

BOOK RECOMMENDATION APP REPORT

8TH – OCTOBER-2023

PREPARED BY

20SW056

20SW062

TABLE OF CONTENTS

- 1.** Real World Problem Identification
- 2.** Proposed Solution
- 3.** Responsive User Interfaces (Screenshots of your app on different screens & platforms)
- 4.** Data Storage (With justification for using a particular database)
- 5.** [Optional Section] APIs/Packages/Plug-ins (if used with justifications for using them).
- 6.** Issues and Bugs Encountered and Resolved during Development

1. Real World Problem Identification:

In the digital age, readers are inundated with an overwhelming number of book options. With the rise of online bookstores, e-book platforms, and digital libraries, readers often find it challenging to navigate through the vast array of books available. Traditional methods of recommendation, such as bookstore displays or word-of-mouth suggestions, may not cater to individual preferences effectively. The sheer volume of choices can lead to decision paralysis and dissatisfaction with the chosen books, hindering the overall reading experience. Readers need a way to sift through this information overload and find books that align with their unique tastes and interests.

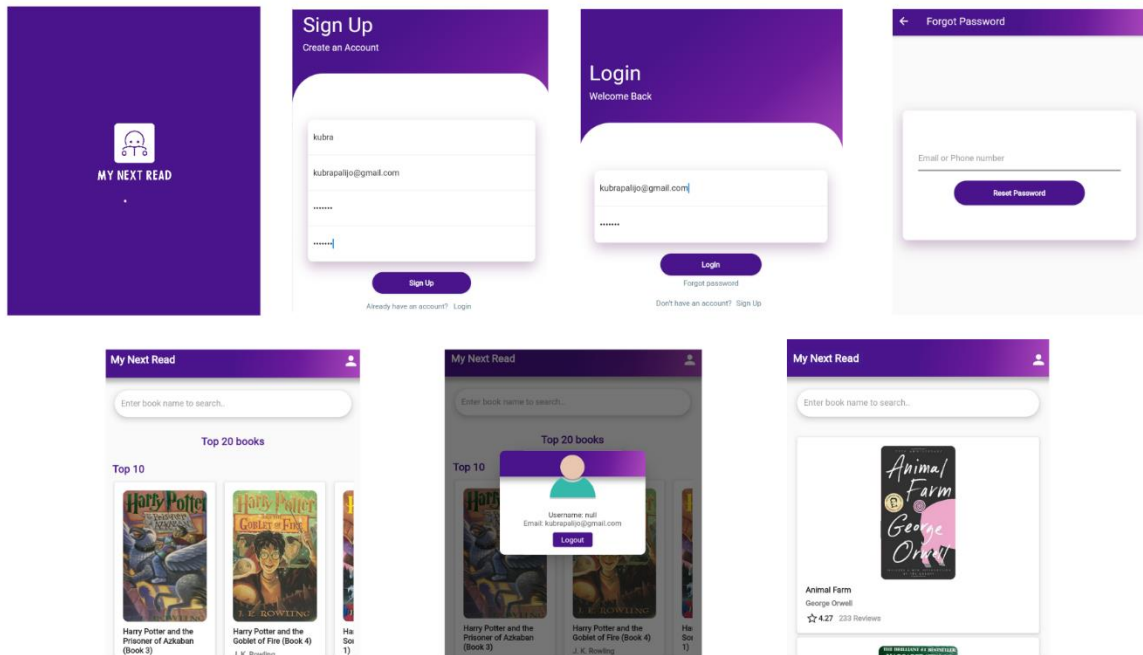
2. Proposed Solution

The Book Recommender System is designed to assist users in finding their next read based on two main techniques: Popularity-Based Filtering and Collaborative Filtering. It aims to solve the problem of information overload by providing personalized book recommendations to users, enhancing their reading experience.

Implementation

- **Data Preparation:** Cleanse and preprocess the books and ratings datasets, ensuring data integrity.
- **Popularity-Based Filtering:** Identify popular books by counting ratings and calculating average ratings, displaying the top 20 on the homepage.
- **Collaborative Filtering:** Implement algorithms to calculate similarity scores based on user input, recommending relevant books.
- **Integration:** Integrate the frontend developed in Flutter with the Flask backend to enable seamless user interaction.

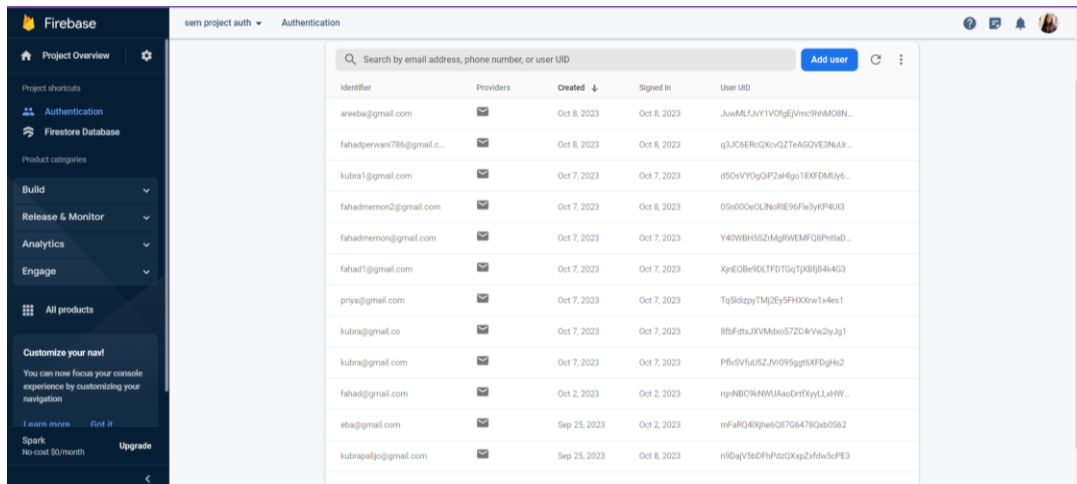
3. Responsive User Interfaces



4. Data Storage (With justification for using a particular database)

For our Book Recommendation App, we have used Firebase for user authentication. Here's why:

- **Seamless Integration:** Firebase integrates smoothly into our app, providing a hassle-free authentication process for users.
- **Secure Authentication:** Firebase offers robust and secure user authentication services, ensuring the safety of user credentials and data.
- **Multi-Platform Support:** Firebase authentication supports various platforms, including web and mobile apps, offering a consistent user experience across different devices.
- **Authentication Methods:** Firebase supports various authentication methods, such as email/password, social media logins, and phone number verification, catering to diverse user preferences.
- **Scalability:** Firebase authentication scales effortlessly with our growing user base, accommodating an increasing number of users without compromising performance or security.
- **Developer-Friendly:** Firebase provides developer-friendly APIs and easy-to-implement SDKs, simplifying the integration of authentication features into our app.



The screenshot shows the Firebase Authentication console. On the left is a sidebar with navigation options: Project Overview, Authentication (selected), and Firestore Database. Below these are product categories: Build, Release & Monitor, Analytics, and Engage. At the bottom of the sidebar is a 'Customize your nav!' section. The main area displays a table of users with columns: Identifier, Providers, Created, Signed In, and User UID. A search bar at the top allows searching by email address, phone number, or user UID. An 'Add user' button is in the top right. The table lists 13 users, all with email providers, created between September 25 and October 8, 2023.

Identifier	Providers	Created	Signed In	User UID
areeba@gmail.com	📧	Oct 8, 2023	Oct 8, 2023	JwvMLJyY1VOlgjVmc9HwM08N...
fahadpervani786@gmail.c...	📧	Oct 8, 2023	Oct 8, 2023	q3JG6ERhQXovQZTsAGQVE3NuJ...
kubra1@gmail.com	📧	Oct 7, 2023	Oct 7, 2023	d50vVYQgQIP2aHgp18XFDMy6...
fahadmemon2@gmail.com	📧	Oct 7, 2023	Oct 8, 2023	05v60OwOLNwRIE9f3yK4UE3
fahadmemon@gmail.com	📧	Oct 7, 2023	Oct 7, 2023	Y40WBH5G2MylRwEMFQ8Pret4D...
fahad1@gmail.com	📧	Oct 7, 2023	Oct 7, 2023	XpE0Be1DLTfDTGqTjKBj84i4G3
priya@gmail.com	📧	Oct 7, 2023	Oct 7, 2023	TqS8zpyTMj2EY5FH00w1v4es1
kubra@gmail.co	📧	Oct 7, 2023	Oct 7, 2023	8fbFohJXVMksoS7ZC4Vv2yJg1
kubra@gmail.com	📧	Oct 7, 2023	Oct 7, 2023	PflvSVfu05ZJv095ggt6XFdg4s2
fahad1@gmail.com	📧	Oct 2, 2023	Oct 2, 2023	rgnNBC9kNWUaaoDrtFxyLLwHw...
eba@gmail.com	📧	Sep 25, 2023	Oct 2, 2023	mFafQ4XjheGf7G6479Qeb0962
kubrapalvi@gmail.com	📧	Sep 25, 2023	Oct 8, 2023	m9DqY5bDFHbzQXpZxf5w5cPE3

5. APIs/Packages/Plug-ins (if used with justifications for using them).

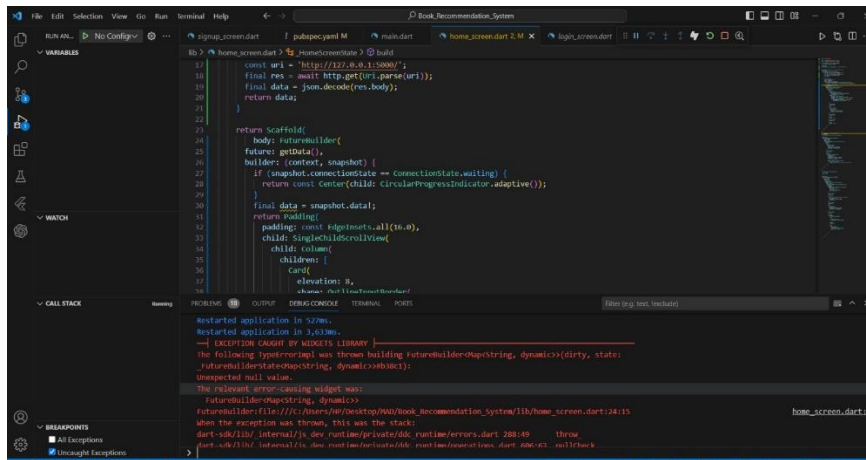
In our Book Recommendation App, we've used:

Provide: Custom module for tailored app functionalities, offering unique features designed specifically for our needs.

Firebase: Chosen for seamless user authentication, real-time data synchronization, scalability, and robust security features.

HTTP Package: Utilized for fetching external data, enabling us to offer diverse and up-to-date book recommendations to users.

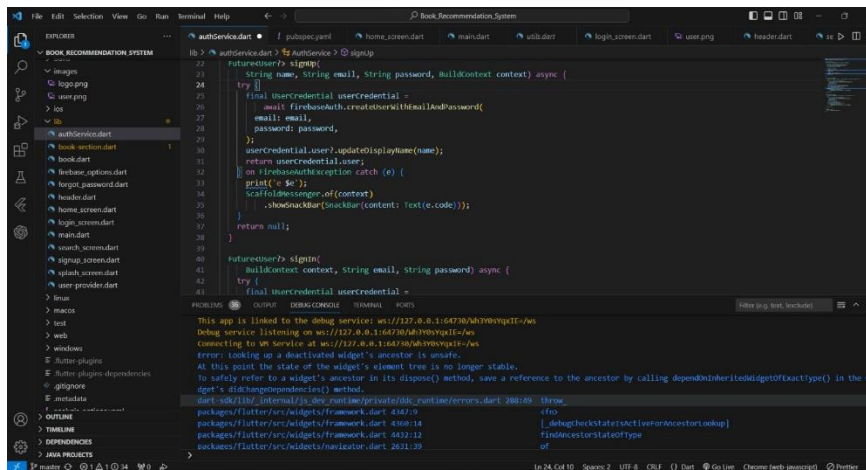
6. Issues and Bugs Encountered and Resolved during Development



Error: "Exception caught by widget library was thrown building FutureBuilder"

Solution:

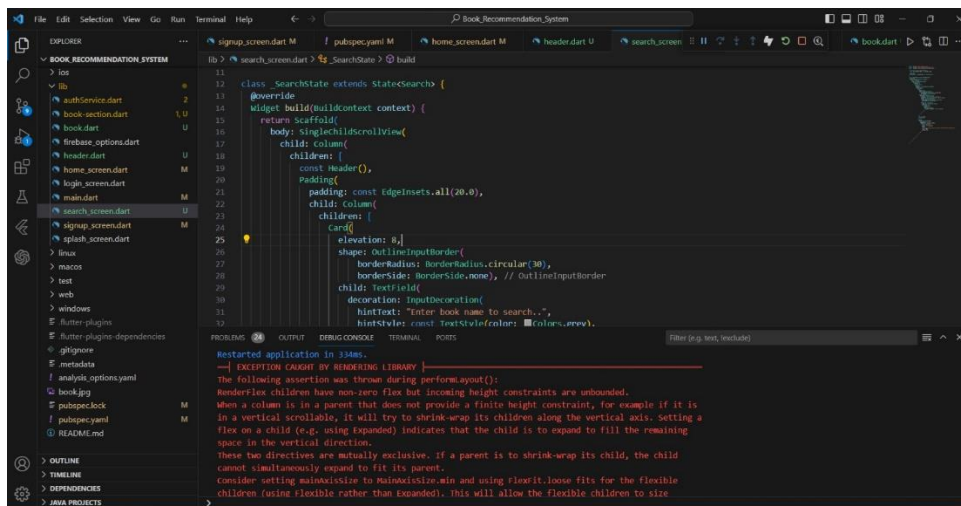
- Null Check: Ensured data fetched by `FutureBuilder` was not null, preventing null value errors.
- Error Handling: Implemented proper error handling within asynchronous functions to catch and rethrow errors.
- Correct Future Implementation: Ensured correct implementation of functions returning a `Future`, enabling smooth asynchronous operations.



The error "Looking up a deactivated widget in an ancestor is unsafe" in Flutter occurs when you try to access a widget that no longer exists in the widget tree. To fix it:

Solution:

- **Dispose Resources:** Properly dispose of resources like streams or controllers when the widget is removed from the tree.
- **Manage Navigation:** Ensure correct navigation flow, using named routes and managing the navigation stack.
- **Avoid setState() on Disposed Widgets:** Refrain from calling setState on widgets that have been removed from the tree.



The error "Caught by rendering library: The following assertion was thrown during performLayout()" in Flutter indicates a layout issue. To fix:

Solution:

- **Review Layout Logic:** Check widget constraints and sizes to ensure they are valid and appropriate.
- **Avoid Fixed Dimensions:** Be cautious with fixed dimensions; consider using Expanded or Flexible for dynamic sizing.
- **Inspect Widget Tree:** Examine widget nesting and configurations for conflicts causing layout problems.