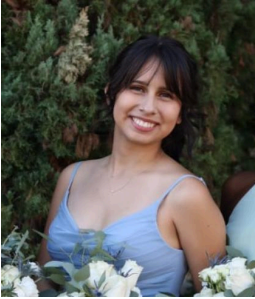




INDY 1 RED — Book Selector App

CS 4850 - Section 03 – Fall 2024

November 8, 2024

 <p>Francesca Del Aguila (Team Leader) Development, Website</p>	 <p>Thomas Roberts Documentation, Test</p>	 <p>Tristan Sanford Development, Website</p>
--	---	---

Team Members:

Name	Role	Cell Phone / Alt Email
Francesca Del Aguila (Team Leader)	Development, Website	678.920.0165 francescydel23@gmail.com
Thomas Roberts	Documentation, Test	404.834.1861 thomas@bwroberts.com
Tristan Sanford	Development, Website	706.461.3929 tristanrsanford@gmail.com
Sharon Perry	Project Owner, Advisor	770.329.3895 Sperry46@kennesaw.edu

Links	
Website	https://bookselectorapp.github.io/
Github	Book Selector App github
Statistics	
Lines of Code	2,400 lines
Number of Components/Tools	16 components
Total Hours	370 hours

1 Introduction.....	4
1.1 Overview.....	4
1.2 Goals.....	4
1.3 Objectives.....	4
2 Requirements.....	4
2.1 Functional Requirements.....	4
2.1.1 Login Screen.....	4
2.1.2 Register Screen.....	5
2.1.3 Home Screen.....	5
2.1.4 TBR Screen.....	5
2.1.5 Profile Screen.....	5
2.1.6 App Navigation.....	6
2.1.7 Backend.....	6
2.2 Nonfunctional Requirements.....	6
2.2.1 Security.....	6
2.2.2 Capacity.....	6
2.2.3 Usability.....	6
2.2.4 Other.....	6
3 Analysis.....	6
3.1 Liabilities.....	6
3.1.1 Data breach.....	6
3.1.2 Copyright infringement.....	6
3.1.3 Misrepresentation.....	7
3.1.4 Mature Content Policies.....	7
3.1.5 Licensing.....	7
4 Design.....	7
4.1 Assumptions.....	7
4.1.1 Operating System.....	7
4.1.2 End-User Characteristics.....	7
4.2 Constraints.....	7
4.2.1 Network Constraints.....	7
4.2.2 Security Constraints.....	7
4.2.3 Server Constraints.....	7
4.2.4 UX Constraints.....	7
4.3 System Architecture.....	8
4.4 Detailed Design.....	9
4.4.1 Screens.....	9
4.4.2 Navigation.....	9
4.4.3 Design Elements.....	10
4.4.4 API.....	10

4.4.5 Render.....	11
4.4.6 Supabase.....	11
5 Development.....	11
5.1 Initial App Creation.....	11
5.2 App Styling.....	12
5.3 App Navigation.....	12
5.4 Screen Components.....	13
5.5 Swipe Functionality.....	13
5.6 API and Supabase.....	14
5.7 Recommendation Algorithm.....	14
5.8 Modifying Backend for Users.....	15
6 Testing.....	15
6.1 Unit Testing.....	15
6.2 Integration Testing.....	15
6.3 Test Report.....	15
7 Version control.....	17
8 Conclusion.....	18
9 Appendix.....	18

1 Introduction

1.1 Overview

The Book Selector app is a mobile application designed to enhance book discovery through a swipe-based interface similar to dating apps like Tinder and Hinge. The app helps users efficiently find new books based on their interests while also providing a platform for authors and publishers to reach readers.

Users can swipe left to skip a book or swipe right to add it to their To-Be-Read (TBR) list. If they are unsure about a book, they can tap for more details such as descriptions and author information.

1.2 Goals

- Enhance Book Discovery.
 - The swipe-based, dating app-like interface provides an engaging and efficient platform for users to discover new books based on their tastes and preferences, reducing the information overload usually associated with finding new books.
- Connect New Authors with New Readers.
 - Rather than only being served top-shelf books as one would be at a bookstore, the app recommends books based on individual user preferences, allowing each reader to explore their own personal bookstore and discover books and authors they would not have previously.

1.3 Objectives

- Develop a mobile application that is responsive, user-friendly, and accessible across a range of devices, ensuring smooth and seamless user interaction.
- Integrate algorithms that tailor book suggestions to individual user preferences.
- Build a strong system to manage book data, user interactions, and user-generated content, ensuring the platform remains scalable and efficient.
- Implement features that encourage ongoing user engagement, such as personalized book recommendations and notifications that keep users returning to the app.

2 Requirements

2.1 Functional Requirements

2.1.1 Login Screen

- Display login screen title.
- Display text fields for username and password.
 - Fields update on text change.
- Display login button.
 - Calls API login function when clicked.
 - Navigates to the home screen if successful.
 - Displays alert if unsuccessful.
- Display register button.
 - Navigates to the register screen when pressed.

2.1.2 Register Screen

- Display register screen title.
- Display text fields for username and password.
 - Fields update on text change.
- Display register button.
 - Calls API register function when clicked.
 - Navigates to the login screen if successful.
 - Displays alert if unsuccessful.

2.1.3 Home Screen

- Load recommendations array upon screen creation.
 - Call API getRecommendations function.
 - Display loading icon while waiting on recommendations.
- Display home screen title.
- Display book cover of first book in recommendations array.
 - If there are no recommendations, display error text.
- Incorporate swipe function on book.
 - Swipe left dismisses book.
 - Swipe right adds the book to TBR.
 - Call API addToTBR function.
 - Tapping on the book displays book description.

2.1.4 TBR Screen

- Load TBR list upon screen creation.
 - Call API viewTBR function.
 - Display loading icon while loading TBR list.
- Display TBR screen title.
- Display TBR books.
 - Make the list scrollable.
 - Display option to remove book from TBR.

2.1.5 Profile Screen

- Load preferences on screen creation.
- Display text field for adding a book title to preferences.
- Display button to add title to preferences.
 - Button calls API addTitle function.
- Display text field for adding a book genre to preferences.
- Display button to add genre to preferences.
 - Button calls API addGenre function.
- Display titles in user preferences.
 - List is scrollable.
 - Display button to remove preference.
- Display genres in user preferences.
 - List is scrollable.
 - Display button to remove preference.

2.1.6 App Navigation

- Implement bottom tab-bar navigation.
 - Add Home, Profile, and TBR screen.
 - Display Icons for each screen.
 - When clicked, the icons navigate to the appropriate screen.
- Implement native stack navigation.
 - Add Login screen, Register screen, and tab-bar component.
 - Components can navigate via function call.

2.1.7 Backend

- Stores user information.
 - Login information, preferences, and TBR list.
- Stores book information.
 - Author, book description, cover.
- Generates recommendations based on user preferences.
- Information is accessible through API.
 - Functions are defined and accessible to all app components.

2.2 Nonfunctional Requirements

2.2.1 Security

- Password text fields obscure password information.
- Username and password information is encrypted.

2.2.2 Capacity

- App and Database can expand to keep up with a growing user base.

2.2.3 Usability

- Intuitive, user-friendly mobile interface designed to minimize friction and maximize user engagement and satisfaction.
- Fast response times when swiping and making modifications to settings and libraries.

2.2.4 Other

- App component's design should be cozy and aesthetic.
 - Use defined fonts and color palettes.

3 Analysis

3.1 Liabilities

3.1.1 Data breach

Since the application requires username and password information, there are potential liabilities involving data breaches. To mitigate the damage of data breaches, sensitive information should be encrypted. To also protect user information, there should be policies to prevent weak passwords.

3.1.2 Copyright infringement

There could be liabilities when displaying content without proper authorization. To prevent copyright infringement, the appropriate licenses would need to be purchased.

3.1.3 Misrepresentation

Misrepresenting a book could lead to damaged reputations and legal repercussions. So, the app will need policies to make sure all descriptions and reviews are accurate and up to date.

3.1.4 Mature Content Policies

The book dating app displays book covers on the home page and it is possible some covers have mature content. The app should have preventative measures such as review of the content before adding it to the production database.

3.1.5 Licensing

As licenses are purchased, proper documentation and records are necessary in case of disputes. Additionally, any purchases need to be paid in a timely manner.

4 Design

4.1 Assumptions

4.1.1 Operating System

It is assumed that the users will access the app through mobile devices running Apple iOS and Android platforms. Additionally, it is assumed the user OS systems will be up to date. The app will be optimized for these operating systems.

4.1.2 End-User Characteristics

It is assumed that users will have the technical literacy to use the app. The app uses a swipe based function that is similar to dating apps like Hinge, Bumble, and Tinder.

4.2 Constraints

4.2.1 Network Constraints

The app is dependent on the network to access external book information and process user requests. Some user requests that may be affected are: login requests, register requests, TBR requests, and recommendation requests.

4.2.2 Security Constraints

The app processes user data which will require the app to be compliant with data protection regulations. Additionally, secure authentication methods will be established to secure sensitive data as necessary.

4.2.3 Server Constraints

The app may be limited by server load and scalability. The server may be overloaded, especially during peak hours which could result in slow response times or crashing. Additionally, the database may have a storage limit which would affect scalability. As the user base grows and as more books are added to the database, it is very likely to be limited by storage.

4.2.4 UX Constraints

The app should provide an interactive user experience similar to common dating apps to provide a positive experience. The app should also accommodate different IOS and Android models, meaning the app should provide a consistent experience for every screen size.

4.3 System Architecture

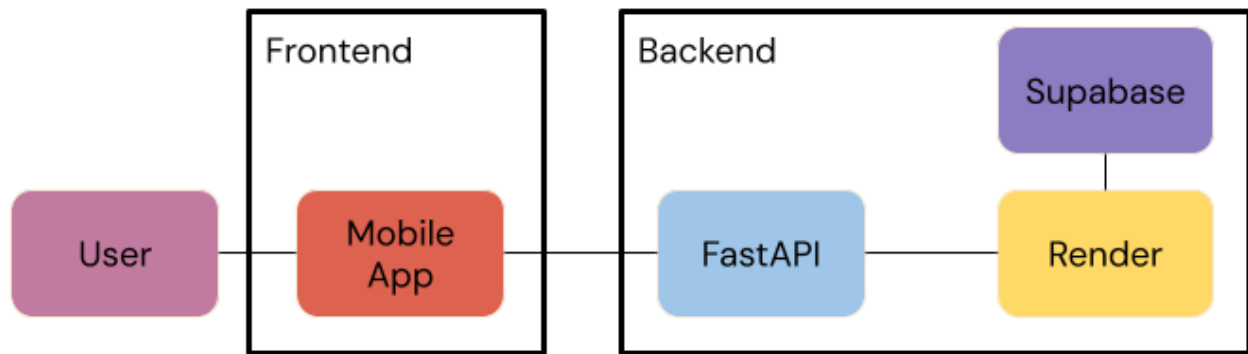


Diagram 1. System Architecture showing the separation of frontend and backend.

The architecture is divided into two categories: frontend and backend. The frontend architecture will focus on the app components: screens, screen navigation, and design elements. The backend architecture will consist of a database (Supabase), cloud application platform (Render), and an API (FastAPI).

The mobile app mainly consists of 6 screen components: Welcome, Login, Register, TBR, Home, and Profile screens. Their specific functions are outlined above in the functional requirements sections. Below, is a flow diagram that illustrates how each screen is connected and the conditions needed to navigate.

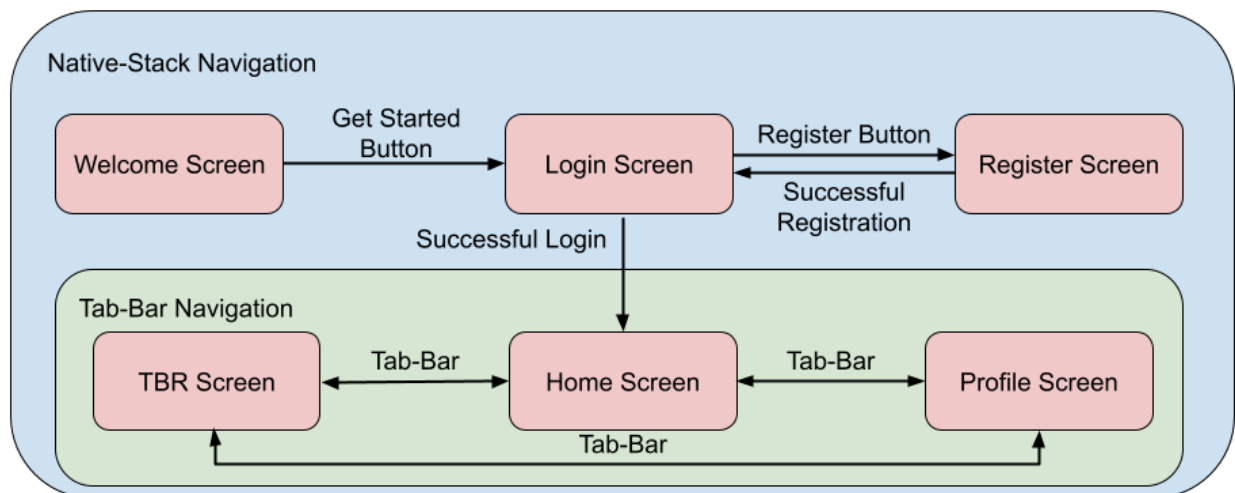


Diagram 2. Displays screen transition flow and key events for screen transition.

The app incorporates two navigation components: Native Stack and Tab-Bar. The Native Stack manages the navigation of the Welcome, Login, Register screens, and the Tab-Bar Navigation. As illustrated in Diagram 2, the Tab-Bar navigation component is nested within the Native Stack component. This nesting is necessary because the screens under the Tab-Bar component (TBR, Home, and Profile) are specific to individual users. Consequently, the Tab-Bar component, which provides easy access to these user-dependent screens, becomes accessible only after a successful login, as depicted in the diagram.

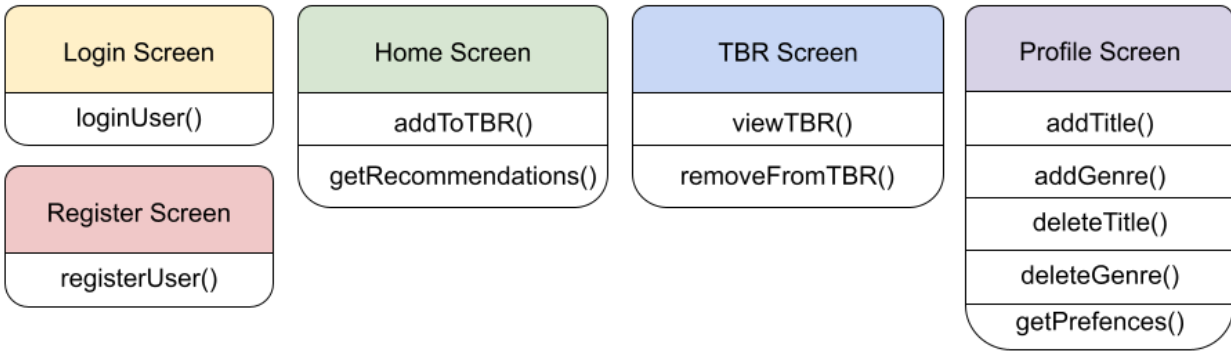


Diagram 3. UML diagram correlating API endpoints to screen components.

An objective of this project is to limit the processing power and memory the screens need to function. Consequently, the screens were designed so that the backend service does all the processing, and the API is responsible for connecting the screens to the backend service . Above, Diagram 3 illustrates the correlation between the API JavaScript functions and the screen components. Each JavaScript function in the diagram is an async await fetch function for an API endpoint. This design allows any component to import the function and use the API resource.

4.4 Detailed Design

4.4.1 Screens

Classification and Definition

A screen refers to a React Native component within its own JavaScript file that acts as a page of the mobile app. Each screen has its own set of tasks related by topic. For example, the home screen focuses on book discovery and the profile screen handles the preferences for the user profile.

Constraints

The home, TBR, and profile screens need to download information before displaying content, so these screens are limited by the network. Additionally, the login and register screen have functions (login and register) that are limited by the network.

Resources

The screen components were designed to utilize very little memory and processing power. Instead, they focus on cloud-based computation and memory storage, so these components heavily rely on communication with network components.

Interface/Exports

Screens compose the app UI and provide a means for the user to interact with the app. This includes displaying information, creating text fields for users to type input, and generating buttons to trigger events. These services are essential for user interaction and facilitate the basic features of the app.

4.4.2 Navigation

Classification and Definition

Navigation refers to a JavaScript file that contains two navigation constants (tab-bar and native stack) from the react library. These constants are defined as react Navigator components which manage the child screens.

Constraints

These navigation components contain screen components so they are limited by memory and performance. As more screens are added, the more memory and processing power are needed to maintain them. Additionally, these components are limited by the complexity of navigation. Complex navigation instructions can result in faulty operation or bugs.

Resources

The navigation components need memory and processing power. It also needs access to the react navigation library.

Interface/Exports

Navigation is essential for app functionality. It allows the user to control which screen is currently displayed. The tab-bar component also displays a set of buttons which navigate to specific screens.

4.4.3 Design Elements

Classification and Definition

The design elements are a collection of fonts, images, and screen styles.

Constraints

One of the main concerns is memory. Additionally, the assets need to be compatible which means there is a limit to the complexity of the designs.

Resources

The design elements require access to Google Fonts, Tailwind, and NativeWind libraries. The image and font files will require memory for storage.

Interface/Exports

The design elements are necessary to create a cozy and aesthetic appearance essential for the user experience. By customizing font weights and sizes, the interface's readability was improved while complimenting the app's cozy feel. Without images, the app would have felt less visually engaging, and by combining all these styling components together, it created a cohesive and attractive interface to attract readers to use the app.

4.4.4 API

Classification and Definition

the API is a set of functions that enable communication between clients and the server, built using FastAPI.

Constraints

The API requests are limited by the number of query parameters as there is a character length of 50. Meaning, the API is limited by query complexity. The API is also limited in the number of API calls allowed as the server has a call rate limit.

Resources

API calls are made to Render which has limits in terms of allocated memory and number of calls.

Interface/Exports

For every API endpoint there is an export-JavaScript-async-await-fetch function. These functions are stored in a JavaScript file, so whenever an App component needs to make an API call, it can import the function and simply pass the necessary information.

4.4.5 Render

Classification and Definition

Render is a cloud application platform to host the backend services.

Constraints

The app uses Render's free subscription which provides limited memory and processing power. Because of this limitation, the mobile app will not be able to perform complex backend services or services that use a lot of memory. Render provides a means of scaling the project in terms of memory and processing power through a paid tier.

Resources

The services will use cloud memory and CPU. For the mobile application, the only resource would be the network.

Interface/Exports

Render hosts the API and Python scripts essential for generating recommendations. Without this service, the mobile app will be unable to generate recommendations and host the API which are critical functions.

4.4.6 Supabase

Classification and Definition

Supabase is an open-source Firebase alternative. For the project, Supabase is a database for storing and retrieving app data.

Constraints

The database is limited by a maximum memory of 50mb. This constraint limits the number of books and number of users the backend service can manage. The subscription can be scaled up but requires payment.

Resources

The services will use cloud memory and CPU. For the mobile application, the only resource would be the network.

Interface/Exports

Supabase is the database that holds book information, user information, and the similarity matrix. This information is needed to generate user-specific recommendations and perform user verification. The information is accessed via API call.

5 Development

5.1 Initial App Creation

Project development began by installing the React Native framework with Expo for mobile development. Once installed, the Expo template and the dependencies were instantiated. This template served as the foundation for the application. The function "npx create-expo-app@latest ./ --template blank" created a blank starting point for the app, allowing the configuration of the navigation, styling, and core components to fit the book dating app's requirements.

During the initial setup, there were issues with starting the app, which resulted in the modification of the package.json file to correctly define the entry point. Next, the mobile app's source files were structured into dedicated folders, including screens for main screen components, navigation for managing app navigation logic, and assets for storing images and other media. The structure by sectioning the files into folders helped maintain the codebase and made it manageable.

5.2 App Styling

An objective of the project was to create a cozy and aesthetic look for the app.

To accomplish this, the Tailwind and NativeWind libraries were downloaded and set up to create Tailwind classes with React Native components. After configuring tailwind.config.js and setting up a custom directory for styling files, elements like color, spacing, and border radius were readily accessible to keep a consistent theme throughout the app.

In addition to Tailwind, the app also incorporates Google Fonts to enhance the typography, giving the text a warm, inviting look that aligns with the overall aesthetic. By customizing font weights and sizes, the interface was optimized to create a clean readable output that compliments the app's cozy feel.

To further enhance the UI, an image folder was created to store assets used across different screens. This folder was essential for the app's visual design as it provided icons and background images that improved the user experience. Without these images, the app would have felt less visually engaging. By combining all these styling components together, it allowed a cohesive and attractive interface to attract readers to use the app.

5.3 App Navigation

The next step focused on configuring the navigation and page structure, calling for managing the different screens in the app. React Native provides several options for navigation, and this app uses a combination of native-stack and bottom-tab navigation to create a user-friendly experience. The bottom-tab navigation was implemented for the Home, TBR, and Profile screens, while the native-stack was used for the Welcome, Login, and Register screens.

To organize the navigation setup, a navigation folder was set up, called AppNavigation.js, to handle both stack and tab navigation. The tab-bar component is displayed on the bottom of the screen and contains @expo/vector-icons to represent/connect to each screen. This setup makes sure that when a user selects an icon on the tab-bar, it navigates to the corresponding screen.

Similarly, screen components were added to the native-stack with assigned names, enabling controlled navigation flows. For example, pressing the register button calls the navigate function to direct users to the Register screen. The tab-bar component was also incorporated into the stack navigation, so upon successful login, the app navigates to the Home screen and displays the tab bar for easy access to other sections.

With these navigation established, the next objective was to continue to develop features for each page and component, making sure to create a pleasant design for users to enjoy as they navigate through the app.

5.4 Screen Components

The login screen is composed of a simple form where the user can type a username and password and press a button to login. The form is designed such that any change in text updates the corresponding username or password value. Upon pressing the login button, the API login function is triggered (Note: API functions are discussed below). If the login is successful, the app navigates to the home page; otherwise, an alert is displayed to the user. Below the login button is a register button. When pressed, the app navigates to the register screen.

The register screen is very similar to the login screen. It consists of a form where the user can type a username and password. Below the form, there is a button to call the API register function. If successful, the app navigates back to the login screen; otherwise, an alert is displayed.

The home screen is responsible for displaying book recommendations and allowing the user to swipe right or left on the book to add to TBR or dismiss. When the app navigates to this screen, it makes an API call to load recommendations based on the user's preferences and stores the data in a local array. The app then displays the cover of the first book in the array. The user can swipe right, swipe left, or tap on the book cover. Swiping right will trigger an API call to add the book to the TBR list and display the next book. Swiping left dismisses the book and displays the next book. Also, `TouchableOpacity` is enabled, so the book cover can be tapped and interacted with. For flipping the app uses an `Animatable` view component for the actual flipping over animation.

The profile screen enables users to set preferences by adding titles or genres. It includes two forms: one for adding titles and another for adding genres, each comprising a text input and a button. To add a title, users type it into the text input and click the button, triggering an integrity check on the input (checking non-blanks or duplicate values). If it passes, the API's `addTitle` function is called. Below these forms, lists display the titles and genres in the user's preferences, which are loaded from the server via an API call. Each preference has a delete button that allows users to remove the item from their profile.

Lastly, the TBR screen displays all the books the user swiped right on, allowing them to remember interesting books and start reading. Additionally, beside each book title, there is an option to remove the book.

5.5 Swipe Functionality

The home screen displays book recommendations, allowing users to swipe right to add a book to their TBR list or swipe left to dismiss it. To implement this feature, the app integrates React Native's gesture-handling libraries, which enable smooth and responsive swipe actions. When a user swipes right, an API call is triggered to save the books to their TBR list, and the next book in the array is displayed. Swiping left dismisses the book without saving it and displays the next recommendation. This setup makes sure it's an intuitive and interactive experience for the users, where each swipe provides feedback and smooth transitions to the next book.

Additionally, tapping on a book cover activates a flip animation, revealing more details on a solid background, similar to turning a book cover. By having this functionality, it aims to create an enhanced user engagement which in turn provides a fun way to explore recommendations.

As the user continues to swipe, the recommendation algorithm updates to offer increasingly relevant suggestions.

5.6 API and Supabase

The backend service was built using FastAPI, Supabase, and Uvicorn. This process started by importing the necessary libraries, setting up Supabase, and obtaining a URL and API key from the project setting. Then, a Supabase client was initialized with the URL and API key. Once this was set up, the API endpoints were defined and connected to the Supabase functions. All these API calls are stored in a JavaScript file. The logic behind the functions is to make an async await fetch function, pass the necessary input, then return a JSON constant containing the information. Then, when a function needs to make an API call, it can import one of these functions from the JavaScript file.

The next step involved hosting the backend Python file for accessibility by any frontend client. Initially, the backend was hosted locally, which rendered the frontend client unusable for others as it wouldn't display any data. Render, a free hosting service, was chosen for deployment but there were issues due to limited system resources on the free server instance.

At this stage, only the CSV dataset was hosted on Supabase. The plan was to host both the backend Python files and the similarity scores dataset on Render. However, the similarity scores dataset was too large for the available server memory. To resolve this, the similarity scores dataset was moved to Supabase and modified the Python file to query the database for similarity information, keeping the previous logic intact but simplifying the backend file.

This adjustment significantly reduced the system resources required, enabling Render to accept and deploy the build successfully. Meaning, the data became accessible from any frontend client.

5.7 Recommendation Algorithm

The goal was to build a book recommendation algorithm leveraging the GoodReads API. However, that API was no longer issuing developer keys, rendering it unusable for the purposes. So, the data source changed to a list of books extracted from the 'Best Books Ever' library on GoodReads. The Pandas package was used to read the Best Books Ever CSV dataset and created a 'content_features' column combining the author, series, genres, and description features of each book. Titles were excluded to avoid false correlations. Series and genres were given the highest weight, followed by the description, with the author given the lowest weight. This weighting was crucial for the recommendation algorithm's accuracy.

Using TfidfVectorizer from scikit-learn, the program created a TF-IDF matrix from the content_features column to represent each book's traits. The cosine_similarity function was used to compare books, identifying similar ones. To obtain the top 50 similar books for each, the program iterated through the TF-IDF matrix and used multithreading to handle the large dataset efficiently.

For recommendations, a new Python file was created to import similarity scores. Functions were implemented to recommend books based on title or genre, returning relevant results. The main recommendation function pulled a mix of top-rated books from titles, genres, and overall ratings.

5.8 Modifying Backend for Users

Once the backend service was running, user-specific functionality such as login, TBR, and profile components were the next priority. This process started by adding a users table in Supabase and used a register function to populate it. When a user registers, the username is checked to make sure there are no duplicates. If it passes, the password is hashed with the bcrypt package and the username and password are stored in the table. For the login, the user is selected from the user table and the hashed password is checked against the hash of the input password.

For the TBR, user id and book ids are paired together and stored in a TBR_list table in Supabase. This means one user can have multiple books stored under their user id. This pairing makes it easy to reference and delete books from the TBR_list.

Next, a preferences section was implemented to influence the recommendation algorithm. Unlike the TBR_list, this list stores all the titles and genres in one JSON string instead of having multiple entries per user. This format facilitates information transfer to the recommendation algorithm. There were downsides to format, such as having to implement string manipulation to remove or add books to the JSON file.

For both the TBR and preferences, Pydantic was used for input validation and FastAPI to create the endpoints that the frontend could interact with.

6 Testing

6.1 Unit Testing

Unit testing consists of testing individual components to verify their integrity and performance. For most components, testing was done using Expo go which was like a sandbox that would generate the app during development. With this tool, the app's individual components were manually tested to make sure the desired action occurred consistently. For backend components such as Supabase, unit testing was done with built-in features such as a feature to query the database. The results from unit testing are shown in table 1 below.

6.2 Integration Testing

Integration testing verified that several components can successfully communicate and perform operations as described in the requirements. For instance, this would include testing if a screen component can successfully make an API call, receive the appropriate information from the backend service, and display the information. This testing was also done using Expo go by manually going through the apps functions as seen by the user. The results of integration testing are shown in table 1 below.

6.3 Test Report

Bottom Navigation Tab	Pass/Fail	Severity of Fail
Display Home, Chat, and Profile buttons	Pass	
Buttons go to appropriate screens	Pass	
Home Screen		

Display Header	Pass	
► Display profile icon, app name, and bell button	Pass	
Bell button goes to notifications page	Fail	Low. Not a priority feature.
Retrieves recommendations	Pass	
Displays book covers	Pass	
Tapping cover displays book information	Pass	
Swipe left on book	Pass	
► Dismiss book	Pass	
Swipe right on book	Pass	
► Add to TBR	Pass	
Login Screen		
Display login form	Pass	
Form retains text input	Pass	
Display login button	Pass	
Button calls login function	Pass	
Password hash function works	Pass	
Backend verifies username/password	Pass	
login function returns correct response	Pass	
Incorrect login alert works	Pass	
Successful login navigates to homepage	Pass	
Display register button	Pass	
Register button navigates to register screen	Pass	
Register Screen		
Display register form	Pass	
Form retains text input	Pass	
Display register button	Pass	
Button calls API register function	Pass	
Password hash function works	Pass	
Backend verifies username/password	Pass	
API register function returns correct response	Pass	
Username alert works	Pass	
Successful register navigates to login screen	Pass	
Profile Screen		

Display form to add titles	Pass	
Display add title button	Pass	
Button adds title to profile	Pass	
Display form to add genre	Pass	
Display add genre button	Pass	
Button adds genre to profile	Pass	
List titles	Pass	
Display delete title button	Pass	
Delete button removes title from profile	Pass	
List genres	Pass	
Display delete genre button	Pass	
Delete button removes genre from profile	Pass	
Lists are scrollable	Pass	
TBR Screen		
Display TBR book titles	Pass	
Display remove button	Fail	Low. Displays button for some entries only.
Remove button removes title from TBR	Pass	
Tap on title to rate it	Fail	Low. Not a priority feature.
Backend Services		
Clients connect to server	Pass	
Data changes are saved	Pass	
API functions work as intended	Pass	

Table 1. Results of unit and integration testing.

Based on the testing criteria, the app performs as intended aside from two features which are low priority. The backend manages data reliably and efficiently. The login and register functions successfully authenticate users and navigate to the appropriate screens. The Profile, TBR, and Home screens accurately display information and call the correct API functions. However, there are some minor issues with the remove button on the TBR screen and the rating function. Despite these low-priority issues, the app demonstrates robust functionality and user experience.

7 Version control

This project uses GitHub to protect the source code and enable safe collaboration. GitHub has several built-in features to protect the source code such as the ability to track

changes and revert changes if needed. It also provides a means for the group to peer review a commit before it is merged into the production branches.

The project source code is separated into two production branches (backend and master) with an additional branch for documentation. Features were developed on local branches and then committed to remote production branches after a brief review.

This project also utilizes documentation files such as READMEs to provide an overview of the code and to facilitate the development process by outlining goals and objectives.

8 Conclusion

The development of the book dating app has been a comprehensive and enlightening journey. By leveraging modern technology, the app presents a robust, scalable, and user-friendly application that meets the objectives of the project. This process involved careful planning and execution, starting from the initial app creation all the way to the development of complex backend functionalities.

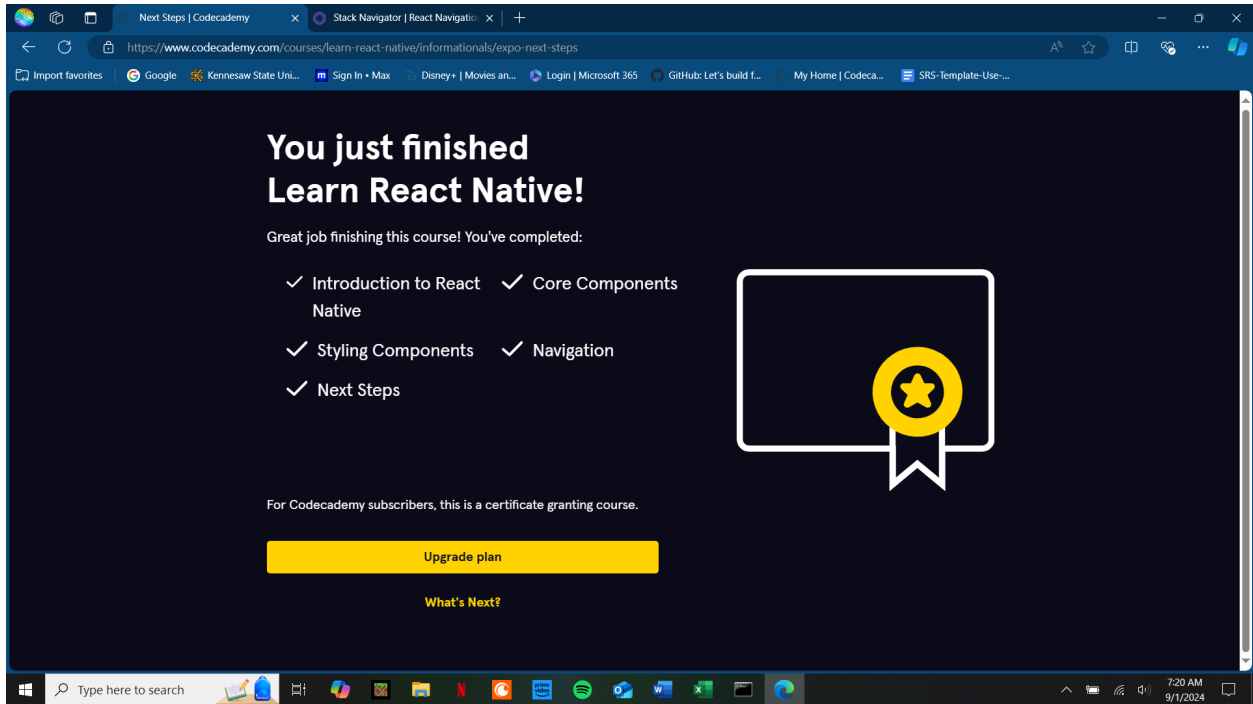
The choice of React Native for the frontend resulted in an interactive, cozy user interface to provide a seamless experience across IOS and Android platforms. The integration of Tailwind and google fonts helped create a visually appealing design. For backend processes, FastAPI was a powerful tool, enabling efficient handling of API requests. Supabase provided a reliable and scalable database while Render hosted the backend services. Together, these tools provided the means to create a mobile app that not only meets the requirements, but also provides a delightful user experience.

Should the opportunity arise, the next objective would be to incorporate more features into the application such as real-time chat features so readers and authors can communicate. The app could facilitate the creation of a friendly online community dedicated to book discovery. For this purpose, the scale of this project would grow exponentially and the backend services would need financial backing; however, with the proper care, the app would be a lucrative investment.

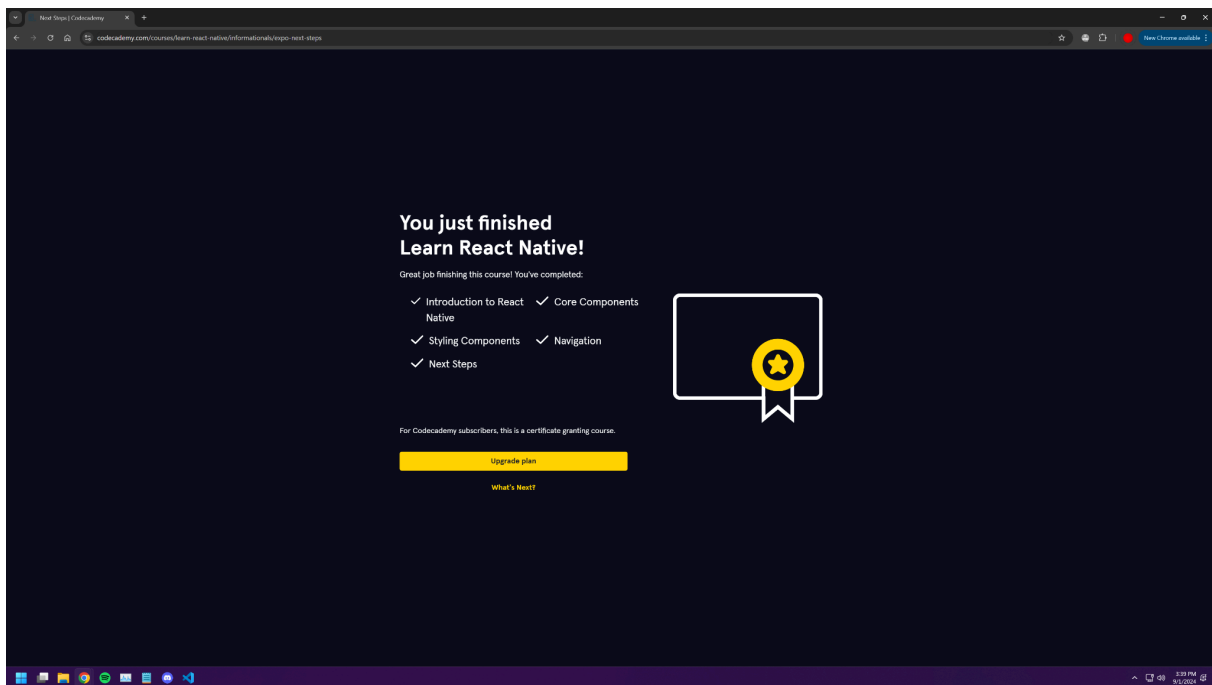
Given these circumstances, the app provides immense satisfaction from its performance. There are limitations to the resources available. Even still, the application meets the goals as described in the report.

9 Appendix

Proof of completion - Thomas Roberts:



Proof of completion - Tristan Sanford:



Proof of completion - Francesca Del Aguila:

My HomeCodecademy

codecademy.com/learn

My HomeCatalogResourcesCommunityPricingCareer CenterFor Teams

Start free trial

Dashboard

My learning

Events

Projects

Workspaces

Try Plus or Pro with a 7-day free trial

Go deeper and learn job-ready skills. Practice with real-world projects, take assessments, and earn certificates.

Try for free

Keep learning

100%

CreateLearn React Native

Current Module: Next Steps

View syllabusStart practice session0/1 todayResume

View all learning in progress

Follow your progress

Skills4,485 XP recently earnedMost progress

JavaScript	066133200	160 XP
Web development		160 XP
Mobile development		135 XP
Computer science		125 XP
Java		125 XP
Web design		35 XP

Learn more about XPGet us feedbackShow all skills

No weekly target set yet

Set target

Your goal

Not sure yet

Edit

View achievements

Recommended for you

Free course

Learn React

In this React course, you'll build powerful interactive applications with one of the most popular JavaScript libraries.

Free course

Learn React: Introduction

Build powerful interactive applications with React, a popular JavaScript library.

Free course

Learn React: Additional Basics

Take your React skills to the next level by

Popular topics: PythonJavaScriptHTML & CSS

9:12 PM9/5/2024