eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Bachelor Degree in Computer Engineering
Software Engineering

Thesis

SocialBook
A social network for sharing books built with Flutter and Firebase

---

Author
Veselin Radoslavov Solenkov

Director
Mikel Larrea Alava

Contents

1. Introduction **5**

2. Project Management Plan **7**

2.1. Project's aim 7

2.2. Scope 8

2.3. Planification 10

2.4. Requirements 11

3. Architecture and design **13**

3.1. Architecture 13

3.2. Design 21

4. Technologies **27**

4.1. State of the art 27

4.2. Flutter 30

4.3. Firebase 33

5. Implementation **35**

5.1. Application setup 35

5.2. Code development 39

6. Testing **47**

6.1. Users feedback 47

6.2. Testing screens 49

7. Monitoring and control **55**

7.1. Scope management 55

7.2. Time management 55

8. Conclusions **57**

8.1. Lessons learned 57

8.2. Personal evaluation 57

8.3. Future work 58

Bibliography **59**

List of Figures

Figures

1. Work Breakdown Structure of the project 9
2. System Architecture Diagram 13
3. Use Case Diagram 15
4. Sequence Diagram 17
5. Firestore Database collections diagram 20
6. Signup screen 21
7. Login screen 21
8. Profile screen 22
9. Feed screen 23
10. Search book screen 23
11. Create a post screen 24
12. Posting screen 24
13. Book/Post screen 25
14. Comments update screen 25
15. Flutter architecture 31
16. Flutter components 32
17. SocialBook Cloud Firestore Database 34
18. Realtime Database example 34
19. Flutter installation command 35
20. Flutter installation command -v 35
21. Flutter new project creation 36
22. SocialBook files 36
23. Location for build.gradle 37
24. Project package dependencies 38

List of Tables

Tables

1. Initial hour estimation table for each task 10
2. User Data Document 18
3. Book/Posts Data Document 19
4. Update Data Document 19
5. Login screen 49
6. Registration screen 50
7. Profile screen 51
8. Feed screen 52
9. Search book screen 52
10. Create a post screen 53
11. Book detail screen 54
12. Comment update screen 54
13. Estimation actual and deviation table for each task 56

List of Code Fragments

Code fragments

1. main.dart main() 39
2. main.dart class MyApp 40
3. login_screen.dart class LoginScreen 41
4. login_screen.dart widget BuildContext 42
5. signup_screen.dart Authmethods() 43
6. profile_screen.dart BoxDecoration() 43
7. feed_screen.dart MediaQuery 43
8. search_book_screen.dart SnackBar 44
9. add_post_screen.dart SimpleDialogOption 45

Chapter 1 - Introduction

To begin with, in this first chapter I'm going to explain the source of my motivation behind the creation of SocialBook, the general idea of the project, and a brief introduction to the technologies used for its development. The whereabouts concerning this particular project are thoroughly explained in the following chapters.

The idea for the creation of this mobile app has its origins in the reprography area of the Faculty of Informatics of EHU/UPV. As I was, there having some copies done, I heard how a student in front of me was asking for a particular book on a subject I had already taken. Since I had that particular book in my possession and did not give it much use after the end of the subject, I spoke with him and we agreed that I will lend him the book for the rest of the semester. He needed a copy of the book as soon as possible, thus he was very pleased with the outcome because he wouldn't have to wait for its arrival. In order to return the favor he asked if I needed any book that he had and fortunately, he did have one book that I really wanted to read, so we made a trade that in the end benefited both of us.

That little experience made me think of how it would be possible for us to end up interchanging books if it wasn't for that accidental encounter. That is when the idea of creating SocialBook came to my mind. A small social network dedicated to students that study in the same area, with the goal of having a profile with books they have read and are willing to lend to others or exchange for another one. Also, as a passionate reader, the idea of being able to obtain more books in such a convenient manner and speak with other students about them seemed a very positive practice.

To create an app capable of connecting many students, I opted for using two key relatively new technologies: Firebase and Flutter. Google Firebase is a Google-backed application development software that enables developers to develop iOS, Android and Web apps. It provides a cloud-hosted NoSQL database that lets you store and sync between your users in realtime, it is designed specifically with app scaling in mind, providing the users with security and privacy and ensuring network resiliency.

Flutter is an open-source UI software development kit also developed by Google. Its first stable launch was released on December 4, 2018 and since has been actively improved and maintained. By allowing to create a native mobile application with only one codebase by using just one programming language, called Dart, it fits perfectly the needs of the project. An advantage that Flutter has is that it is built directly into the machine code, which eliminates any performance bugs in the interpretation process, contrary to the approach of most cross-platform frameworks.

The integration of these two tools, as both are property of Google and are meant to be used in cooperation, is seamless and very convenient. The documentation available for them is quite extensive, making it accessible for programmers to peek up and learn at a relatively fast pace. Further explanations about the technologies and the development process can be found in the chapters: *Architecture and design (see chapter 3), Technologies (see chapter 4) and Implementation (see chapter 5).*

Chapter 2 - Project Management Plan

This chapter's focus is on the project's aim, general and specific objectives, along with the scope, as well as the organizational structure for the project (WBS), the minimum requirements and the respective exclusions.

2.1 Project's aim

Currently, there are some applications that offer users an environment in which they can search for books and interact with others. The most well-known one is *"Goodreads"* , an American social cataloguing website that allows individuals to search its large database of books. Other examples are *"The story graph"* and *"Booktrib" .* However, none of them focus on book trading in a specific area.

As it was mentioned in the previous chapter, the project intends to provide a tool to the students that will permit them to share and find books that can be interchanged with each other. To obtain such an application a couple of essential objectives have been priorly established and in case that it is possible some additional ones have been also added. All these objectives are listed as follows:

- Main objective:
    - Create a scalable social network native application for sharing books.
    - Obtain fast and smooth performance.

- Specific objectives:
    - Incorporate Firebase as a cloud-hosted NoSQL Database.
    - Document all the code written.

- Additional objectives:
    - Design a simple and pleasant user experience.
    - During development obtain feedback from university students and implement additional features based on the recommendations.

- Optional objectives:
    - Upload application to play store.
    - Translate user interface in more languages.

2.2 Scope

The scope of this project is to meet the project's main, specific and additional objectives. Therefore, here are indicated all the specific tasks that need to be performed in order to achieve the proper development of the project. These tasks are included in the Work Breakdown Structure *(see figure 1)* and their description is as follows:

1. **Management**: this section refers to the essential tasks that need to be carried out for the project to be properly executed, monitored and controlled.

    1.1. Scope statement: outlines the entire project.
    1.2. Planification: defines an estimation of the hours needed to complete each task *(see table 1)*.
    1.3. Monitoring and control: tracks the project's development and tasks completion.
    1.4. Requirements: identifies the requirements and possible future additions of the project.

2. **Product**: all the tasks involved in development of the application.

    2.1. Research: a study of the stare of the art and decisions around the overall objective of the project.
    2.2. Technologies: deciding on the technologies to be used and learning to manage them.
        2.2.1. Flutter.
        2.2.2. Firebase.

    2.3. Architecture:
        2.3.1. Database: development of the database.
        2.3.2. Design: define the components and communications between them.

    2.4. Implementation:
        2.4.1. Set up: first steps in creating the application.
        2.4.2. Interfaces: explanation of the implementation of various components.

    2.5. Testing:
        2.5.1. User experience: Obtain feedback from users and implement changes.
        2.5.2. Use cases: Test every use cases' correct functionality.

3. **Documentation**: includes the development of this thesis and later the defence of the project in front of the tribunal.
    3.1. Thesis
    3.2. Defence

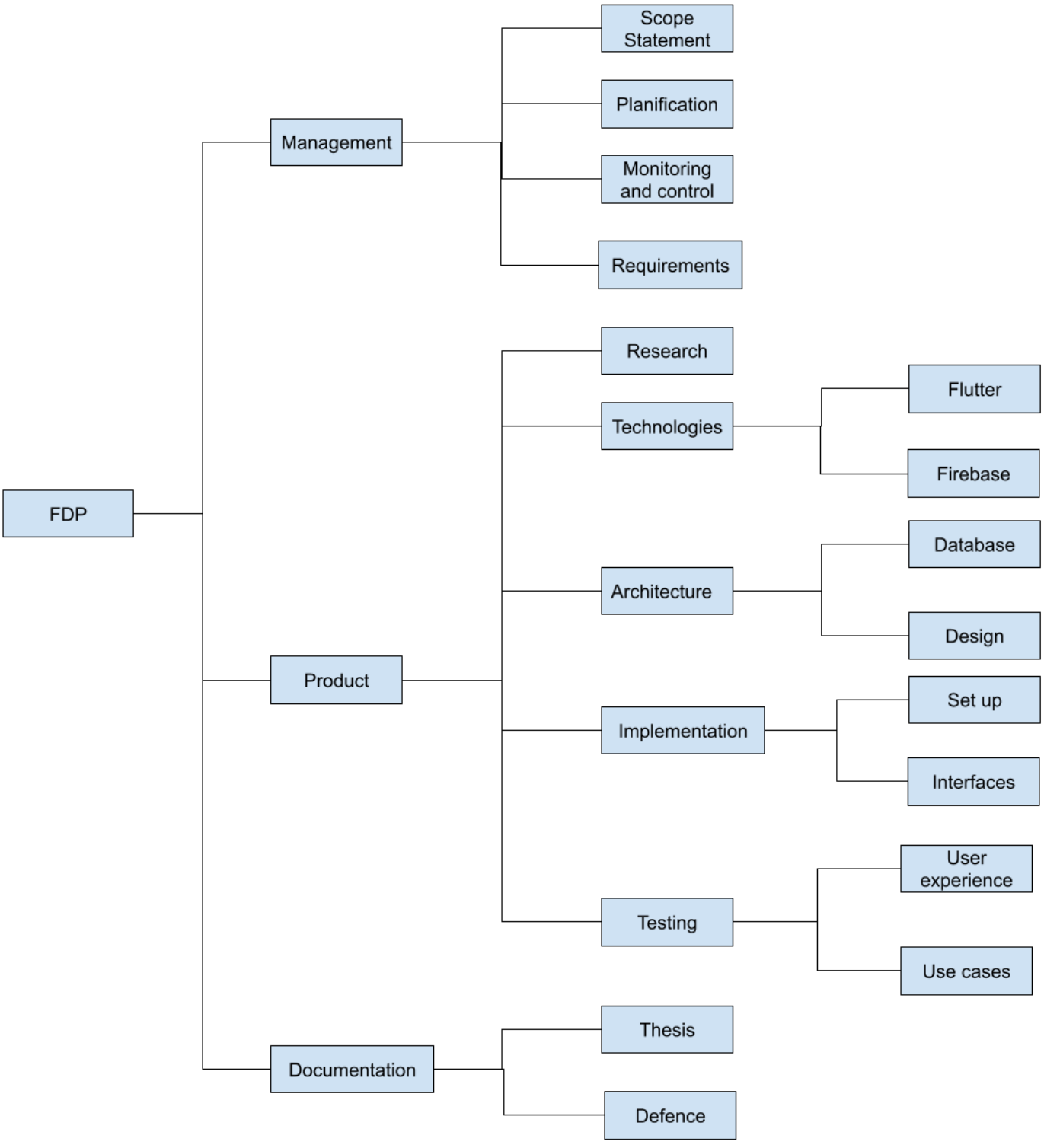


***Figure 1**: Work Breakdown Structure of the project*

2.3 Planification

Code	Tasks	Estimated duration (h)
1	**Management**	
1.1	Scope statement	3
1.2	Planification	6
1.3	Monitoring and control	9
1.4	Requirements	2
		20
2	**Product**	
2.1	Research	**10**
2.2	Technologies	**25**
2.2.1	Flutter	18
2.2.2	Firebase	7
2.3	Architecture	**55**
2.3.1	Database	20
2.3.2	Design	35
2.4	Implementation	**90**
2.4.1	Set up	5
2.4.2	Interfaces	85
2.5	Testing	**15**
2.5.1	User experience	12
2.5.2	Use cases	3
		195
3	**Documentation**	
3.1	Thesis	75
3.2	Defense	10
		85
	Total	**300**

Table 1: *Initial hour estimation table for each task*

2.4 Requirements

The requirements for the scope of the application are the following:

- Must run properly in iOS and Android.
- Must be integrated with Firebase.
- Code must be commented and well maintained.
- Performance must be very good, no errors or lag.
- Users must be able to at least:
    - Create an account.
    - Publish a book/post.
    - Search for books.
    - Leave a comment on books.
    - Follow and unfollow users.
    - See if books are available or not.
    - See the format and theme of the books.
    - Delete books/posts.

Due to the scope and time limitations the following aspects will be excluded from this iteration of the project, but are to be taken into account for future iterations :

- Publish the application in iOS and Android shops.
- File management system for PDF book sharing.
- Addition of an administrator and professor/teacher roles.
- Translation to other languages.

Chapter 3 - Architecture and design

This chapter revolves around the user experience of the app and the sequence of actions a user can experience, it also describes the architecture behind SocialBook and the data models.

3.1 Architecture

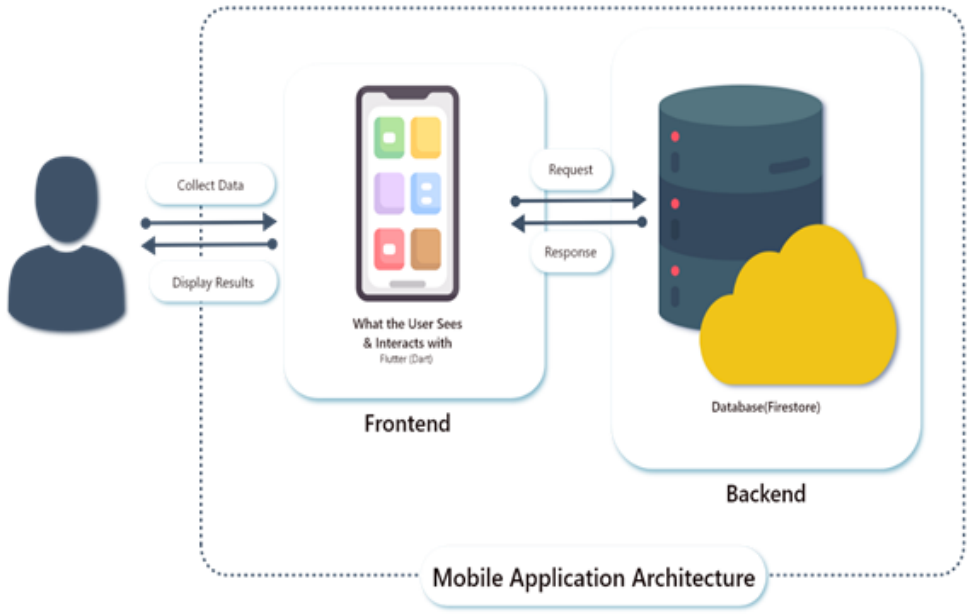


***Figure 2**: System Architecture Diagram*

The front end is visible to the users of the application *(see figure 2)*. Users interact with the interface and the requested data is sent to the database to be processed and given as a response to the users on their screen, after getting data to the users queries from backend (Firebase). SocialBook stores the user data that is created while logging in to the system, creating book/post, adding comments and responses. Firebase helps in storing information, can retrieve and display it to user, also involves authentication, storing files and communication between users of the platform with the help of comments.

In this architecture, the app only consists of static content and assets, and all your dynamic content and user data is stored and retrieved from Firebase with Cloud Firestore Realtime, a cloud hosted NoSQL database. It manages realtime data in the database. So, it easily and quickly exchanges the data to and from the database.

A use case diagram is used to identify and organise the requirements of the system. The use case diagram below *(see figure 3)* describes the functional requirements of this application from an end user's view. It consists of the interactions between the user and system and includes the main flow. The system boundary is defined around the use cases so that the developer avoids scope creep. The diagram of the SocialBook application consists of:

- Actor: the user, an actor, can interact with the system to perform tasks. This user can be either a registered on or an anonymous one.

- System Boundary: system boundary present around the use cases, represents the entire system.

- Use case: these are the system functions with which users interact to perform certain activities.

- Extend relation: this relation includes the optional behaviour or functionality of the system.

- Invariant : a condition that must always be true throughout the use case in order to be executed.

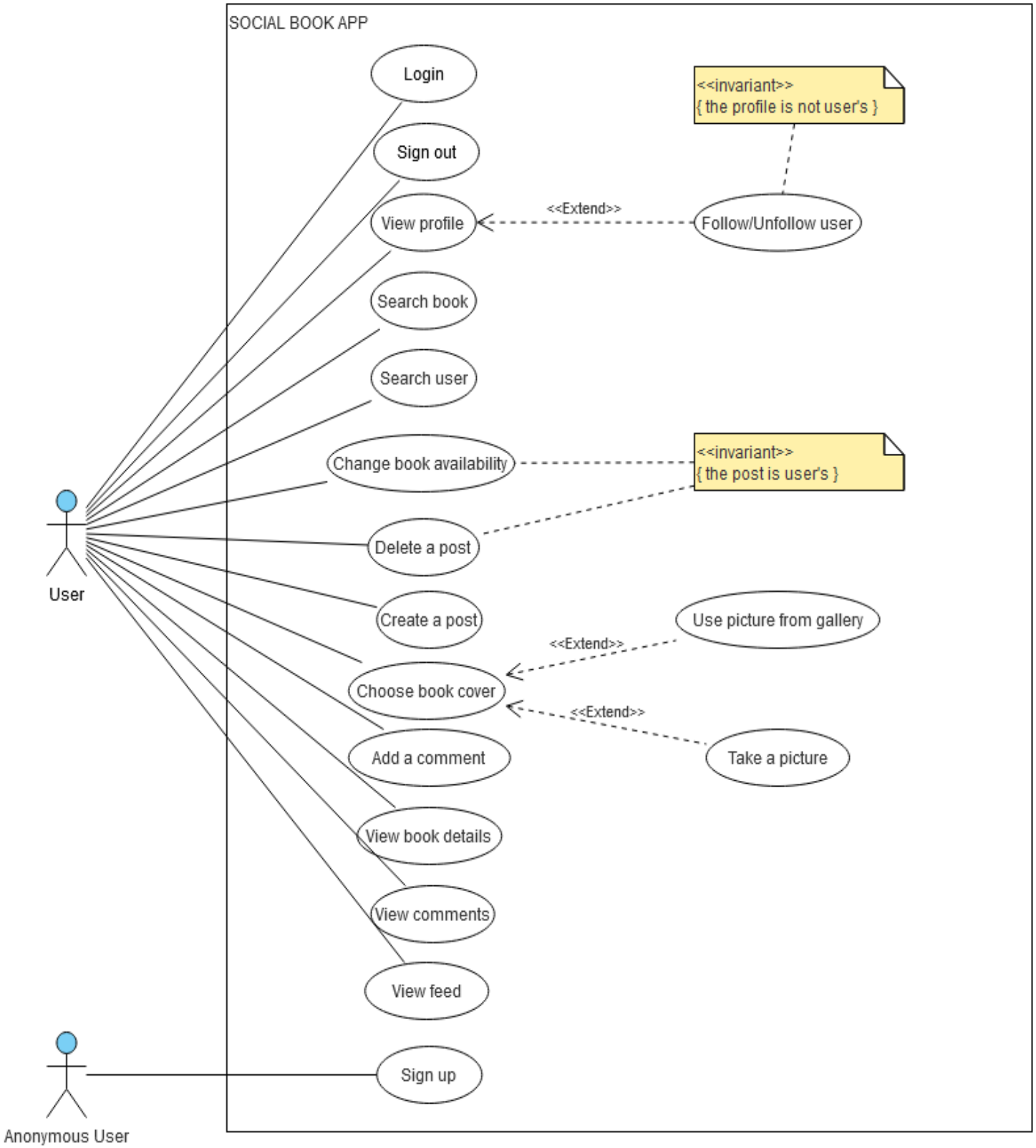


***Figure 3**: Use Case Diagram*

A sequence diagram is an interaction diagram of how the operations are being carried out. It models the high level of interactions between user and the system. The objects that are involved are listed from the left side to the right *(see figure 4)*. The messages are exchanged between the diagrams to show the interaction among objects. It involves:

- An actor: the entity that interacts with the objects by sending messages to them.
- Lifeline: it presents the actor's participation during interaction. It activates when an actor is performing an operation.
- Message: It is a message from the user of the system towards the system that invokes certain functions to process their data and respond to them.
- Return Message: This is the response of the system in return to the message sent by the user.
- The object: It represents the class or an object and how it will behave in the context of a system.

The sequence diagram below describes the detailed user interaction with the system for various use flows. One possible scenario of a sequence in which the data jumps from layer to layer would be the following:

1. User A attempts to log in into the system with his email and password.
2. A call is made to firebase, which will verify whether the that is being served is stored in the database.
3. Once the data is verified, the user A will be able to land on the profile page.
4. The user A searches for a specific book by its name.
5. Firebase retrieves the specific book and displays it to the user.
6. The user A can check the availability of the book.
7. The user A writes a comment on the book.
8. The message is stored in firebase and sent to the owner of the book, which is user B.
9. User B receives the messages in his updates page and replies to it.
10. User A and user B communicate via comments on the book and user B decides to give the book to user A.
11. User B changes the availability status of the book to *“taken”*, which is all stored in Firebase.

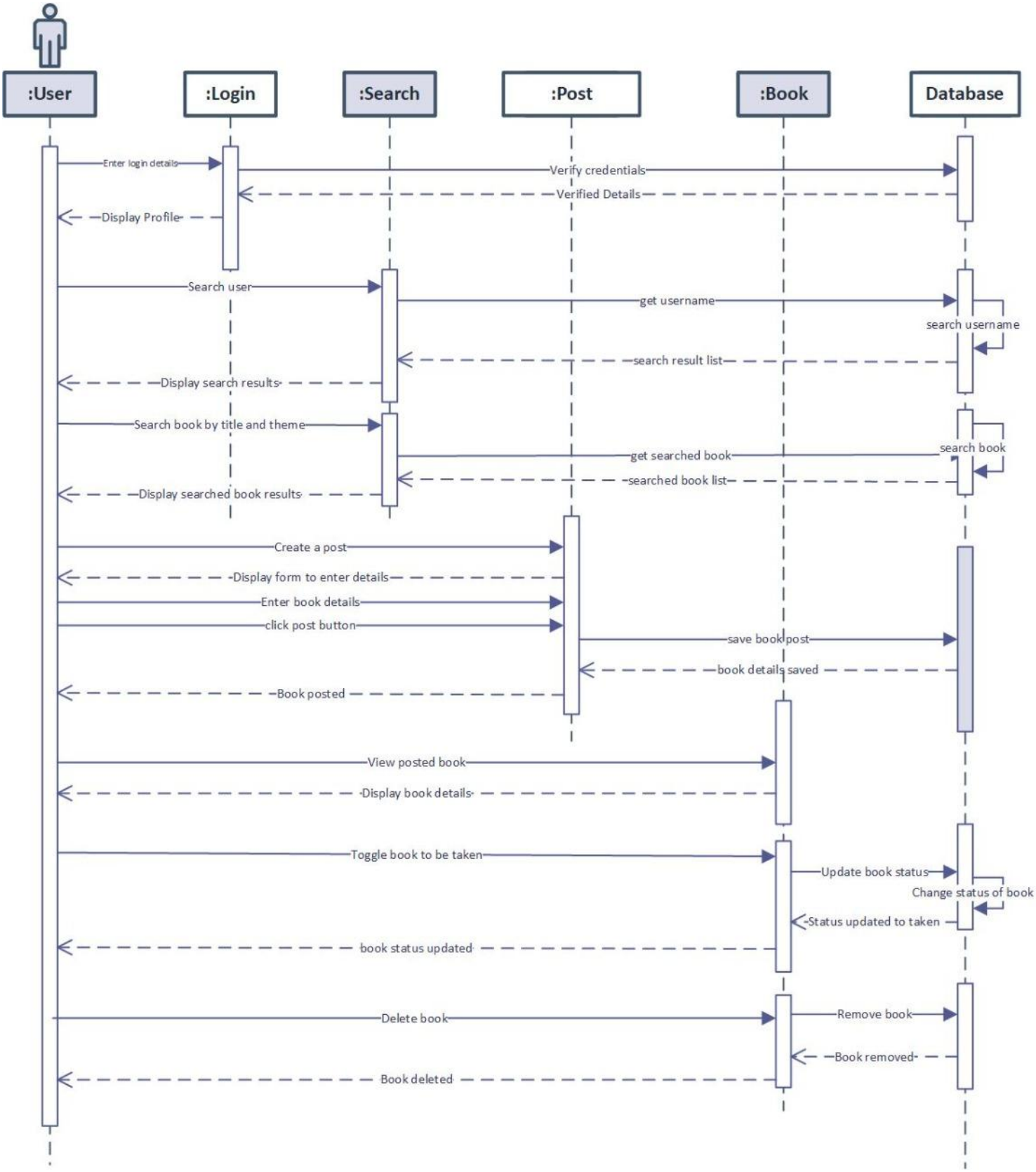


***Figure 4**: Sequence Diagram*

All the information that user generates is stored in the Firebase Firestore Database, from there the data is retrieved and displayed on screens. Data dictionary includes metadata, which is data about data. It represents the name or the collection of the attributes being used, their types and what are they used for, i.e. the description of attributes. Below are the data dictionaries for collections present in Firestore *(see table 2,3 and 4)*.

Field	Type	Description
User_uid	Int	Unique id given to each account holder
Bio	String	Description about the user
Username	String	Name of the account holder
Email	String	Email to log in to the account
Password	String	Password to log in to the account
Photo	URL	Pictorial representation of added data
Follower_uid	String	Unique id of follower followed user account

***Table 2**: User Data Document*

Field	Type	Description
Description	String	Description about added book
Theme	String	Book belongs to which domain or category
Book Availability	Boolean	Book can be reserved by user or not
Post_id	Int	Unique id given to each posted book
Title	String	Title of book
Photo	URL	Cover of book
Date Published	Date	Date of creation of post
Comment	String	Comment on book by users

***Table 3**: Book/Posts Data Document*

Field	Type	Description
CommentId	Int	Unique given to comments posted
Text	String	Comment on the post
Title	String	Username of user posted comment
Post_id	Int	Unique id given to each posted book
Date Published	Date	Date of posting of comment
postUser_Uid	Int	User id of user who comments

***Table 4**: Update Data Document*

Firebase is a NoSQL database which is document-oriented. Thus, there are no tables and rows or specified relations among them. The data store is in the form of documents, which are then organized into collections. Each document in a collection has a unique random IDs associated with it. If a document of a collection does not exist, then Firestore creates it. The data reside inside the document is in the Jason format. The document includes the attributes of their types in it.

The representation below *(see figure 5)* describes the SocialBook application data that resides in Firestore and the collections with the name Users, Posts, and Update. User collection all profile and login details in the document. Posts contain the book posted details and the user who posted the books. While update collection contains all the chat and comment data exchange among different users. Also, it contains the UIDs of the users that chatted along with the dates on which the users commented on book posts.

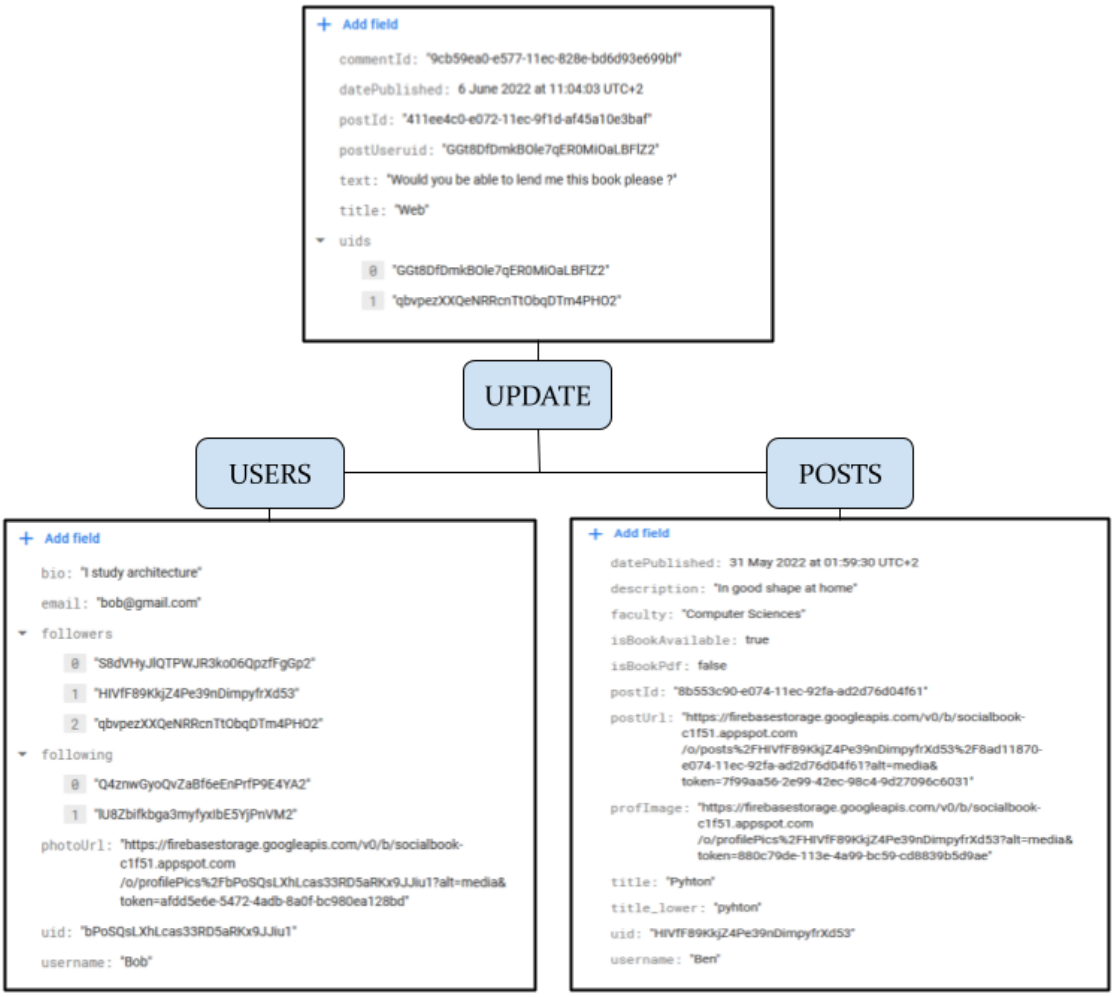


***Figure 5**: Firestore Database collections diagram*

3.2 Design

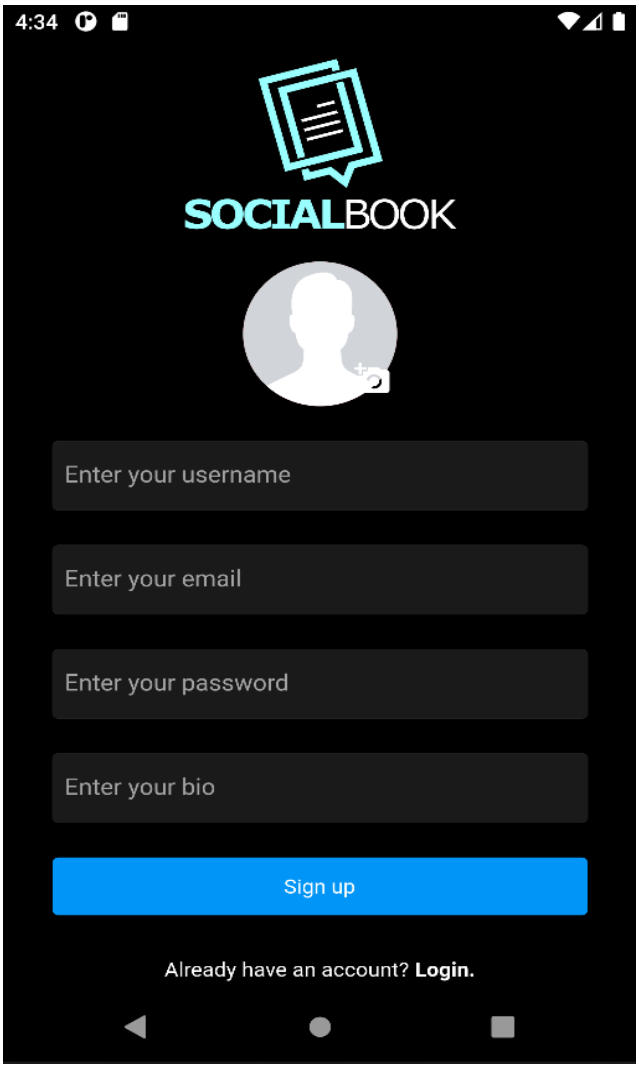


***Figure 6**: Signup screen*

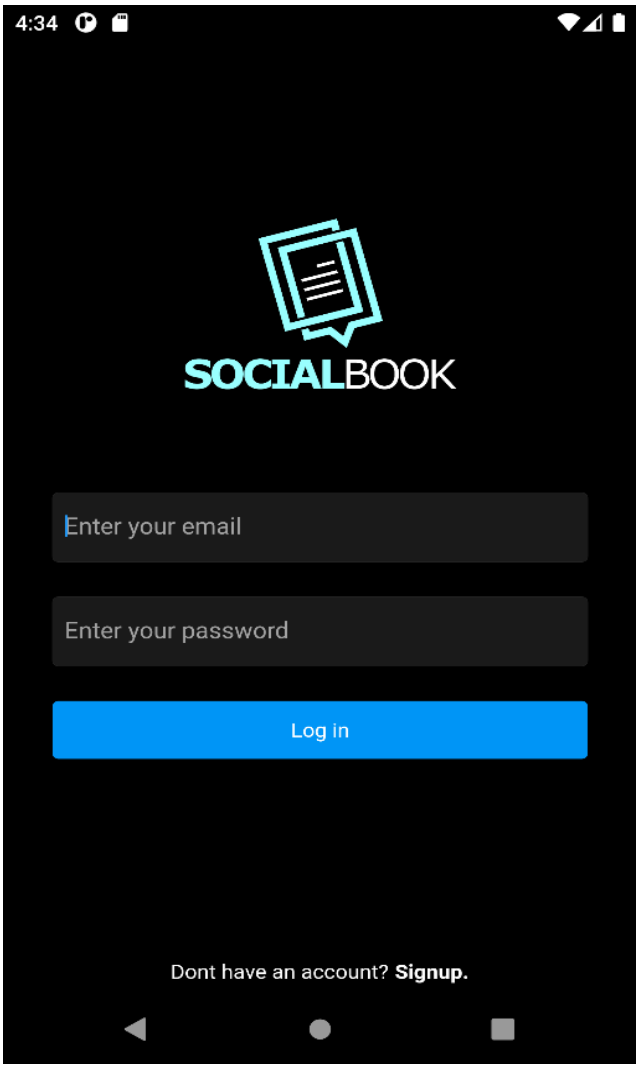


***Figure 7**: Login screen*

The signup screen built in SocialBook is simple in its visual aspects *(see figure 6)*. The screen holds a bright logo of the app. Then there is a circle avatar with image icon, to add user's profile image. There are four text fields below the circle avatar. The first one is for the user to type username he pleases, afterwards is the email to be registered with in the app. A password in the password text field and then at the end, filled up the text field of bio with their experience or background. There is a button *"Sign up"*, below these text fields that the user will click on upon entering all the required details necessary to create an account. If user already have an account, then can click on the text button *"Login"* at the bottom of the screen.

On the Login screen *(see figure 7)*, firstly, there is once more the logo of the app including also the app name "*SOCIALBOOK*". After that, there are two text fields visible on the screen. One for email and other for password. Users filled them with their credentials. Then, a

button is being placed below these text fields so that after entering correct and verified credentials, the user will click on it to log in to the system. If a user is not registered into the app, then will be able to sign up, into the app by clicking on the *"Sign up"* button at the bottom of the screen.

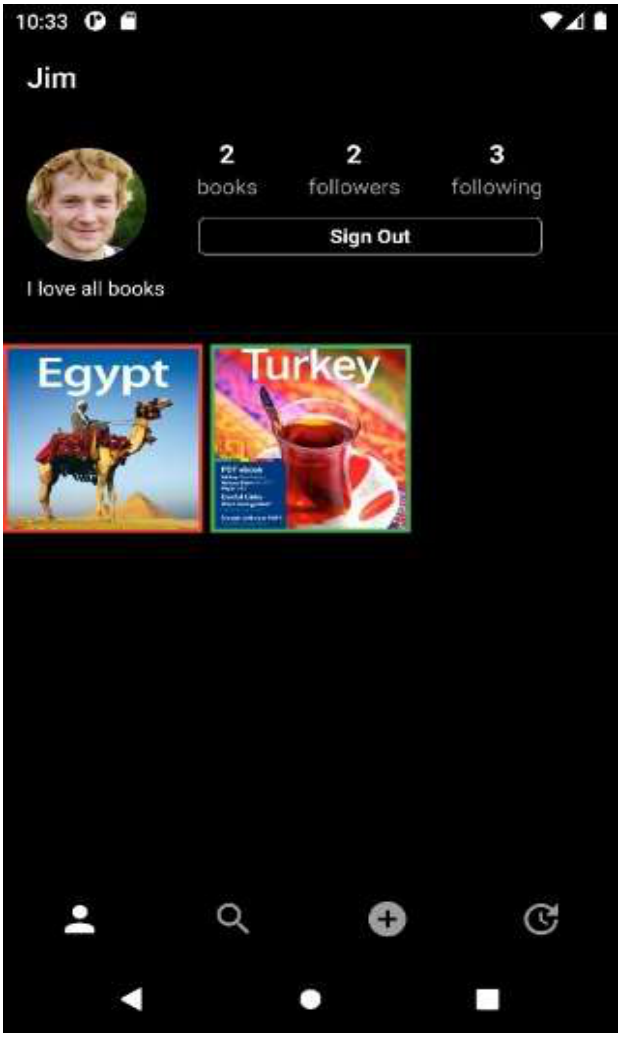


***Figure 8**: Profile screen*

Once logged in, the Profile screen is displayed *(see figure 8)*. On the top the username is being placed, after which below there is a circle avatar having a user profile picture shown along with *"number of books*", "*followers*" and "*following*" text and number is placed in parallel. Below the profile picture, there is the bio of the user. Next there is a container in which an image of all the users books/posts are being shown, upon clicking will reveal book details. The images of the books are being blinked with the corner edges red and green. If book is reserved or in use, then it won't be available and the corner frame on the image will turn red else green if book is available as shown in the image of profile screen.

At the bottom of the app, there is an app navigation bar. By clicking on each icon on the bar, it will navigate the user to the different screens, it is also possible to navigate there through a swipe gesture to the left and right.

***Figure 9**: Feed screen*

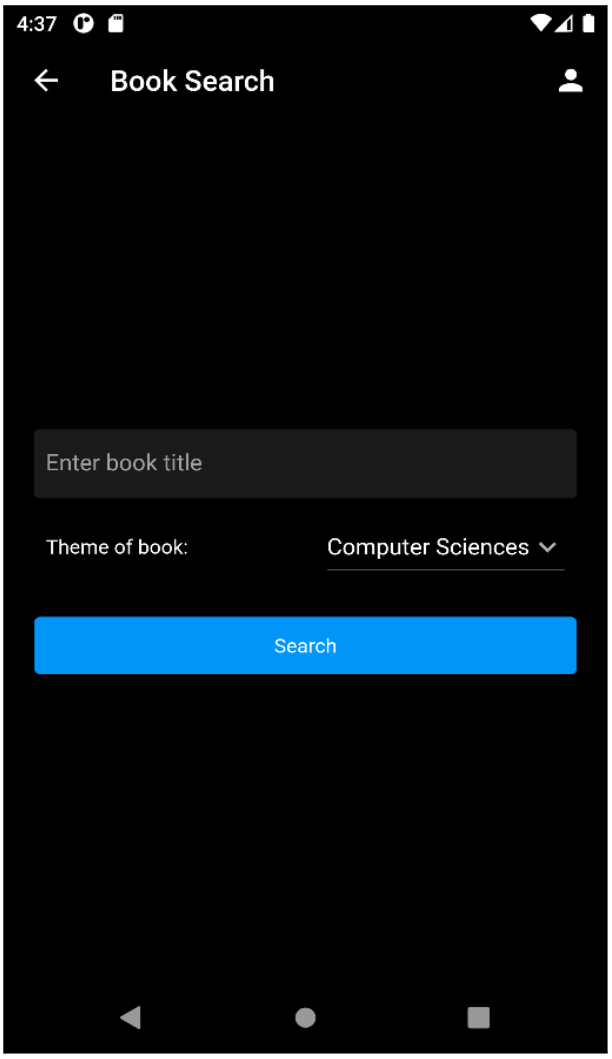
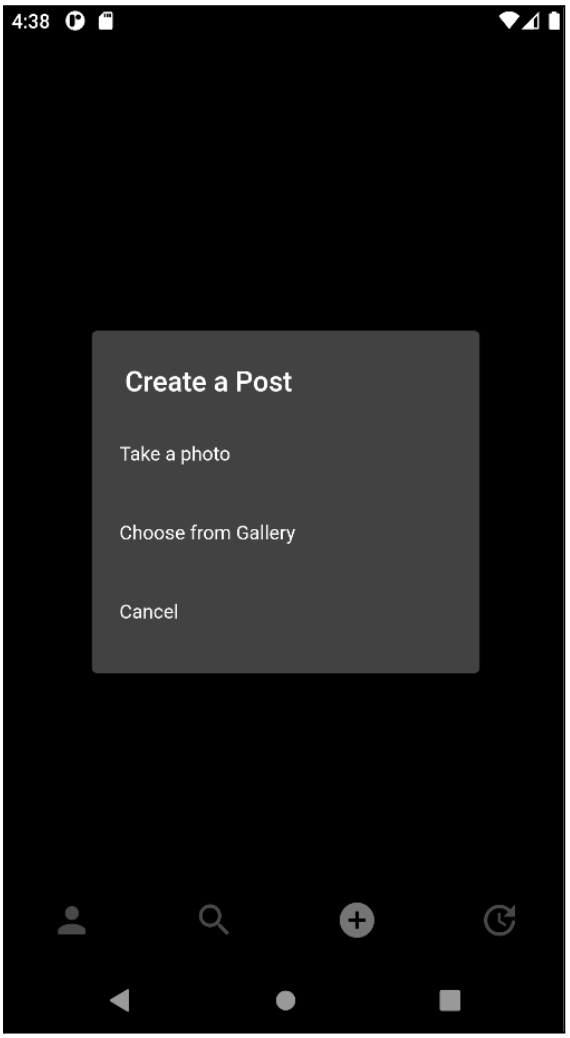


***Figure 10**: Search book screen*

The feed screen *(see figure 9)* is accessed by click on the search icon in the button bar. This screen contains a search bar on top where one can search other users. A book icon placed parallel to the search bar, on clicking by the user, can lead to the book search page *(see figure 10)*. After that, picture containers are displayed. These are the primary image for the book cover images posts, representing the books uploaded by different users. These book images are clickable to view details of that particular book. Again, there is this navigation app bar at the bottom of every page.

In the search book screen there is a "*back arrow*" to return to feed page, a text representing name of the page as "*Book search*" and on top right corner of the page there is a profile icon. In the centre of the page, a text box is placed to enter the desired book name to search and then select the theme of the book by scrolling through the categories provided in the drop-down. After that, click on the *"search button"* below to search for the book.



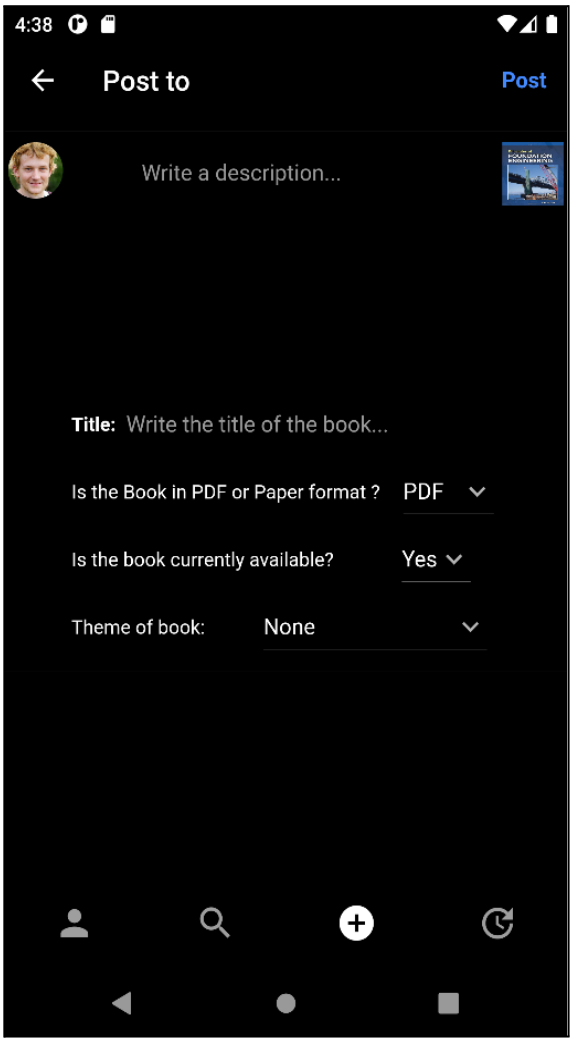


***Figure 11**: Create a post screen*

***Figure 12**: Posting screen*

After clicking on the "*plus icon*" on the bottom navigation app bar, it will open into another screen where a modal pop-up to show 3 clickable text options *(see figure 11)*. In order to create a post, one can, take a picture by using their camera or can click on "*Choose from Gallery*". If a user does not want to post a book, can choose "*cancel*" to discard their actions.

The head of create post screen *(see figure 12)* contains "*back arrow*" to return to feed page, a text "*Post to*" and on top right corner of the page there is a text button named as *"Post"*. In the centre of the page, a text box is placed to enter the title of the book to be posted, then a text with a drop-down is displayed to select the format of the book. After that select from drop down, about whether the book is currently available or not and then select the theme of the book i.e., in which category that book lies in. After entering and selecting all the relevant detail, the user can click on the post button in the top right corner so that the book will be available to other users on the feed page and by the search functions.

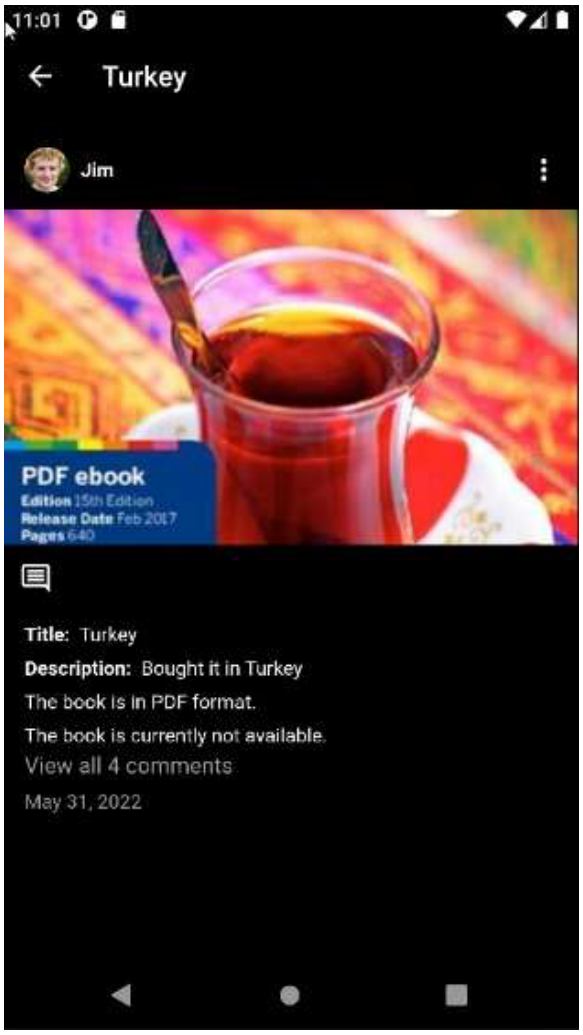


***Figure 13**: Book/Post screen*

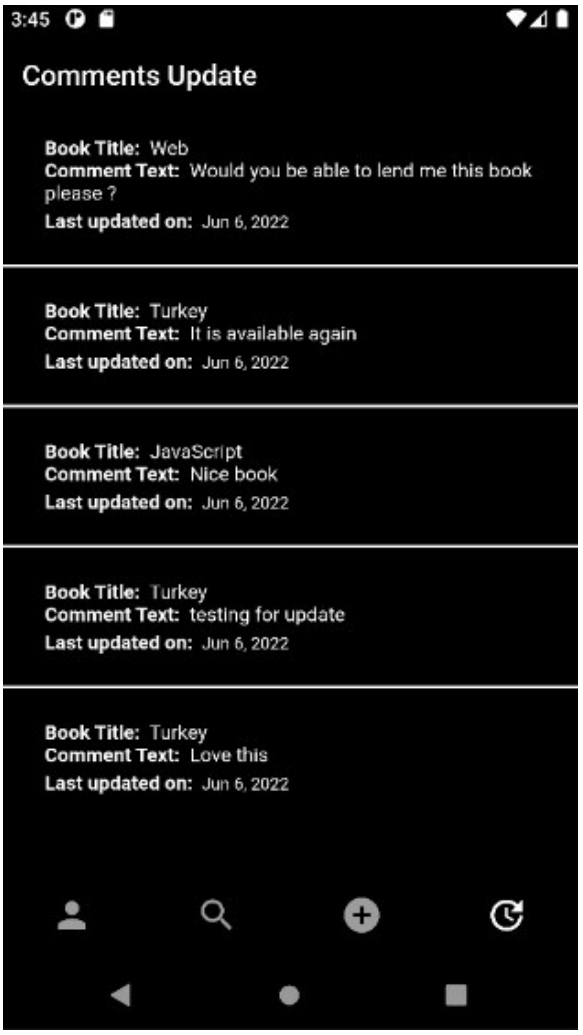


***Figure 14**: Comments update screen*

After clicking on any book post on the feed page, it will direct the user to the detailed view page of that selected book post *(see figure 13)*. In the top left corner, a back arrow along with text "*Name of the book*" is displayed. Then the image and name of the user who posted it and a three dots menu in parallel on the right corner is placed. In case if, a user is viewing his/her own book then the menu will show two options, a "*delete*" button and other is "*toggle to available or reserved*" to show the availability of book but if other users view book details the "three dots menu is not there" on the screen. Below the image, a chat icon is placed so that a user can click to post a comment on the book. After that, book title description and other details are written in text format. Comments can be viewed by clicking on the text *"view comments"*. The day of posting is displayed at the end of the book description.

The comments page includes the text "*Comments Update*" on the top of the page *(see figure 14)*. After which the comments details are visible on screen that include the book title, the comment text, and the last update. These are the comments posted by a user against different books he viewed and then commented upon. The user can click on them and open the thread of that message and reply if wanted.

Chapter 4 - Technologies

In this chapter, four types of mobile app development technologies are described and compared. Afterwards, Flutter and Firebase are more thoroughly explained.

4.1 State of the art

There are a number of different ways to build a mobile app and each one has a unique approach and features. In order to choose the right one for creating this project four different paths are analysed below.

4.1.1 Native apps

Applications that were made with one specific technology dedicated to a concrete platform (Android or iOS) are called native apps. Each mobile operating system has its own languages used for coding native apps. For Android mobile app development, the examples could be Kotlin or Java. For iOS native mobile app solution would be Swift or Objective C.

Compared to other types of products, native apps offer more consistent performance and are more reliable. Native apps allow for the use of the existing system resources provided by the chosen platform in a very convenient manner.

Advantages:

- Very good functionality in an offline environment.
- UI components are custom to each platform, which boosts user experience.
- Has support for device APIs which means higher usability

Disadvantages:

- No reusable code, If a developer wants to create native apps both for Android and iOS, he would have to develop two separate native apps which takes longer and is more expensive to create than other methods.
- As users of different devices may be using different versions of the app, it makes it difficult for the developers to maintain and offer support.

Some examples of native apps are the calculator, notes, camera and all the other built-in applications that you get when you buy a device. Another example are the majority of mobile games, since they all require very good performance and resource management.

4.1.2 Hybrid apps

Hybrid mobile app development is characterized by combining features of a native app and a web app. That means we can build a hybrid mobile app using well-known languages and frameworks, for example, JavaScript, HTML and CSS. Basically, it's about mixing web and mobile elements to quickly create a product available on different platforms.

The process of making hybrid apps consists of creating a backend codebase that will then be covered with a native shell that allows them to be uploaded to Google Play or App Store.

Advantages:

- Maintenance is much easier because it's based on web solutions – native ones are much more complex when it comes to code.
- Lower development cost, especially when you make a hybrid app for many different platforms.
- Adding new features to hybrid mobile apps is simple with one codebase.

Disadvantages:

- Complex apps won't perform perfectly with this solution, more features slow it down.
- Due to the fact that hybrid apps are essentially web-based, they don't work without the Internet connection.

Some of the most popular hybrid apps used nowadays are Instagram and Gmail.

4.1.3 Cross-platform apps

Might resemble hybrid development, but they are not the same. Cross-platform mobile applications are using native elements to give the user great experience despite the device they use. So they share the same codebase for all platforms, but they can have their own respectful differences in UI.

A popular framework for this type of app is Flutter. Which utilises Dart programming language and this gets compiled to ARM C/C++ library. Compiled apps provide better performance than hybrid apps because the WebView wrapper is absent in this.

Advantages:

- Time and cost-effective thanks to developing both app versions at the same time.
- UI performance can be comparable to native apps because it is rendered with native solutions.
- Reusability of code so we don't have to create a separate base for each platform.

Disadvantages:

- Dependency on framework when it comes to hardware, operating system and UI features.
- There's a need to keep all the little differences between operating systems and the hardware they run on, especially when it comes to implementing a complex interface and features.

Three good examples of cross-platform apps are: Facebook, Skype and Slack.

4.1.4 Progressive web apps (PWA)

The main attribute of progressive web apps is running in a web browser, even though it gives the user the native feeling. That means you can install the app on your device, you can use it offline and it can send you push notifications. It can also use hardware features (camera, GPS), but it is much easier on Android than on iOS.

Their progression is based on the user experience that is optimised for each platform. There are no dedicated languages or frameworks for Progressive Web Apps. They can be done in Angular or React. This type of app coding is most popular in e-commerce projects.

Advantages:

- They adapt to different screen sizes – great responsiveness.
- Interaction and navigation resembles native apps, so it is easy for the user to understand how to use a PWA.
- No installation process needed.

Disadvantages:

- Limitations when it comes to hardware and operating system features.
- Usually battery consumption is higher.

Since consumption of visual content should be easy and available, Forbes uses this type of technology for their main business news hub. Another example could be Uber.

4.1.5 Why Flutter

SocialBook is a social network and similarly to Facebook one of its objectives is to reach as many users as possible and thus going with a framework that provides easy development for all platforms is very efficient.

4.2 Flutter

Flutter was previously launched as a project named *"Sky"* and was only executable for android. It came on Google's radar and just a few months afterwards they bought it and renamed it Flutter in 2015. They officially released the project to the public in may 2017 as Flutter Alpha Release, which allowed users to create native applications for iOS and Android platforms with only one code. Flutter uses Dart as a coding language, which is a very flexible client-optimized language for developing fast apps.

The main focus of flutter are the following three main concepts:

1. **Fast development.**

   Stateful hot reload allows changing code and in a moment, this changes can be seen on the screen, without losing the state of the app and developers are able to fix bugs, if appeared, in no time.

2. **Expressive and Flexible UI**

   Flutter moves the widgets, rendering, gestures, and animation into the framework, to give a complete control over every pixel on the screen to give flexibility to build custom designs.

3. **Native Apps for Android and IOS**

   Flutter apps follow platform conventions and interface details, such as icons, fonts, scrolling, navigation, and more. That's the reason why the apps built with flutter features on both the Google Play Store and App store.

Flutter is free and open source that is used by developers and organisations around the world, including enterprise, agencies, and startups. Flutter is a toolkit that makes it easy for the developers to design beautiful interfaces for all sorts of screen sizes and devices and it comes with pre-built widgets that help to lay out your app easily. Flutter asks for a blank window on a device either it's iOS, an android, desktop or a website and it draws on that blank window.

Everything in a flutter is a widget. The main idea is to build an interface with widgets that describes how the UI view should be like in the current state and whenever the state changes, it rebuilds the description. Widget creates a hierarchy in an app where every widget acquires the properties from its parent widget. Flutter contains Cupertino packs and all sets of widgets that are from material design, help to build a basic visual layout structure. The basic widgets that are commonly used are text, row, column, stack, and container. Widgets can be stateful or stateless. Stateful widget can be updated based on user action or data change and will be able to re-render if the widget state changes, while stateless widget is not able to change its state during the runtime of an app.

The layer system is used to design a flutter framework. It consists of three layers: framework, engine and embedder *(see figure 15)*. The layers don't have access to the layer below. The layer present on the top i.e., Framework layer, represents the UI of an application that provides with the reactive and modern frameworks written in Dart. It includes all the required packages, layouts, set of platforms. In framework layer form bottom we have foundational classes and animation painting, gestures that are the building block services. Then comes the rendering layer that helps to build a tree of renderable objects, and it automatically updates the layout to show changes.

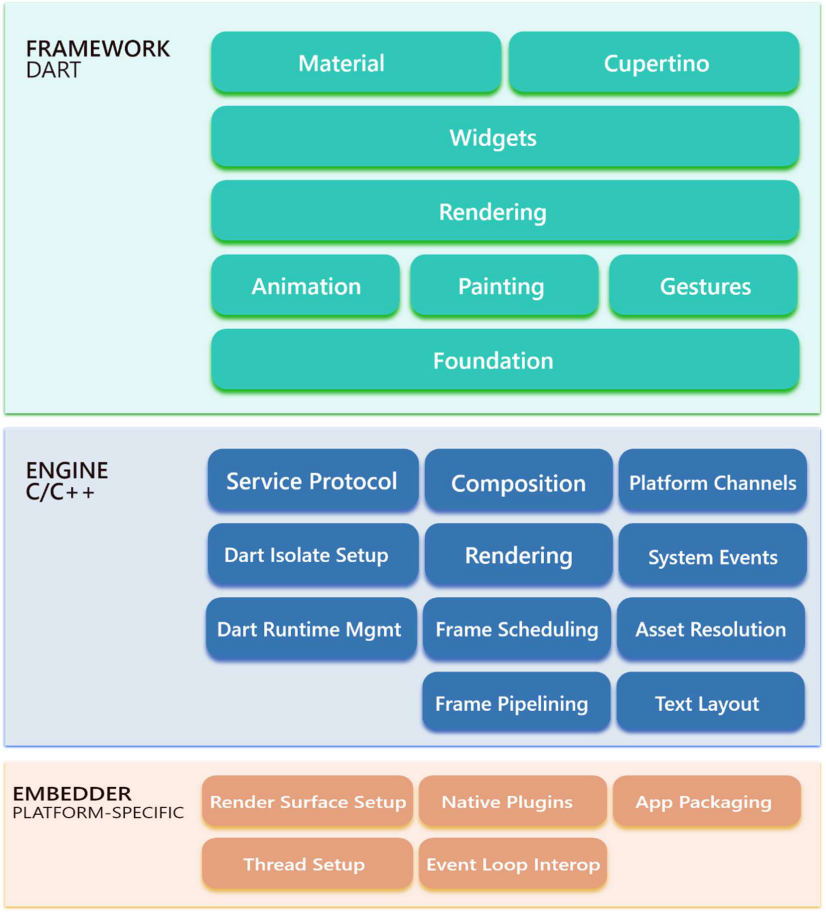


***Figure 15**: Flutter architecture*

Above rendering, a Widget layer is present that defines the combination of classes that can be reused and in the rendering layer each render object has a corresponding class in the widget. The material and Cupertino are the library that uses the widget layer to implement the material or IOS design languages.

The Flutter engine is written mostly in C++ and supports all applications of flutter. It is liable for converting drawing instructions into pixel data i.e., rasterization, whenever a new frame needs to be painted. Engine gives the low-level implementation of core API of flutter including text layout, graphics, network I/O, file, plugin architecture, accessibility support,

Dart runtime and compile toolchain. It is presented to the framework layer through dart: ui, that covers the basic C++code in Dart classes.

The embedder provides the entry point and coordinates with the OS for accessing services like input, accessibility, rendering surfaces and manages the event loop. Java and C++ are the languages used for the Android platform in embedder, C++ for Windows, and Objective-C++ /objective-C for IOS. Embedder helps to integrate the flutter code as a module in an existing application or the code for the whole content of the application.

The Flutter SDK provides with the following:

- A modern react-style framework.
- Rich set of ready-made widgets.
- 2D rendering engine.
- Unit and integration testing APIs and plugin APIs to connect to the system and 3 party SDKs.
- Dart Dev tools for debugging, testing.
- Command-line tools for compiling and building applications.

Flutter contains widgets and independently draws UI components by using its own rendering engine and *"Skia"* graphics library *(see figure 16)*. The code compiles to Android or IOS native code beforehand that ensures that all the UI elements behave in the same manner on any platform or device.

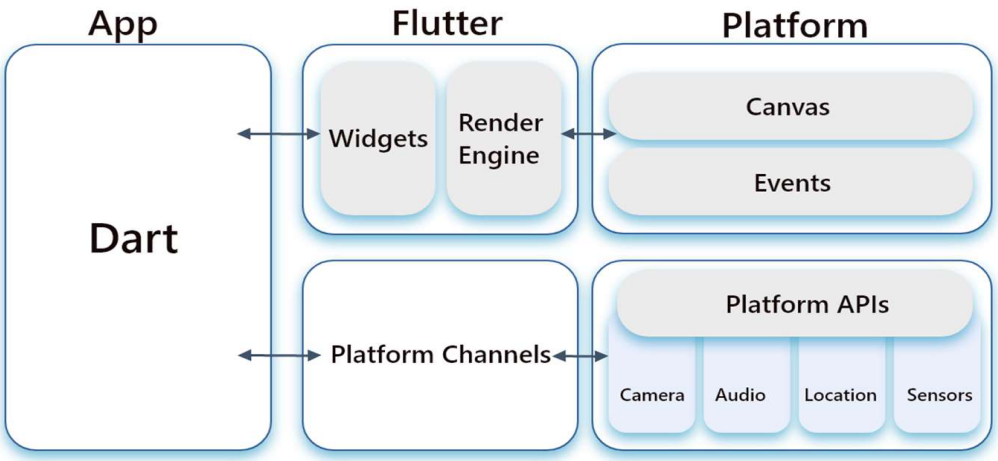


***Figure 16**: Flutter components*

4.3 Firebase

Firebase is a backend application development software that provides with the backend services such as cloud storage, Realtime database, authentication, reporting and fixing app crashes , machine learning, remote configuration, tracking analytics, and hosting for static files. It is a NoSQL cloud hosted database that stores and sync data between users in realtime and the data is stored in the form of JSON file. These are some of his key features:

- Firebase authentication, secure users accounts and let them access these accounts under authenticated credentials. It supports the authentication process by using passwords, phone number, Google, Facebook, Twitter etc.
- The realtime database that sync the data across all clients in realtime and data remain available even when the app is offline.
- It provides fast hosting for the web app. The contents are being cached into the content delivery networks worldwide.
- The applications are tested on physical and the virtual devices that are in Google's data centers.
- Notifications send through firebase does not involve any additional coding.

Firebase provides with two types of cloud-based databases that are accessible by clients and support realtime data syncing.

1. **Cloud Firestore Database**
   The Cloud Firestore is a flexible and scalable database for web, mobile and server development and lets you store your data in the cloud and sync it across all the devices or can even share them among multiple users. It has robust client libraries, full support for offline mode so that your app continues to work fine whether you are connected or not. Involve comprehensive set of security rules that help you manage access, easy to use data browsing tool and also lets you structure the data that makes sense because of its querying and fetching capabilities. It works in near real time, automatically fetch changes from database or one can request and fetch data manually.

   It supports flexible and hierarchical data structures to store data in documents and organize it into collections. Queries can be used to retrieve an individual, specific document or to retrieve all the documents in a collection. Cloud Firestore uses data synchronization to update the data on any device that is connected and can also cache the data that your app is using actively so that the app can read, write listen, and query data even if the device is offline. The figure below shows how the data is represented in Firestore *(see figure 17)*.

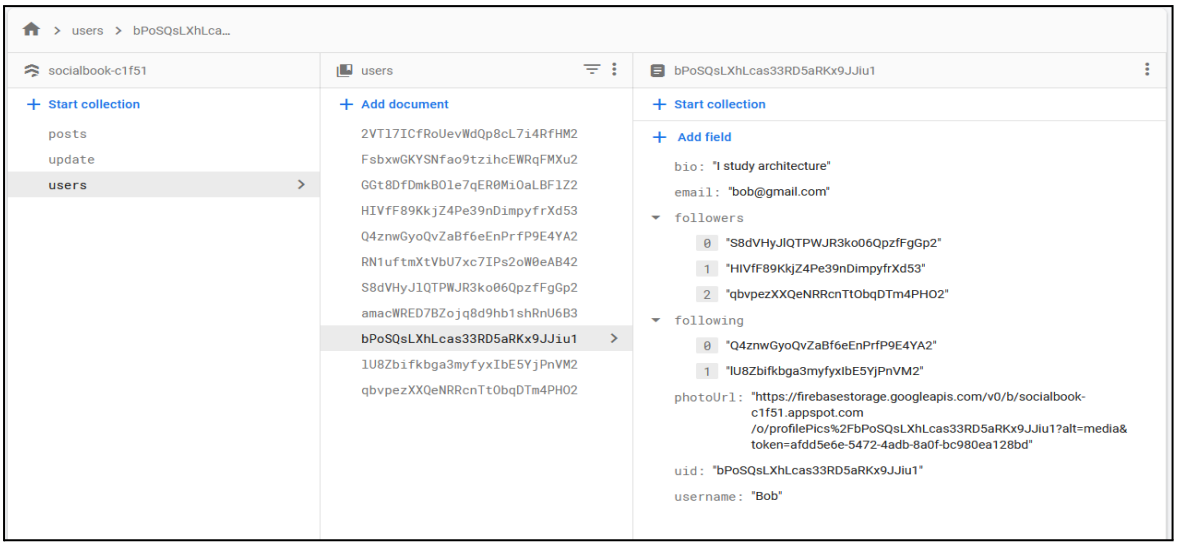


***Figure 17**: SocialBook Cloud Firestore Database*

2. Realtime Database

Realtime database allows storing and sync data between users in real time. This makes it easy for users to access the data from any device, mobile or web and helps users to collaborate with one another. Whenever the data is being updated in realtime database, it stores the data in the cloud and simultaneously notifies all the interested devices in milliseconds. It is optimized for offline use so if a loser loses connection the database SDK uses a local device to serve and store changes. So, when the user gets connected again the data will automatically be synchronized.

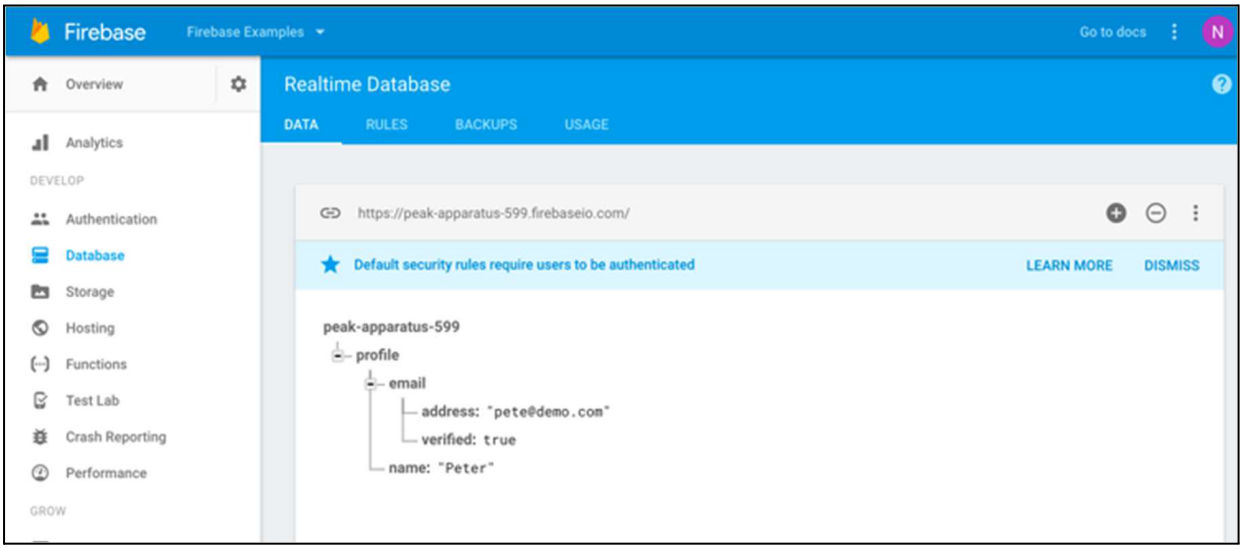


***Figure 18**: Realtime Database example*

The realtime database uses data sync. The connected devices receive the updates in seconds, every time data changes. The data validation and security remain available through realtime database security rules. The data inside it stores as a one large JSON tree *(see figure 18)*.

Chapter 5 - Implementation

This chapter begins with a tutorial explaining how flutter applications are created to align with the requirements. The most important files and folder are also briefly described. In the second part fragments of the codes are introduced and explained.

5.1 Application setup

SocialBook app is a flutter mobile application. Flutter is a Google's UI toolkit for building advanced custom UI designed applications for mobile, web and desktop. In order to install and run Flutter, the following requirements must be met for your development environment.

- Operating system: Chrome OS (64-bit) with Linux (Beta) turned on or macOS.
- Disk Space: 600 MB for windows or 2.8 GB for macOS (not including disk space for IDE/tools).
- An IDE or text editor of your choice, such as Android Studio or VS Code configured with the Dart and Flutter plugins.
- Git for Windows, for running git commands.
- The latest stable version of Flutter.
- A browser, such as Chrome.

We can browse through flutter documentation to install all the required tools for our application setup. After that, run the following command "*flutter doctor*" *(see figure 19)* to ensure that if there are any other dependencies that one need to complete the setup:

```
PS C:\Users\Lenovo\Downloads\socialbook_last-20220602T210316Z-001\socialbook_last> flutter doctor
[√] Flutter (Channel stable, 2.8.0, on Microsoft Windows [Version 10.0.22000.675], locale en-US)
```

***Figure 19**: Flutter installation command*

Flutter Doctor assesses which tools are installed on the local machine and what other software need to be installed and to be configured. It checks the current environment and then displays a report of the status of your Flutter installation or notify if any problem is encountered. In this app, everything is perfectly installed and by running flutter doctor -v command, no issues have been found *(see figure 20)*.

```
Doctor summary (to see all details, run flutter doctor -v):
[√] Flutter (Channel stable, 2.8.0, on Microsoft Windows [Version 10.0.22000.675], locale en-US)
[√] Android toolchain - develop for Android devices (Android SDK version 29.0.3)
[√] Chrome - develop for the web
[√] Android Studio (version 4.2)
[√] VS Code (version 1.63.2)
[√] Connected device (2 available)

• No issues found!
```

***Figure 20**: Flutter installation command -v*

When configuration is done, it's the time to create an app in the IDE. Visual Studio Code is the environment being used for creating flutter app. Open the command palette in VS code to create a project. Enter the desired project name *(see figure 21)* and open the created project in IDE.

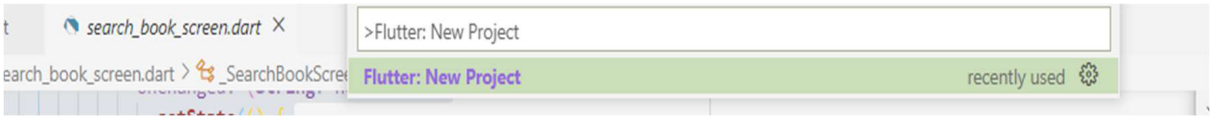


***Figure 21**: Flutter new project creation*

When a project is created , it generates a list of files and folders important to develop a project *(see figure 22)*. The structure of the project is as follows:

***Figure 22**: SocialBook files*

Lib Folder: The implemented dart code part is inside the lib folder. The lib folder of this app contains subfolders models, providers, resources, screens, utils and widgets. By default, this lib folder contains only the **main.dart** file. It's the entry point for the dart programs when a flutter app is run, it first runs main function.

Android/ IOS Folder: This folder contains the folders and files that are needed for running the application on an android operating system. The flutter code, after converting into native code, is injected into this android folder and then through this android app build is formed. So, if required to implement platform specific feature or configuration related to android, then it is done through this android folder. Same for IOS folder that contains files related to IOS app creation.

Build Folder: It contains flutter application compiled code and is generated and managed by flutter SDK. Flutter SDK makes changes inside it automatically with app development.

Test Folder: This folder is used to write test dart code that tests the flutter application.

Pubspec.yaml: This configuration file contains all the dependencies (third-party packages), related to our application, in side it. User is also able to configure any type of assets (images, fonts, mp3 files etc.) inside this file.

Firebase *(Backend for App)*:
SocialBook involves firebase as a backend to link with the flutter application. Create a project in firebase console. Then adding firebase to your app by adding package name from build.gradle file that resides in android folder with in a flutter app *(see figure 23)*.

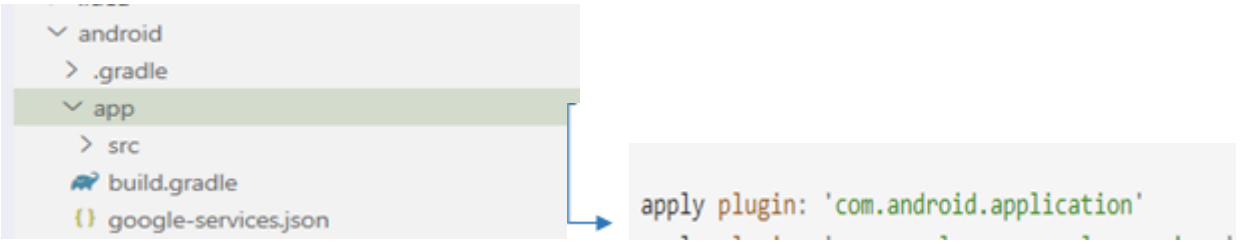


***Figure 23**: Location for build.gradle*

After registering the app, the next step is to download the *"google-services.json"* file. This file is placed inside the folder structure of flutter app. This file connects the backend to the front end of the application. Add the file into the android folder, dependencies are added in the pubspec.yaml file. The firebase auth dependency to authenticate users to the app, cloud_firestore in order to read the value of a document or a collection. The connection of firebase with the flutter app is completed with these basic configurations.

```yaml
dependencies:
  flutter:
    sdk: flutter
  cloud_firestore: ^3.1.4
  cupertino_icons: ^1.0.2
  firebase_auth: ^3.2.0
  firebase_core: ^1.10.5
  firebase_storage: ^10.2.3
  flutter_staggered_grid_view: ^0.4.1
  flutter_svg: ^1.0.0
  image_picker: ^0.8.4+4
  intl: ^0.17.0
  provider: ^6.0.1
  uuid: ^3.0.5
dependency_overrides:
  firebase_messaging_platform_interface: 3.1.6
  firebase_crashlytics_platform_interface: 3.1.13
  cloud_firestore_platform_interface: 5.4.13
  firebase_auth_platform_interface: 6.1.11
  firebase_storage_platform_interface: 4.0.14
  cloud_functions_platform_interface: 5.0.21
  firebase_analytics_platform_interface: 3.0.5
  firebase_remote_config_platform_interface: 1.0.5
  firebase_dynamic_links_platform_interface: 0.2.0+5
  firebase_performance_platform_interface: 0.1.0+5
  firebase_app_installations_platform_interface: 0.1.0+6
dev_dependencies:
  flutter_lints: ^1.0.0
  flutter_test:
    sdk: flutter
```

***Figure 24**: Project package dependencies*

Other dependencies shown in the figure above *(see figure 23)* involve firebase cloud messaging that let users exchange and deliver messages. Cloud functions dependency that run backend code automatically in the response to triggered events by HTTPS request or firebase features, a firebase crashlytics to fix crashes, storage dependency to store and serve user content like photographs, videos etc., firebase dynamic links for handling link across platforms.

5.2 Code development

Main.dart:

The main.dart file contains main function, is the first file to be executed when the user runs the flutter application. This void main function contains a runApp() method that loads the app layout. Main function uses async keyword to await for firebase.initializeApp method, which creates and initialises the Firebase app instance. A firebaseOptions is used to configure the app's services.

```
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  if (kIsWeb) {
    await Firebase.initializeApp(
      options: const FirebaseOptions(
          apiKey: "AIzaSyCP8YDr2cMnYSCAcDoJQiPzY_YCw3jg7L",
          appId: "1:1070797723393:web:d72b71883dcbeab7fced00",
          messagingSenderId: "1070797723393",
          projectId: "socialbook--c1f51",
          storageBucket: 'socialbook-c1f51.appspot.com'),
    );
  } else {
    await Firebase.initializeApp();
  }
  runApp(const MyApp());
}
```

Code fragment 1: *Main.dart main()*

As stated earlier everything in flutter app is a widget from buttons to text boxes, logos etc. Class MyApp extends the StatelessWidget. StatelessWidgets are used where the state never changes during the runtime of an application. So, the appearance and the properties remain unchanged. For example, icons, texts and icons buttons are examples of stateless widget. Inside MateriaLApp widget then introduces the theme and title of the book. The users' authentication state is provided as a snapshot to the stream builder in order to check whether the use is authenticated or not. Using the snapshot property, we can display the login screen to a user if the user is not signed in.

```
class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        ChangeNotifierProvider(
          create: (_) => UserProvider(),
        ),
      ],
      child: MaterialApp(
        debugShowCheckedModeBanner: false,
        title: 'SocialBook',
        theme: ThemeData.dark().copyWith(
```

```
          scaffoldBackgroundColor: mobileBackgroundColor,
        ),
        home: StreamBuilder(
          stream: FirebaseAuth.instance.authStateChanges(),
          builder: (context, snapshot) {
            if (snapshot.connectionState == ConnectionState.active) {
              if (snapshot.hasData) {
                return const ResponsiveLayout(
                  mobileScreenLayout: MobileScreenLayout(),
                  webScreenLayout: WebScreenLayout(),
                );
              } else if (snapshot.hasError) {
                return Center(
                  child: Text('${snapshot.error}'),
                );
              }
            }
            if (snapshot.connectionState == ConnectionState.waiting) {
              return const Center(
                child: CircularProgressIndicator(),
              );
            }
            return const LoginScreen();
          }
```

Code fragment 2: *Main.dart class MyApp*

Login interface:

The packages placed in pubspec.yaml file are imported in different class files to use them while implementing the app logic. The first screen visible to a user of the app is the login screen. The class loginScreen extends the statefulWidget as stateful widgets are dynamic and, based on user actions, are updated during runtime. TextEditingController for both password and email text fields as it is for an editable text field. Whenever a user modifies text fields, the text fields updates and controller notify it to the listeners. A void Dispose() method is used to release the memory allocated to the variables when the state object is removed. Inside loginUser(), setState() is used to notify the framework that the internal state of the object is changed. Authmethod to check if user enters verified credentials and is not left empty, then they will be navigated to another screen by using MaterialPageRoute that helps to give animation when transitioning between the two pages occur. It takes in the builder, and one needs to provide context and the screen on which the user will be going to navigate upon after logging in.

```
class LoginScreen extends StatefulWidget {
  const LoginScreen({Key? key}) : super(key: key);

  @override
  _LoginScreenState createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {
  final TextEditingController _emailController = TextEditingController();
  final TextEditingController _passwordController = TextEditingController();
  bool _isLoading = false;
  @override
```

```
  void dispose() {
    super.dispose();
    _emailController.dispose();
    _passwordController.dispose();
  }
  void loginUser() async {
    setState(() {
      _isLoading = true;
    });
    String res = await AuthMethods().loginUser(
        email: _emailController.text, password: _passwordController.text);
    if (res == 'success') {
      Navigator.of(context).pushAndRemoveUntil(
          MaterialPageRoute(
            builder: (context) => const ResponsiveLayout(
              mobileScreenLayout: MobileScreenLayout(),
              webScreenLayout: WebScreenLayout(),
            ),
          )
```

Code fragment 3: *login_screen.dart class LoginScreen*

The build (BuildContext context) describes the part of the user interface represented by this widget. This widget contains scaffold, a widget that is used to implement the basic material design visual layout structure. Inside it are containers that carry a logo of the app, then text boxes for email and password. After that a login button. Don't have text is added after login button that navigates user towards Signup screen in case user is not registered in the app.

```
Widget build(BuildContext context) {
    return Scaffold(
      resizeToAvoidBottomInset: false,
      body: SafeArea(
        child: Container(
              Image.asset(
                'assets/socialbooklogo.png',
                height: 130,
              ),
              const SizedBox(
                height: 64,
              ),
              TextFieldInput(
                hintText: 'Enter your email',
                textInputType: TextInputType.emailAddress,
                textEditingController: _emailController,
              ),
              const SizedBox(
                height: 24,
              ),
              TextFieldInput(
                hintText: 'Enter your password',
                textInputType: TextInputType.text,
                textEditingController: _passwordController,
                isPass: true,
              ),
              const SizedBox(
                height: 24,
              ),
```

```
            InkWell(
              child: Container(
                child: !_isLoading
                    ? const Text(
                        'Log in',
                      )
                    : const CircularProgressIndicator(
                        color: primaryColor,
                      )
              onTap: loginUser,
            ),
            const SizedBox(
              height: 12,
            ),
            Flexible(
              child: Container(),
              flex: 2,
            ),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Container(
                  child: const Text(
                    'Dont have an account?',
                  ),
                  )
```

Code fragment 4: *login_screen.dart widget BuildContext*

Sing up interface:

The class signup screen extends Statefulwidget. It contains TextEditingController so that if you modify the [text] or [selection] properties, the text field will be notified and will update itself appropriately. Inside UinBlist image file is being stored. Authmethods performs authentication and are responsible for assigning identity and a set of policies to a user. If the credentials provided are accepted by the application, then the user account has been created. This navigates the user towards the profile screen or else display a pop-up error message on screen using ShowSnackBar widget.

```
    String res = await AuthMethods().signUpUser(
        email: _emailController.text,
        password: _passwordController.text,
        username: _usernameController.text,
        bio: _bioController.text,
        file: _image!);
    if (res == "success") {
      setState(() {
        _isLoading = false;
      });
      Navigator.of(context).pushReplacement(
        MaterialPageRoute(
          builder: (context) => const ResponsiveLayout(
            mobileScreenLayout: MobileScreenLayout(),
            webScreenLayout: WebScreenLayout(),
          ),
        ),
      );
    } else { //Loading to false
```

```
      setState(() {
        _isLoading = false;
      });
      // show the error
      showSnackBar(context, res);
    }
  }
```

Code fragment 5: *signup_screen.dart Authmethods()*

SelectImage method is created to set state in order to display the image selected from gallery into the circle avatar. The buildwidget contains the interface layout that will be shown on the user interface to the user to enter their required credentials into the fields. Container class is used to position different widgets' on the screen according to the convenience. For signup, it involves a circle avatar that contains an icon button. On pressing will direct user to select a picture from gallery. Text fields for username, email, password and for user bio. Inside InkWell widget Signup text is placed. InkWell is a rectangle area that responds to the touch events performed by users. If user already have an account, the user will be navigated to the log in screen.

Profile Screen:

The profile screen is the main landing screen after logging into the application. The ProfileScreenState involves declaring the datatypes of the attributes that are visible on screen, like followers and following. Get the data of the user from firebaseFirestore with a collection name *"user"*. Also get the books from *"posts"* collection in Firestore to display on the profile page of user. The build widget holds the design layout of the profile page. CircularProgressIndicator(), a small animating circular icon, will be displayed first in case if data is loading. After that, in scaffold title that includes the username along with the circular avatar is placed on the profile screen. In column, followers, books and following text is placed along with the numerical data loaded above the texts. The follow button will change into a sign-out button if the user is on his/her own profile.

Also check the availability of the book. If book available, then it will be having green borders, else red.

```
child: Container(
      decoration: BoxDecoration(
                border: Border.all(color: snap['isBookAvailable'] == true ?
Colors.green : Colors.red, width: 3)
```

Code fragment 6: *profile_screen.dart BoxDecoration()*

Feed Screen:

Class feedScreen contain variables with different keywords. MediaQuery is used in order to retrieve the width of the screen. Querying the current media using MediaQuery.of will cause your widget to rebuild automatically whenever the MediaQueryData changes (e.g., if the user rotates their device).

```
  Widget build(BuildContext context) {
    final width = MediaQuery.of(context).size.width;
```

Code fragment 7: *feed_screen.dart MediaQuery*

Scaffold implements a basic material design visual layout structure. Appbar is used to display the title above. Streambuilders allow apps to display the async data. It will navigate the way to the collection from where you want to read data from, like in this app to read data from posts collection to display that collection on the feed screen and then calling snapshots at the end that will grab the relevant posts found in Firestore. In case data not returned of is still loading, a CircularProgressIndicator() is displayed on screen. After receiving the data, use ListView to build the list of documents present in the snapshot documents list.

Search Screen:

The search bar, present on the top of the feed screen, helps to search for a user. Firestore returns the data of all the users that match our typed search name or related to it. Listview will build a list with each user having their specific profile picture and username. Tapping on any of the listed users present on screen will lead to their profile screen, where the user can see other books posted by them as well.

Search Book Screen:

Search Book screen offers the users of the application to find the books posted by other users. The theme of the book dropdownlist is created at first, and the initial value is set to "*Ignore Theme*". If nothing is filled in the title text field and theme is not selected, then the search won't happen. SnackBar will show a message on the screen as 'please fill at least one of the fields.'

If the user only wants to search through the title, then the Firestore searches the book with the title entered by the user and returns it to it if found. And if the user searches with title and added theme of the book too then will get the book from that theme user selected. Else SnackBar will display a message *"No matching results"*.

```
if (theme == 'Ignore Theme') {
        snap = await FirebaseFirestore.instance
            .collection('posts')
            .where(
              'title_lower',
              isGreaterThanOrEqualTo: _titleController.text.toLowerCase(),
            )
            .get();
        print(snap.runtimeType);
      } else {
        snap = await FirebaseFirestore.instance
            .collection('posts')
            .where('theme', isEqualTo: theme)
            .where(
              'title_lower',
              isGreaterThanOrEqualTo: _titleController.text
                  .toLowerCase(),
            )
            .get();
        print(snap.runtimeType);
      }

      setState(() {
        isLoading = false;
      });
      if (snap.docs.length == 0) {
```

```
        return showSnackBar(context, 'No matching results');
```

Code fragment 8: search_book_screen.dart SnackBar

The title text field and a DropDownMenuItem will be visible to the users on their screen. On tapping the search button, a CircularProgressIndicator will be shown on screen, so the user can wait for the required data to be loaded on to their screens.

Add_Post Screen:

Class AddPostScreen first shows a popup to the screen. SimpleDialogOption let users have a choice between several options. If user selects *'Take a photo'* then will be able to take picture from gallery, else can choose from gallery or cancel the operation. SimpleDialogOption widget will call onPressed callback that uses Navigator.pop to close the dialog.

```
            SimpleDialogOption(
              padding: const EdgeInsets.all(20),
              child: const Text("Cancel"),
              onPressed: () {
                Navigator.pop(context);
              }
```

Code fragment 9: add_post_screen.dart SimpleDialogOption

The Build widget involves the User interface display fields. Write a description, title, the dropdownbuttons for book format, availability, theme of the book. On pressing the post button, the book details entered will be posted. Title and description of the book must be fulfilled by the user in order to post a book.

Comments Screen:

This screen involves the comment posted by different users on the Book post. Scaffold method contains the title *'comments'* on top of the page. Then using streamBuilder that helps in managing the stream's state. When performing the query, Firestore returns a QuerySnapshot that gives access to the documents in the collection and returns to the screen the book title comment that posted on it, date on which it is published. On tapping post text button after entering the comment in the comment field. The comment will be posted along with the username, profile picture and the comment they wrote on a particular book post.

Comments Update Screen:

This screen contains the comments posted by user on different books. Fetches details from Firestore and displays it on the screen. The title of the book on which the comment was made, comments text, and the last updated date on which the comment was posted.

Chapter 6 - Testing

This chapter is divided in two parts. The first one revolves around the feedback data gathering process and the decisions taken accordingly. The second part consist of a series of tests for each screen and the results yielded.

6.1 Users feedback

The SocialBook application has been evaluated by different real users through various stages of the development, using an agile methodology. They have been looking for any changes required in user interface elements, in usability, functionality, and the design, to improve the app experience. The user testing conducted was with the help of a small sample group of peers of different ages and educations. The participants had to test SocialBook on numerous occasion and answer a *Google form*, their feedback provided me with valuable insight into where my application needs to improve. Each user had different devices to check flow across multiple devices. Following are the main functionalities for which testing was carried out:

- Register and Login to the application
- Create a post
- Search posted book
- View details of posts
- Comment/reply on post
- Check availability
- Log out and close app

Also, the testing is done based on user experience with the interface design in order to know whether the created application meet the expectations of user and how it looks, feel and the ease of use from user point of view. Following are some elements of interface that users were advised to into look during testing phase:

- Input controls involving buttons, text fields, dropdown list and list boxes.
- The navigational components that include navigation bar at the bottom of the page, navigation icons, menu icons, search field for book search and user search by name.
- Pop-ups, modal for post creation, progress bar, message boxes.

After getting the required feedback from the users and their experience with the application, I got to learn many things. Their perspectives on the application interface and its functionalities have been very valuable. Following are some suggestions by the user towards SocialBook:

- Users want to search book not only by name but can also be able to add the theme of the book from which category that book belongs so that the search could be more efficient for them to access their desired book.

- Users want to delete their posted books and edit the details.

- Users want to have an option where they can follow/unfollow other users in order to feel more secure.

- Users want to select the posted book image not just from the gallery but also be able to open up their camera and take an image of the book to post it.

- Users would like to view all of their comments that are posted on different book on one page and not have to check each book one by one.

- User want to be able to check if a book is available or not in a faster way, rather than clicking on each one.

- User would like to have a sorting function in the profiles for arranging books in various ways, such as new/old or pdf/book.

All this feedback and more was collected and carefully analysed. Afterwards, it was considered if those changes were going to be added SocialBook or not. Following are some changes and adjustments that are made in this final version of the application:

- The search option is updated where users can not only search by book name but can select book theme too for more precise results being shown to them.
- A search bar on the top f the feed page is created so that the user will be able to find the publisher of the book if they know their username.
- The user who posted the book will be able to delete their respective book from their profiles.
- Users will be able to change the status of their posted books. If the book they posted is acquired by other user they updated the status of the book to be reserved. If book is reserved or in use, the corner frame on the image will turn red else green if book is available.
- A choice will appear before users before posting a book, to select image from gallery or take picture from their camera, otherwise there is also the option to cancel the action.
- A comments update page is created for users to view their own comments they posted on different books that are clickable and navigate them to that particular book's comments where they posted it and will be able to view any update regarding their query or whatever they asked in comment.

6.2 Testing screens

In the following tables numerous tests are performed for each individual screen to check if the expected result of the action is the desired on. The intention is to ensure that SocialBook is performing as it was intended and to know if there is any bug or error that can be resolved.

TEST	EXPECTED RESULTS
Verify if the login page contains necessary input fields, login button and sign-up text button i.e., in case not registered	All the necessary input fields i.e., email and password fields are present along with login button and sign-up text button.
Verify that if user can enter correct credentials, i.e., valid email and password in the text field.	User login to the application, landed on profile page.
Verify user navigates to signup page when selecting *"don't have an account? Signup"*	User successfully landed on the signup page.
Verify if the user can log in with incorrect email and valid password.	An error message appears *"enter valid username or password*". User cannot log in to the system.
Verify if user can enter into the system using valid email and incorrect password.	An error message appears *"enter valid username or password"*. User cannot log in to the system.
Verify if user can log in using invalid password and email.	An error message will appear *"enter valid username or password"*. User cannot log in to the system.
Verify if user can click on *"don't have an account? Signup"*	User landed on register of signup screen to register the credentials to create an account.
Verify if login button working correctly after pressing it.	With correct credentials entered and pressing login button will land the user on main screen i.e., profile screen

Verify that the validation messages appear or displayed in case user left password of email field empty	The system will generate an error message *"enter required credentials"*.
Verify that the helping text is placed in the text fields.	The system displays the helping text in the text fields before entering credentials in it.

***Table 5**: Login screen*

TEST	EXPECTED RESULTS
Verify that if user can add profile picture in circular avatar.	Picture successfully added in the avatar.
Verify that if user can register into the application if left the text fields empty and click on register button.	An error message appears *"please fill the required fields to continue"*. User is not directed to login screen.
Verify if user enter valid credentials and press registration button.	User landed on login screen.
Verify to check email test field by entering email without @ symbol.	A validation message appears "*please fill valid email".*
Verify if user can enter email that is already registered.	An error message appears "email already registered".
Verify to check the password limit when enter value less than min	A validation message appears *"password should be 8 characters long".*
Verify by adding spaces in the email and password fields.	A validation message appears "enter valid details".
Verify if user can register without adding bio.	An error message appears *"please fill the required fields to continue".*
Verify if user can land to login page using text button *"Already have an account? Login"*	User successfully landed on the login page.
Verify if signup button working correctly after pressing it.	Signup button successfully lands the user to the login screen on pressing i.e., if correct credentials are entered.

***Table 6**: Registration screen*

TEST	EXPECTED RESULTS
Verify to check if basic profile information i.e., user name, profile picture, bio, sign out button are visible to user.	Profile information is visible to user.
Verify if the number of the books posted by the user, number of followers and following are displayed on page.	Number of books, followers and following are visible to user.
Verify after uploading books the number of books updated on the profile page.	The number of books updated on profile screen.
Verify if the book posted is added to the profile screen of user.	The book posted by user is successfully added in the photo grid on profile page.
Verify if user can land on the profile screen by clicking on profile icon in the navigation bar	The user successfully navigates to the profile screen.
Verify if user can land on the feed screen by clicking on search icon in the navigation bar	The user successfully navigates to the feed screen.
Verify if user can land on the create a post screen by clicking on + icon in the navigation bar	The user successfully navigates to the post screen.
Verify if user can land on the comment update screen by clicking on processing icon in the navigation bar	The user successfully navigates to the comments update screen.
Verify if the corner frame of the image is shaded red in case reserved.	The book with red shaded corners is visible to user as being reserved.
Verify if the corner frame of the image is shaded green in case available.	The book with green shaded corners is visible to user as being available.
Verify if sign out button working correctly after pressing it.	Sign out button successfully lands the user to login screen on pressing.

***Table 7**: Profile screen*

TEST	EXPECTED RESULTS
Verify if the search placeholder is functional and generating the correct results after entering username.	Successfully generated list of users with the same names.
Verify if an error message is displayed by entering invalid keywords in the search field.	An error message *"enter valid credentials"* is displayed on the screen.
Verify if by clicking on the book icon on the top right corner, navigates user to the book search page.	User is taken to the book search screen.
Verify a list of users is being shown when username is entered.	A list of users are displayed when that username exists.
Verify if feed screen displays the posted books to users.	All the posted books are successfully displayed to the users.
Verify by clicking on any of the posted book leads to that book details page.	User is successfully taken to the detailed book page screen.

***Table 8**: Feed screen*

TEST	EXPECTED RESULTS
Verify that if user can enter correct credentials i.e., add book title and select theme of the book.	User successfully gets the desired results.
Verify if an error message is displayed if one of the fields is left empty.	An error message *"please enter details to proceed"* is displayed on the screen.
Verify if search button is working correctly after entering details.	Results for the searched book will be displayed on screen.
Verify if the back arrow icon on the top left corner navigates the user back to feed page.	User is redirected to the feed page.
Verify if the profile icon on the top right corner navigates the user back to profile page.	User is redirected to the profile page.
Verify if dropdown menu for selecting theme of the book is working properly.	Dropdown successfully displays a list of themes to select from them.

***Table 9**: Search book screen*

TEST	EXPECTED RESULTS
Verify if a form appears on screen when + icon is clicked on the bottom navigation bar.	Create a post form appears after the user is navigated to this screen.
Verify if user will be able to choose from the given options in the form.	User successfully choose options from form.
Verify if user can cancel action by clicking on cancel text button in the form.	Action is successfully cancelled.
Verify if user landed to *"post to"* screen after selecting an image.	User successfully landed on *"post to"* screen after adding an image.
Verify if user left the title empty and click on post button.	An error message popped on screen *"enter all necessary details".*
Verify if the by default selected value is shown on the dropdown or not.	The default value for dropdown is visible to user.
Verify if the dropdown values can be selected by clicking on them.	The user successfully selects a value from dropdown after clicking on it.
Verify on clicking on the dropdown it displays the list containing values.	A list is displayed too user after clicking on the dropdown.
Verify if the post button on the top right corner is clickable or not.	The post button generated message *"book is being posted"* when clicked i.e., the post button is working.

***Table 10**: Create a post screen*

TEST	EXPECTED RESULTS
Verify if a book name is present on top of the screen.	A book name is displayed on the top of page.
Verify if user profile picture and name is displayed.	User is able to view the name and profile picture of the user who posted the book.
Verify if the menu icon i.e., the three-dot icon is functional or not.	By clicking on icon, a pop list is shown on the screen.
Verify if use can delete a book posted by his/her by selecting delete option from popup list by pressing menu icon.	User successfully deleted the book.
Verify if user will be able to update the status of the book.	User successfully updated the status of the book to available.
Verify by clicking on comment icon, user is navigated to comments screen.	User landed on the comments screen where user can post a comment.
Verify if the description book title and its availability details are visible to user or not.	All of the details of book present below the book picture are visible to user.
Verify if the date on which the book was posted is displayed at the end of the book description.	The date on which the book is posted is displayed and is visible to user.

***Table 11**: Book detail screen*

TEST	EXPECTED RESULTS
Verify if all the posted comments are displayed on the screen.	All the posted comments are being displayed on screen and are visible to user.
Verify if the comment detail box is clickable.	By clicking on the comment box user will be navigated to the comment section of book on which the comment was posted.

***Table 12**: Comment update screen*

Chapter 7 - Monitoring and control

This chapter reviews the processes followed during the project and how they have been managed. It analyses the scope management and the main changes it has undergone during the development with respect to the original version. The actual number of hours dedicated and the tasks to which they have been associated with are also included.

7.1 Scope management

At the beginning of the project an initial scope was established, which has varied during the development of the project. This is partially due to the fact that I did not have the knowledge to use Flutter and Firebase and also because new use cases were implemented, so the estimated times at the beginning did not have a sufficiently firm basis.

The main objective and result of this project was to create an application that would serve as a gateway to a social network for book sharing among students. In the current state SocialBook is working and scaling properly and therefore that aspect of the target has been accomplished. The minimum requirements stated at the beginning *(see section 2.4)* were met and further some have been upgraded, such as the comment section with the comment page and reply use case.

7.2 Time management

During the development of the project the hours dedicated to each task, as defined in the Work Breakdown Structure *(see figure 1),* have been recorded and accounted. The result of this can be visualized in the following table *(see table 13)*.

The main deviation from the initial estimation can be seen in the implementation phase, where 28 more additional hours were committed. This change comes from the expansion of the original scope and from issues that came along during the development. Having to incorporate and alter some of the initial uses cases while still not having enough experience with flutter was the main issue of this increment.

Preparation of the thesis and defence has also seen an increment, fortunately not in such an impactful manner.

Code	Tasks	Estimated (h)	Actual (h)	Deviation (h)
1	**Management**			
1.1	Scope statement	3	3	0
1.2	Planification	6	6	0
1.3	Monitoring and control	9	11	2
1.4	Requirements	2	2	0
		20	**22**	**2**
2	**Product**			
2.1	Research	**10**	**10**	**0**
2.2	Technologies	**25**	**28**	**3**
2.2.1	Flutter	18	20	2
2.2.2	Firebase	7	8	1
2.3	Architecture	**55**	**54**	**-1**
2.3.1	Database	20	17	-3
2.3.2	Design	35	37	2
2.4	Implementation	**90**	**118**	**28**
2.4.1	Set up	5	8	3
2.4.2	Interfaces	85	110	25
2.5	Testing	**15**	**17**	**3**
2.5.1	User experience	12	12	0
2.5.2	Use cases	3	6	3
		195	**227**	**32**
3	**Documentation**			
3.1	Thesis	75	80	5
3.2	Defense	10	15	5
		85	**95**	**10**
	Total	**300**	**344**	**44**

Table 13: *Estimation actual and deviation table for each task*

Chapter 8 - Conclusions

This is the last chapter and all the work has been done and being that the case, here I am going to share my assessment of the project. I will evaluate the work that has been done, the knowledge I have acquired and explain what my vision of the future of SocialBook should be.

8.1 Lessons learned

Working on a project of this scale has taught me many lessons about developing native apps. It is true that at the learning curve might be a little stiff since there is a lot to tackle, but it gets much better once you begin using some key features like:

- Hot-Reload, which was very helpful, I could see immediately what the code I was trying translated to.
- Built in Widgets, a vast array of widgets that are available for all types of uses , they helped me make great progress with just a few lines of code.
- Firebase functions, were also a time saver and very interesting to learn about.

Flutter alone has definitely great potential, but once you add the seamless integration it offers with Firebase it becomes a wonderful developing environment. Although, there is one thing I found that did give me quite the trouble. That is the constant update process, they indeed do bring a lot of good features with every new update, but they also make old packages and widgets obsolete and the migration process was sometimes quite difficult. I have spent many hours fighting with the deprecated version errors.

Overall I do think Flutter updates are worthwhile, especially since its last 3.0 update, where now with just coding once you can compile the code even for macOS and Linux.

8.2 Personal evaluation

The begging of the project was quite challenging. I had previous knowledge of Android and Kotlin, since I studied them in the course of Software Engineering II, where we built a few simple applications. Human-Computer Interaction had also taught me useful things like how to design and focus on the user's experience. I had also taken three subjects on databases and felt quite comfortable on my skills of dealing with data. Other subjects such as Web Systems and Software Quality has as well helped me feel confident in approaching to build a social network application.

However, even thought I had also this experience, the start was quite overwhelming. Having to learn a new framework from scratch, using a new type of database model, planning ahead a huge project and much more was something I found quite challenging.

Good thing was that all this was just a short-lived initial phase. Once I started taking the first steps with SocialBook all begun to feel more natural and clear. Flutter and Dart language were reasonably easy to take a grasp of, after only a short 10-hour course I was able to create my first Flutter mini programs and begun to familiarize with the environment.

Completing this project has taken me a lot of time, and during that I have seen how flutter has evolved and become a very popular tool for mobile app development. By seeing this evolution I was able to try out and learn much more than I initially expected, for which I am grateful.

8.3 Future work

SocialBook development stops at this stage for now, but from all the feedback I have received and from seeing how the project has performing I have prepared this list of features that I think would being value to the purpose of the social-network:

1. With longevity in mind, since people will upload more and more books, one useful feature would be to be able to sort in different manners the books that are uploaded on each user's page, not only for the one that uploads them but also for people that are checking others profiles.

2. Regarding scalability, a good thing to incorporate would be a report system for inappropriate content uploaded and for misbehaving users.

3. Add different type of user roles, for example a teacher, this way students will know that they can trust that user. A community type user account could also be an interesting addition, for example a user for an entire classroom.

Bibliography

1. Official Flutter documentation:
   https://docs.flutter.dev/

2. Official Firebase documentation:
   https://firebase.google.com/docs

3. Goodreads - Book cataloging website
   https://www.goodreads.com/

4. The story graph - Book tracking website
   https://thestorygraph.com/

5. Book trib - Online literary books
   https://booktrib.com/

6. Native apps
   https://magenest.com/en/native-app-example/

7. Hybrid apps
   https://magenest.com/en/hybrid-app-examples/

8. Cross Platform apps
   https://www.netsolutions.com/insights/cross-platform-app-frameworks-in-2019/

9. Progressive web apps
   https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

10. A complete guide to Flutter architecture
    https://blog.logrocket.com/complete-guide-flutter-architecture/

11. Official Dart documentation:
    https://dart.dev/

12. Choose a Database: Cloud Firestore or Realtime Database
    https://firebase.google.com/docs/database/rtdb-vs-firestore

13. Monitoring your Flutter app's stability with Firebase Crashlytics
https://www.youtube.com/watch?v=cIFLFpKTy7c

14. Flutter - Stateful vs Stateless Widgets
https://www.geeksforgeeks.org/flutter-stateful-vs-stateless-widgets/

15. Exploring StreamBuilder In Flutter
https://medium.flutterdevs.com/exploring-streambuilder-in-flutter-5958381bca67

16. Flutter 3.0 - What's New In Flutter
https://medium.flutterdevs.com/flutter-3-0-whats-new-in-flutter-12259bf090ba