration tools for software development projects. It offers a range of benefits for individuals and teams working on code-based projects.

Benefits of using GitHub as our collaboration tool:

1. **Version Control:** GitHub is built on Git, a distributed version control system. It allows developers to track changes, manage different versions of their code, and collaborate effectively. With Git's branching and merging capabilities, multiple developers can work on the same project simultaneously without conflicts.
2. **Collaboration:** GitHub provides a collaborative environment for developers to work together on projects. It enables multiple contributors to easily collaborate, review, and provide feedback on code changes. Developers can create pull requests, manage issues, and conduct discussions within the platform, streamlining the development workflow.
3. **Code Hosting:** GitHub serves as a centralized repository for hosting code. It provides a secure and reliable infrastructure to store and manage codebases, ensuring accessibility and availability for team members. Developers can clone, fork, and download code repositories, making it easy to distribute and share projects.
4. **Issue Tracking and Project Management:** GitHub offers built-in issue tracking and project management features. Developers can create and manage tasks, track bugs, assign responsibilities, and set milestones. This helps in organizing and prioritizing work, enhancing project management and coordination within teams.

5. **Continuous Integration and Deployment:** GitHub integrates with various continuous integration (CI) and deployment tools. It allows developers to automate build processes, run tests, and deploy applications directly from code repositories. This streamlines the software development lifecycle and improves efficiency.
6. **Documentation:** GitHub provides a platform for creating documentation for projects. We can document code, APIs, project guidelines, and best practices. This helps in maintaining a centralized and up-to-date source of information, making it easier for team members and contributors to understand and collaborate on projects.

Chapter V

System Implementation

5.1 Mobile Application

Splash Screen



figure 5.1

Intro First Screen

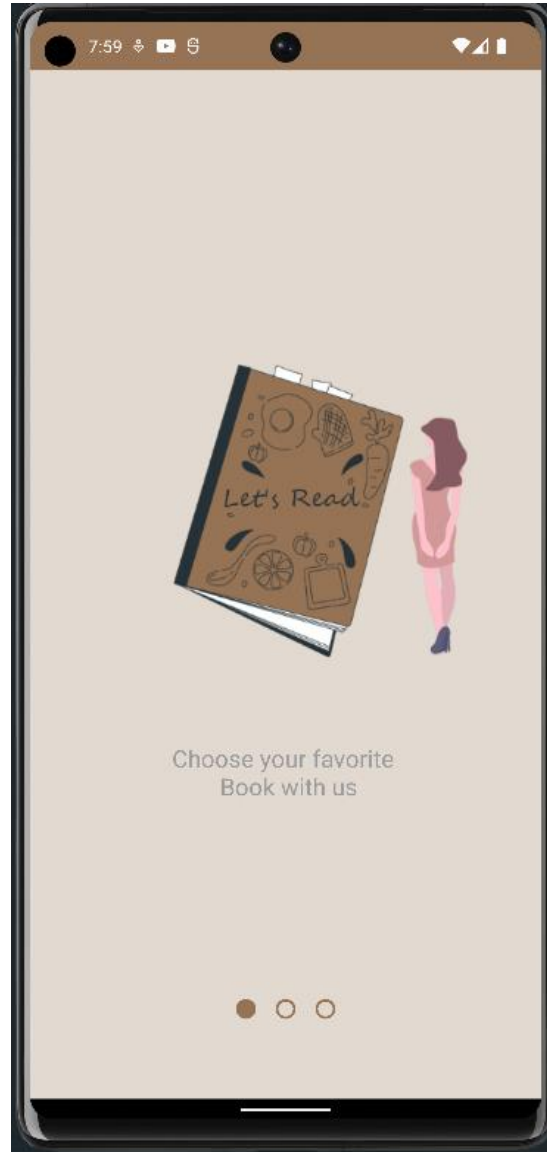


figure 5.2

Intro Second Screen

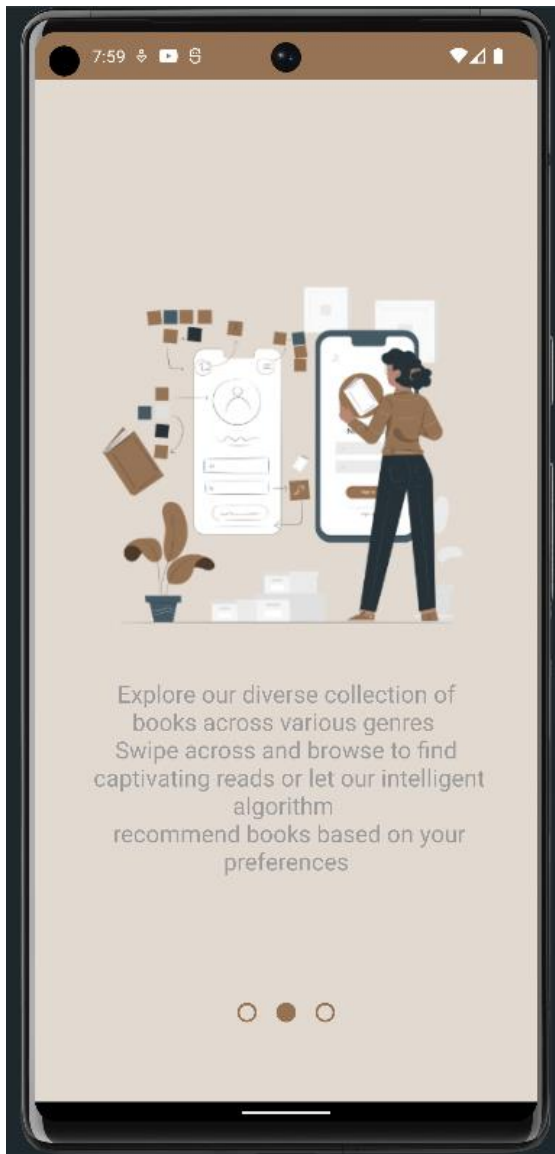


figure 5.3

Intro third Screen

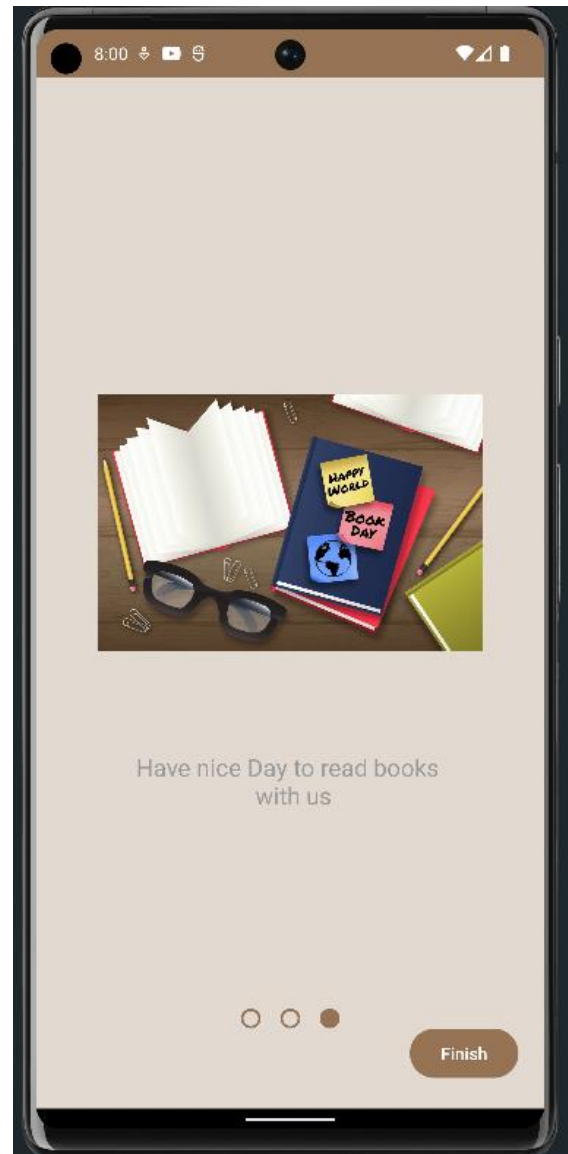
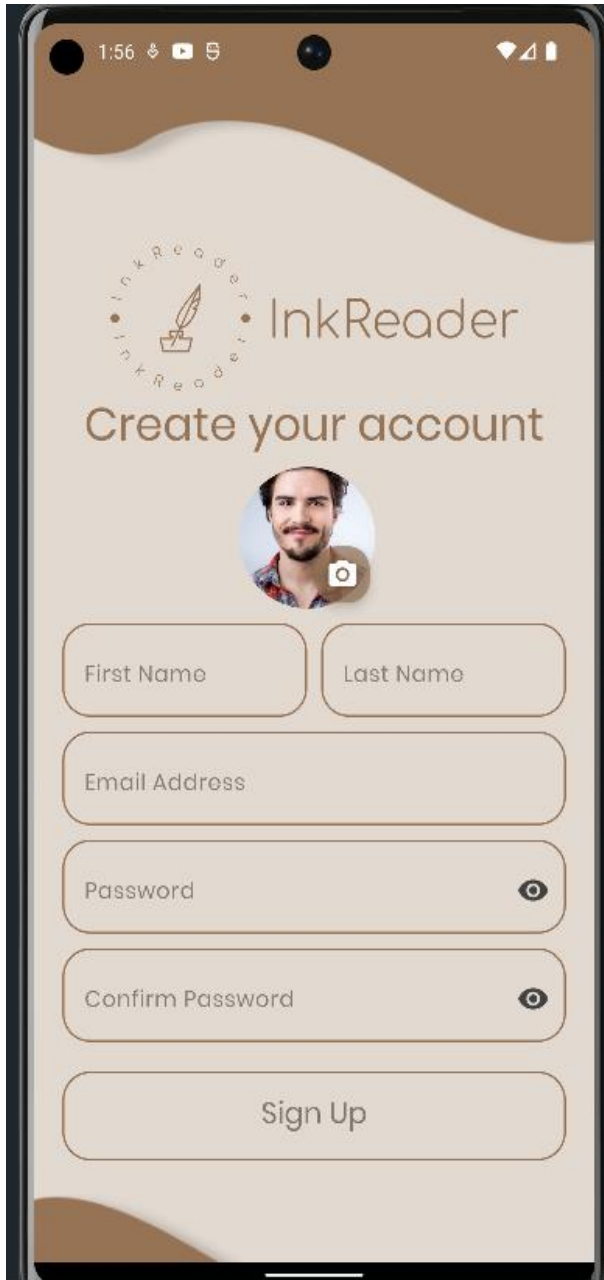


figure 5.4

Sign up



The sign-up screen for InkReader features a light beige background with a brown header and footer. At the top, the status bar shows the time 1:56 and various icons. The InkReader logo, which includes a quill pen icon, is centered. Below the logo, the text "Create your account" is displayed. A circular profile picture placeholder with a camera icon is shown. The form consists of several input fields: "First Name" and "Last Name" (side-by-side), "Email Address", "Password" (with a toggle icon), and "Confirm Password" (with a toggle icon). A "Sign Up" button is at the bottom.

1:56

InkReader

Create your account

First Name Last Name

Email Address

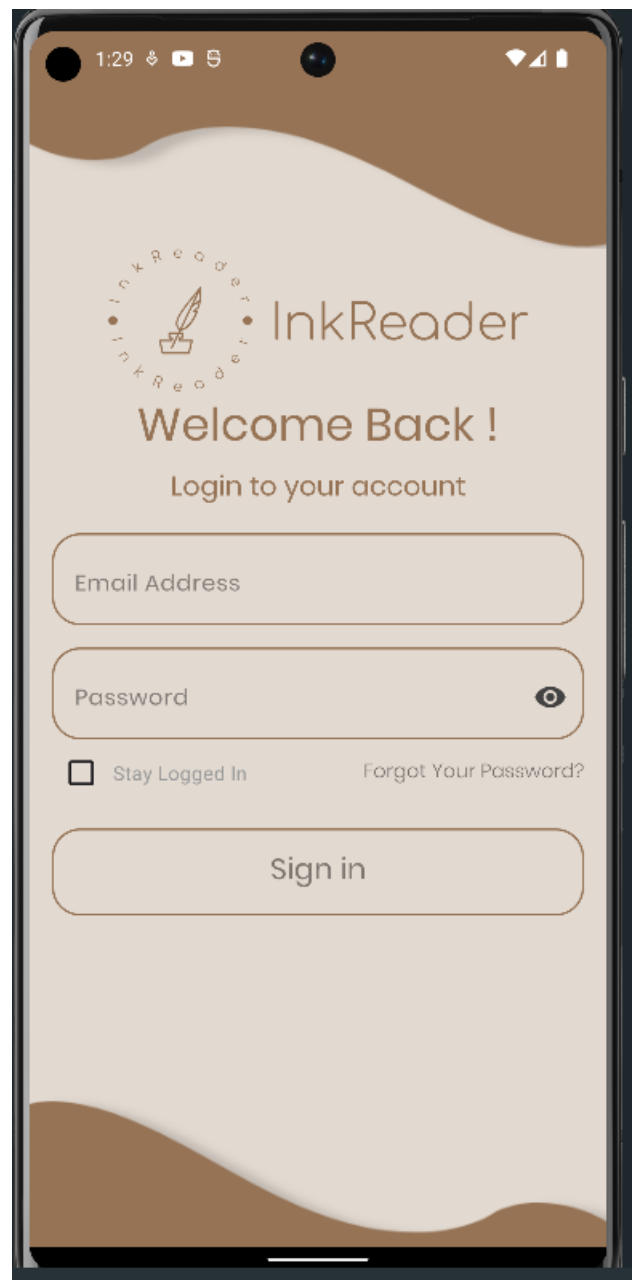
Password

Confirm Password

Sign Up

figure 5.5

Login



The login screen for InkReader has a similar design to the sign-up screen. The status bar shows the time 1:29. The InkReader logo is at the top. Below it, the text "Welcome Back !" is displayed, followed by "Login to your account". The form includes an "Email Address" field, a "Password" field (with a toggle icon), a "Stay Logged In" checkbox, and a "Forgot Your Password?" link. A "Sign in" button is at the bottom.

1:29

InkReader

Welcome Back !

Login to your account

Email Address

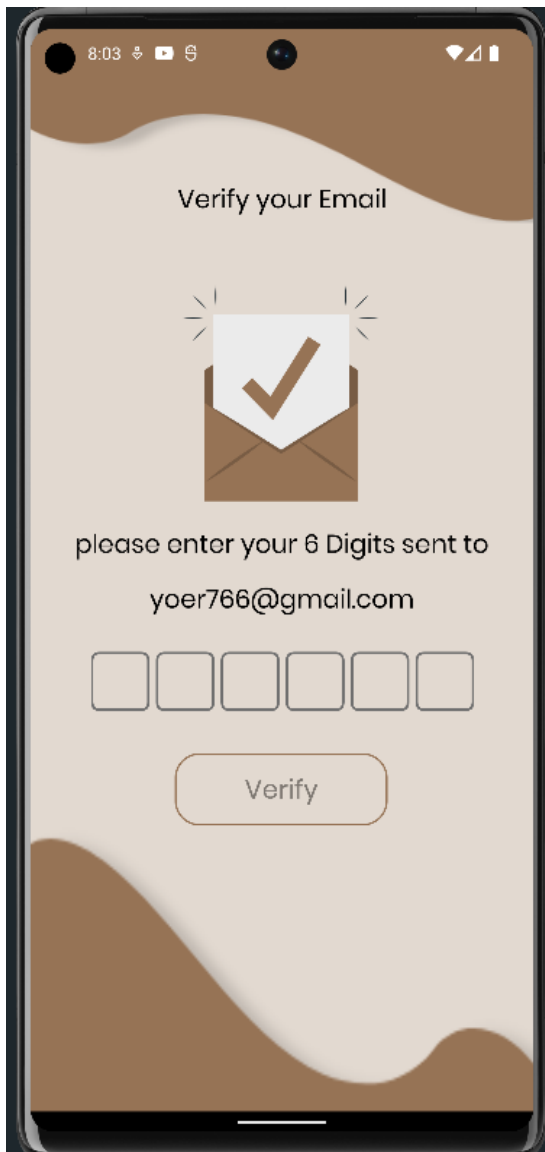
Password

☐ Stay Logged In [Forgot Your Password?](#)

Sign in


figure 5.6

Verify Email



8:03

Verify your Email

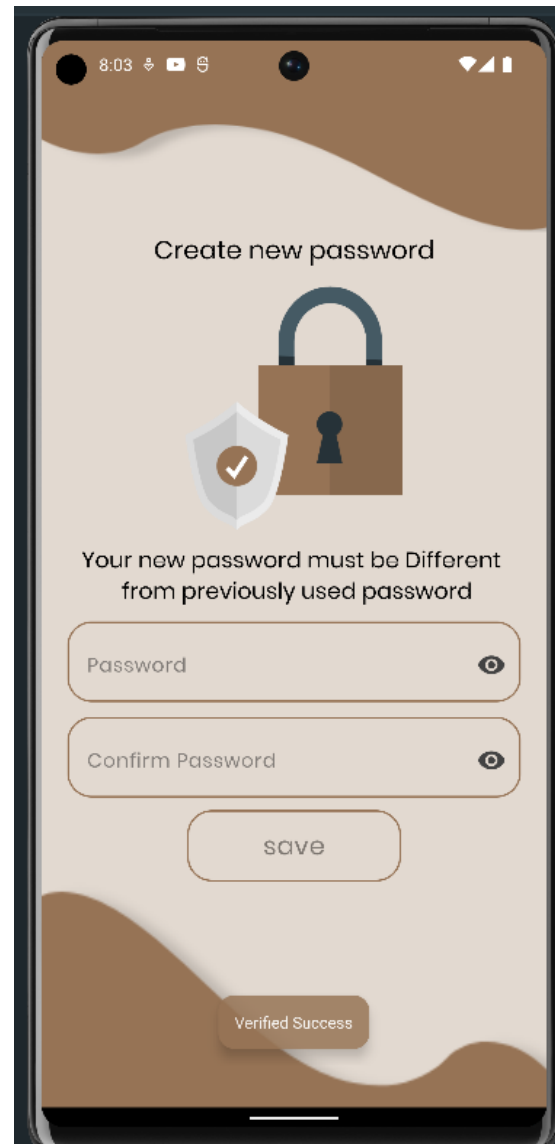


please enter your 6 Digits sent to
yoer766@gmail.com

Verify


figure 5.7

Create New Password





8:03

Create new password



Your new password must be Different
from previously used password

Password 

Confirm Password 

save

Verified Success

figure 5.8

Home Page



figure 5.9

Search Page



figure 5.10

Search for Books

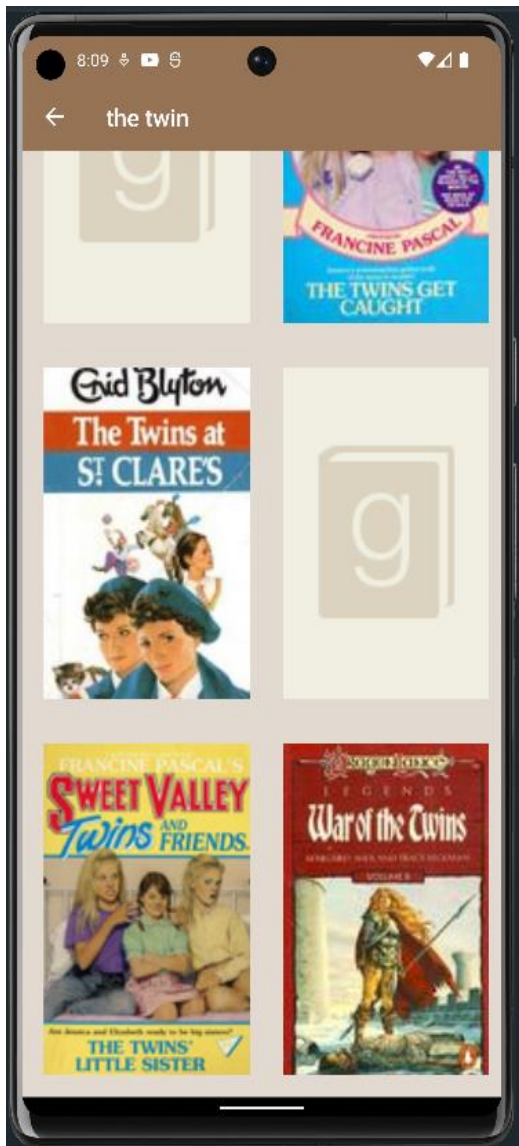


figure 5.11

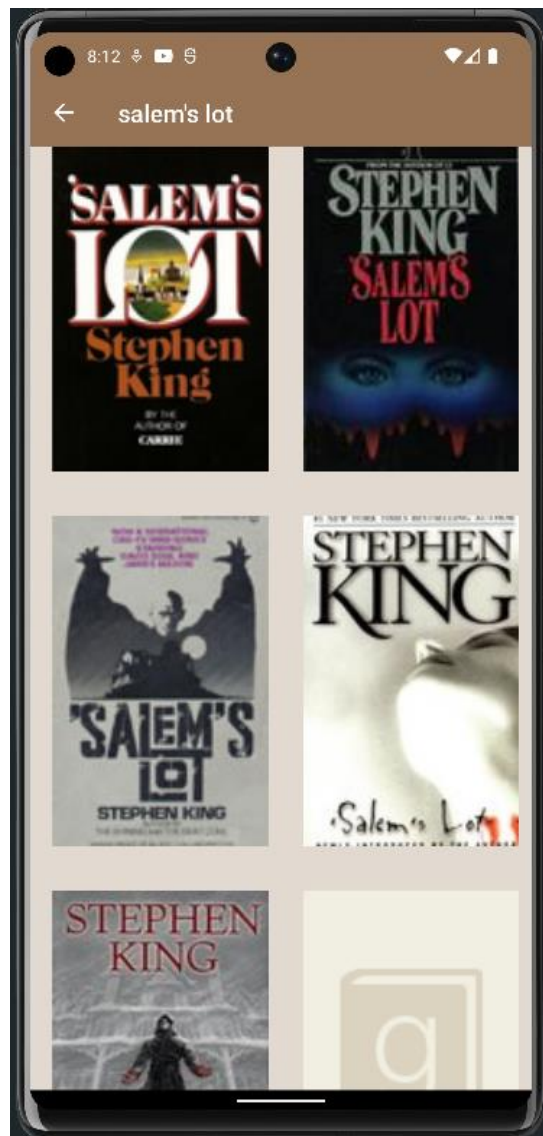


figure 5.12

Search History

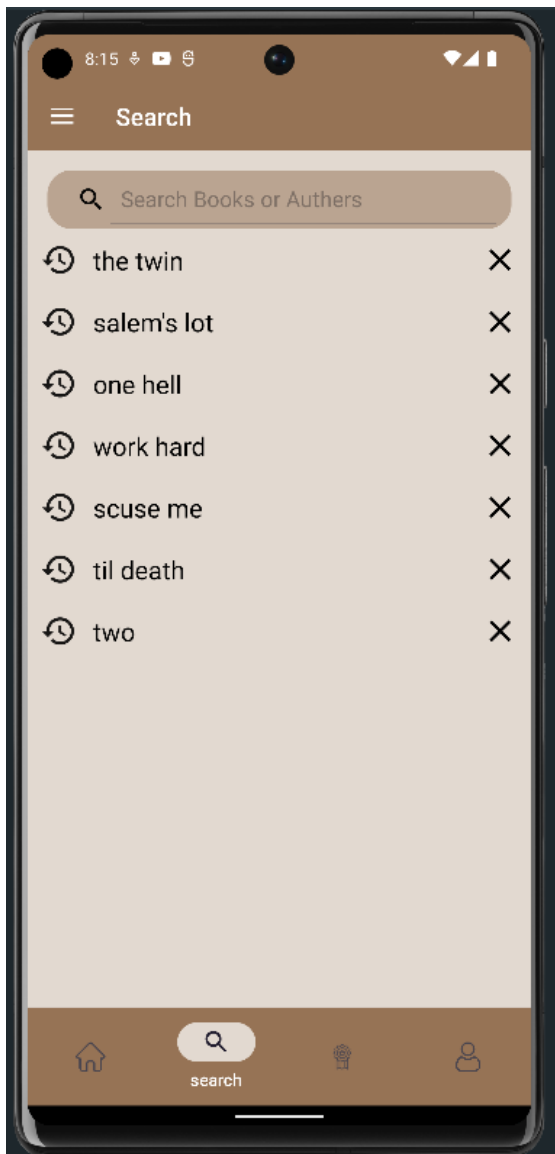


Figure 5.13

Recommendation Page



figure 5.14



Wishlist Page



figure 5.13

Already Read Page



figure 5.14

Book Details

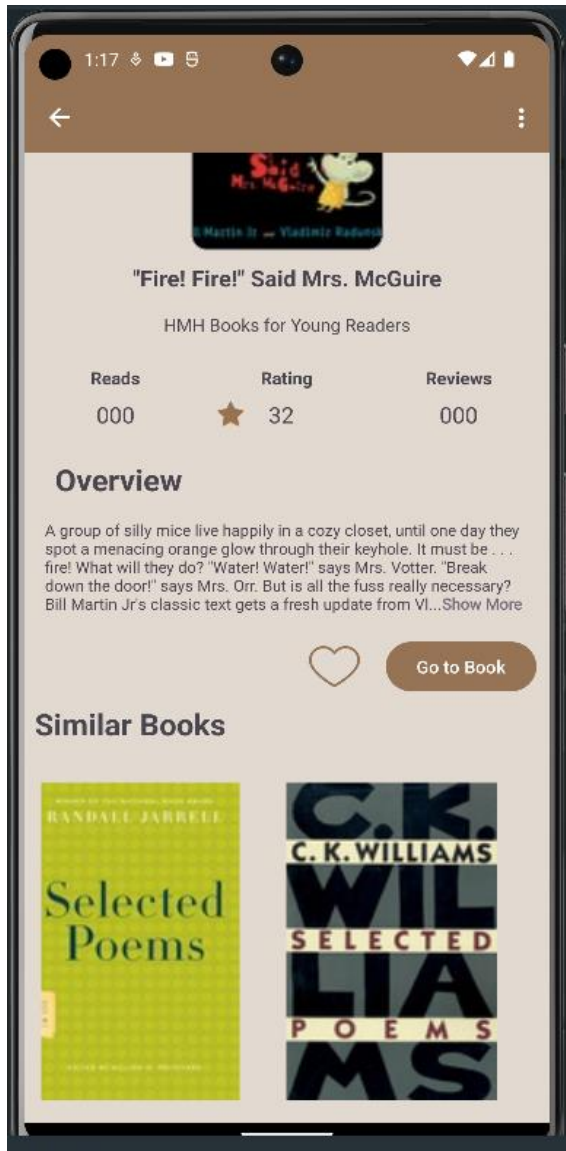


figure 5.19

Rate Book

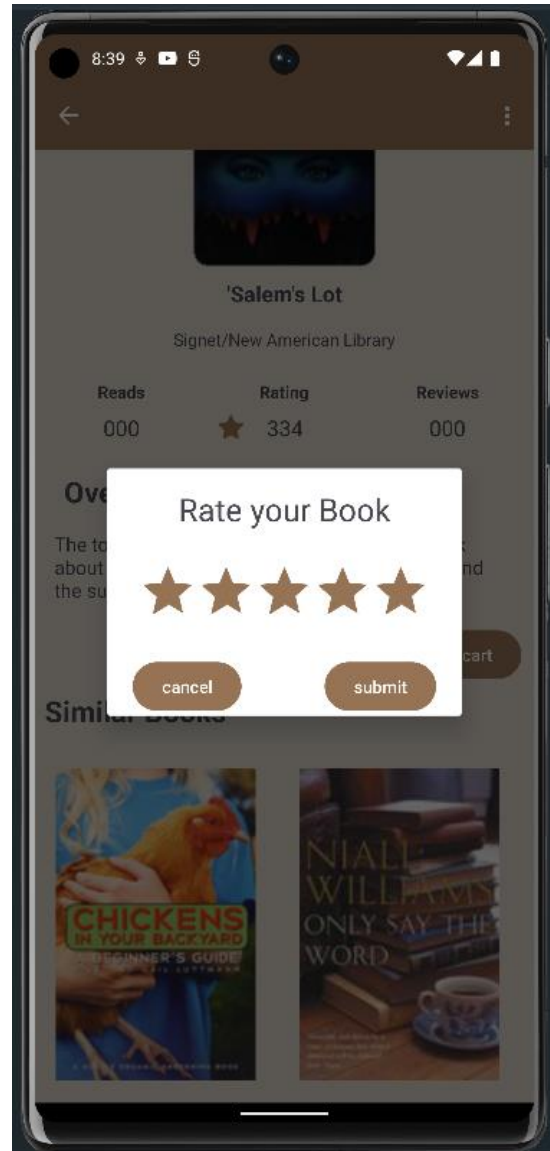


figure 5.20

Chapter VI

System Testing

6.1 Introduction

Software testing is a critical element of software quality assurance and represents the ultimate review of specification, design and coding. In fact, testing is the one step in the software engineering process that could be viewed as destructive rather than constructive.

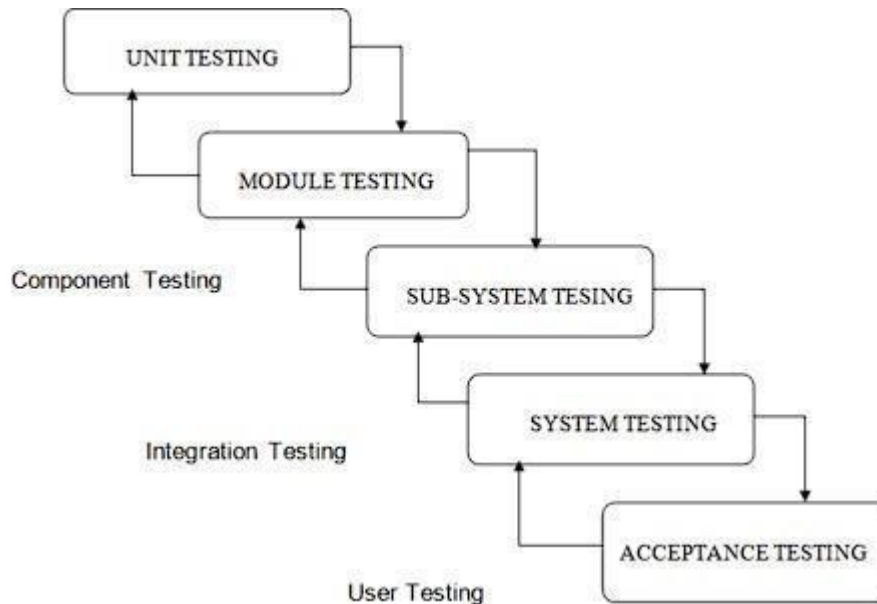
A strategy for software testing integrates software test case design methods into a well-planned series of steps that result in the successful construction of software. Testing is the set of activities that can be planned in advance and conducted systematically. The underlying motivation of program testing is to affirm software quality with methods that can economically and effectively apply to both strategic to both large and small-scale systems.

6.2 Strategic approach to software testing

The software engineering process can be viewed as a spiral. Initially system engineering defines the role of software and leads to software requirement analysis where the information domain, functions, behavior, performance, constraints and validation criteria for software are established. Moving inward along the spiral, we come to design and finally to coding. To develop computer software, we spiral in along streamlines that decrease the level of abstraction on each turn.

A strategy for software testing may also be viewed in the context of the spiral. Unit testing begins at the vertex of the spiral and concentrates on each unit of the software as implemented in source code. Testing progress by moving outward along the spiral to integration testing, where the focus is on the design and the construction of the software architecture. Talking another turn on outward on the spiral we encounter validation testing where requirements established as part of software requirements analysis are validated against the software that has been constructed. Finally, we arrive at

system testing, where the software and other system elements are tested as a whole.



6.3 Levels of testing

Levels of testing that software goes through are:

ACCEPTANCE TESTING: Testing to verify a product meets customer specified requirements. A customer usually does this type of testing on a product that is developed externally.

BLACK BOX TESTING: Testing without knowledge of the internal workings of the item being tested. Tests are usually functional.

COMPATIBILITY TESTING: Testing to ensure compatibility of an application or Web site with different browsers, OSs, and hardware platforms. Compatibility testing can be performed manually or can be driven by an automated functional or regression test suite.

CONFORMANCE TESTING: Verifying implementation conformance to industry standards. Producing tests for the behavior of an implementation to be sure it provides the portability, interoperability, and/or compatibility a standard defines.

FUNCTIONAL TESTING: Validating an application or Web site conforms to its specifications and correctly performs all its required functions. This entails a series of tests which perform a feature-by-feature validation of behavior, using a wide range of normal and erroneous input data. This can involve testing of the product's user interface, APIs, database management, security, installation, networking, etc. testing can be performed on an automated or manual basis using black box or white box methodologies.

INTEGRATION TESTING: Testing in which modules are combined and tested as a group. Modules are typically code modules, individual applications, client and server applications on a network, etc. Integration Testing follows unit testing and precedes system testing.

LOAD TESTING: Load testing is a generic term covering Performance Testing and Stress Testing.

PERFORMANCE TESTING: Performance testing can be applied to understand your application or WWW site's scalability, or to benchmark the performance in an environment of third-party products such as servers and middleware for potential purchase. This sort of testing is particularly useful

to identify performance bottlenecks in high use applications. Performance testing generally involves an automated test suite as this allows easy simulation of a variety of normal, peak, and exceptional load conditions.

REGRESSION TESTING: Similar in scope to a functional test, a regression test allows a consistent, repeatable validation of each new release of a product or Web site. Such testing ensures reported product defects have been corrected for each new release and that no new quality problems were introduced in the maintenance process. Though regression testing can be performed manually an automated test suite is often used to reduce the time and resources needed to perform the required testing.

SMOKE TESTING: A quick-and-dirty test that the major functions of a piece of software work without bothering with finer details. Originated in the hardware testing practice of turning on a new piece of hardware for the first time and considering it a success if it does not catch on fire.

STRESS TESTING: Testing conducted to evaluate a system or component at or beyond the limits of its specified requirements to determine the load under which it fails and how. A graceful degradation under load leading to non-catastrophic failure is the desired result.

Often Stress Testing is performed using the same process as Performance Testing but employing a very high level of simulated load.

SYSTEM TESTING: Testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System

testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

6.3.1 Unit Testing

Unit testing focuses verification effort on the smallest unit of software design, the module. The unit testing we have is white box oriented and some modules the steps are conducted in parallel.

6.3.1.1 White box testing

This type of testing ensures that:

- All independent paths have been exercised at least once
- All logical decisions have been exercised on their true and false sides
- All loops are executed at their boundaries and within their operational bounds
- All internal data structures have been exercised to assure their validity

To follow the concept of white box testing we have tested each form, we have created independently to verify that Data flow is correct, All conditions are exercised to check their validity, All loops are executed on their boundaries.

6.3.1.2 Conditional testing

In this part of the testing each of the conditions were tested to both true and false aspects. And all the resulting paths were tested. So that each path that may be generate on particular condition is traced to uncover any possible errors.

6.3.1.3 Data flow testing

This type of testing selects the path of the program according to the location of definition and use of variables. This kind of testing was used only when some local variables were declared. The definition-use chain method was

used in this type of testing. These were particularly useful in nested statements.

6.3.1.4 Loop testing

In this type of testing all the loops are tested to all the limits possible. The following exercise was adopted for all loops:

- All the loops were tested at their limits, just above them and just below them.
- All the loops were skipped at least once.
- For nested loops test the inner most loop first and then work outwards.
- For concatenated loops the values of dependent loops were set with the help of connected loop.
- Unstructured loops were resolved into nested loops or concatenated loops and tested as above.

Each unit has been separately tested by the development team itself and all the input have been validated.

6.4 Methodology

With this testing approach, all the specs were ready for a prototype, and a plan was already built to be shown; the tester started writing the code and saw if it could obtain the same results that the specs mentioned. This way, the specs were tested on each prototype, and continuous testing was applied. This also helped to minimize the testing that would have to be implemented at the end of the software lifecycle. In the process, all aspects of the software were tested.

Steps to follow while implementing the methodology are as follows:

1. Start with a base function that you want to implement.
2. Create a document with the detailed requirement definition, an activity diagram with a description of the flow, database tables to be used, a component diagram, and a description of each component with the precondition and tables that would be affected by the component.
3. Give the document to the tester, and work with the tester while they write the code to check if the steps in the document can be implemented and if the result of each use case can be achieved.
4. If the tester finds a step difficult to implement or if it is missing additional information to implement the functionality, then go to step 2; otherwise, go to step 3.
5. Ask the tester to log on all the errors and difficulties they encountered while working on the prototype implementation
6. Once the prototype is done and the results between the developer's prototype and tester's prototype match, work on the other requirement, and expand the prototype to final software.
7. When the testing approach was implemented, the following pros and cons regarding the testing approach were realized.

6.4.1 Pros of using the methodology

- Helps give a better understanding about the requirements.
- Better design at the end of the cycle.
- Reduced testing to be performed at the end of the cycle
- Documents produced would be of higher quality.
-

6.4.2 Cons of using the methodology

- The person working on the document should be experienced.
- There are increased time and money involved with testing.

- Different viewpoints for the same problem can lead to varying results.

6.5 Interface Testing

6.5.1 Functional requirements

This section lists the functional requirements used for creating the test-case table, the test cases that were used to verify the interface table, and the results for the test-cases table. the table include test case and his Test results

Test Case 1:

Title	Receiving confirmation Email.
Description	Verify the email address of the user.
Precondition	Enter all needed personal information.
Assumption	The system should send email with a link which verifies the email of the user.
Test Steps:	<ol style="list-style-type: none"> 1. Enter the personal information (The email address should be the google email "Gmail"). 2. Click on the button "Register." 3. Check Gmail. 4. Click on the link in the Email.
Expected Result:	The login page will open, and the user can login to the website.
Actual Result:	The login page will open and login to the website successfully.

Test Case 2:

Title	Login System.
Description	Login to the application.
Precondition	The user should register and verify the email.
Assumption	The home page will open.
Test Steps:	1.Enter the email address and password. 2.Click on the button “Login.”
Expected Result:	the home page will open.
Actual Result:	the home page open.

Test Case 3:

Title	View all books
Description	User can view all books
Precondition	After the user logged-in he can view all books in home page.
Assumption	The home page will show all books.
Test Steps:	1.logging-in 2.open home page
Expected Result:	the home page show all books.
Actual Result:	the home page show all books.

Test Case 4:

Title	book interaction
Description	User can interact with book
Precondition	After the user opened the book page he can interact with by making like "add to favorite",add to wishlist,add to already-read give rate
Assumption	User can interact with book
Test Steps:	<ol style="list-style-type: none">1. open book page2. add to favorite3. add to wishlist4. add to already-read5. give rate
Expected Result:	User can interact with book
Actual Result:	User can interact with book

Test Case 5:

Title	buy book
Description	User can buy book
Precondition	After the user click on add to cart button he will redirected to buy page in on of the book stores platforms

Assumption	User can buy a book
Test Steps:	<ol style="list-style-type: none"> 1. open book page 2. click on add to cart button 3. user redirected to books store platforms
Expected Result:	User can buy a book
Actual Result:	User can buy a book successfully

Test Case 6:

Title	search for a book
Description	User can search for a book
Precondition	After the user click on search button in home page he can write the title of the book he want to find
Assumption	User can search for a book
Test Steps:	<ol style="list-style-type: none"> 4. open home page 5. click on search button 6. user write the of the book 7. click on the result 8. open the book page
Expected Result:	User can can search for a book
Actual Result:	User find the book successfully

Chapter VII

System Security

7.1 Introduction

The purpose of security in a book recommendation mobile application is to protect the app and user data from unauthorized access, theft, and other malicious activities. Security is important in this type of application because it deals with user data, such as reading preferences, search history, and user profiles, which may contain sensitive information. If this data is compromised, it may lead to privacy violations and financial losses. Therefore, it is crucial to ensure that the application is secure and protected against common security threats such as hacking, malware, and phishing attacks.

A secure book recommendation mobile application can provide a sense of trust and confidence to users. It ensures that their data is handled responsibly and that their privacy is respected. Additionally, it can help to maintain the reputation of the application and the company behind it. A security breach can have severe consequences, including loss of customers, legal action, and damage to the brand image. Therefore, it is essential to prioritize security in the development and maintenance of book recommendation mobile applications.

7.2 Threat Assessment

There are several potential security threats and vulnerabilities that a book recommendation mobile application may face. Here are some common examples:

1. **Insecure data storage:** If the application stores user data in an insecure manner, such as in plain text or with weak encryption, it may be vulnerable to data breaches and theft.
2. **Insecure communication:** If the application communicates with servers or other devices using insecure protocols, such as plain HTTP, it may be vulnerable to man-in-the-middle attacks and other forms of interception.
3. **Injection attacks:** If the application does not properly validate user input, it may be vulnerable to injection attacks such as SQL injection and cross-site

scripting (XSS).

4. **Malware:** If the application is infected with malware, it may be vulnerable to data theft, unauthorized access, and other malicious activities.

5. **Phishing attacks:** If the application sends emails or other messages to users, it may be vulnerable to phishing attacks, where malicious actors attempt to trick users into disclosing sensitive information.

6. **Unauthorized access:** If the application does not properly implement access controls, it may be vulnerable to unauthorized access by malicious actors.

7. **Social engineering:** If the application relies on user input for authentication or other security measures, it may be vulnerable to social engineering attacks where malicious actors trick users into disclosing sensitive information or performing unauthorized actions.

It is important to note that this list is not exhaustive, and other security threats and vulnerabilities may exist depending on the specific design and implementation of the book recommendation mobile application. Developers should perform thorough security testing and consider working with security experts to identify and address potential threats and vulnerabilities.

7.3 Security Objectives

The specific security objectives that a book recommendation mobile app aims to achieve may vary depending on the application's design and the type of data it handles. However, here are some general security objectives that a book recommendation mobile app may aim to achieve:

1. **Confidentiality:** The book recommendation mobile app should aim to ensure that user data, such as reading preferences and search history, is kept confidential and protected from unauthorized access or disclosure.

2. **Integrity:** The book recommendation mobile app should aim to ensure that user data is accurate and reliable and has not been tampered with or modified by unauthorized parties.

3. **Availability:** The book recommendation mobile app should aim to ensure that users can access the application and their data when needed, without any disruptions caused by security breaches or other issues.

4. **Authentication:** The book recommendation mobile app should aim to ensure that users are properly authenticated and authorized to access their data, and that only authorized users can perform sensitive actions, such as changing account settings or making purchases.

5. **Authorization:** The book recommendation mobile app should aim to ensure that users have access only to the data and features they are authorized to use, based on their roles and permissions.

6. **Non-repudiation:** The book recommendation mobile app should aim to ensure that users cannot deny their actions, such as making a purchase or providing a book review, by providing mechanisms for tracking and logging user activities.

By achieving these security objectives, a book recommendation mobile app can provide users with a secure and trustworthy experience, protect their data from unauthorized access or disclosure, and maintain the integrity and availability of the application.

7.4 Security Controls

The security controls and measures that may be implemented to mitigate the identified threats in a book recommendation mobile app can vary depending on the specific threats and vulnerabilities that have been identified. However, here are some examples of security controls and measures that may be implemented:

1. **Secure data storage:** Implementing proper encryption and access controls for user data in databases and file systems to prevent unauthorized access in case of a data breach.
2. **Secure communication:** Implementing secure communication protocols, such as HTTPS, to encrypt data in transit and prevent interception by third parties.
3. **Input validation:** Implementing proper input validation and sanitization techniques to prevent injection attacks such as SQL injection and cross-site scripting (XSS).
4. **Malware protection:** Implementing anti-malware solutions to detect and remove malware from the mobile app and prevent unauthorized access by malicious actors.
5. **Role-based access control:** Implementing role-based access control (RBAC) to restrict access to sensitive data and functionalities based on user roles and permissions.
6. **Logging and monitoring:** Implementing logging and monitoring mechanisms to track user activities and detect any suspicious activities or security breaches.

These security controls and measures can help mitigate the potential threats and vulnerabilities in a book recommendation mobile app, but it is important to note that the implementation of security is an ongoing process and should be continuously reviewed and updated to ensure the application's security posture. Additionally, developers should work with security experts to identify and address potential threats and vulnerabilities specific to their application and use case.

7.5 Data Protection

Sensitive data such as user preferences and search history should be protected throughout the book recommendation mobile app lifecycle to prevent unauthorized access, disclosure, and modification. Here are some general practices that can be implemented to protect sensitive data throughout the app lifecycle:

1. **Secure data storage:** Sensitive data should be stored in an encrypted format, either on the mobile device or on the server, with access controls in place to ensure that only authorized users can access the data.
2. **Secure communication:** All communication between the mobile app and the server should be encrypted using secure protocols such as HTTPS to prevent interception and unauthorized access.
3. **Data minimization:** Collect only the minimum amount of user data required for the app's intended functionality and delete any unnecessary data as soon as possible.
4. **User authentication:** Implement strong user authentication mechanisms, such as password policies, biometric authentication, or multi-factor authentication, to ensure that only authorized users can access the sensitive data.
5. **Access controls:** Implement role-based access controls (RBAC) to restrict access to sensitive data based on user roles and permissions.
6. **Logging and monitoring:** Implement logging and monitoring mechanisms to track user activities, detect any suspicious activities or security breaches, and respond accordingly.
7. **Regular security assessments:** Conduct regular security assessments and penetration testing to identify and address any vulnerabilities or weaknesses in the app's security posture.

It is important to note that the protection of sensitive data is an ongoing process, and developers should continuously monitor and update their security measures to adapt to changing threats and vulnerabilities. By implementing these best practices, developers can help ensure that sensitive data is protected throughout the book recommendation mobile app lifecycle, maintaining the privacy and trust of their users.

7.6 Access Control

Access control mechanisms are a critical part of securing the book recommendation mobile app and its resources. Here are some examples of access control mechanisms that may be implemented to restrict access:

1. **Authentication:** Authentication mechanisms such as passwords, biometric authentication, or multi-factor authentication can be used to verify the identity of users and ensure that only authorized users can access the app and its resources.
2. **Authorization:** Authorization mechanisms such as role-based access control can be used to limit access to specific resources based on the user's role or permission level.
3. **Access control lists:** Access control lists can be used to define which users or groups have access to specific resources or features within the app.
4. **Time-based access control:** Time-based access control can be used to limit access to the app or specific resources based on the time of day or specific dates.
5. **Session management:** Sessions can be used to manage user access to the app and its resources, controlling the duration and scope of the user's access.
6. **Encryption:** Encryption can be used to protect sensitive data stored on the

mobile device or transmitted over the network, ensuring that only authorized users can access the data.

7. Security policies: Security policies can be defined and enforced to ensure that all users and system components adhere to security best practices and policies.

By implementing these access control mechanisms, developers can help ensure that only authorized users have access to the book recommendation mobile app and its resources, reducing the risk of unauthorized access, data breaches, and other security incidents.

7.7 Network Security

Protecting the book recommendation mobile app's network infrastructure is critical to ensuring the security and availability of the application. Here are some security measures that can be implemented to protect the app's network infrastructure:

- 1. Firewall:** A firewall can be installed to prevent unauthorized access to the network infrastructure by blocking traffic from unauthorized sources and allowing only authorized traffic.
- 2. Network segmentation:** Network segmentation can be used to separate the app's network infrastructure into logical segments, reducing the risk of security breaches and limiting the impact of any potential security incidents.
- 3. Intrusion detection and prevention system:** An intrusion detection and prevention system (IDPS) can be implemented to detect and prevent unauthorized access attempts or suspicious network activities.

4. **Virtual private network:** A virtual private network (VPN) can be used to encrypt all traffic between the mobile app and the server, protecting the data from interception and unauthorized access.

5. **Network monitoring:** Network monitoring tools and techniques can be used to monitor the network infrastructure for any suspicious activities or security breaches, enabling prompt response to mitigate the impact of any incidents.

6. **Regular security updates and patches:** Regular security updates and patches can be applied to the network infrastructure to address any known vulnerabilities or weaknesses.

7. **Access control:** Access control mechanisms, such as authentication and authorization, can be implemented to restrict access to the network infrastructure only to authorized personnel.

By implementing these security measures, developers can help protect the book recommendation mobile app's network infrastructure from unauthorized access, data breaches, and other security incidents, ensuring the security and availability of the application.

Chapter VIII

Conclusion and Future Work

8.1 Conclusion

Book Recommendations Android App

Our Android app aims to provide personalized book recommendations to users using collaborative filtering models. We have implemented both user-based and item-based collaborative filtering techniques to enhance the accuracy and relevance of the recommendations.

The app utilizes a combination of Node.js, Express.js, MongoDB, and Mongoose as the backend technologies. Node.js allows us to build a fast and scalable server, while Express.js provides a robust framework for handling HTTP requests and building APIs. MongoDB, a NoSQL database, is used to store and manage the book data efficiently. Mongoose is employed as an Object Data Modeling (ODM) library to simplify the interaction with the MongoDB database.

On the client side, we have chosen to use Android as the operating system for our app. Android offers a vast user base and a developer-friendly environment. With the help of the Android SDK and Java, we have developed a responsive and intuitive user interface that allows users to browse and explore books easily.

The collaborative filtering models are the core of our recommendation system. User-based collaborative filtering analyzes the preferences of similar users to generate recommendations for a specific user. It takes into account the ratings and preferences of users with similar tastes to suggest books that the user might enjoy. Item-based collaborative filtering, on the other hand, focuses on the similarities between books and recommends items that are similar to the ones the user has shown interest in.

By combining these two techniques, our app provides accurate and personalized book recommendations to users based on their preferences and behaviors. Users can rate and review books, which further refines the recommendation algorithm.

In conclusion, our Android app leverages user-based and item-based collaborative filtering models to deliver personalized book recommendations. The backend implementation using Node.js, Express.js, MongoDB, and Mongoose ensures a robust and efficient server-side infrastructure. With its intuitive user interface and accurate recommendations, our app aims to enhance the reading experience of book lovers.

8.2 Benefits

Our book recommendations Android app utilizes user-based and item-based collaborative filtering models, backed by a powerful technology stack consisting of Node.js, Express.js, MongoDB, Mongoose, and Flask. The app offers a wide range of features that provide numerous benefits to users. Some of these benefits include:

1. **Convenient user management:** Easy sign-in, sign-up, and log-out features allow personalized experiences and access to saved preferences and book lists.
2. **Personalized book preferences:** Users can update their profiles with reading preferences, favorite genres, and relevant information, leading to personalized book recommendations.
3. **Enhanced book interactions:** Users can like, add to favorites, mark as already read, or add to the wishlist, enabling effective management of reading choices and personalized recommendations.
4. **Seamless book purchasing:** Users can buy books directly within the app, adding to the cart, specifying quantities, and completing secure purchases without switching to external platforms.
5. **User ratings and reviews:** Users can provide ratings and reviews for books, aiding others in making informed decisions and fostering community engagement.
6. **Comprehensive book search:** Robust search functionality allows users to find books easily based on title, author, genre, or keywords.
7. **Personalized recommendations:** The app generates personalized recommendations based on users' favorite books and reading history, expanding their choices and introducing new authors and genres.
8. **Discover similar books:** Users can explore books similar to their favorites, exposing them to a wider range of related books and themes.

9. **Online book list and accessibility:** Users can save their book lists online, ensuring access from any internet-connected device, providing convenience and flexibility.

8.3 Future work

As any project being developed there are some limitations for the current system to which solutions and improvements can be provided as a future development:

1. **Performance Optimization:** Currently, the system is deployed on a free web host, which may result in suboptimal performance. To enhance the user experience, it is recommended to explore premium web hosting options that offer better performance and reliability.
2. **Upgrade MongoDB Cluster:** To ensure scalability and improved performance, upgrading the MongoDB cluster to a more robust and scalable configuration is recommended. This will allow for better handling of increased data volume and user interactions.
3. **Integration of Content-Based Recommendation Model:** Implementing a content-based recommendation model would enhance the book recommendation system by considering the characteristics and attributes of books. This model will analyze factors such as book titles, authors, genres, synopses, and other relevant textual features to generate recommendations based on content similarity.
4. **Expansion of Book Collection:** Increasing the number of books in the app's database will provide users with a broader selection of recommendations and improve the overall user experience. Continuously updating and expanding the book collection will cater to diverse reading preferences and interests.
5. **Addition of Admin Dashboard React Webpage:** Introducing an admin dashboard

using React will enable administrators to manage the app more efficiently. This dashboard will provide a user-friendly interface for administrators to monitor user activities, manage book data, analyze system performance, and perform administrative tasks.

6. Implementation of Author Signup/In React Webpage: Creating a dedicated webpage using React for author signups and logins will facilitate the onboarding process for authors. Authors can create accounts, manage their profile information, and access features like uploading their books and interacting with readers.
7. Author Form for Book Upload: Developing an author form within the app will streamline the process of book uploads. Authors can provide relevant book details, cover images, synopses, and other essential information through a structured form. This will ensure accurate and consistent data entry and ease the management of book information.
8. Addition of Comments Section for Books: Including a comments section for each book will encourage reader engagement and foster a sense of community. Users can leave reviews, ratings, and comments about the books they have read, enabling other users to make more informed decisions when selecting books.

Chapter IX

References

1. <http://www.agilemodeling.com/artifacts/dataFlowDiagram.htm>
for definition of Data Flow Diagram.
2. <https://www.nodejs.org/en/docs> for information about Node.js
3. <https://www.expressjs.com> for information about Express.js
4. <https://www.mongodb.com/docs/> for information about MongoDB
5. <https://mongoosejs.com/docs/> for information about Mongoose
6. cs.github.com/en for information about GitHub

الملخص

يصف هذا الملخص تطبيق الهاتف المحمول الذي يوصي المستخدمين بالكتب بناءً على تفضيلات القراءة الخاصة بهم. تم تصميم التطبيق لتوفير تجربة قراءة مخصصة، مع واجهة سهلة الاستخدام تجعل من السهل العثور على كتب جديدة واكتشافها.

يستخدم التطبيق خوارزمية متطورة تأخذ في الاعتبار تاريخ قراءة المستخدم وتفضيلات النوع وتقييمات الكتب السابقة لتقديم التوصيات. يمكن للمستخدمين أيضًا تصفح القوائم المنسقة، مثل «الإصدارات الجديدة» أو «الأكثر مبيعًا»، أو البحث عن عناوين أو مؤلفين محددين.

يتضمن التطبيق جانبًا اجتماعيًا، مما يسمح للمستخدمين بالتواصل مع الأصدقاء ومشاركة توصيات الكتب. يمكن للمستخدمين أيضًا إنشاء قوائم القراءة الخاصة بهم وتتبع تقدمهم عند القراءة. يتضمن التطبيق ميزات مثل ملخصات الكتب والمراجعات والتقييمات من المستخدمين الآخرين، بالإضافة إلى روابط لشراء الكتب مباشرة من التطبيق. كما يقدم توصيات مخصصة للكتب الصوتية والكتب الإلكترونية، مما يجعله رفيق قراءة شامل.