RMIT UNIVERSITY

ASSIGNMENT 3 PROJECT REPORT

COSC2659 - IOS DEVELOPMENT

Submitted by Group 3

Le Hong Thai	(s3752577)
Hua Nam Huy	(s3881103)
Phan Vinh Loc	(s3938497)
Tran Hoang Vu	(s3915185)
Nguyen Giang Huy	(s3836454)

Table of Contents

1. Introduction

For software engineering students, one of the most effective methods to acquire coding experience is through collaborating with fellow team members on software development projects. Participating in hackathons organized by tech companies is particularly beneficial as it allows you to tackle real-world issues and find innovative solutions. These hackathons typically ask participants to create software in a limited timeframe centered around a specific theme. In assignment 3, the project will aim to replicate a tech hackathon with the theme of "Discovery." For this project, our team will create an iOS app focused on book discovery, drawing inspiration from well-known apps like Amazon Kindle, Goodreads, and Audible. The app will follow the MVVM design pattern [1].

2. Project Description

Figure 1: BlogReads logo

Our app BlogReads aims to create a user-friendly iOS application that will provide avid readers with a platform for discovering, buying, and organizing their favorite books. The app will feature a collection of the latest books and works of literature from different genres for the user to browse, as well as more information on the description, reviews, and links to sites where the user can purchase the book that they like. Moreover, BlogReads offers a search feature, allowing users to look up specific titles of interest and add them to their personal wishlist, ensuring that users can effortlessly keep track of their growing book collection. In addition, users can personalize their experience by creating a personal account that will store their preferences and settings whenever the user logs in to the app. Behind the scenes, BlogReads makes use of Firebase's capability to store user data, book information, and other key app elements in both Firestore and Realtime Database.

3. Main Features

3.1. User authentication

In our app, user authentication is managed using the **FirebaseAuthService** class which contains key functions for handling user login and registration. To create a new account, users need to provide their email, username, and password on the sign-up screen. Once the user has successfully registered, we will utilize Firebase Authentication to create their account and store their information within the Firestore database. For existing users, the app offers a sign-in feature, where Firebase Authentication verifies user credentials, granting access upon successful authentication. Users can also reset and change their passwords by clicking on the corresponding option in the app. When they initiate this process, Firebase promptly sends them an email containing a clickable link, directing them to a Firebase-hosted webpage where the user can

replace their existing password with a new one. In addition, the app also supports biometric authentication using FaceID for a more convenient and secure login process.

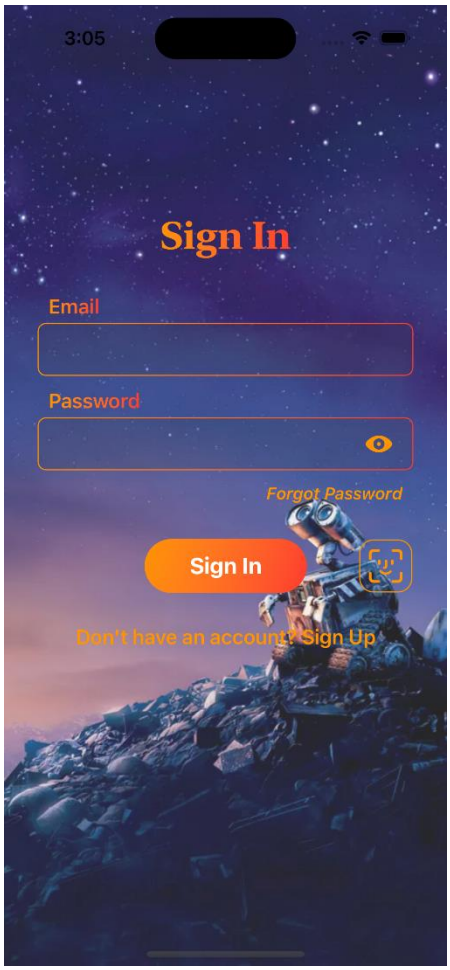


Figure 2: Sign In View

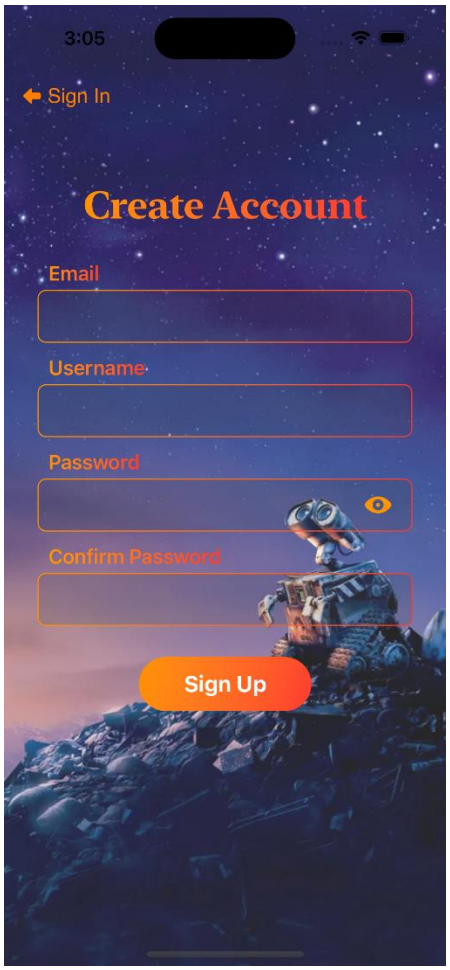


Figure 3: Sign Up View

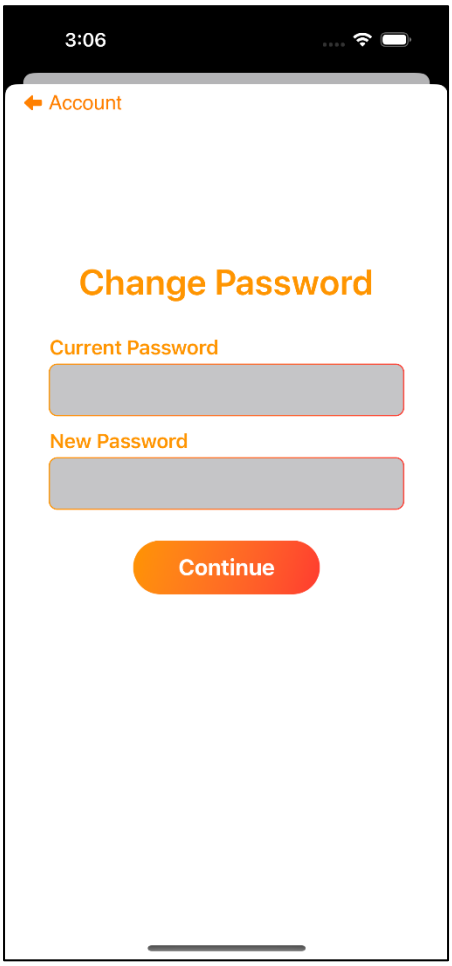


Figure 4: Change Password

3.2. Books display

For displaying books in our app, we use three main views which are Home, Browse, and BookDetails. The home view is the primary view of our app and it is what the user would first see after they have successfully signed in with their account. It consists of an event banner at the top and a list of all of our available books below. Users seeking specific genres can turn to the Browse view which offers a variety of categories for exploration. Additionally, we also provide users with filtering options in each view so that they can sort books by their ratings or names. When a user clicks on a book image, the app will redirect them to the book details view where users can view more information on a specific book. In here, the user can find the name of the book's author, categories, description, ratings, and reviews. The user can choose to contribute their own review to the book they're viewing or add it to their wishlist. For those interested in purchasing a book,

they can click on the buy button at the bottom of the details page and they'll be directed to the Amazon website.

Book details and images are retrieved from Firebase using SwiftUI's @ObservedObject property wrapper. The BookViewModel, serving as an observed object, manages book-related data. When a user navigates to the BookDetailView, the selected book's details are fetched asynchronously using the book View Model. This triggers data retrieval for properties like book name, author, and rating, which are then automatically displayed in the UI. In our app, there exists a Book struct that contains the string variables imageURL which stores the book's Firebase URL and image which checks if the image is stored locally. If found, it's loaded from local storage, avoiding redundant downloads each time we need to display the book image again. If not, it fetches the image from Firebase using asynchronous URLSession tasks. Once downloaded, it's converted to a UII message and saved locally for future use, allowing for better performance and image handling.

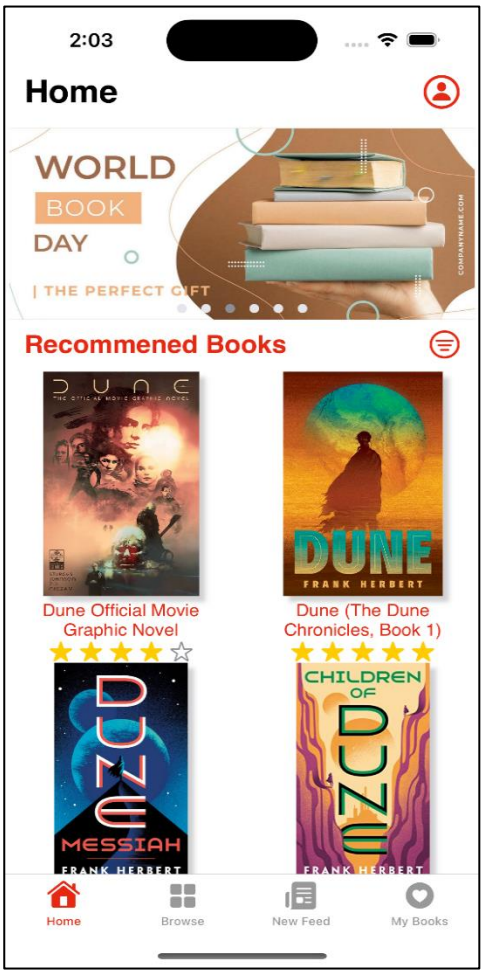


Figure 5: Home View

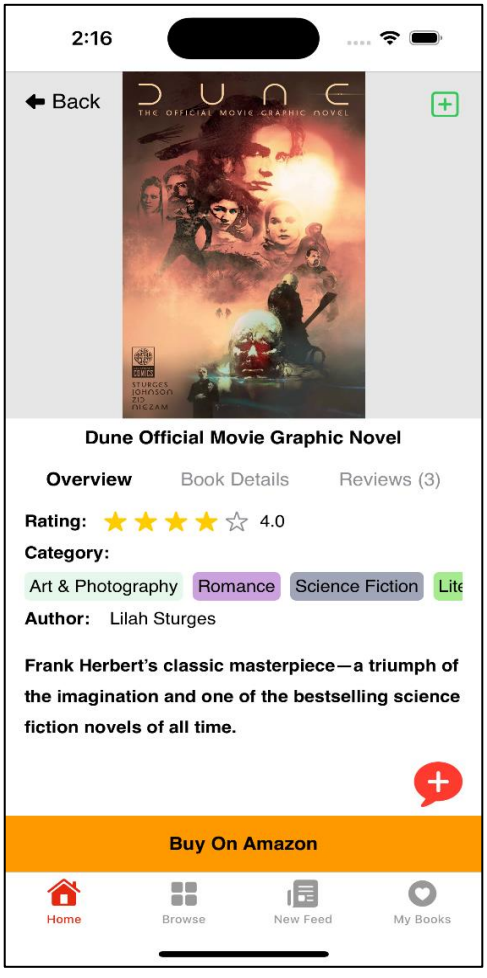


Figure 6: Book Details View

3.3. Search

In the search view, the user can enter the name of the book they want to look for in the search bar at the top. When a user enters a search query, the app will invoke the searchBooks function to query and iterate through the Firebase Realtime Database to check if there are books containing the name in the query. When a match is found, the book will then be added to a result array. When all matching books are processed, the search results array will then be passed to the SearchView to be displayed below the search bar as a list. The user can then click on a name displayed in the search results list which will take them to the details page of that specific book.

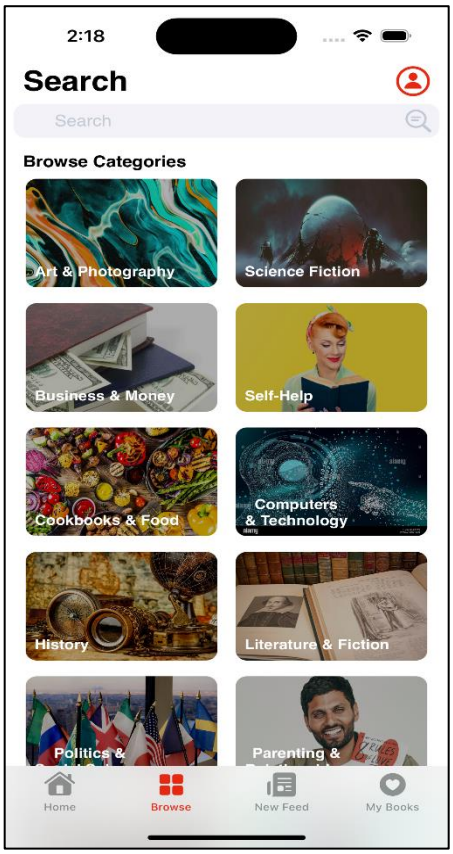


Figure 7: Search View

3.4. Wishlist

After the user has added a book to their wishlist, it will be displayed here in the wishlist view. This view will contain all of the user's favorite books that they would like to read. Each user's wishlist is saved to their account on Firebase and displayed accordingly when they log in. The logic for adding a book to the wishlist is handled by the wishlistToggle function. When the user

presses the add button, it appends the book's ID to the user's wishlist array; if removed, it removes the book's ID. After modifying the wishlist, wishlistToggle calls the updateUser function, passing the current user object. The updateUser function updates the user's data in Firestore by finding the user's document based on their email and replacing the wishlist data with the updated version. This two-step process ensures that changes to the user's wishlist are synchronized with Firestore, maintaining an accurate record of the user's preferences.

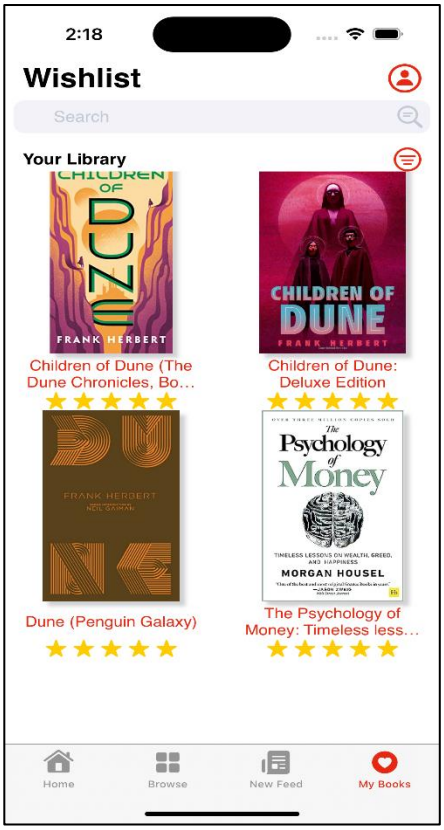


Figure 8: Wishlist View

3.5. Newsfeed

The NewsFeed View is responsible for displaying a news feed of user reviews and related book information. Each time a user adds their review to a book, it will be displayed in a list, showing the user's rating and the name of the work. Additionally, for each review, it looks up the corresponding book from the Realtime Database. If a matching book is found, it creates a NavigationLink to the BookDetailView for that book, displaying the book's name, author, and cover image. The user can navigate to the book's details by clicking on the displayed book shortcut below the review.

The logic behind this view involves populating an array called allReviews with user reviews for all books in the database. It achieves this by making asynchronous calls to Firebase for each book, fetching the associated reviews, and adding them to the array. The reviews are then sorted by date, ensuring the most recent ones appear at the top of the feed. This way, users can see a chronological list of reviews and easily access the details of the books being reviewed.

Figure 9: Add review view

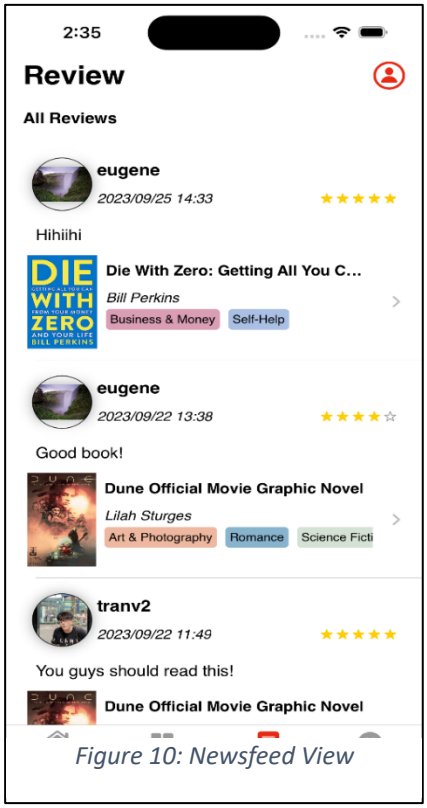


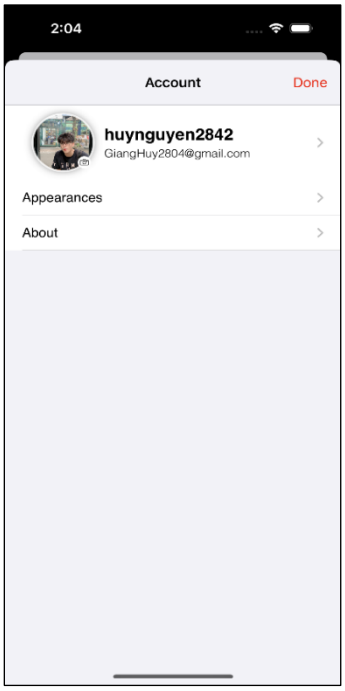
Figure 10: Newsfeed View

3.6. User profile / Settings

Within the settings, the user can configure the app appearance to their liking and also view their user profile. Users can edit their account information such as profile image, username, address, and bio. The new changes will then be added onto Firebase when the user presses update. For this, we use the updateUser function in our FireBaseDB view model to search the real-time database for documents with the email field matching the user's email. After which it will update the correct user with new information based on the document's ID. Additionally, the user can change their password, sign out, and delete their account using the options at the bottom of the view.

The delete user option also follows the same approach except it first finds the current user using their ID and deletes their account on Firebase Authentication before proceeding to delete the user's information and reviews in the collections stored in the Firestore Database. Signing out

is a simple process where the user's authentication session is terminated, effectively logging them out of the app. This ensures that the user's session is ended, and they are required to log in again to access their account. Changing the password is handled securely by first re-authenticating the user using their current credentials, ensuring that they are the legitimate account holder. Once re-authenticated, the user can update their password to a new one, ensuring that only the account owner can change their password.



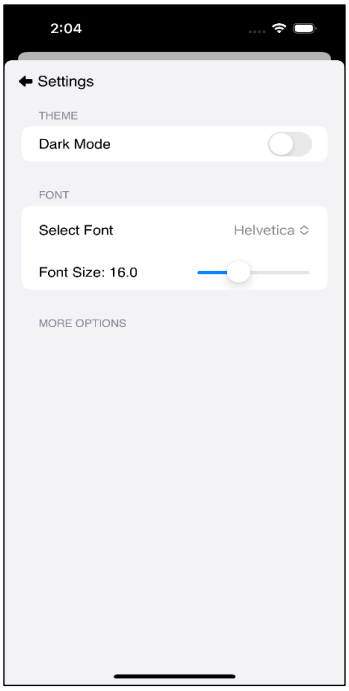


Figure 11: Settings View

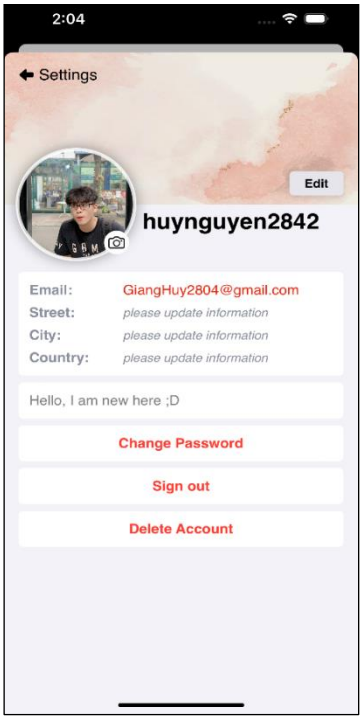


Figure 13: User Profile View

Figure 12: Appearances View

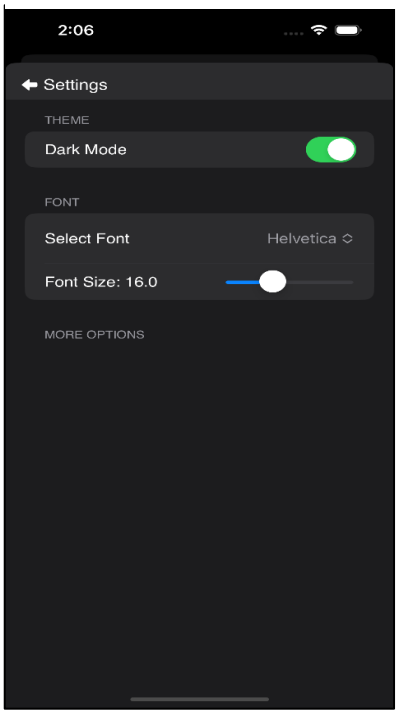


Figure 14: Dark Mode View

4. Application Flow

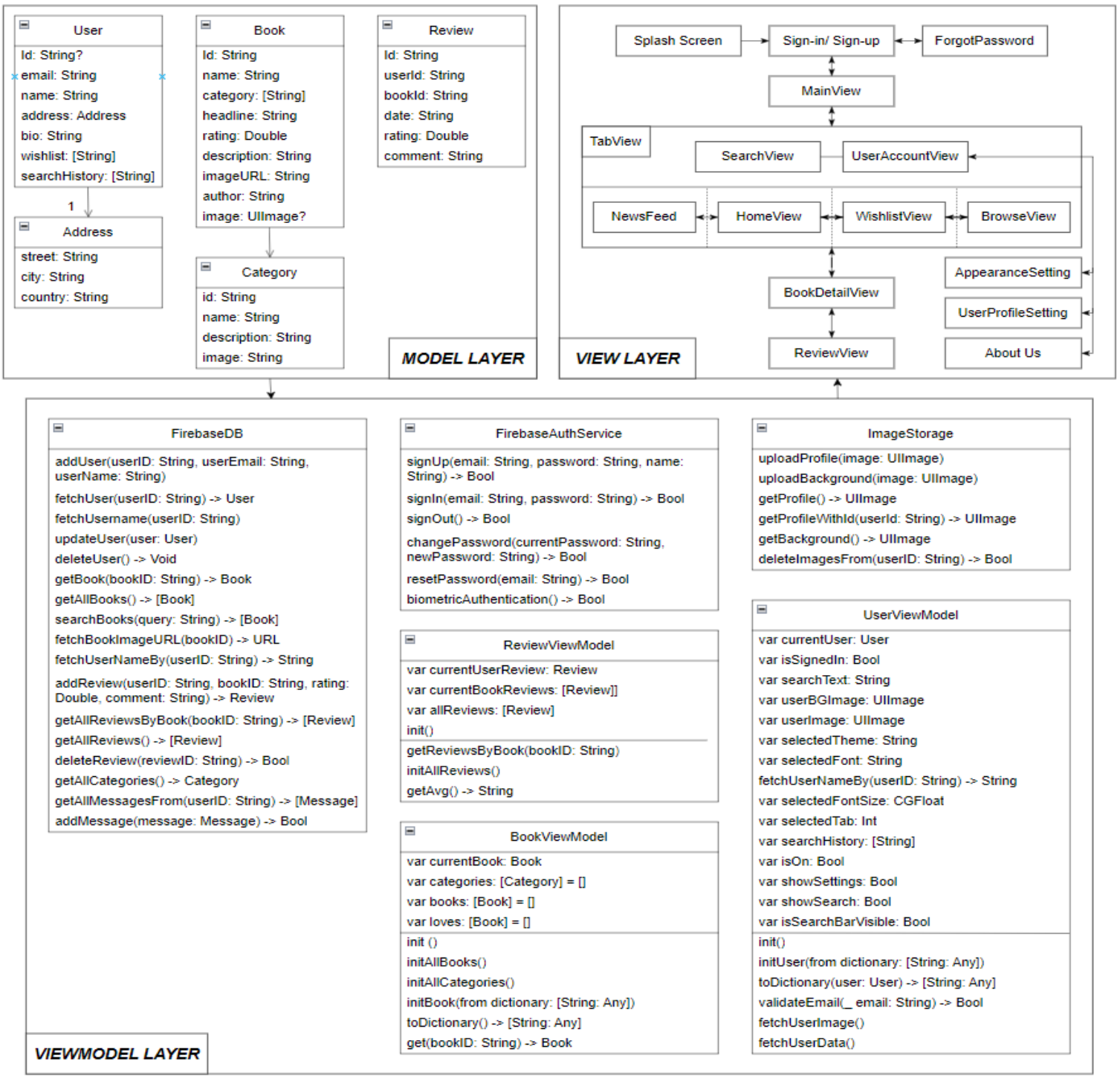


Figure 15: Application Flow Diagram

5. Known Bugs and Problems

By the conclusion of our development phase for the BlogReads app, we have conducted many system, performance and user acceptance tests, and we're pleased to report that we haven't identified any major bugs or issues. However, we acknowledge that there could exist unknown bugs in our app that we haven't managed to discover, therefore we will proceed to monitor and routinely test our app for any issues that many occur during usage and continue to improve upon the features and performance of BlogReads as time goes on.

6. Project Responsibilities

Member name	Tasks	% of contribution
Le Hong Thai	Book Display, Newsfeed, Settings, UI/UX design, User Profile	**20%**
Hua Nam Huy	User authentication, Sign In View, Face ID, Search, report	**20%**
Phan Vinh Loc	Firebase Setup, CRUD operations, Data Synchronization	**20%**
Tran Hoang Vu	Book Reviews, Wishlist, Dark Mode, Splash Screen	**20%**
Nguyen Giang Huy	UI/UX design, Newsfeed, report	**20%**

Table 1: Project Responsibilities

7. Conclusion

In conclusion, our app has successfully met and even exceeded all of the requirements outlined in the assignment. In terms of User Management and Customization, we have implemented a complete user authentication process, allowing users to create accounts and sign in securely as well as change their passwords. In addition, we have also implemented Face ID authentication as an additional layer of security, allowing users to quickly sign in to the app. For a more customized experience, users can also personalize their profiles with image avatars, bios, and other relevant information. For Search and Filter functionalities, BlogReads also includes a search function and various filtering options, making it easy for users to look up any books that interest them. To handle Data Persistence and CRUD operations, we have chosen Firebase as our

main method for storing and managing book data because of its real-time database which ensures that any changes to the data are reflected across all devices. Lastly, users can change the interface of the app to their liking by changing text font, and text size, and toggling between light and dark modes in the settings. While our app has included all of the basic features, it's worth noting that, due to time constraints, we were unable to fully integrate some planned features. These include allowing the user to add other users to their friend list and chat with each other in real-time. In addition, the user would have also been able to see more notifications about other users' activity in the app instead of only the book reviews in our current implementation.

8. References

[1] M. Aljamea and M. Alkandari, “MMVMi: A Validation Model for MVC and MVVM Design Patterns in iOS Applications.” Available: https://www.iaeng.org/IJCS/issues_v45/issue_3/IJCS_45_3_03.pdf

9. Appendices

```
func searchBooks(query: String, completion: @escaping ([Book]?) -> Void) {
    // Get a reference to the Firebase Realtime Database
    let databaseRef = Database.database().reference()

    // Convert the query to lowercase for case-insensitive search
    let lowercaseQuery = query.lowercased()

    var result: [Book] = []

    // Construct a query to search for books by name
    let booksRef = databaseRef.child("books")

    booksRef.observeSingleEvent(of: .value) { snapshot in
        for child in snapshot.children {
            if let childSnapshot = child as? DataSnapshot,
               let bookData = childSnapshot.value as? [String: Any] {
                var book = emptyBook
                book.id = childSnapshot.key
                if let bookName = bookData["name"] as? String {
                    // Convert the book name to lowercase for comparison
                    let lowercaseBookName = bookName.lowercased()

                    // Check if the lowercase book name contains the lowercase query
                    if lowercaseBookName.contains(lowercaseQuery) {
                        book.name = bookName
                        // Add other book attributes as needed

                        result.append(book)
                    }
                }
            }
```

Figure 16: searchBooks function