INDY-1 Blue — Book Dating App
Final Report
CS 4850 - Section 02 – Fall 2024
Professor Perry – Nov 4, 2024

Chelsea Mandel
Documentation

Sebastian Arechaga
Developer

Obie Ezeobi
Developer

Keaton Moneymaker
Developer

Team Members:

Name	Role	Cell Phone / Alt Email
Obie Ezeobi	Developer	678.655.0628 oezeobi@students.kennesaw.edu
Chelsea Mandel	Documentation	423.285.4514 cmandel1@students.kennesaw.edu
Sebastian Arechaga	Developer	678.213.6014 sarechag@students.kennesaw.edu
Keaton Moneymaker	Developer	706.260.5079 kmoneym1@students.kennesaw.edu
Sharon Perry	Project Owner or Advisor	770.329.3895 Sperry46@kennesaw.edu

Lines of Code: 2985
Total man hours: 491
Total Project Components: 38
Website Link: INDY-01-Blue-Book Dating App
GitHub Link: book-dating-app

Table of Contents

1.0 Project Plan

1.1 Project Overview / Abstract (Research)

Bridging the gap between technology and literature, our app offers a fresh and interactive approach to book discovery, turning the search for a good read into a delightful and personalized journey.

Our Book Dating Software redefines the convenience and carefree experience of modern dating apps as a revolutionary tool for book lovers interested in finding their next great read. Users are presented with a queue of potential candidates for new stories that can be "accepted" or "rejected" at their discretion. Each item in this queue contains information on the book such as its genre and a brief description of its contents. Titles that seem interesting to the user are added to a "To Be Read" list, and users can set filters to choose which types of books to receive in their queue.

1.2 Platform

Front-End Tech Stack:

- HTML
- CSS
- JavaScript
- Thymeleaf Spring Boot

Back-End Tech Stack:

- Framework: Spring Boot MVC.
- Language: Java (Java 21).
- Database: MongoDB Atlas, (NoSQL database).
- Security: Spring Security.

1.3 Collaboration Tools

Communication — Teams, Cellphones (Call/Text)
Collaboration — Discord
Version Control — GitHub

1.4 Deliverables

- Team/Project Selection document (Individual Assignment)
- Weekly Activity Reports (WARs – Individual Assignment)
- Team Status Report (TSR – Group Assignment)
- Peer Reviews (Individual Assignment)
- Project Plan (Group Assignment)
- SRS, SDD, STP & Dev Doc (Group Assignment)
- Present Prototype for Peer Review (Group Assignment)
- Final Report Package (Group Assignment)
  - Final Report (Group Assignment)
  - Source Code (Group Assignment)
  - Website (Group Assignment)
  - Video Demo (Group Assignment)
- C-Day Application/Submission

1.5 Meeting Schedule

Aug. 26 4:00pm – 5:00pm
Sep. 9 4:00pm – 5:00pm
Sep. 16 4:00pm – 5:00pm
Sep. 23 4:00pm – 5:00pm
Sep. 30 4:00pm – 5:00pm
Oct. 7 4:00pm – 5:00pm
Oct. 14 4:00pm – 5:00pm
Oct. 21 4:00pm – 5:00pm
Oct. 28 4:00pm – 5:00pm
Nov. 4 4:00pm – 5:00pm

2.0 Software Requirements Specification

2.1 Introduction

2.1.1 Overview

This document defines the functional and non-functional requirements for the Book Dating Application software product. This document provides a detailed description of the system's requirements so that they can be easily understood by stakeholders and developers.

This document is not intended to address development-related issues such as scheduling, cost analysis, or development methods. Such issues will be addressed in a separate design document.
The Book Dating App is a web-based application which aims to be a service for readers to find books and discover what matches their reading preferences.

2.1.2 Project Goals

The goal of the Book Dating App project is to develop a website utilizing an extensive collection of books, categorized and labelled by relevant distinguishers such as genre, for users to peruse and select from. Users will create profiles that will house saved collections of the books chosen while browsing. Suggested books will account for user preferences. Advertising will be added as a method of monetization.

2.1.3 Definitions and Acronyms

Application or Product – the software system specified in this document
DBMS – Database Management System
Candidate- A book in the user's queue of suggestions pending acceptance or rejection

2.1.4 Assumptions

- It is assumed that users will possess compatible devices with a stable internet connection with which to use the Application.
- It is assumed that third-party APIs such as book databases and authentication services will be reliable and consistently available.
- It is assumed that the development tools, libraries, and platforms needed to build the app are accessible and reliable.
- It is assumed that users have a basic understanding of how to navigate and interact with apps.
- It is assumed that the books featured in the app are available for purchase, loan, or download in the regions where the app is released.

2.2 Design Constraints

2.2.1 Environment

The Application will interact with existing third-party book databases such as Google Books to populate the user's feed with relevant book information. Additionally, the Application will cooperate with authentication services such as Google to allow users to create accounts or sign in to existing accounts. Furthermore, the Book Dating Application will make use of a non-relational DBMS for storing data such as users' account information and book information such as title, genre, etc.

2.2.2 User Characteristics

The primary end users of the Application are individuals who enjoy reading and are looking for a convenient way to discover new books based on their preferences. Users will come

from a wide range of demographics with diverse reading habits, ranging from casual readers to devoted literature enthusiasts. Most end users are expected to be somewhat comfortable with web applications, but the Application is intended to be accessible to users of all skill levels.

2.2.3 System Constraints

In some cases, the design and implementation of the Application are subject to specific constraints imposed by external requirements. Factors such as compliance with external standards and regulations, budget limitations, and organizational policies may impact the options available for development of the Application.

2.3 Functional Requirements

This section outlines the functional requirements of the Application, including various features and the expected behavior of the system.

2.3.1 Login

Account Creation

The system shall allow users to create an account using their email address or a third-party authentication service such as Google or Facebook. Users will be prompted to create a password, which will be stored in a secure manner. Finally, users will be sent a verification email upon account creation and will be signed in to the application

Login with Username and Password

The system shall allow users to log in to their existing accounts using their email address or the third-party authentication service of their choice. The credentials entered will be securely validated, and access will be provided only if the credentials are valid. An error message will be shown if either the username or password are incorrect.

2.3.2 Display Main Page

Book Suggestions

On the main page, the system shall present users with a queue of books that are personalized to the user's interests. Each book serves as a potential candidate for being accepted or rejected at the user's discretion. Books in the queue will be presented along with relevant information about the title such as genre, theme, and more.

Accept/Reject Book

Users shall be able to accept or reject the candidates they will be presented with using dedicated "accept" and "reject" interactions. Upon acceptance of a candidate, the book and its relevant information will become available in the Saved Books section of the Profile page. Candidates that are rejected are not added to Saved Books and are not recommended again. Regardless of the user's decision to accept or reject a candidate, the user's preferences will be updated within a recommendation algorithm, which will affect the future candidates that appear in the suggestion queue.

User Preference Filtering

The system shall provide filtering options that allow users to refine their searches based on their preferences and specific criteria. Users will be able to select filters to update their search queue, changing their book suggestions. These preferences shall be saved for future uses of the Application.

2.3.3 Display Profile Page

Profile Details

The profile page shall provide details to the user such as personal information, including but not limited to their username and email address. The users' preferences and filters will also be available via the Profile page, allowing them to update these preferences. Users will also be able to securely update their information such as their username and password.

Saved Books

The system shall allow the user to view and manage the candidates they have saved. In the Saved Books section of the Profile page, users will be able to organize their saved books by genre (e.g. "Action," "Horror," etc.), remove books from their Saved Books list, and review information regarding the accepted candidates.

2.4 Non-Functional Requirements

2.4.1 Security

The system shall enact measures to protect sensitive data such as user login credentials, user preferences and other personal information, and more. Passwords shall be stored securely using strong hashing algorithms and all sensitive data shall be encrypted to high standards. All client-server data transfers will make use of the HTTPS protocol for more secure messaging.

2.4.2 Capacity

The system shall make significant accommodations in its design for factors such as scalability, performance under heavy load, and data storage. The system shall be able to accommodate many concurrent users during peak usage times and ensure that multiple book recommendation queries may be processed simultaneously. Furthermore, the system shall be able to accommodate as many registered users as necessary as well as support the storage of detailed information from a large database of books. This will be accomplished by taking measures such as efficient algorithm and database design, and advanced cloud infrastructure.

2.4.3 Usability

The system shall prioritize usability by having a simple yet efficient user interface that requires minimal actions to reach a desired page. The system shall be designed for use in multiple environments with varying noise levels, so dependence on audible output for feedback will be kept at a minimum. Furthermore, accessibility features shall be

incorporated into the web page in accordance with Web Content Accessibility Guidelines. Finally, the system shall provide efficient UI performance with minimal response time.

2.4.4 Other

Other non-functional requirements may include, but are not limited to:

1. Support for multiple common web browsers such as Google Chrome and Microsoft Edge
2. Consistent server uptime
3. Sufficient performance, including fast startup times and low latency

3.0 Software Design Document

3.1 Introduction and Overview

This document outlines the design of the software application, detailing how the requirements specified in the SRS will be met. The document is intended to provide a high-level overview of the system architecture and provide details on the individual components that make up the system. This document will provide guidance to the development team regarding the structure and architecture of the system being developed.

3.2 Design Considerations

3.2.1 Assumptions and Dependencies

The following are assumptions or dependencies regarding the software and its use. The use of the application is dependent on a stable internet connection. Account creation is dependent on the user having access to an email address or an accepted third-party authentication service. It is assumed that end users will interact with the product in a predictable manner. Familiarity with online profile management, as in the case of selecting a username and password and interacting with profile settings, is assumed. Familiarity with the concept of online dating website norms will, to an extent, be assumed on the part of the user. Such norms include an understanding of the binary accept or reject system and the user-preference guided suggestions as these will not be explicitly explained.

3.2.2 General Constraints

Describe any global limitations or constraints that have a significant impact on the design of the system's software (and describe the associated impact). Such constraints may be imposed by any of the following (the list is not exhaustive):

- Hardware or software environment
- End-user environment
- Availability or volatility of resources
- Standards compliance
- Interoperability requirements
- Interface/protocol requirements

- Data repository and distribution requirements
- Security requirements (or other such regulations)
- Memory and other capacity limitations
- Performance requirements
- Network communications
- Verification and validation requirements (testing)
- Other means of addressing quality goals
- Other requirements described in the requirements specification

3.2.3 Development Methods

The Agile Software Development methodology will be the primary approach for this software design. Focus will be placed on maintaining a steady work pace, delivering working software consistently, keeping open and frequently used communication channels, prioritizing simplicity of design over intricate design, and adapting to change. Some portions of the methodology that will not be applicable to this project are the maintenance process phase and the distinction of multiple teams.

3.3 Architectural Strategies

HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) will be used for the structure and layout of the pages. JavaScript will be used for constructing the UI (user interface). JavaScript was chosen in consideration of the Agile Software Development methodology for its simple approach to UI construction.
The source code will utilize the JavaScript programming language. MongoDB will be used for document storage. The Google Books database of books will be accessed for book candidates. A hosting platform will be chosen closer to the completion of the application. The application may be extended to a mobile application that functions on IOS and Android. Additional monetization may be added such as premium accounts with exclusive features.

3.4 System Architecture

The subsystems of this system will include the book candidate search and recommendation subsystem and the user profile system. The book candidate search and recommendation subsystem will include components performing book database retrieval and book filtering. The user profile system will include components performing user personal information retrieval and update and saved books list management.

3.5 Detailed System Design

Most components described in the System Architecture section will require a more detailed discussion. Other lower-level components and subcomponents may need to be described as well. Each subsection of this section will refer to or contain a detailed description of a system software component. The discussion provided should cover the following software component attributes:

3.5.1 Classification

Database Retrieval Component

Database retrieval will be handled by a class performing the necessary operations by accessing the database of books.

Filtering Component

Filtering will be performed through a function.

User Personal Information Customization

Retrieval and updating of the user profile information will be handled by a class accessing the database of user information.

Saved Books Component

The list of saved books will be managed by a class.

3.5.2 Definition

Database Retrieval Component

As mentioned in 3.2.1 of the SRS, there will be a selection of books which can be suggested to the user. The database retrieval component uses Google Books to populate a local non-relational database with book information.

Filtering Component

In this case, filtering refers to skewing the book search to a user-defined filter chosen based on their book interests. Referring to 3.2.3 of the SRS, the filtering component will aid in narrowing down the results of a user's book candidates.

Profile Customization

Addressing the requirement 3.3.1 in the SRS, profile customization displays and allows editing of saved user information including login credentials and book preferences.

Saved Books

Addressing the requirement 3.3.2 in the SRS, the saved books component will allow viewing, updating, and sorting of the saved books list.

3.5.3 Constraints

Database Retrieval Component

As the database will fuel our supply of books for the program, there needs to be an adequate response time to maintain the user's reservoir of goodwill. There will need to be a limit on the storage used by the database as the cache of books may overflow a user's device if is not managed.

Filtering Component

A precondition constraint for the filtering component would be the scope of categories in the filter. There will be categories to select from on the filtering component such as genre or read length, but we will have to have enough options to satisfy the use of a filtering feature in the first place.

Profile Customization

There are a handful of constraints to consider for profile customization. A few would include appropriate profile update periods, limiting profile biographies and how to store them, and a profile's state being saved and updated each session.

Saved Books

A constraint for the book saving component would be the ensured storage and access to each individual user's saved books list and the ability to update it in real time. The component should maintain the saved state of each user's list synced between the server and the user's latest changes.

3.5.4 Resources

Database Retrieval Component

The Google Books database is needed to provide the collection of books that the application draws from which will be stored using MongoDB. A potential race condition would be many users attempting to fetch the same data about a book at once and bog down the system. The solution we could apply would be a client-based cache to be used instead of making iterative external calls to the database.

Filtering Component

Using MongoDB, we will create our own filtering component which can surf through the queries of profile and books alike. There is a possibility of simultaneous searches slowing down overall performance, but this would be resolved by a stateless filter component.

Profile Customization

A MongoDB database is needed to store user information. Also, a race condition can be found when a user attempts to update their profile while logged into two separate sessions, on different devices for example, and they attempt to update their profile. Optimistic locking would be a solution to this race condition, the record of the profile is versioned, so each update checks the database and refers to the server and session copy before making an edit.

Saved Books

A connected MongoDB database will link a user's profile with their saved list of books. Similar to the profile customization, the saved books component would have a race condition if updating their list on multiple devices, which could also be solved by optimistic locking.

3.5.5 Interface/Exports

The set of services (resources, data, types, constants, subroutines, and exceptions) that are provided by this component. The precise definition or declaration of each such element should be present, along with comments or annotations describing the meanings of values, parameters, etc. …. For each service element described, include (or provide a reference) in its discussion a description of its important software component attributes (Classification, Definition, Responsibilities, Constraints, Composition, Uses, Resources, Processing, and Interface).

Database Retrieval Component

Classification:

A database retrieval class. (BookDBFetcher)

Definition:

'BookDBFetcher' will interface with whichever external database source is chosen and fetch it for the user. (Google Books, Open Library, etc.)

Responsibilities:

- Retrieve book data from external sources.
- Cache the data and format it to suit the interface.

Constraints:

- Speedy response time for positive user experience.
- Limited cache size to prevent heavy storage use.

Composition:

- Functions: queryBook(String input), fetchDetails(String bookName), cacheData(List<String> bookData)
  - ♣ 'queryBook' will search for a particular book (input) and display the results
  - ♣ 'fetchDetails' will be a function that runs once someone inspects a book closer and it will return the data necessary to display its metadata
  - ♣ 'cacheData' will run in the background as someone is loading or reading about a book and will return data which is cached for later use

Uses:

- The Filtering component will use 'BookDBFetcher' to retrieve data for the search to apply its filter to.

Resources:

This component depends on Google Books or OpenLibrary to retrieve data about multitudes of books which is then stored by a MongoDB database.

Processing:

The class is called to search for many books, then the results are formatted for use, and cached for later.

Interface:

- 'queryBook(String input) returns a List<String>
- The returned data is a list of many books.

Filtering Component

Classification:

The filtering will be done through a function (searchFor).

Definition:

This function uses the user-defined filter to search for a narrowed selection of books.

Responsibilities:

- Find books based on their genre, read time, time written, etc.
- Narrow the user's choices and start searching through a smaller sample.

Constraints:

- The limiting constraint of a filter component is the pre-defined categories necessary to facilitate a proper layer of filters which aids the user experience.

Composition:

- searchFor(bookFilter filter1) returns a list of queries matching the filter

Uses:

- The saved books component will likely use this to quickly sort through certain types of books a user has previously saved.

Resources:

The filter will use a MongoDB database, but it will be developed by us. Another resource used by the filter would be system memory used to cache the results.

Processing:

A user chooses on the website UI what preferences they want to look for in a book and the 'searchFor' component will fetch books using the 'BookDBFetcher' and apply a filter to the fetched results and then convert them to a viewable format for the user.

Interface:

- 'searchFor(bookFilter filter1)' returns the population of books that fit the filtered search.
- 'filter1' is the filter object created after the user has selected their choices.
- bookFilter is an object type which will consist of the many attributes chosen by the user to filter the search.

Profile Customization

Classification:

A class which stores a profile object in an independent database (ProfileManager).

Definition:
'ProfileManager' is a class that notes all the changes a user wants to save and updates the database with their decision.

Responsibilities:
- Fetching user data to display.
- Taking in user input and updating database.

Constraints:
- Limited biography lengths.
- Updating and saving each profile state.

Composition:
- Functions: fetchProfile(String username), updateProfile(String username, Profile newProfile)

Uses:
- Used by the front-end to fetch data for a user's page.
- Used by the back-end to update data for user changes to their profile.

Resources:
An external database will store all of the profiles, we will use MongoDB to make one.

Processing:
A user will view their page and the 'fetchProfile()' method will retrieve their details. If they want to update them they can press and button and the 'updateProfile()' will send the updated 'Profile' object to the database to update.

Interface:
- fetchProfile(String username)
  - Returns a List<String> which stores all the data for someone's profile
  - Username is the id to target a certain user's profile.
- updateProfile(string username, Profile newProfile)
  - Profile is an object type which stores all the attributes that make up a profile and newProfile is the updated version of someone's profile after they finish changing it.

Saved Books
Classification:
A class which stores the saved books for a user (Library).

Definition:
'Library' deals with fetching a certain user's list of saved books and handles editing and making changes to the list.

Responsibilities:

- Store and revise user-saved books.
- Synchronize identical lists across devices.

Constraints:

- The class must have an adequate response time as the many changes will bog down the experience. There must also be ensured access to the database.

Composition:

- Functions: add(Book book) returns void, remove(Book book) returns void, getBooks() returns Book[], an array of books

Uses:

- This class fetches the data displayed by the front-end while also updating the data stored by the back-end similar to the "Profile Customization" component.

Resources:

Once again, 'Library' will have its own MongoDB database created and utilized solely by this class.

Processing:

When a user goes to their saved books page the class uses getBooks() to retrieve the saved data for that user's account. Then use either add() or remove() to make changes and send updates to the database.

Interface:

- getBooks()
  - Returns an array of saved books tied to the user's profile.
- add() and remove()
  - Returns void.
  - Updates the data tied to the user's account and updates the user's UI to show their changes, of either removing or adding books.

3.6 Glossary

- **Agile Software Development:** A methodology for software development emphasizing adaptability and communication.
- **Candidate:** A book in the user's queue of suggestions pending acceptance or rejection.
- **CSS:** Cascading Style Sheets, a language for styling HTML elements.
- **HTML:** HyperText Markup Language, a language for constructing the basic elements of web pages.

3.7 Bibliography

A list of referenced and/or related publications.
Optimistic Locking with Version Number - Amazon Dynamodb. Amazon Web Service. (n.d.). https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/DynamoDBMapper.OptimisticLocking.html
Don't Make Me Think, Revisited | Krug, Steve
https://www.geeksforgeeks.org/stateful-vs-stateless-architecture/

4.0 Design and Architectural Drawings

4.1 UI Flow Diagram

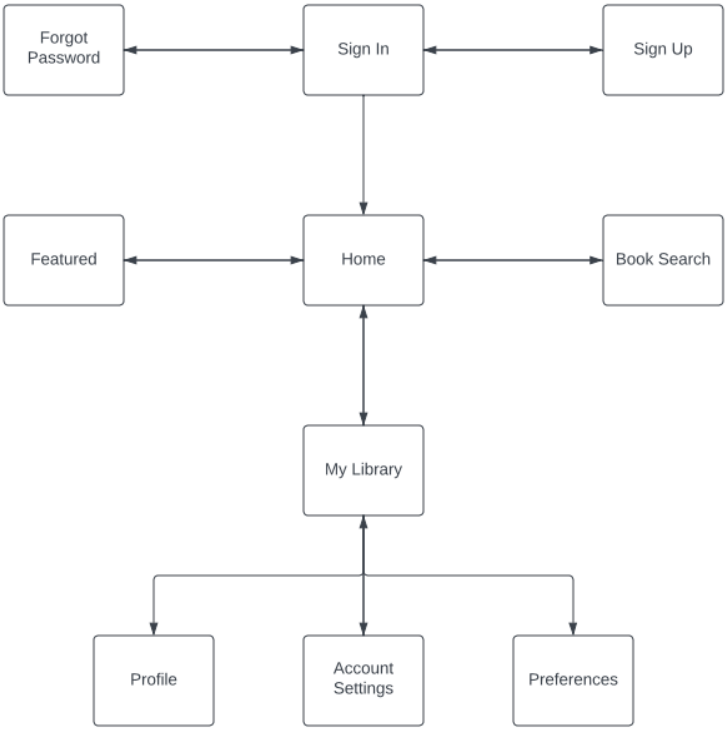

Figure 1: Flow of Screens UI Diagram

Figure 1 represents the user interface journey within our book dating app, detailing the sequence of screens a user encounters and the interactions available at each step. Each flow in the diagram demonstrates how users navigate the app and interact with its core features, providing a visual guide to user experience and helping identify intuitive pathways through the app. By mapping the user's journey, the UI flow diagram ensures that the design remains user-centered and logically organized.

4.2 UI Mock-Ups

4.2.1 Login

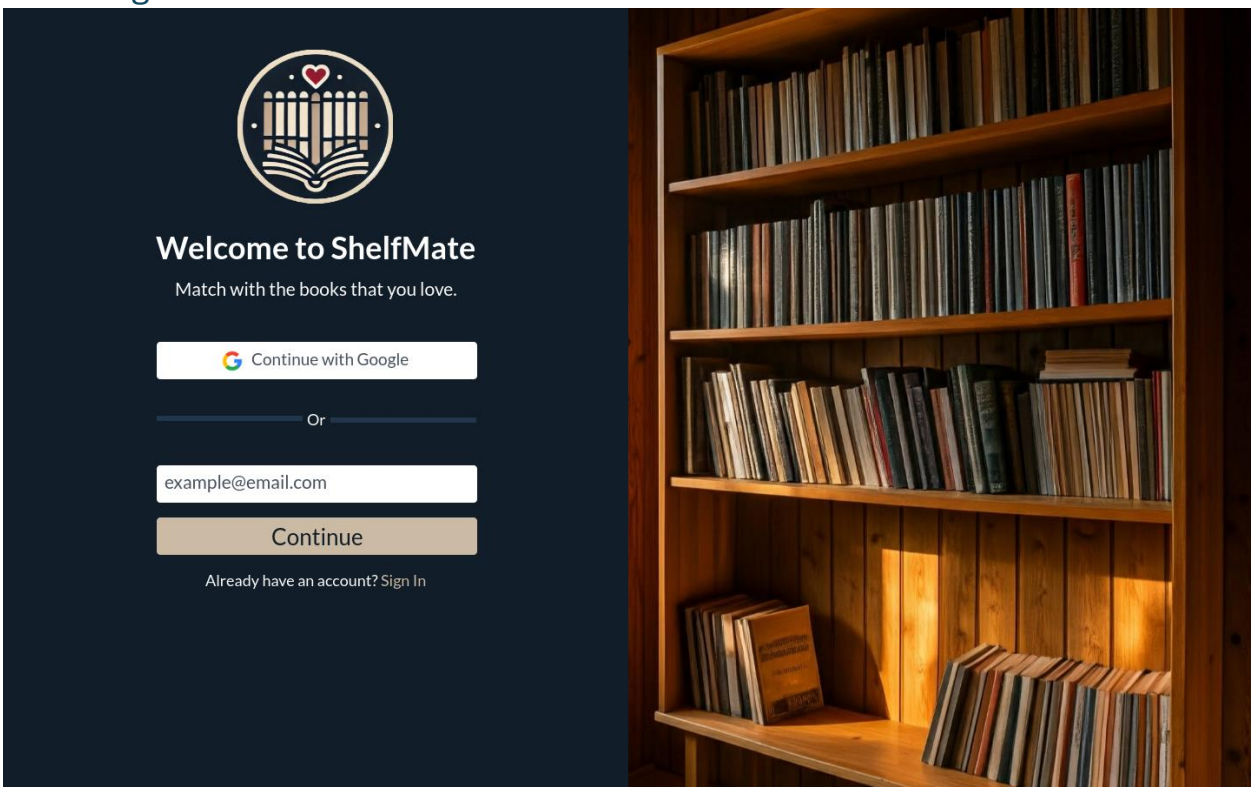

Figure 2: Login Screen

Users will begin with the Sign In page, where they will be able to input their login credentials and access the main page. If the user is unable or does not wish to login, they are able to access the Forgot Password and Sign Up page from this screen. Users will be given the option to sign in with their Google accounts.

4.2.2 Home

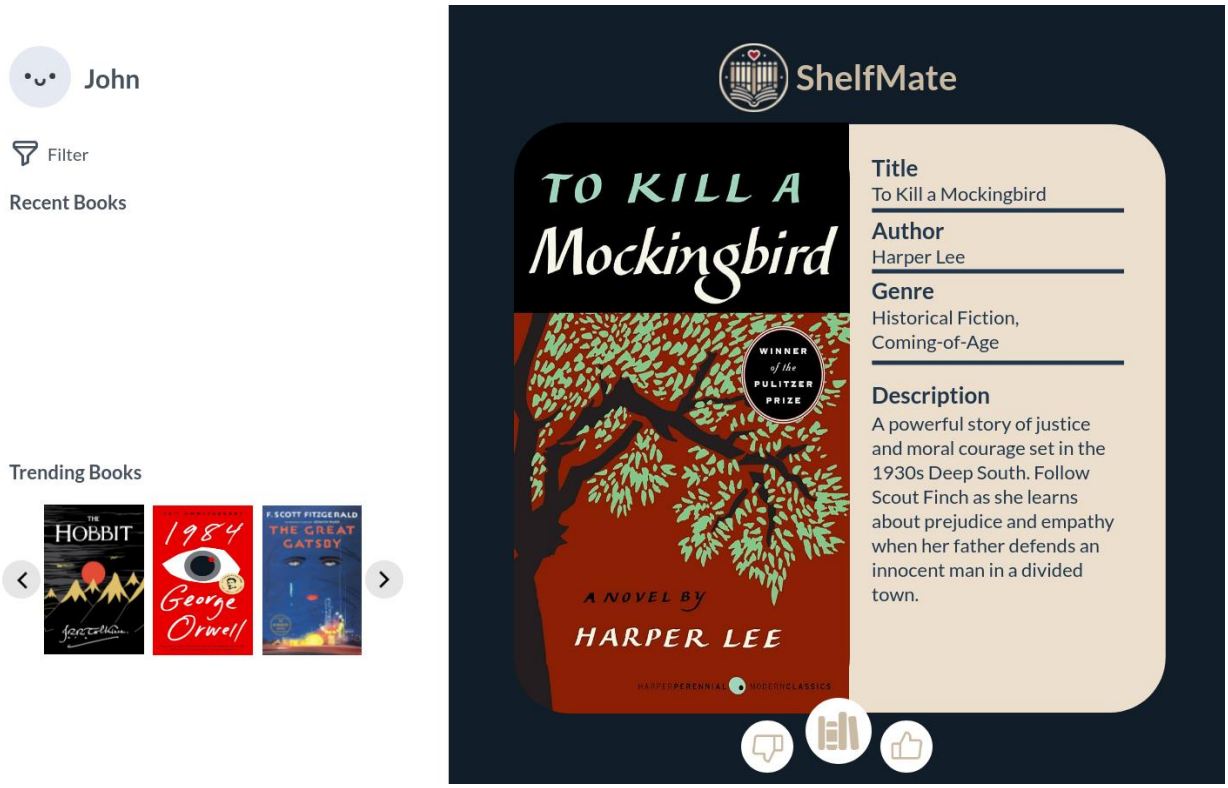

Figure 3: Home Screen

After logging in successfully, users are presented with the Home screen. This screen displays the user's recommendation queue which enables them to accept and reject candidate books. Information within the queue includes an image of the currently recommended book's cover as well as the title, author, genre, and a brief description of the book's contents. Additionally, the Home screen includes a list of recently viewed books, a list of trending books, access to preference filters, and a link to access the My Library screen.

4.2.3 My Library

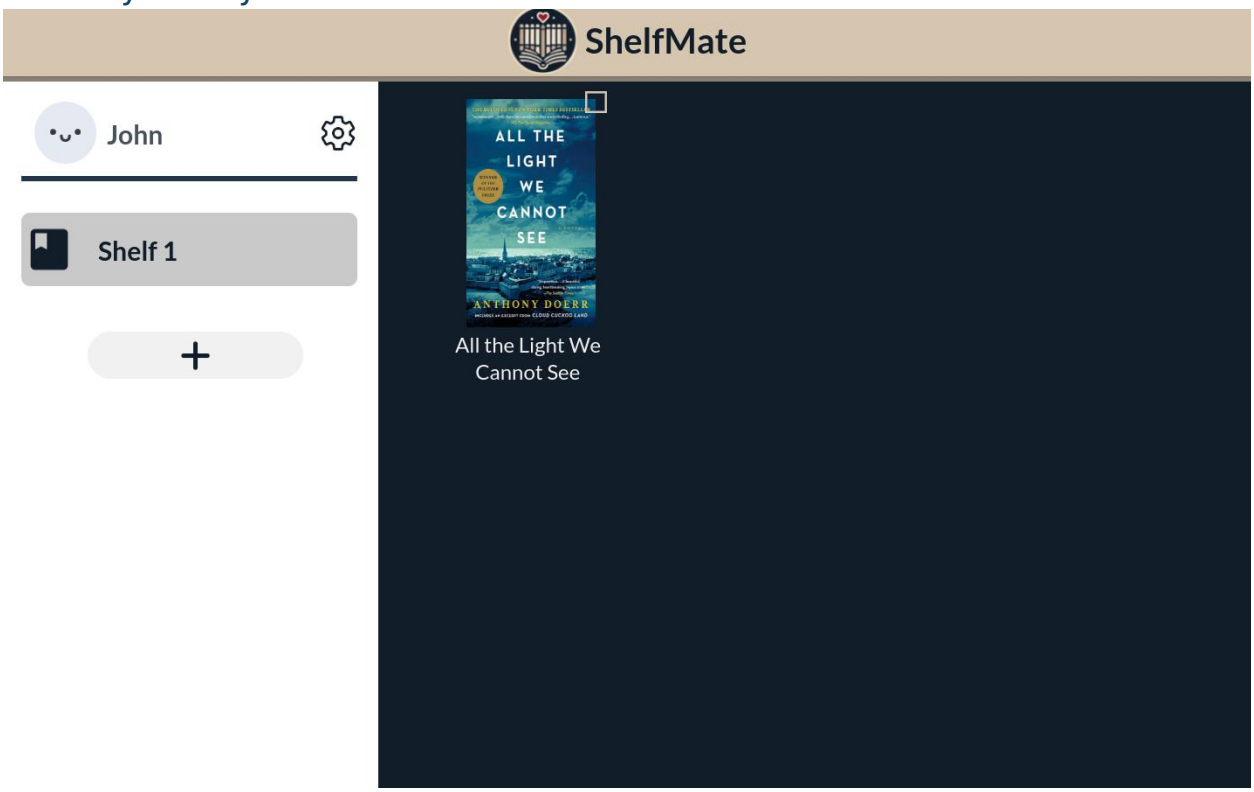

Figure 4: My Library Screen

From the My Library screen, users are able to view the books that they have accepted and filter them by genre. Users can also remove books from the library from this screen. The Profile, Account Settings, and Preferences screens can be accessed from here.

4.3 Architectural Class Diagram

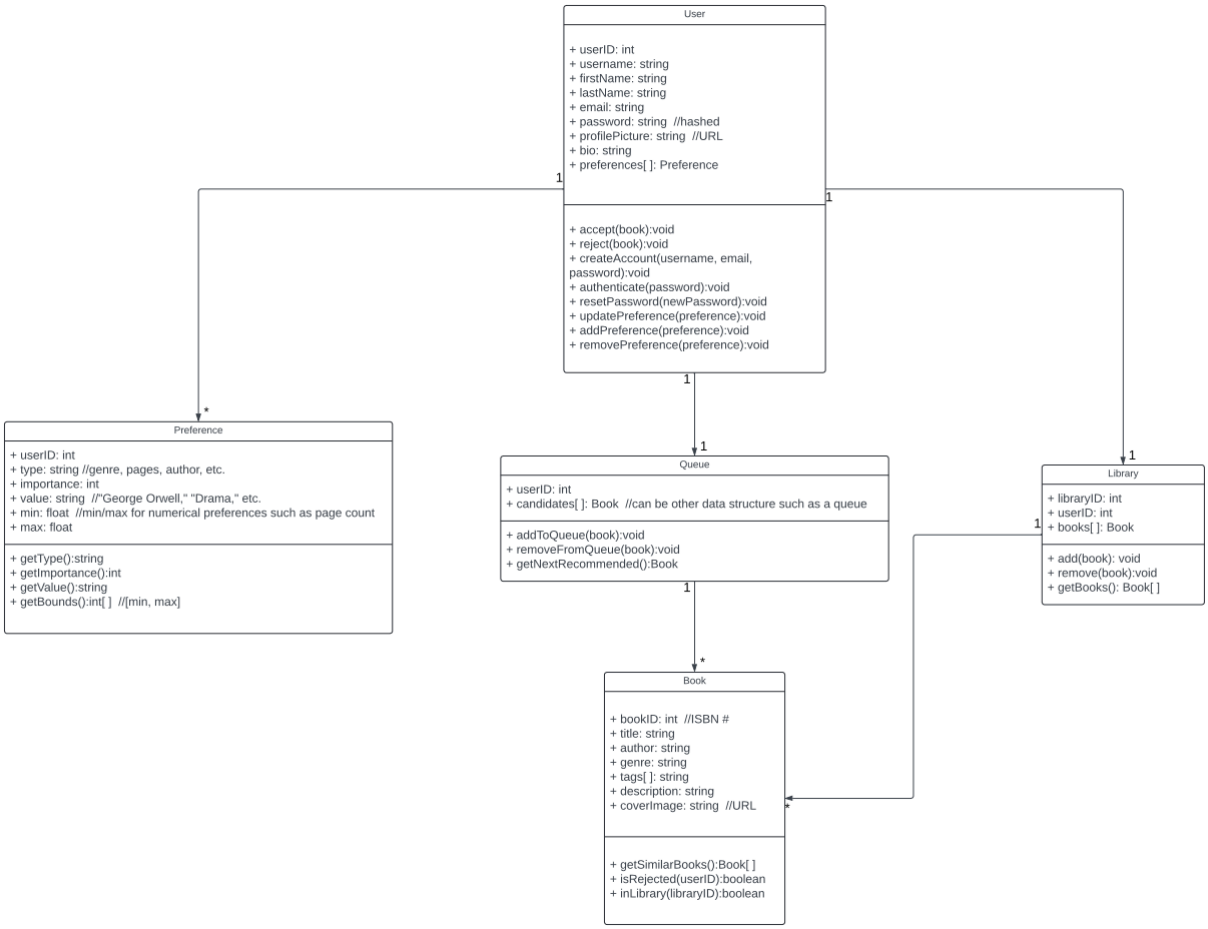

Figure 5: Architectural Class Diagram

Figure 5 demonstrates the organization, relationships, and functionality of the app's data, showcasing how different data elements interact and support core features. Key classes in this diagram include *User*, *Book*, *Library*, *Queue*, and *Preference*, each with specific attributes and methods to support actions like adding books to the library or managing user profiles. This diagram showcases how different data entities interact and are stored, making it clear how the backend manages user requests and data flows.

5.0 Version Control Description

For the book dating application, we utilized Git as our version control system, hosted on GitHub. Our repository is part of a shared organization called “INDY 01 BLUE Book Dating App.” The repository was organized with a main branch for stable code and a separate "Implementation" branch for developing new functionalities. We established the Implementation branch to keep unfinished features isolated from the main application, maintaining stability in the main branch. We followed a consistent commit practice, committing changes early and often, with clear and descriptive commit messages to effectively track the evolution of our project.

The files stored on the shared repository include source code, documentation, database-related files, and other backend components. This diverse collection of files allows the repository to serve as a central hub for viewing and managing all project aspects. Using version control facilitated seamless collaboration among team members, enabling efficient tracking of changes and allowing us to revert to previous versions when necessary.

6.0 Narrative Discussion (5-10 pgs)

This section will present a comprehensive overview of the entire development process that led to the creation of our book dating application. It will outline the key phases, including planning, implementation, and testing, while highlighting the methodologies and technologies utilized at each stage. By examining the progression of our project from initial concept to final product, this section aims to provide insight into the collaborative efforts and critical decisions that shaped the development of our application.

6.1 Overview

In today's digital landscape, readers are confronted with an overwhelming number of choices when it comes to selecting books. Online retailers often utilize algorithms and trends to recommend titles, but these methods can be time-consuming and may not always align with a reader's personal preferences.

Our book-dating application addresses this challenge by offering a personalized book-finding experience. By employing a user-friendly format that resonates with readers, our app delivers tailored recommendations that enhance the discovery process. This ensures that users can efficiently find books that match their tastes, making their reading journey more enjoyable and less daunting.

6.2 Planning Phase

The planning phase of our book dating application was crucial in establishing a solid foundation for the project's development. We began by defining our primary goals and objectives, focusing on the need for a personalized book-finding experience that caters to the diverse preferences of readers. Some of the features that we decided to implement include, but are not limited to a main recommendation queue, an algorithm for suggesting books, filters for user preferences, a means of viewing one’s profile and account information, and a dedicated area where users can view their saved books.

Another significant step in our planning process was creating UI mockups to use as a template for creating our web pages. This was done using Marvel, a free design platform allowing teams of developers to create prototypes of web pages as well as demonstrations of user interface flow. This software was chosen because of its accessibility and ease of use, and due to the fact that some team members were already familiar with it. This initial visual representation of our application not only facilitated clearer communication among team members but also served as a valuable reference point during the subsequent development stages.

Our technology stack was chosen after careful consideration of the specific needs of our application, the skill sets of our team, and the long-term scalability of the application. Like all web pages, HTML and CSS were used to create the structure and style of our pages. While we had originally decided on React JS for the interactivity of our website, we ultimately opted to use plain JavaScript due to the app's simplicity. Thymeleaf and Spring Boot were chosen to provide a robust framework for building modern web applications, enabling easy templating and rapid development.
Regarding the back end, we opted for a non-relational database for book and profile information storage to prioritize performance and horizontal scalability. Java was chosen for the main back-end language, enabling modern development practices and features. Finally, Spring Security was used for authentication.

6.3 Implementation

This section outlines the key aspects of our implementation process, detailing the tools we used and the features we developed.

6.3.1 Front End

The front end of our book dating application serves as the primary interface through which users interact with the platform. Designed with user experience in mind, this component focuses on delivering an intuitive and engaging layout that allows users to effortlessly navigate through features such as book recommendations, user profiles, and search functionalities.
The user interface was developed based on the UI mockups created in Marvel. HTML documents were structured according to these designs, while style sheets were created to achieve the desired visual appearance. To enhance the layout, Flexbox was utilized in the style sheets, allowing for responsive and flexible designs that adapt to various screen sizes. JavaScript was employed to add interactivity to the front end, enabling dynamic content updates and enhancing user engagement.
The key features implemented in the front end include user registration and log-in forms, book search functionality, presentation of personalized recommendations, the user interface for viewing and editing profiles, and more. These features were designed to enhance user engagement and streamline the process of discovering new books. For instance, the user registration and login forms provide a secure way for users to access personalized content, while the book search functionality allows for quick and efficient navigation through the vast catalog. By implementing these features using technologies like JavaScript for dynamic interactions and CSS for responsive design, we aimed to create an intuitive interface that caters to user needs.

6.3.2 Back End

The backend of the Book Dating app serves as the foundation for handling application logic, data management, and communication with the database. It was designed to provide efficient, secure, and scalable support for front-end operations. The development process relied on several tools and frameworks. Spring Boot was chosen for its ability to

simplify application development, enabling us to create RESTful APIs for communication between the front end and back end. Java served as the primary programming language, while MongoDB, a NoSQL database, was utilized to store user information, book details, and library data due to its flexibility in handling unstructured data. Additionally, Gradle was employed for dependency management and project build automation, and Thymeleaf was used as a template engine to dynamically render HTML pages based on user-specific data. Several key features were implemented in the backend to enhance the application's functionality. A custom authentication and authorization system using Spring Security allowed users to log in with either a username or email, ensuring secure password storage and session management. The recommendation logic displayed one book at a time on the homepage, offering 'Accept' or 'Reject' options. Accepted books were saved to the user's library, while rejected books were skipped. Users could also manage their libraries, with accepted books dynamically updated and displayed using Thymeleaf templates.

The backend included robust RESTful APIs to handle core functionalities such as user registration, book retrieval, and updates to the user's library. Data was stored efficiently in MongoDB, with each user's profile including a books field in the User model to maintain a list of book IDs corresponding to the user's library. Furthermore, session management was implemented to track the logged-in user's activity across the platform, ensuring personalized content delivery.
One significant challenge in the backend was implementing the dynamic recommendation functionality. This was resolved by designing an efficient state-tracking system to manage the current book displayed and update it upon user interaction. Additionally, handling session data dynamically across different user interactions posed initial difficulties but was addressed with robust Spring Security configurations.

6.3.3 Integration

The integration phase focused on seamlessly combining the front end and back end to deliver a cohesive user experience. This involved connecting the user interface with backend APIs and ensuring consistent data flow and interaction.
The integration began with REST API integration, where endpoints developed in Spring Boot were connected to the front-end using JavaScript to fetch data dynamically. For example, the book recommendation logic retrieved data from the server via API calls and updated the user interface based on user preferences. Thymeleaf templates were utilized to bridge the gap between static front-end HTML and dynamic backend data, allowing real-time updates such as displaying a user's library or generating personalized book recommendations.
Session-based personalization was another key aspect of integration. The userId stored in sessions was used to customize content, such as book suggestions and user-specific pages. Sessions were managed efficiently to ensure a smooth and personalized user experience. Comprehensive error handling was also implemented to address potential issues during data transmission, including invalid session data, database errors, and API response delays.

As a result, the integration phase successfully ensured the seamless functionality of key features such as real-time book recommendations, secure user authentication, and dynamic updates to user libraries. The coordination between the responsive front end and robust back end created a unified platform that met user expectations.

6.4 Challenges and Solutions

This project offered a new level of complexity compared to most other projects in our academic career. The project was a unique idea that offered our team something that would explore new solutions using familiar skill sets. Additionally, the group struggled to coordinate tasks to best suit everyone. Originally, everyone was willing to learn something new, but our real issue was establishing roles. Our consistent schedule of meetings as a team every week allowed us to iron out some miscommunications and allowed for a smoother workflow. In these meetings, we were able to outline everyone's strengths and define a clear role for everyone to carry their weight on the team.

From a more technological standpoint, our group was unexperienced with web development. We decided to take on the challenge because of the new skills we could develop along the way. A lot of the struggles and obstacles for our development was this inexperience. However, the developers for this team worked to learn the HTML and CSS skills necessary to implement a user-friendly front-end. Meanwhile, the back end was made using some prior experience and bits of trial and error with the new database implementation of MongoDB.

6.5 Reflection

Through this project, we were able to experience the full Software Development Life Cycle. We were able to gain many new skills and have the opportunity to learn many job-related tasks like progress reports as well as documents like the SRS or SDD. Some of us on the team got insight into HTML development coupled with experience in CSS to better prepare for web development opportunities. Some of us got experience with database management to prepare for back-end development work. Learning the development cycle with a fundamental project like a website was beneficial for everyone on the team.

The reports and documents that were necessary for the project are a good foundation for what we might see in the workplace. Progress reports and Gantt charts are some of the documents that we are likely to see in a future workplace. In addition, the early cycle documents like the RAD and the Project Plan are commonplace in the design phases of a project, so it is great to become acquainted early. Furthermore, the peer reviews gave us a few interesting questions and insights as to how the project could have been better executed.

Our website's idea is unique but also has a narrow scope. A handful of the peer reviews concurred that the idea could be better accomplished as an ancillary feature to a larger system like a book marketplace or forum. Upon reflection, it's plain to see that the website could implement more features to give a more complete user experience. However, the project was sufficiently complex given the time allotted and our level of experience, so we will refine what we have and see what else the remaining time may allow.

Overall, this project was an adventurous idea that taught or at least reinforced a lot of skill sets that will continue to be useful in our future careers of development.

7.0 Software Test Plan

7.1 Introduction

7.1.1 Purpose

The purpose of this Software Test Plan (STP) is to outline the testing approach for the Book Dating App, a web application that enables users to discover, save, and explore books in a queue-based format. This document will guide all testing activities to ensure that the application meets functional requirements, is user-friendly, performs efficiently, and remains secure across various scenarios and usage levels.

7.1.2 Definitions and Acronyms

CRUD: “Create, Read, Update, Delete” database operations
UI: User Interface
UX: User Experience
STP: Software Test Plan

7.2 Test Objectives

This section defines the objectives that testing aims to accomplish, focusing on ensuring the app’s functionality, usability, performance, and security. Meeting these objectives will help confirm that the Book Dating App delivers a reliable and user-friendly experience.

7.2.1 Primary Objectives

The primary objective of the testing process is to ensure that the Book Dating App operates as expected in terms of functionality, usability, performance, and security. Specific objectives include, but are not limited to, intuitive user experience, performance efficiency, and data security.

A key objective of testing is to ensure that the Book Dating App provides a seamless and enjoyable user experience. This includes evaluating the app’s navigation, layout, and ease of use, aiming to make the interface intuitive for new and returning users. Testing will verify that users can effortlessly browse books, manage their queues and libraries, and access all primary features without confusion. A positive user experience will contribute to increased user engagement and satisfaction with the website.

Another important objective of testing is to analyze the performance of the web application in a variety of conditions, ensuring it remains responsive and efficient under different user loads. Multiple components of the web application will be tested including database interactions and server communications. Performance testing will determine whether the website can handle high traffic volumes without encountering significant issues or slowdowns.

One of the most important aspects of any software application is data security. We plan to conduct testing to ensure that user data is properly protected against unauthorized

access, breaches, and other security vulnerabilities. We will test various aspects of the application, including user authentication, data encryption, and secure storage practices, to ensure that sensitive information remains safe. The goal is to verify that all security mechanisms are functioning correctly and provide the necessary protection for user data.

7.2.2 Quality Standards

To ensure that the web application meets quality expectations, the following standards and criteria will be put into place:
Compatibility – The application will be able to run on multiple common browsers, such as Google Chrome, Firefox, Microsoft Edge, Safari, and more. The website's UI will be analyzed on a variety of mobile devices.
Functional Correctness – All core features, including user registration, book search, queue management, and library functions, must operate according to their defined requirements.
Performance – Page load times should not exceed 2 seconds under normal operating conditions.
Error Handling – The application must effectively handle errors and exceptions, displaying informative error messages when something goes wrong.
Maintainability – The architecture must support future growth, allowing for easy addition of features and handling increased user traffic without major changes to the core functionality.

7.3 Test Scope

This section defines which features and functionalities will be tested, and which ones will not be included in this testing phase.

7.3.1 In-Scope

The following features and components of the Book Dating App will be covered in the testing process:
1. User Registration and Authentication: Testing of account creation, login/logout functionality, password recovery, and session management.
2. Book Discovery and Search: Verifying that the book search function works as expected, including searching by title, author, or genre.
3. Queue and Library Management: Testing the functionality of adding, removing, and viewing books in the user's queue and library.
4. Book Recommendations: Testing the functionality of the recommendation algorithm for accurate and relevant suggestions.
5. User Interface and Usability Testing: Ensuring that the user interface is intuitive, easy to use, and responsive.
6. Performance Testing: Analyzing the app's ability to perform under various traffic conditions.
7. Security Testing: Ensuring that data is handled securely, including user authentication and data encryption.
8.

7.3.2 Out-of-Scope

The following features and components of the Book Dating App will not be covered in this testing cycle:

1. Advanced Features: Any additional features that are planned for future releases but are not part of the current version.
2. Third-Party integration: Testing of external integrations, such as payment systems or external APIs.
3.

7.4 Test Strategy

The testing strategy for the Book Dating App will focus on ensuring the application's functionality, performance, and security. Our approach includes a combination of different types of testing, such as functional, performance, and security testing, to cover all aspects of the application.

7.4.1 Types of Testing

The testing strategy for the Book Dating App will involve several types of testing to ensure the application meets all required standards. For instance, functional testing will be employed to ensure that each of the core features of our application, such as user registration, book discovery, and queue management, are all functioning properly. Functional testing will then be followed by usability testing, which will assess the app's user-friendliness and ease of navigation. It is critical that users can interact with the app intuitively, and this testing will ensure that the interface is simple to use.
Performance testing will then evaluate how the app behaves under different levels of user load, ensuring the system remains responsive even under heavy traffic. Additionally, security testing will be conducted to verify that sensitive user data is properly protected, ensuring there are no vulnerabilities in the app's security mechanisms. Compatibility testing will also be conducted to ensure that the web application functions effectively and smoothly on multiple common web browsers. Finally, regression testing will ensure that any new code changes or updates do not interfere with existing functionality.

7.4.2 Test Levels

Testing will occur at several levels to ensure the system functions as expected at both individual and integrated stages. Unit testing will be conducted on individual components to confirm that each part of the application performs as expected in isolation. Furthermore, integration testing will be employed to ensure that each component, such as the front end and back end, interacts with one another in an effective manner. This will be followed by system testing, which will be conducted to analyze the system as a whole and ensure that every aspect of the system works seamlessly together. Finally, Finally, acceptance testing will verify that the app meets the user requirements and is ready for production release.

7.4.3 Test Focus

We will prioritize our testing efforts based on the areas of the Book Dating App that carry the highest risk, complexity, or user impact. The goal is to ensure that critical features are thoroughly tested and that any vulnerabilities or issues are identified early in the development process. Areas that will receive focused attention during testing include data security, core functionality, and usability.

Given the sensitive nature of personal data such as user profiles and login credentials, security must be a top priority in the design and testing process. Special attention will be directed to authentication services and password security. Furthermore, the core functionality of the application will also be a top priority, as we need to ensure that key features such as book recommendations and library management are functioning properly. Finally, since the app relies heavily on user interaction, ensuring a smooth and intuitive user experience is essential. Usability testing will focus on the ease of navigation, accessibility, and visual clarity of the interface.

7.5 Test Environment

This section outlines the hardware, software, network setup, and other resources that will be used to conduct testing for the Book Dating App. The test environment is designed to replicate the production environment as closely as possible to ensure the app functions as intended across all stages of testing.

7.5.1 Hardware Requirements

Testing will be performed on a variety of devices to ensure that the User Interface behaves as expected on multiple screen sizes and orientations. Such devices include, but are not limited to desktop computers, laptops, tablets, and mobile phones. Testing will also be conducted to ensure that the website performs well on various levels of hardware, including low-end or older devices. Mobile devices will be tested in both portrait and landscape orientations.

7.5.2 Software Requirements

Testing will be conducted to verify that the Book Dating App is compatible with a wide range of software configurations. The app will be supported on major operating systems, including Windows 11, iOS 18, Android 15, and other UNIX-based systems. Testing will be conducted to ensure that the Book Dating App functions properly across the most used web browsers, including the latest versions of Google Chrome, Firefox, Safari, Microsoft Edge, Opera, and others.

The Book Dating App will use MongoDB as the primary database for storing user and book data. Testing will ensure that the database performs as expected under different load conditions, supports seamless data retrieval and storage, and is compatible with the chosen backend architecture. Additionally, database compatibility with operating systems and web servers will be validated to ensure smooth interaction.

7.5.3 Network Configuration

Network conditions will be simulated using Chrome DevTools' Network Throttling feature to mimic various real-world scenarios, such as 3G, 4G, or custom network speeds with high latency and low bandwidth. This will allow testing of how the app performs under slow or fluctuating network conditions without requiring a complex infrastructure setup. Testing will be performed under both high-latency scenarios and low-bandwidth scenarios to assess the app's performance.

7.6 Test Deliverables

The following deliverables will be produced throughout the testing process to track progress, results, and any identified issues.

6.1 Test Cases

We intend to develop test cases that will verify the functionality of the core features of the application. Such cases include, but are not limited to:

1. User Registration and Authentication: Verifying that users can successfully sign up for an account and log in to their accounts.
2. Book Search: Verifying that the search functionality performs as expected, allowing users to search by title, author, and genre.
3. Library Management: Verifying that users can add books to their library or remove them.
4. Profile and Account Management: Verifying that users are able to update their profiles and account information such as preferences.

6.2 Test Report

The test report will be a comprehensive summary of all testing activities, results, defects, and recommendations based on severity. This report will provide stakeholders with insights into the quality of the Book Dating App and whether it is ready for release. The Test Report will include details such as test descriptions, test coverage, and severity of unresolved defects.

7.7 Conclusion

Overall, this Software Test Plan outlines the comprehensive strategy and approach for testing the Book Dating App. Our test process aims to assess the functionality, usability, performance, and security of our application. The testing process will involve various levels and types of testing, including functional, performance, and security testing, executed across a range of hardware and software environments to ensure compatibility and reliability.

8.0 Software Test Report

Requirement	**Pass**	**Fail**	**Severity**

Create Account	Pass		
Login	Pass		
Forgot Password		Fail	High
Logout	Pass		
Session Creation	Pass		
Book Search - Title	Pass		
Book Search - Author	Pass		
Book Search - Genre	Pass		
Book Queue		Fail	Low
View Library	Pass		
Add To Library	Pass		
Remove From Library		Fail	Mid
Book Recommendations		Fail	High

9.0 Conclusion

The Book Dating website is a solution to a problem that avid readers face while packaging it in a format that is familiar. To streamline the design process our group used Marvel to create interactive mockups to show user-flow and to quickly illustrate our proposed UI. Using HTML, CSS, and MongoDB our team has been working to create a modern system to recommend books with a user-catered experience. As development continues, we have been implementing our Test Plan to fine-tune the issues in the design or to find where users may be confused. In summary, our group has worked to create an inventive solution for book readers while offering a user-centric design to make the website flow as cohesive as possible.