CineLibra

Software Design Specification

17.05.2024

Mehmet Toprak Balıkçı 150121032
Tolga Fehmioğlu 150120022
Enes Torluoğlu 150121002
Muhammed Enes Gökdeniz 150121538

Prepared for
CSE3044 Software Engineering Term Project

Table of Contents

1. Introduction

This Software Design Specification (SDS) document is a guide detailing the architecture and design considerations for the development of CineLibra, a mobile application for managing and tracking movies and books. Building upon the foundation laid out in the Software

Requirements Specification (SRS) document, this SDS document focuses on the constraints of the software's architecture, design principles, and implementation strategies.

1.1. Purpose

The primary purpose of this SDS document is to provide an in-depth exploration of the software architecture and design choices of the CineLibra application. It aims to explain the structural components, system behaviors, and interaction patterns to define the software's design.

1.2. Statement of scope

The scope of the CineLibra software is the following functionalities, categorized based on their importance within the time frame of this semester:

Essential Requirements:

Authentication: Implementing user authentication using Firebase to ensure secure access to the application.

Basic Functionality: Allowing users to add movies and books to their favorites, mark them as watched or to watch later, and provide personal ratings for items.

API Integration: Utilizing API to fetch information about popular, trending, and newly released movies and books, ensuring up-to-date content availability.

Desirable Requirements:

User Interaction: Enabling users to comment on movies and books, fostering community engagement and discussion.

Global Rating: Implementing a system to display the average rating of movies and books, providing users with valuable insights from the community.

Basic Recommendation System: Developing a basic recommendation system, not AI-driven, to suggest movies and books based on user preferences.

Future Requirements:

Advanced Recommendation System: Integration of a more sophisticated recommendation system powered by AI algorithms, enhancing user experience and personalization.

The prioritization of these functionalities reflects a balance between meeting essential user needs, enhancing user engagement with desirable features, and considering future advancements to enrich the application experience.

1.3. Software context

CineLibra exists within the realm of entertainment and media management, catering to users' needs for efficiently organizing and tracking their movie and book preferences.

Positioned within the competitive landscape of mobile applications, CineLibra aims to offer a comprehensive platform that integrates both movies and books.

Business/Product Line Context:

Digital Media Management: In an era dominated by digital media consumption, CineLibra addresses the growing demand for tools that streamline the organization and tracking of media preferences.

User-Centric Approach: The design and development of CineLibra prioritizes user experience and satisfaction, aiming to provide a user-friendly UI and best UX possible with valuable features that meet users' needs and preferences.

Community Engagement: By incorporating features such as user comments, ratings, and recommendations, CineLibra aims to foster a vibrant and engaged community of movie and book enthusiasts.

Innovation and Differentiation: Continual innovation and differentiation through unique features, personalized recommendations, and community engagement initiatives are key strategies for positioning CineLibra as a compelling alternative within the competitive landscape.

Strategic Considerations:

Established Players: The presence of well-known brands such as Letterboxd and IMDb signifies a competitive marketplace with established platforms that already offer comprehensive solutions for managing movie preferences and tracking viewing habits.

Unique Value Proposition: Despite the presence of competitors, CineLibra distinguishes itself by offering a unified platform that integrates movie and book management functionalities, catering to users with diverse media preferences.

1.4. Major constraints

Time Limitations: The project should finish in a semester timeline, needs efficient allocation of resources and prioritization of tasks to finish it.

Budget Limitations: The development budget for CineLibra is zero, so only free to use APIs and resources are going to be used in the project.

Platform Compatibility: CineLibra must be compatible with various operating systems, like android and iOS, imposing constraints on the selection of development frameworks and technologies. And should be tested to work on both operating systems.

API Integration: The integration of external APIs, such as the TMDB API for movie data, introduces dependencies and constraints related to API availability, functionality, and usage limitations.

1.5. Definitions

CineLibra: The mobile application designed to manage and track movies and books, providing users with features such as adding favorites, marking as watched, and personal rating functionalities.

Firebase: Google's mobile and web application development platform used for user authentication and database management in the CineLibra application.

TMDB (The Movie Database): An external API used to fetch information about movies, including popular, trending, and now playing titles, to ensure up-to-date content availability in CineLibra.

API (Application Programming Interface): Interface that allows different software applications to communicate and interact with each other, facilitating the integration of external services such as TMDB into the CineLibra application.

UI (User Interface): The graphical interface through which users interact with the CineLibra application, including screens, menus, buttons, and other visual elements.

UX (User Experience): The overall experience and satisfaction users derive from using the CineLibra application, encompassing factors such as ease of use, efficiency, and enjoyment.

Personal Rating: A user-specific rating assigned to movies and books within CineLibra, visible only to the user who assigned the rating, to capture individual preferences and opinions.

Global Rating: The average rating assigned to movies and books by all users of CineLibra, providing a collective measure of the community's opinion on specific items.

Recommendation System: A feature of CineLibra that suggests movies and books to users based on their preferences, viewing history, and community ratings, aimed at enhancing user engagement and content discovery.

1.6. Acronyms and Abbreviations

API: Application Programming Interface

SRS: Software Requirements Specification

SDS: Software Design Specification

TMDB: The Movie Database (API used for movie data)

UI: User Interface

UX: User Experience

1.7. References

CineLibra Software Requirements Specification Document
https://docs.google.com/document/d/1TZftvomkRG5TG840li6TIJDRCbEBsDbU

CineLibra UML Diagram

https://drive.google.com/file/d/16xwC_gtwKXlJcPuDSeZSD_4rILGhidcR/view

CineLibra Wireframe

https://lucid.app/lucidspark/9e534edc-06b7-4827-bc4a-4d67ab1e6699/edit?invitationId=inv_8d989321-ab0d-422e-a206-9eeb942f3a1c&page=0_0#

2. Design Consideration

2.1. Design Assumptions and Dependencies

Describe any assumptions or dependencies regarding the software and its use. These may concern such issues as:

- Related software or hardware
- Operating systems
- End-user characteristics
- Possible and/or probable changes in functionality

- **1-Related Software or Hardware**
  Assumption: CineLibra will integrate with movie and book API to fetch up-to-date datas
  Dependency: Compatibility with mobile devices for accessing ios or android platforms. Integration with databases or APIs should be reliable and well-maintained.

- **2-Operating Systems**

Assumption: CineLibra will be accessible on web browsers (Windows, macOS, Linux) and mobile platforms (iOS, Android).

Dependency:compatibility testing across mobile devices is crucial to understand how much the system is reliable

- **3-End-user characteristics**
  Users will have varying levels of movie/book knowledge and preferences.
- **4-Possible And/Or Probable Changes in functionality**
  Since our time is limited with a given amount of time, we decided to make basic functionalities that are required for this kind of application.In future the functionalities can be specialized and added new ones to enhance better user experiences.
  Possible changes can be listed as;
  Social features**: (**to provide more friendly user interface such as discussion section, follow profiles and creating an interaction between users by chat environment)
  User-Generated Content: Enable users to contribute their own content, such as movie reviews, ratings, and lists. This could enhance community engagement and provide valuable insights for other users.
  Probable changes can be listed as;

Interactive features: Offering an enjoyable and exciting environment for its specific users such as allowing them to challenge some polls/quizes related to their interests, creating a rating user list based on their success, and some daily/weekly activities to catch users' attention regularly.
Integration with Streaming Platforms: Partner with streaming services to provide compatible integration, allowing users to watch recommended movies directly within the CineLibra platform. This could enhance user convenience and make the app preferable for stakeholders (movie-makers-book writers) to be used.

2.2. General Constraints

- **1-Hardware or software environment:**
  The platform must be compatible with a wide range of devices supporting ios or android devices and maybe can be expected to be compatible with desktops by simulators or web browsers to be able to check the system's behavior simultaneously.

- **2-End-user environment:**
  Since this is a portable platform many users across the world may access the system by just network connectivity. For users, CineLibra manages a user friendly environment in terms of user's preferences and expects users to have well known subjects they are interested in to increase the interaction.

- **3-Availability or volatility of resources:**
  Since we use api services for fetching the movie or book data, in the future the usage of it can affect our system. Besides, the usage of firebase can be updated in near future so it's important to know that from the point of maintainability, the system should be opened to new updates or features dynamically.

- **4-Standards compliance**
  It's expected that the user interface is well designed for users and it should be adaptable to similar systems. After all, the system should obey some basic non-functional criterias(in terms of performance) to be able to compete with other well known systems.

- **5-Interoperability requirements**
  API Design: CineLibra needs well-defined APIs (Application Programming Interfaces) that adhere to industry standards and best practices. These APIs should use common protocols like HTTP/HTTPS and support popular data formats such as JSON or XML
  Compatibility with Industry Standards: CineLibra should comply with relevant industry standards and specifications to ensure compatibility with other systems.
  Authentication and Authorization: Firebase Authentication provides a robust and secure identity verification system, making it easier to implement authentication and authorization for CineLibra's APIs.
  Documentation and Testing: CineLibra should provide comprehensive documentation for its APIs, including usage instructions, data models, and error codes.

- **6-Interface/protocol requirements**

HTTP/HTTPS Protocol: CineLibra's APIs should be accessible over standard HTTP/HTTPS protocols, allowing clients to communicate with the server using RESTful API calls

JSON (JavaScript Object Notation): CineLibra's APIs should use JSON as the primary data interchange format for request and response payloads.

Error Handling: CineLibra's APIs should define standardized error formats and status codes to communicate error conditions to clients effectively.

Versioning: CineLibra's APIs should support versioning to ensure backward compatibility and smooth evolution of the API over time.

Caching Mechanisms: CineLibra's APIs may implement caching mechanisms, such as HTTP caching headers or in-memory caching, to improve performance and reduce server load

It's important to see that some of them exist in firebase but when the system grows, these criterias will be important in terms of usability.

- **7-Data repository and distribution requirements**
  CineLibra provides a robust data storage solution to store various types of data, including movie information, user profiles, ratings, reviews, and activity logs. The choice of data storage technology is firebase which includes many services such as data backup and recovery, real time data updates and data privacy.

- **8-Security requirements**
  CineLibra can enhance its resilience to cyber threats, protect customer data, and maintain compliance with applicable laws and industry standards, in terms of authentication and Data Encryption, Secure Communication, data privacy with the help of firebase services.

- **9-Memory and other capacity limitations**
  Caching strategy plays an important role to reduce the workload of the system. As long as limited internet connection and storage is provided, the other criterias will not have a huge effect on the system as significantly unless the platform is adaptable with the system's environment.

- **10-Performance requirements:**
  Response time and throughput should obey some criterias:
  The time period for searching for movies/books or the time when a user enters the system and login verification should be max 2-3 seconds at worst. For many users, the operation of waiting time that requires social connection should not exceed 1 or 2 minutes unless the system crashes.

11-Network communications:

CineLibra currently communicates api services for fetching current book(api to connect the database of ımdb) and movie datas, beside that for authentication firebase auth service will be used and other related user event actions(marking books/movies,add delete fetch operations etc.) firestore will be used to communicate.

12-Verification and validation requirements (testing):
The system should obey the principle of modularity so that unit testing can be done at each stage of the implementation.The general structure could be isolated from its sub-components so that system based testing could be achieved(The pages, the page components separately should be testable).

13-Related to the rsd, time constraints made a huge impact on designing the general structure of the system, since all team members work separately it's crucial to make task division as efficient as possible.

2.3. System Environment

List collection of hardware and software tools a system developer uses to build software systems.

Development Environments:
Integrated Development Environments (IDEs): Such as Visual Studio Code, JetBrains IntelliJ IDEA, Microsoft Visual Studio.
Command-Line Interfaces (CLIs): Such as Terminal on macOS or Command Prompt on Windows for running commands and scripts.
Helper Devices: IOS/Android simulators are used to keep track of the behavior of the system.

Version Control System:
Git is a distributed version control system used for tracking changes in code and collaborating with other developers. We used github for keeping track of our changes in remote environment

Collaboration Tools:
Generally we used discord for team communication and collaboration and whatsapp to manage some small issues related to our software. It's also the same for the design period.
Development Frameworks and Libraries:
We created our application using javascript language and to handle ui interaction, react native library was used which is also common for many developers since providing cross-platform on mobile devices. For the backend side, firebase services are used so it is imported to our project to be able to use its functionalities.

Documentation:
React-native official documentation page is used to understand how the basic components are used and also firebase documentation page is followed regularly to be familiar with the usage of services/functionalities

2.4. Development Methods

The approach that was applied during the period of the project was the waterfall method in which the requirements were determined at the design stage and within those requirements, we tried to develop all the functionalities at different stages. From that aspect, there would be no way to understand how the components work at system level since the system was partially implemented by collaborators of the project. It would be good practise to make it iterative development but since we were not well adapted to the coding environment and did not know how some basic requirements should be implemented or how the interaction flow will continue, waterfall approach educated us to be more familiar with the relations of the components and the structure is built within the effort and time that all of the project members tried to learn and make research.

For a more detailed description of Agile methodologies, The references could be given to have detailed understanding of how or when these methods should be applied:

"Agile Software Development: Principles, Patterns, and Practices" by Robert C. Martin and Scrum

"Object-Oriented Design and Patterns" by Cay S. Horstmann and "Design Patterns: Architectural and component-level design

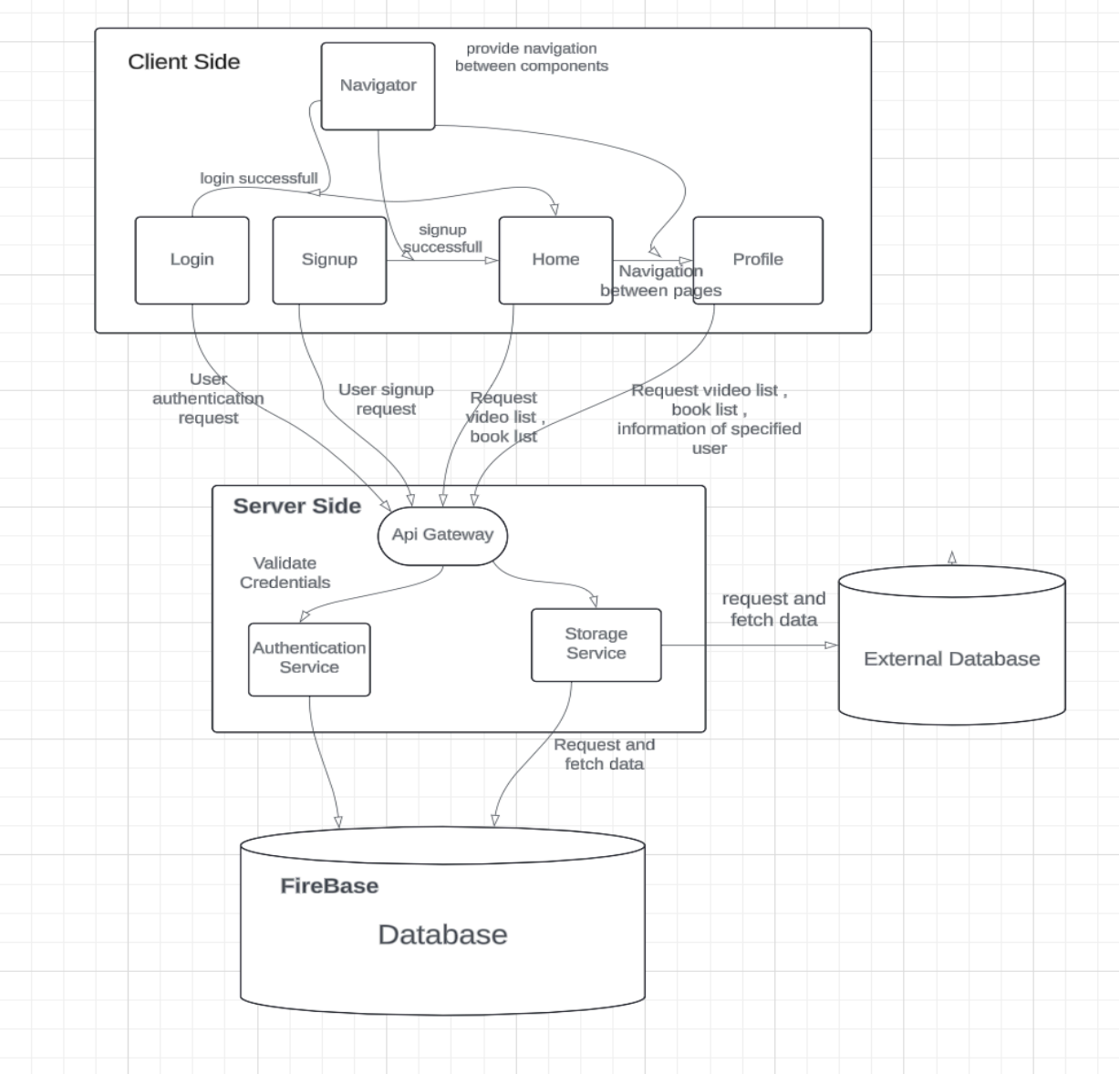
3. Architectural and Component-level design

The design has been influenced by the following decisions:

- The purpose of the application is to provide a platform to keep track of and to follow movie and book publishings. The user shall be aided with the IMDB information and other users' opinions while doing so by the item information pages. The user shall be able to keep track of their desired movies/books by adding them to their personal lists. All the functionality in the application shall be provided with an intuitive, easy to learn user interface. The IMDB api shall aid with the movie/book data.
- The authentication shall be required only upon the initial log-in process to the application. The nature of the application does not require a more secure system.
- As for a database, FireBase shall be used. Only user related data shall be managed in this database, such as: Log-in information, User saved lists, comments to movies/books.
- The system shall be designed on a client-server approach. This helps ensure the client side will be lighter in size and computation.
- Database operations shall be handled by FireBase built-in systems.
- JavaScript shall be used for the client side of the application with the help of IMDB api to populate the data on the client side.
- The JavaScript files shall emulate an object oriented approach by modularisation.
- Each page shall take advantage of modularisation of components, as such, each component such as buttons, text fields shall be imported from their own JavaScript files. Similarly each page shall be kept in their respective JS files.
- The only communication of the user with the server side shall be the following: Log-in information, user saved lists, user comments. These shall be ensured with a session-id on the server side with the FireBase built in functions.
- All communication towards the user shall be done inside the application by JavaScript files.
- The client-side shall be constructed using JS with the help of React-Native, React-Native was selected due to providing a wider coverage of consumer devices and the use of JavaScript. React-Native provides many beneficial properties that will aid with the development of the application

- The system shall be easy to expand by the means of modularisation, if the scope of the application is desired to be wider, the pages shall accept newly constructed module files and expand quickly.

3.1.1. Architecture diagram



3.2. Description for Components

Overview of Primary components:

1. **Navigator** : This component serves as a central hub for defining and configuring navigation within our React Native app, utilizing stack-based navigation and providing custom screens and navigation options to create a seamless user experience.
2. **LogIn :** This component provides users to log in the mobile app with password and email after credentials are verified . Also , this component provides users to log in mobile app with google account

3. **SignUp :** This component provides new users to sign up mobile apps and new users need to sign up with their username , email , password . Also new users can sign up with their google account.
4. **Home :** This component is composed of lots of sub-component .This component has four lists of movies . These lists are Trending , Now Playing , Top Rated and Popular Movies . Also this component has a search area where users can search movies or books that users want . Lastly , this component has a LeftBar component which navigates to the new screen .
5. **Profile :** This component is composed of a profile picture and three list components for the user to save their desired movie/books into. It provides a page for the user to see their personal information.
6. **Authentication Service :** This component identifies and logs in the client matching it with a session id for later use in the database. Built in FireBase functions will be used to help.
7. **Storage Service :** This component stores the user related information such as their account details and their saved lists. As well as managing IMDB's movie and book data.
8. **Database:** This is the FireBase database that will be the server side of the application and handle any and all user related data.
9. **External Database:** This component is the IMDB database we will use the api to get the movie and book information.

A detailed description of each software component contained within the architecture is presented. Section 3.2 is repeated for each of n components.

3.2.1. Processing narrative (PSPEC) for components

1. **Navigator :** The navigator controls all the navigation done between pages by holding each page's id.
2. **Login :** The login component is responsible for taking the input of the login information and using it to let the client login to the app on first start up.
3. **SignUp :** Sign up is much like login, it just lets the user enter the details for an account sign up and uses it to match the database and let the user sign up.
4. **Home :** The home page is the hub of the application with navigation to every page possible from here, it serves as a quick glance at the movies/books and a navigation middleground.
5. **Profile :** Profile page is responsible for displaying the user's account information and their saved lists for access.
6. **Authentication Service :** Authentication service is the server side of the login and sign up operations. it matches the data of login or sign up with the ones in the database and authenticates the action.
7. **Storage Service :** This component is responsible for putting and pulling all the user related data in the FireBase system. As well as pulling the IMDB's book and movie data from the external database.
8. **Database :** This component is responsible for storing all the user related data in the FireBase system.
9. **External Database :** This component is IMDB's database that lets the application display relevant movie and book information.

A processing narrative for component n is presented. It should describe the responsibilities of the component.

3.2.2. Components' interface description.

1. **Navigator**:
   - 1.1. **Input Interface:** List of Pages
   - 1.2. **Output Interface:** ID list of pages
2. **LogIn :**
   - 2.1. **Input Interface:** Text input for user information and buttons to log in or sign up.
   - 2.2. **Output Interface:** Provides user authentication details for the authentication service.
3. **SignUp :**
   - 3.1. **Input Interface:** Text input for user information and buttons for sign up.
   - 3.2. **Output Interface:** Provides user authentication details for the authentication service to sign-up.
4. **Home :**
   - 4.1. **Input Interface:** Gets the information of the lists to be displayed as lists of item buttons.
   - 4.2. **Output Interface:** Displays the item lists as well as a search bar and navigation buttons to get to the other pages.
5. **Profile :**
   - 5.1. **Input Interface:** Gets the saved lists data and subsequent information of the saved books/movies.
   - 5.2. **Output Interface:** Displays the user information and the saved lists with the items as buttons to navigate to the item's page.
6. **Authentication Service :**
   - 6.1. **Input Interface:** Gets its input from either the login or sign up page in the form of user information.
   - 6.2. **Output Interface:** Authenticates the user for the requested component and provides a response.
7. **Storage Service :**
   - 7.1. **Input Interface:** Gets the requested data information or the data to be written.
   - 7.2. **Output Interface:** Provides the information request results from the FireBase database.
8. **Database:**
   - 8.1. **Input Interface:** Gets the requested information about the user.
   - 8.2. **Output Interface:** Provides the requested information result to the callee.
9. **External Database:**
   - 9.1. **Input Interface:** Gets the requested information about the item.
   - 9.2. **Output Interface:** Provides the information results for the request.

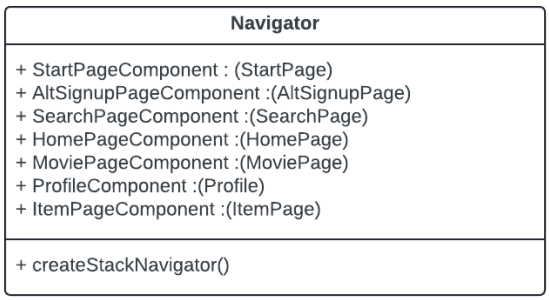
A detailed description of the input and output interfaces for the component is presented.

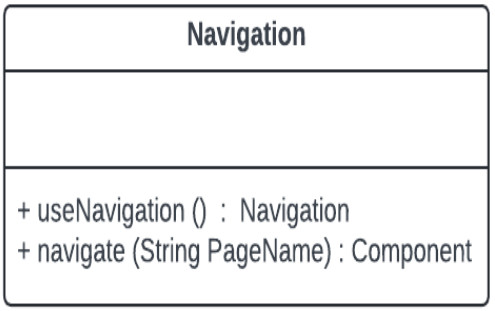
3.2.3. Components' processing detail

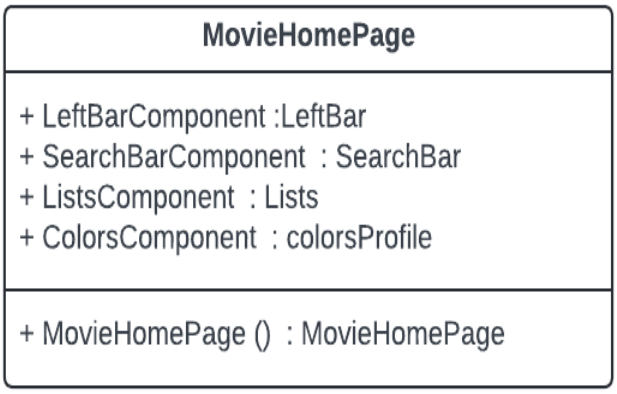
A detailed algorithmic description for each component is presented.

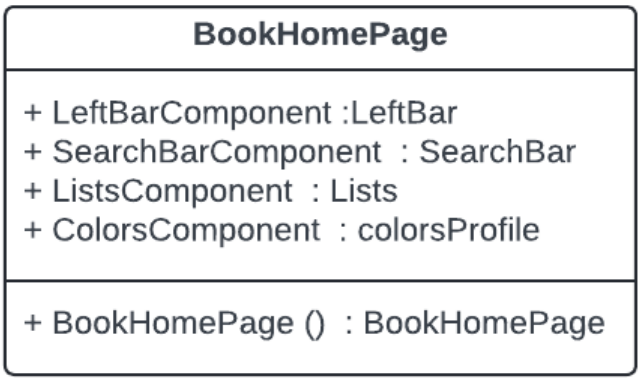
3.2.3.1 Design Class hierarchy for the components

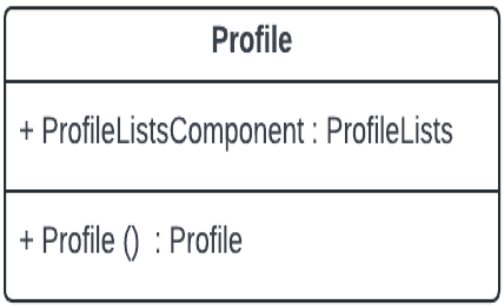
ClientSideComponents:

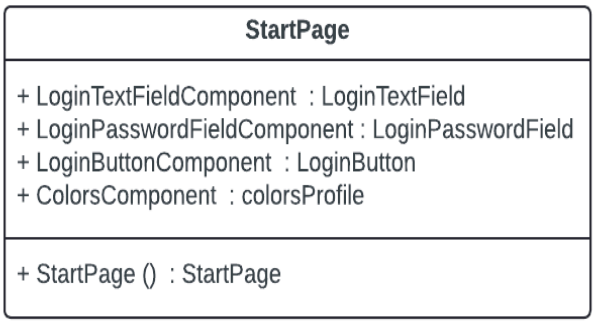


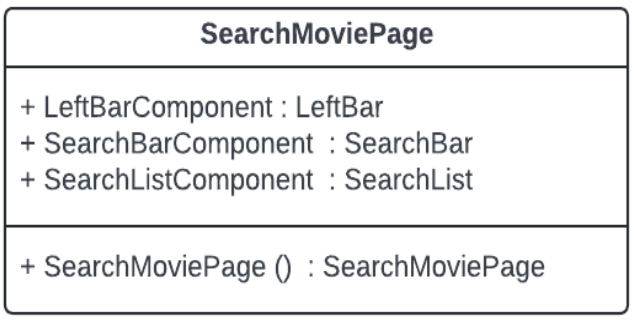


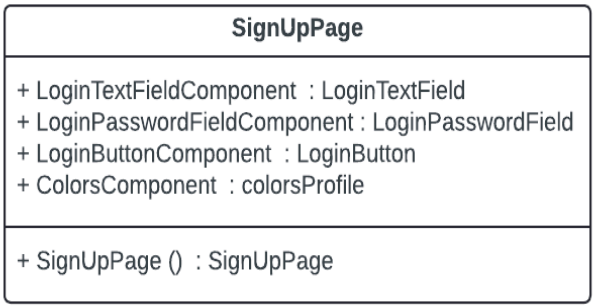


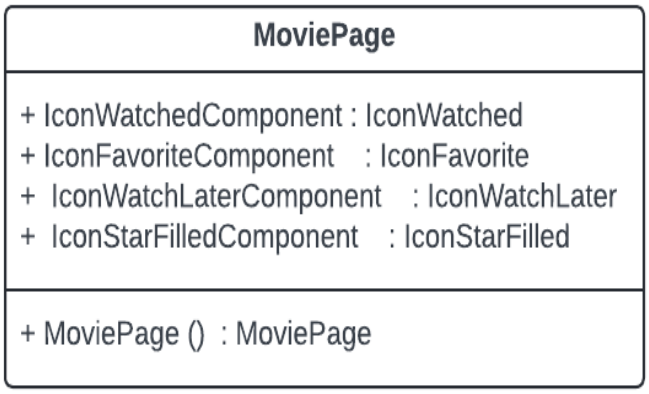


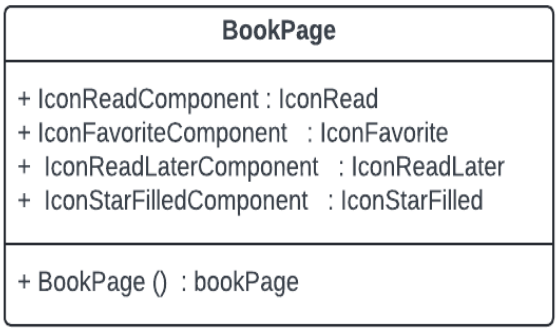




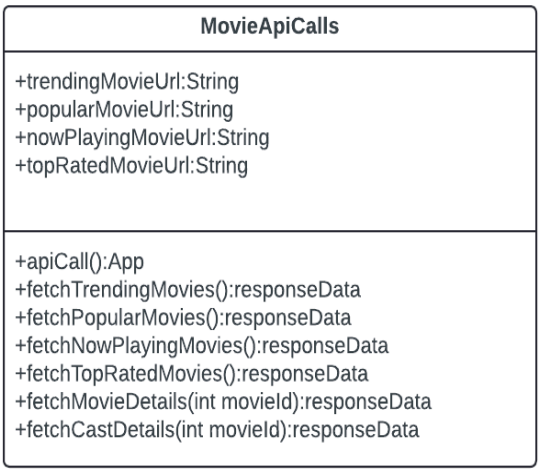


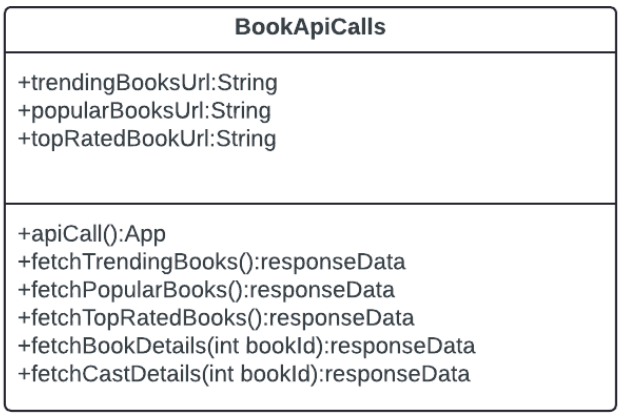




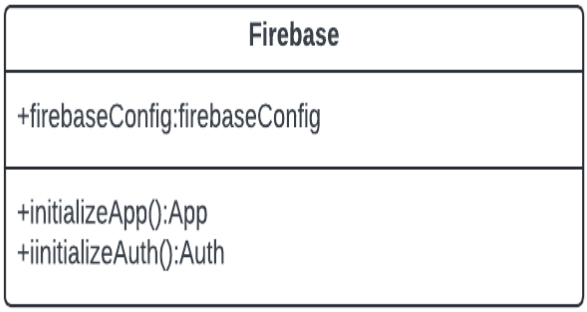


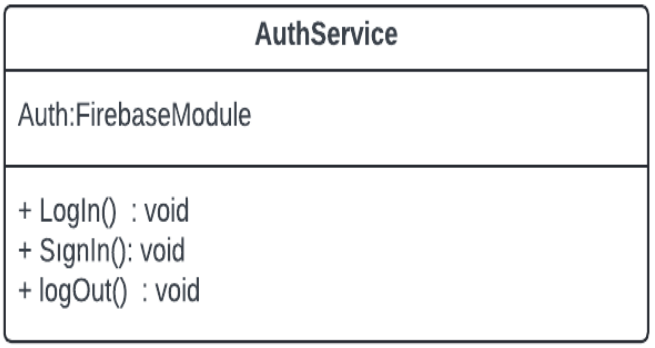
ApiComponents:

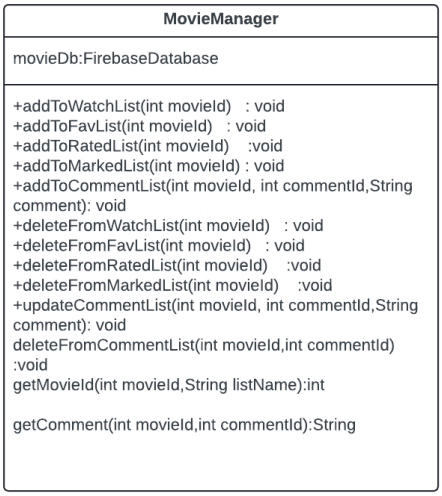


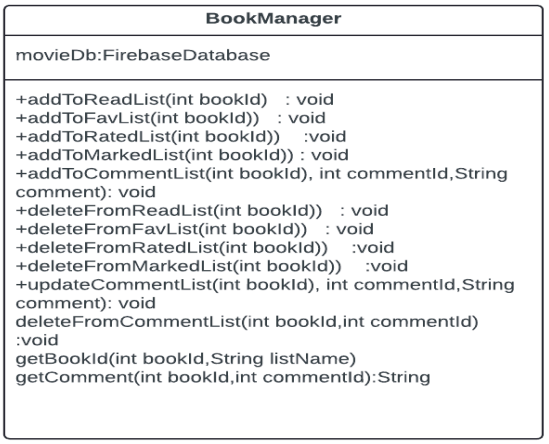


FirebaseServiceComponents:









3.2.3.2 Restrictions/limitations for the components

1. **Navigator :** The navigator can only use the page linking method of react-native, this has proved troublesome while implementing the extra book section of the application as well as the drawer navigation system in the home page. workarounds were put in place to achieve the desired look and feel of the navigation.
2. **Login :** For simplicity with the firebase implementation, the login only can use email and password authentication unlike some other applications which can use username and/or phone numbers to authenticate.
3. **SignUp :** The signup component is similarly limited in the input fields much like the login component. For simplicity with the FireBase implementation only three fields are presented to the user, username, password and email.
4. **Home :** The home component is limited with its navigation capabilities because of space limitations with the presented items on it, we had to introduce an intermediary, a drawer navigator which houses the majority of the navigation on this component.
5. **Profile :** The profile page is limited to only three user saved lists due to FireBase implementation simplicity. Three was deemed appropriate for the

number of lists the user could have. Alongside not being able to display the users all time comments here due to FireBase simplicity and the comments being tied to items instead of users.

6. **Authentication Service :** The authentication service is bound to FireBase built in functions, therefore it is not fully customizable and requires extra steps for the sign up procedure, namely the username being held in the database is not built in.
7. **Storage Service :**
8. **Database :** The database is bound by FireBase's limitations and therefore cannot be customized fully, however, there has been no forthcoming limitations from this approach.
9. **External Database :** External Database is limited to what the IMDB provides for the time data, therefore the data in the application represented is completely reliant on this database.

3.2.3.3 Performance issues for the components

1. **Navigator :** The navigation visuals used by this component rarely can take long enough to notice while using the application.
2. **Login :** There seems to be no performance issues with this component except possible internet connection problems.
3. **SignUp :** There seems to be no performance issues with this component except possible internet connection problems.
4. **Home :** This component can take a while to display the contents due to large amounts of data being pulled from the IMDB database each time it is loaded.
5. **Profile :** This component can take a while to load its contents of items.
6. **Authentication Service :** There seems to be no performance issues with this component except the possible internet and FireBase related slowdowns.
7. **Storage Service :** There seems to be no performance issues with this component except the possible internet and FireBase related slowdowns.
8. **Database :** There seems to be no performance issues with this component except the possible internet and FireBase related slowdowns.
9. **External Database :**There seems to be no performance issues with this component except the possible internet and api related slowdowns.

3.2.3.4 Design constraints for the components

1. **Navigator :** The navigator controls all the navigation done between pages by holding each page's id.
2. **Login :** The login component is responsible for taking the input of the login information and using it to let the client login to the app on first start up.
3. **SignUp :** Sign up is much like login, it just lets the user enter the details for an account sign up and uses it to match the database and let the user sign up.
4. **Home :** The home page is the hub of the application with navigation to every page possible from here, it serves as a quick glance at the movies/books and a navigation middleground.
5. **Profile :** Profile page is responsible for displaying the user's account information and their saved lists for access.
6. **Authentication Service :** Authentication service is the server side of the login and sign up operations. it matches the data of login or sign up with the ones in the database and authenticates the action.
7. **Storage Service :** This component is responsible for putting and pulling all the user related data in the FireBase system. As well as pulling the IMDB's book and movie data from the external database.

8. **Database :** This component is responsible for storing all the user related data in the FireBase system.
9. **External Database :** This component is IMDB's database that lets the application display relevant movie and book information.

3.2.3.5 Processing detail for each operation of the components

1. **Navigator :** The navigator links each page using their string names as IDs. This is then used inside the page files to properly navigate.
2. **Login :** The login component uses text fields to get the authentication details and sends them to the authenticator when the relevant button is pressed. it can alternatively navigate to the sign up component if the button is pressed.
3. **SignUp :** The sign up component uses text fields to get the sign up details and sends them to the authenticator when the relevant button is pressed. it can alternatively navigate to the login component if the button is pressed.
4. **Home :** The home page has various buttons connecting other pages, a search button for the search page, a drawer button for the drawer component, and item lists which are fetched from the IMDB database, the drawer includes various buttons of navigation alongside the profile photo which is fetched from the database.
5. **Profile :** The profile component has three saved lists that fetch the information from the api, and a profile photo with the username, these are fetched from FireBase.
6. **Authentication Service :** The authentication service is a bridge between the FireBase authentication service and the relevant components needing authentication.
7. **Storage Service :** The storage service is the intermediary between any component needing something stored or fetched by the FireBase. it gets the needed information and requests related data from FireBase
8. **Database :** The database holds the user related information and provides it with javascript code in the components.
9. **External Database :** This component is accessed through the api provided by IMDB. Regular javascript code is used.

3.3. Dynamic Behavior for Components

1. **Navigator** : This component connects every page in the application via assigning IDs and matching them. It is the backbone of the navigation system in every page.
2. **LogIn :** This component provides the authentication service with the user information to be checked.
3. **SignUp :** This component provides the authentication service with the new user information to be signed up.
4. **Home :** This component connects the search, profile, login, item page via button navigation. It requests the item information from the storage service as well.
5. **Profile :** The profile page requests the user information from the FireBase database to be presented. As well as the saved lists of the user from the database.
6. **Authentication Service :** The authentication system takes the login information from the Login or SignUp components. It then transmits the information to the FireBase database to be checked.

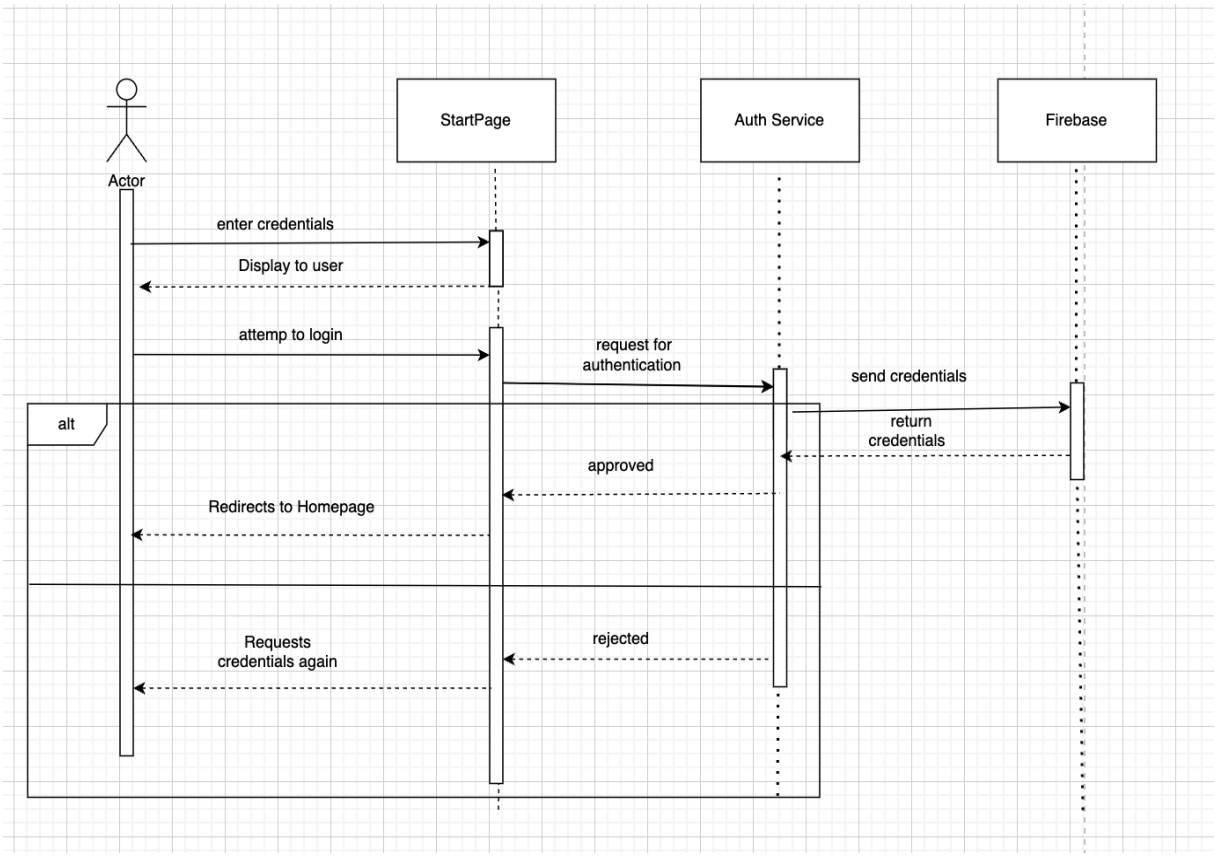
7. **Storage Service :** The storage component provides an interface for the various databases that are present in the system.
8. **Database:** This component provides the authentication data for the authentication service on request. As well as storing the user information via the storage service.
9. **External Database:** The storage service requests data from this component which is then displayed on; home page, search page and the profile page.

A description of the interaction of the classes is presented.

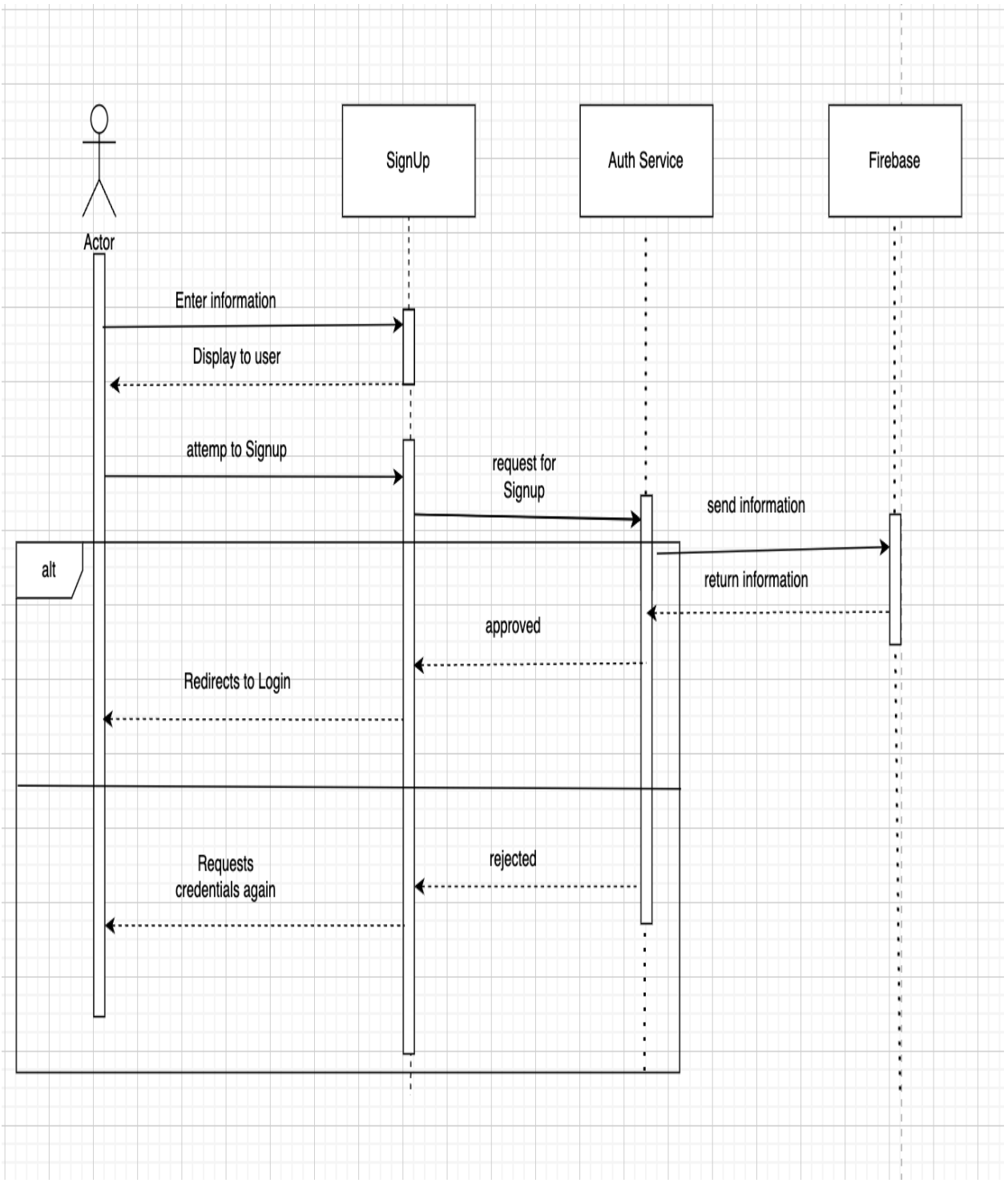
3.3.1. Interaction Diagrams

A sequence diagram, for each use case the component realizes, is presented.

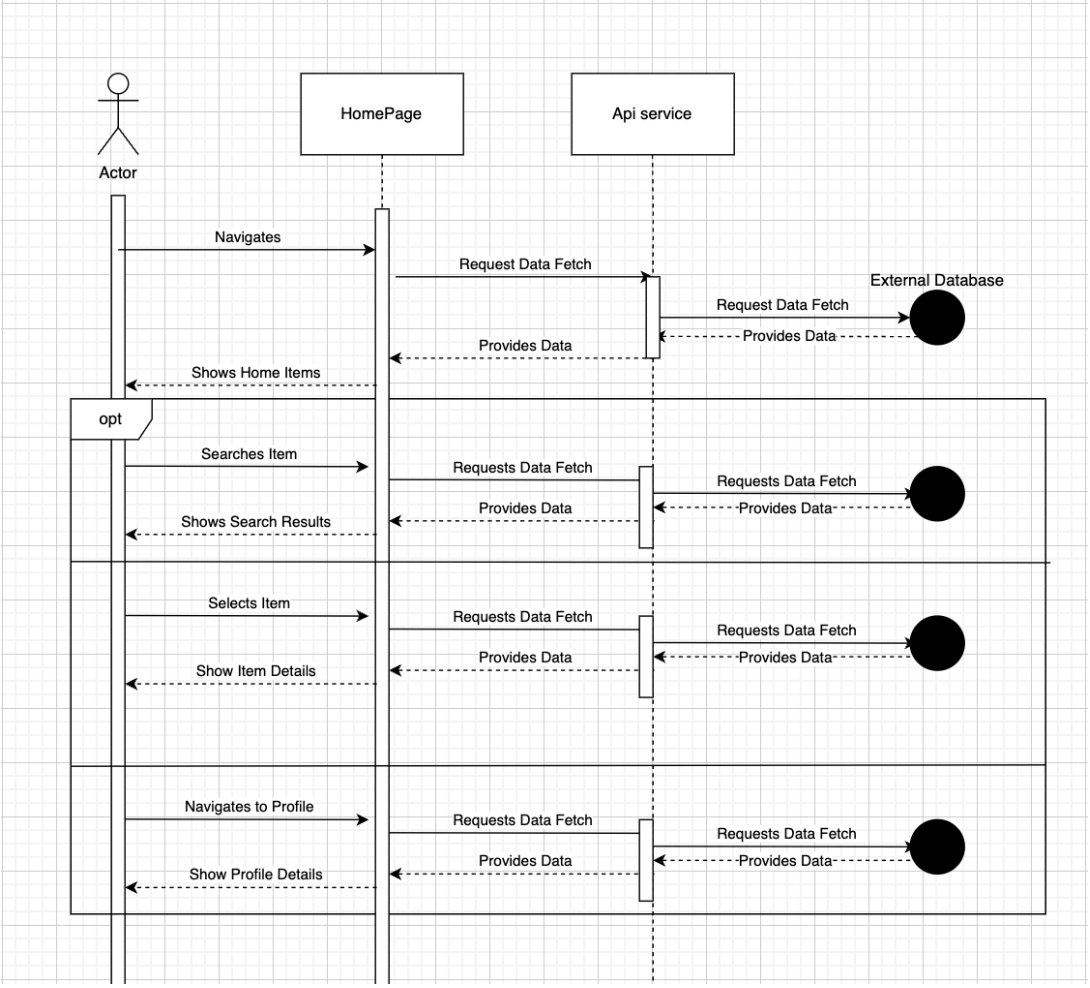
1- LogIn :



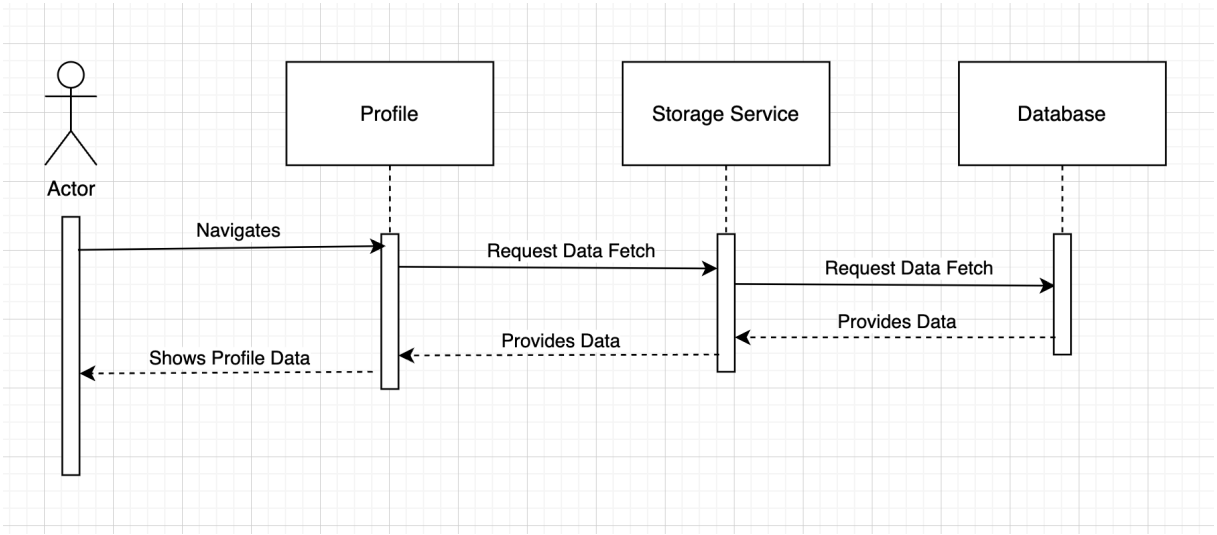
2- SignUp :


Actor
SignUp
Auth Service
Firebase
Enter information
Display to user
attemp to Signup
request for Signup
send information
alt
return information
approved
Redirects to Login
Requests credentials again
rejected

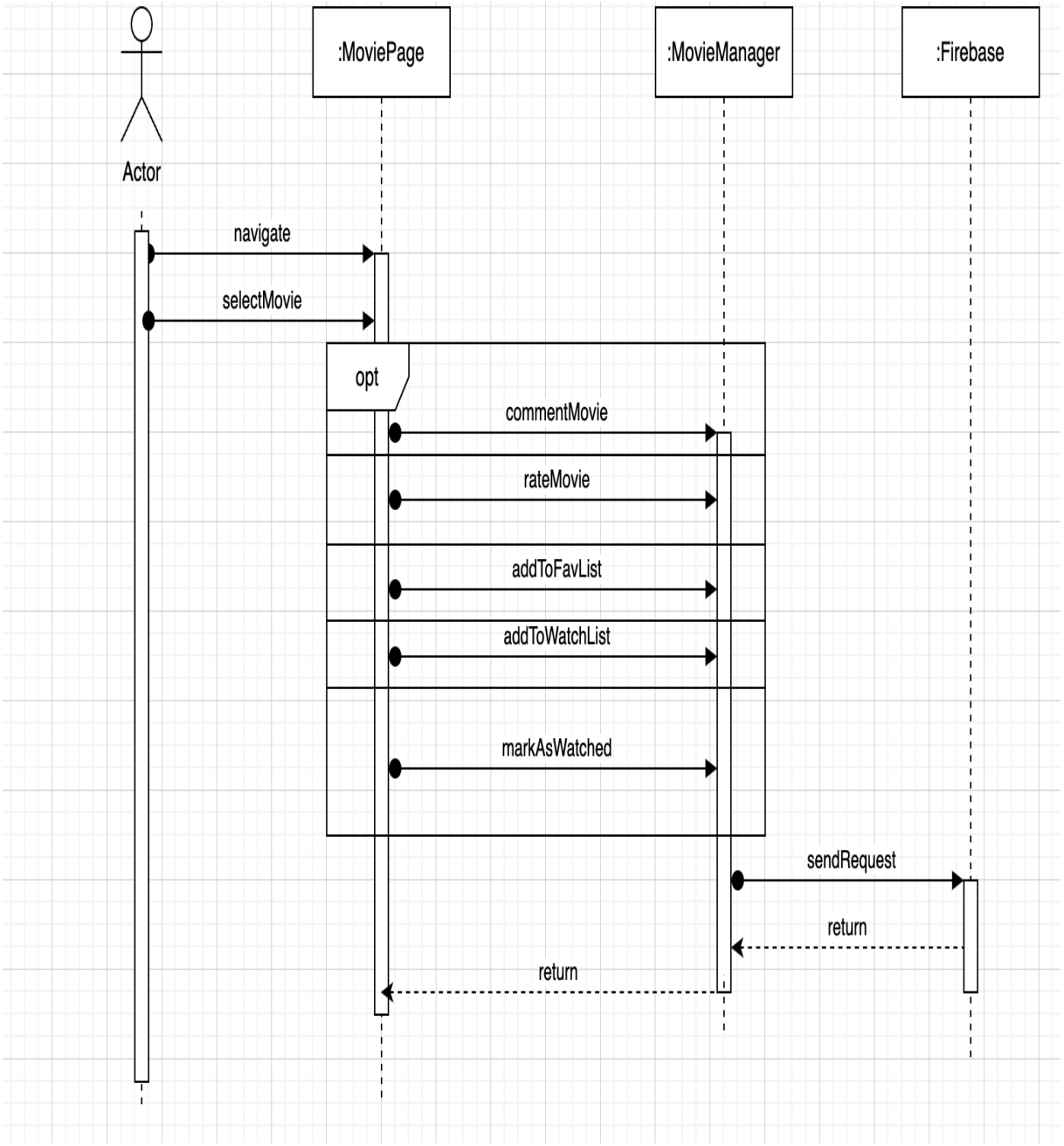
3- Home :



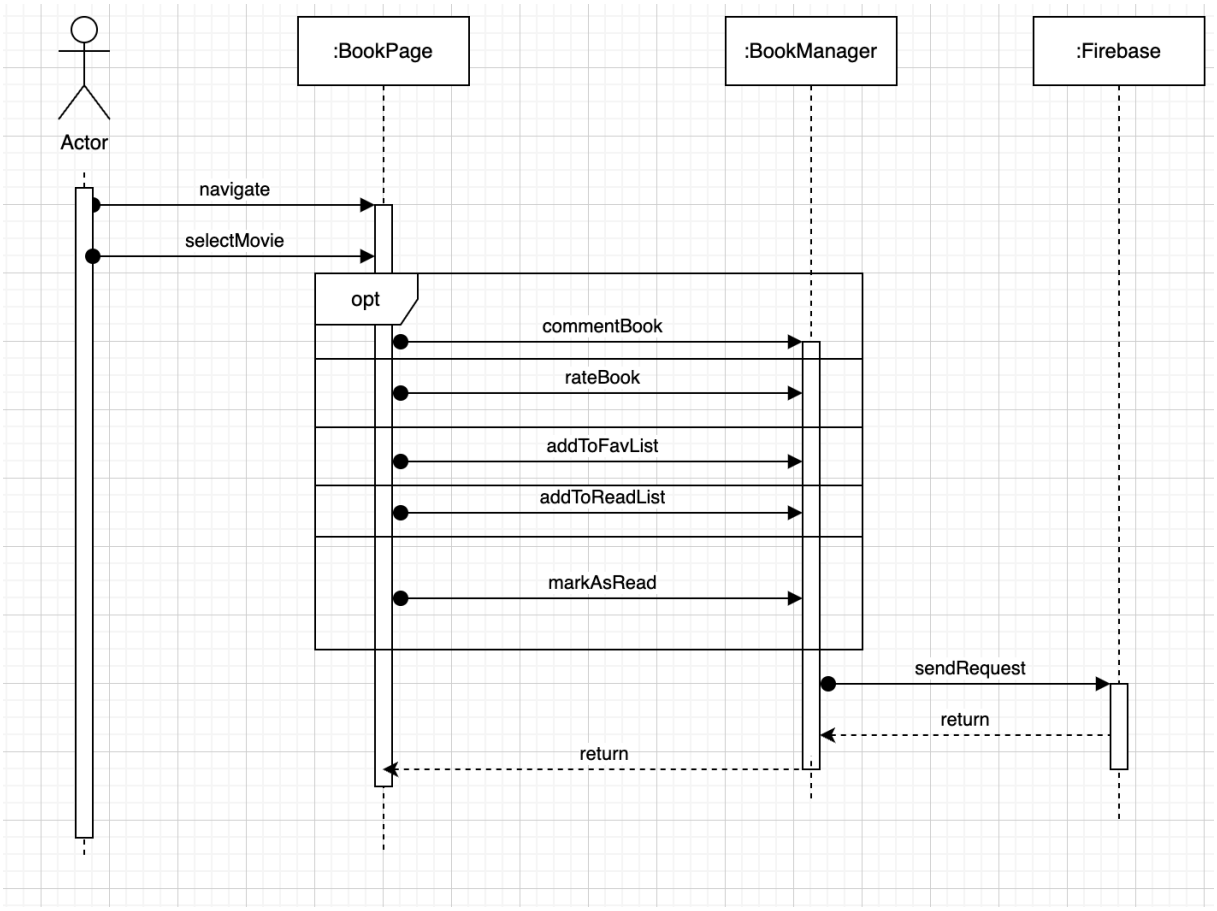
4 - Profile :



5-MoveItemSelection



6)BookItemSelection:



4. Restrictions, limitations, and constraints

- The use of React-Native and thus the provided functionalities necessitated different design choices between platforms of IOS and Android due to the aforementioned platforms having different properties in their designs.
- Due to different sizes of devices any and all design elements in the user interface must be dynamically sized instead of statically
- The current trend of multiple device themes, such as, dark and light mode, necessitated a way to implement multiple application wide themes on the run.
- The margin of users scattered on different platforms such as IOS and Android necessitated a common service, React-Native, to reach higher numbers of consumers.
- The vast amounts of media necessitated the use of external api's from IMDB to be able to include the movie and books in the amount that would make the application usable.

5. Conclusion

CineLibra aims to be a user-friendly mobile application where users can track, find movies and books, and share their opinions. This document addresses essential functionalities such as user authentication, API integration for up-to-date movies and books, and database system for keeping track of users' activities. Firebase for authentication and data management, and integrating external APIs, CineLibra ensures a robust backend while providing a seamless user experience on both Android and iOS platforms.

In conclusion, the CineLibra project demonstrates the integration of multiple technologies to create a functional mobile application. Future iterations can build upon this foundation by incorporating advanced features such as AI-driven recommendations and enhanced social interactions, further enriching the user experience.