**CS 3354 Software Engineering**

**Final Project Deliverable 1**

Online Library App

Mehdi Devjani, Anna Ernst, Chunnu Ghimire, Jonathan Gruber, Kyle Keeton, Stefan Stojanovic, Hannah Swiney, Micah Warner

# Addressing of Feedback to Project Proposal

## Original Topic Description

We will design a library app to allow users to check out eBooks from a local library. This idea is quite like the suggested bookshelf project but adds support for user accounts and ensures only the allocated number of eBooks are lent out at one time. This app can be used in community outreach programs that provide services for those who want to utilize library resources.

## Feedback

Good choice for a topic! A comprehensive library management system truly is useful and will promote community spirit. Please contact UTD Library staff, as well as local library staff to get more information about the details of the workings of the current systems, what could be improved, etc. for a better design.

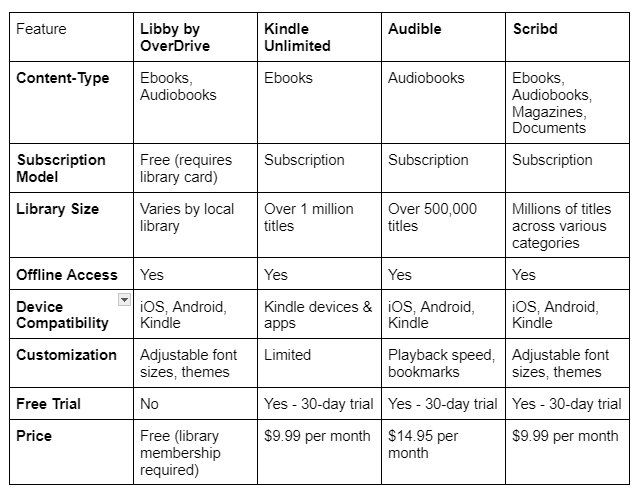
It is great to see a detailed breakdown of the tasks you have considered already. Good job.

In the final report, please make sure to include comparison with similar applications -if any-, make sure that you differentiate your design from those, and explicitly specify how.

Fair delegation of tasks.

## Response to Feedback

Our group has created a chart that lists and compares the features of four similar library applications of various sizes including Libby, Kindle Unlimited, Audible, and Scribd. We intend to use this chart as a guide and a metric for the design of our software project. Many of the features of our project are already decided on, such as content type and customization. Other features, such as the price of the finished product, however, depend on the cost of the software and profit margins that are yet undetermined. Furthermore, we intend to contact local libraries to determine what software, if any, they use for online content distribution.



# GitHub Repository

<https://github.com/micahwarner/3354-OnlineLibraryApp>

# Task Delegation

Mehdi Devjani

* Create use-case diagram
* Create test plan

Anna Ernst

* Create use-case diagram
* Create project planning estimate

Chunnu Ghimire

* Create sequence diagrams
  + Your Books
  + Read Book
  + See Chapters List
  + Search Book Contents
  + Change Appearance
  + Create Accounts
  + Modify Book Categories
* Create project staffing estimate

Jonathan Gruber

* Create sequence diagrams
  + Log In
  + Verify Login Credentials
  + Display Login Error
  + See List of Books/Search Books
  + Check out Books
  + Verify Book Availability
  + Add/Delete Book
* Create cost and effort pricing estimate

Kyle Keeton

* Commit “project\_scope” document to GitHub repository
* Create class diagram
* Create cost of personnel

Stefan Stojanovic

* Create first commit to the GitHub repository (README file with team name)
* Create class diagram
* Research hardware costs

Hannah Swiney

* Create architectural design
* Cost of software projects

Micah Warner

* Set up GitHub repository and add all team members and TA as contributors
* Create architectural design
* Comparison of our work to similar designs

# Group-Member Contributions

Mehdi Devjani

* Deciding upon architectural design
* Use-case diagram

Anna Ernst

* Deciding upon architectural design
* Use-case diagram

Chunnu Ghimire

* Deciding upon architectural design
* Sequence diagrams
  + Your Books
  + Read Book
  + See Chapters List
  + Search Book Contents
  + Change Appearance
  + Create Accounts
  + Modify Book Categories

Jonathan Gruber

* Deciding upon architectural design
* Sequence diagrams
  + Log In
  + Verify Login Credentials
  + Display Login Error
  + See List of Books/Search Books
  + Check out Books
  + Verify Book Availability
  + Add/Delete Book
* Functional requirements 1, 2, and 3
* Nonfunctional requirements
  + Usability requirements
  + Dependability requirements
  + Ethical requirements

Kyle Keeton

* Committed “project\_scope” document to GitHub repository
* Deciding upon architectural design
* Collaborated with Stefan on the class diagram
* Class diagrams
  + Book
  + Catalog
  + Personal shelf
* Software process model
* Functional requirement 6
* Nonfunctional requirements
  + Space requirements
  + Environmental requirement
  + Regulatory Requirement
  + Legislative Requirement
  + Operational Requirement
  + Development Requirements

Stefan Stojanovic

* Created first commit to the GitHub repository (README file with team name)
* Deciding upon architectural design
* Functional requirements 4 and 5
* Nonfunctional requirements
  + Performance requirements
  + Security requirements
* Collaborated with Kyle on Class Diagram
* Class diagram
  + User
  + Librarian
  + Person
  + Authorization
  + Light/Dark mode and Chapter List functions/attributes in Book

Hannah Swiney

* Deciding upon architectural design
* Addressing feedback
* “Architectural Design” section of this report

Micah Warner

* Set up GitHub repository and added all team members and TA as contributors
* Deciding upon architectural design
* “Architectural Design” section of this report

# Software Process Model

We are using scrum for the following reasons:

1. This is an area with significant competition. The waterfall model is definitely impossible to do reasonably.
2. It is extremely likely that many things are going to be completed in parallel. Components are not going to be designed one at a time. Rather, for the sake of time efficiency, it is important to develop, say, the books, library, and user classes at the same time during points of the project.
3. To work on everything in parallel, it is important to organize things into smaller projects, in the form of sprints on a scrum agile development cycle.
4. In the modern day, most programmers have experience with Agile development methods (scrum and similar). In a real-world environment, this means cutting down on training.
5. Customers and other beneficiaries of the software (i.e. both libraries and end users) do not know what they want. A process model that can pivot easily will help deal with that.

# Software Requirements

## Functional Requirements

1. A customer shall be uniquely identified by their username and have an account in the system under that username. *(An account system is a natural way for storing information that is private to each user, and accounts are* *generally identified by a unique username.)*
2. A customer shall be able to browse the list of books in the library catalog and to filter them according to a text query. *(A customer must be able to determine what books a library has in order for the library to serve its purpose, and filtering books allows one to find a book without needing to examine each book in the catalog one-by-one.)*
3. A customer shall be able to check out books listed in the library catalog and to return books that they have checked out. *(A necessary function of a library is for one to be able to check out books that the library possesses and to return books that one has checked out.)*
4. A customer shall be able to log in with their username and password and log out of the application when finished. *(This creates instances in the system and ensures none are taking up space and processing power when not in use.)*
5. A customer shall be able to select any checked-out books and read them using the Online Library Application. *(This accommodates digital readers if it is preferred over physical books.)*
6. A customer should be allowed to reserve books to automatically check them out when they are available. *(This is a standard function in most libraries to allow people to check out books that are unavailable at the moment without a huge time investment from the user for checks.)*

## Nonfunctional Requirements

### Product requirements

* Usability requirements
  + The software shall be interoperable with screen readers. *(The software ought to be accessible to those who are blind or otherwise visually impaired.)*
* Efficiency requirements
  + Performance requirements
    - The Online Library Application system shall not exceed 500 milliseconds in operation delay which includes enabled or disabled animations. *(This ensures average usage is fast, for example when opening a book or changing the view mode and heavy usage can still offer decent performance at 500 milliseconds.)*
  + Space requirements
    - The Online Library Application client size shall not exceed 50 MB. *This is based on a rough average of library app download size on a mobile platform. Sources used are Open Library Reader, Libby, and My Library on the Google Play Store.* *BorrowBox was looked at on the Apple Play Store, but its astronomical size in comparison to competitors determined it to be an outlier. Extra space is accounted because it is unclear if the extra space is due to Apple’s required development environment)*
* Dependability requirements
  + Total daily application downtime for a customer shall not exceed 100 seconds, on average. *(A day is* *86,400 seconds, so this requirement ensures that the application is functional for well over 99% of a given day.)*
* Security requirements
  + The Online Library Application will securely store and transmit user data using up to date encryption algorithms. *(It is important that personal information is not compromised otherwise no one will trust the system.)*

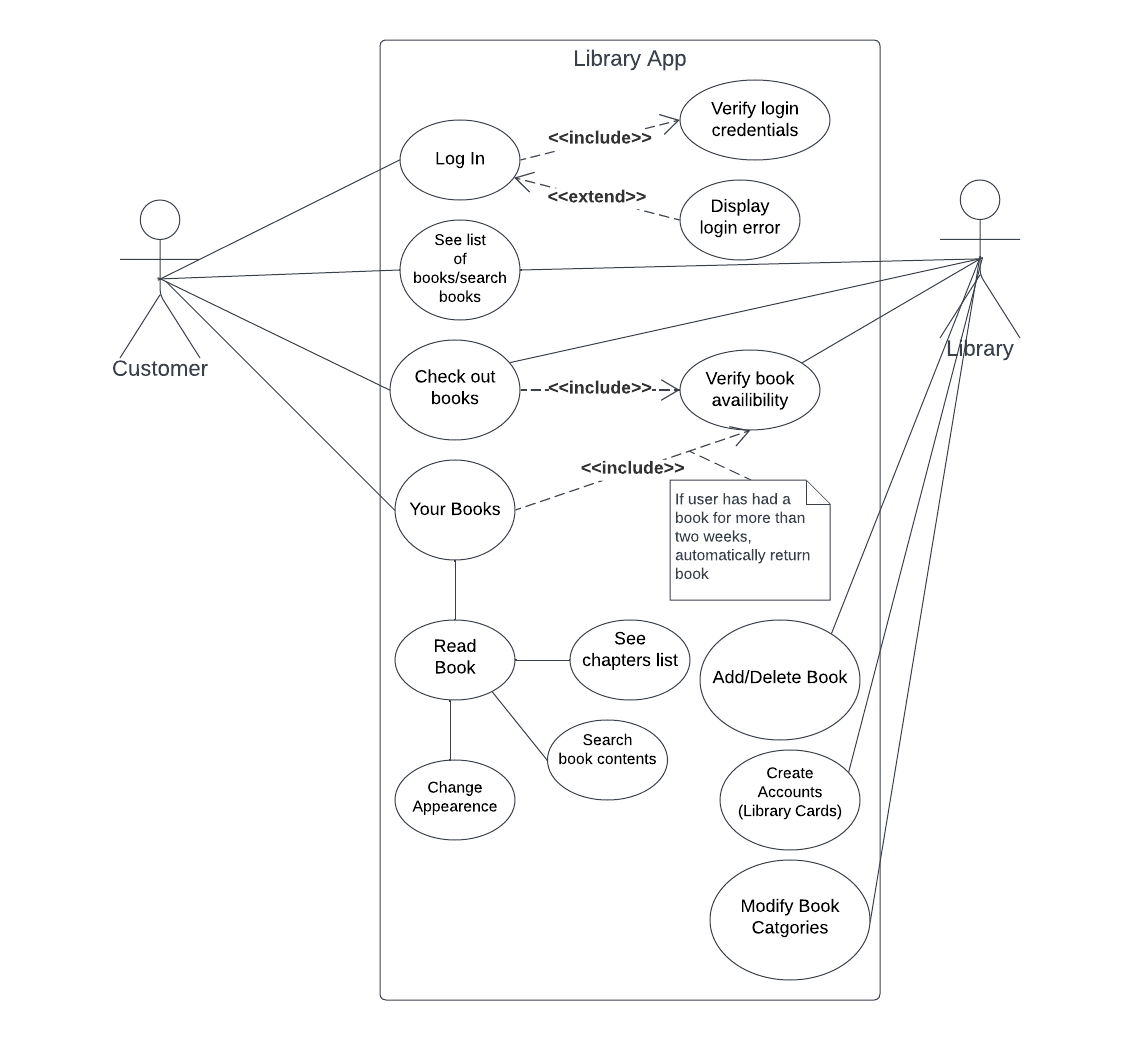
### Organizational requirements

* Environmental requirements
  + The client-side app should work on Android and IOS operating systems. It will also work on base Chromium and Firefox. *(They are the most common mobile platforms for phones and tablets. Browsers work similarly to containers. As a result, making the web-app side work on the two main base browsers will give Linux, MacOS, and Windows functionality with very minimal testing. On Linux, this avoids dependency-based conflicts without the need for a container in a Snap or* *Flatpak style installer.)*
  + The server will work on the AWS (Amazon Web Services) platform with an Amazon Aurora DBOS. *(Using an AWS platform drastically reduces the barrier of entry for any libraries that want to use the app. The operating system has better integrations for the database than a standard Red Hat system. Aurora is also well-integrated in the AWS environment, making it a better choice than other DBOS. The DBOS is also highly scalable allowing for many different libraries of varying sizes to use the app.)*
* Operational requirements
  + Web traffic will be dictated by the authentication level given in login. *(Using the authentication level provided in a key or similar from the authentication app will prevent unauthorized users from accessing things they should not be accessing.)*
  + Server-client communications will be encrypted with an RSA-based encryption protocol*. (RSA-based encryption will prevent people from intercepting and understanding signals originating from the client app and ending at the server. This will help prevent unauthorized users from intercepting signals for usernames and passwords. An RSA encryption also allows for digital signatures that depending on client needs can prevent man in the middle attacks that hijack cookies.)*
* Development requirements
  + The database will be hosted in a DynamoDB database system. *(The server will* *likely be hosted on a cloud service to aid in easy scaling. DynamoDB is the AWS* *noSQL database.)*
  + The authentication will use Java. *(One premier open-source project for authentication, Auth0, is written in Java. Integrating Auth0 is best as it is a proven solution to the authentication problem. Java allows for the easiest integration into the overall system.)*
  + The web-app client will use the ReactJS framework *(React is one of the most used JavaScript frameworks. It has a wealth of integrations and support to assist with the server-client model.)*
  + The app will be written in Kotlin. *(Kotlin is well-supported by the Android Studio and has integrations with the JVM. Android-Apple conversion can be handled by* *MechDome or J2ObjC style tools. Complete parallel development in Swift and a non-Apple language would create ballooning costs due to being forced to remake the same application twice.)*

### External requirements

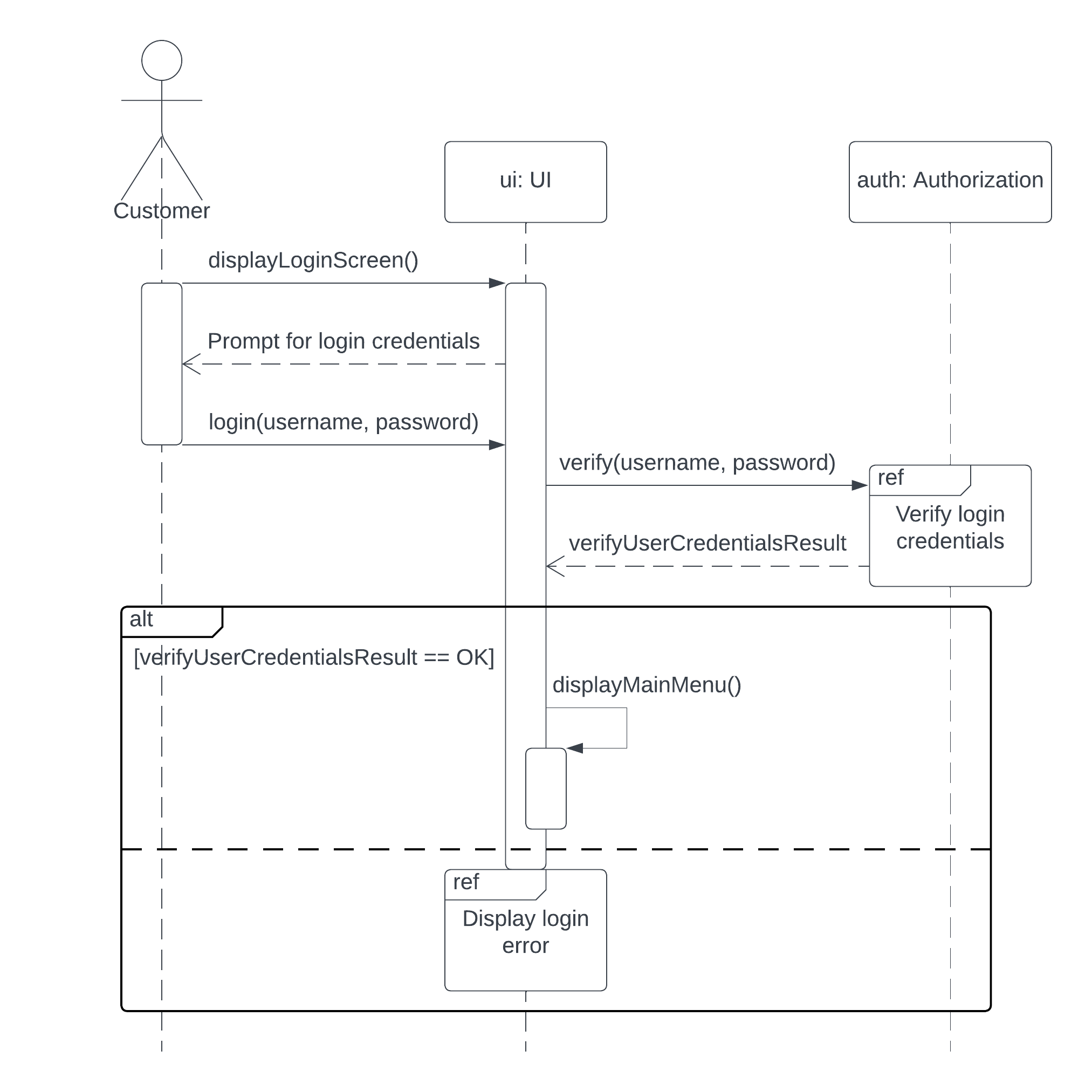
* Regulatory requirements
  + This app must comply with the California Consumer Privacy App (CCPA), Children’s Online Privacy App (COPA), and any other state regulations that may arise during the development of the app. *(This allows the application to be used in any state in the United States. We do not plan on it being used outside of the United States.)*
* Ethical requirements
  + The software shall not collect telemetry from users without their explicit consent. *(Collecting telemetry from users without their explicit consent would violate user privacy, which would be unethical.)*
* Legislative requirements
  + Accounting requirements
    - There will not be any accounting requirements. *(This app is meant to assist an existing library in online distribution. As a result, membership fees and other payments should be processed by the relevant parties in a separate secure site if necessary.)*
  + Safety/security requirements
    - This app must comply with the California Consumer Privacy App (CCPA), Children’s Online Privacy App (COPA), and any other state regulations that may arise during the development of the app. *(This allows the application to be used in any state in the United States. We do not plan on it being used outside of the United States.)*

# Use-Case Diagram

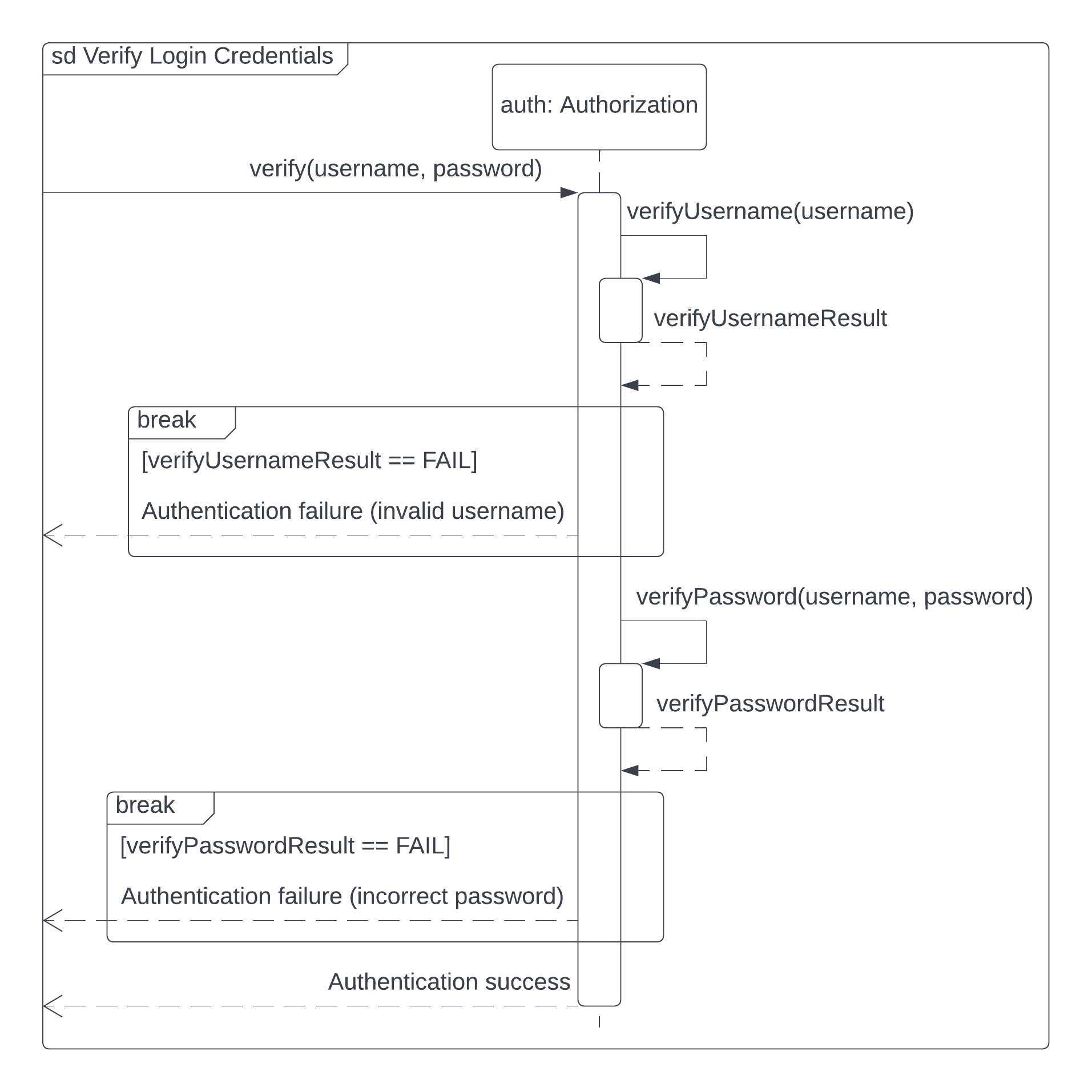


# Sequence Diagrams

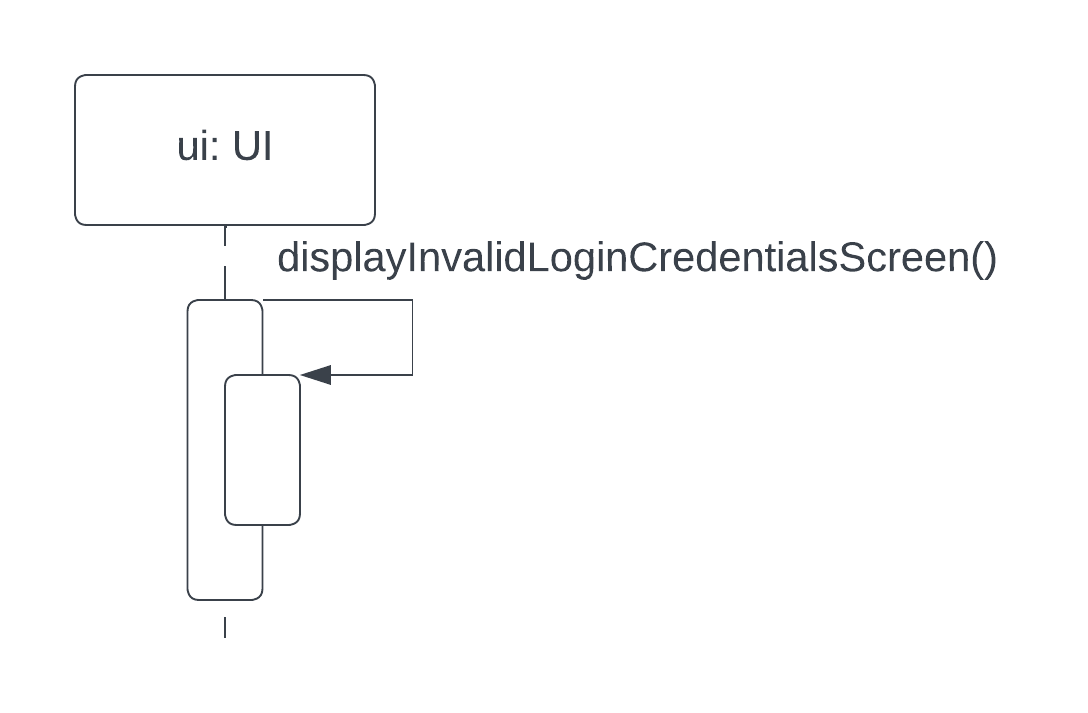
Log In



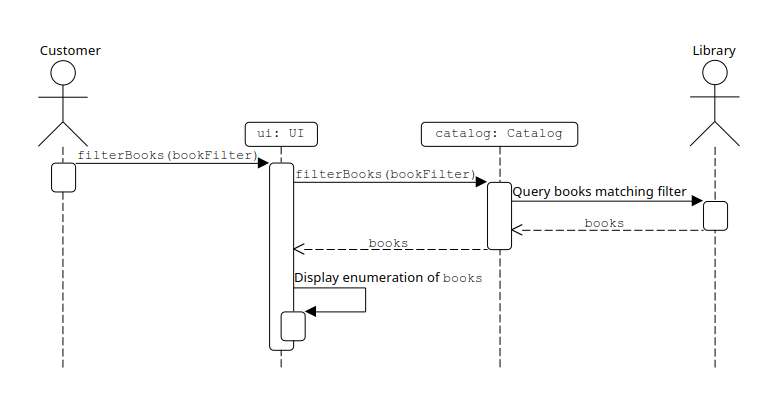
Verify Login Credentials



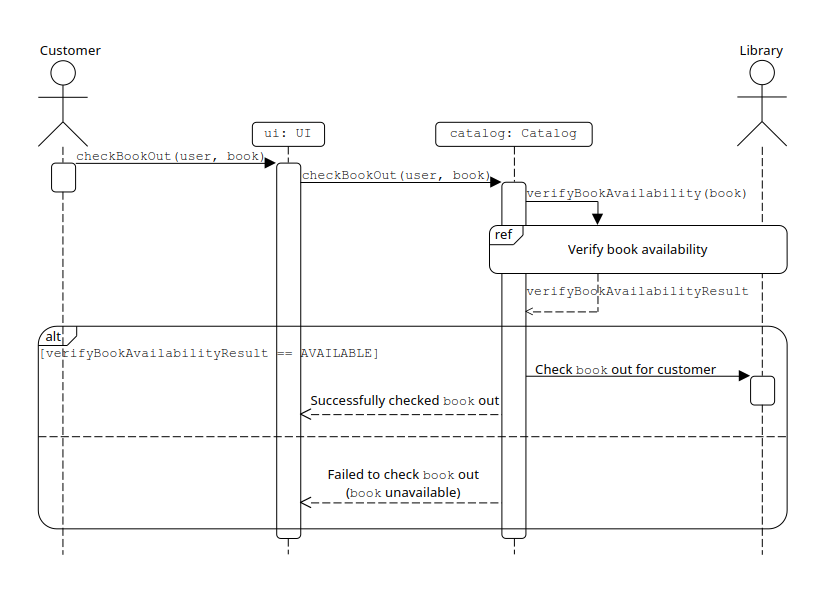
Display Login Error



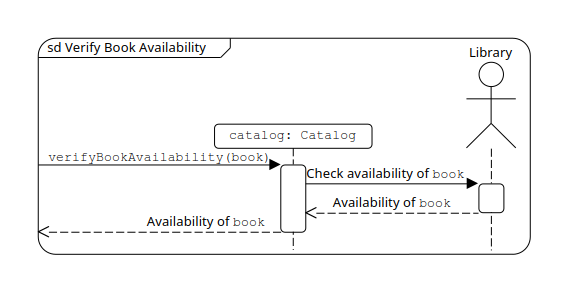
See List of Books/Search Books

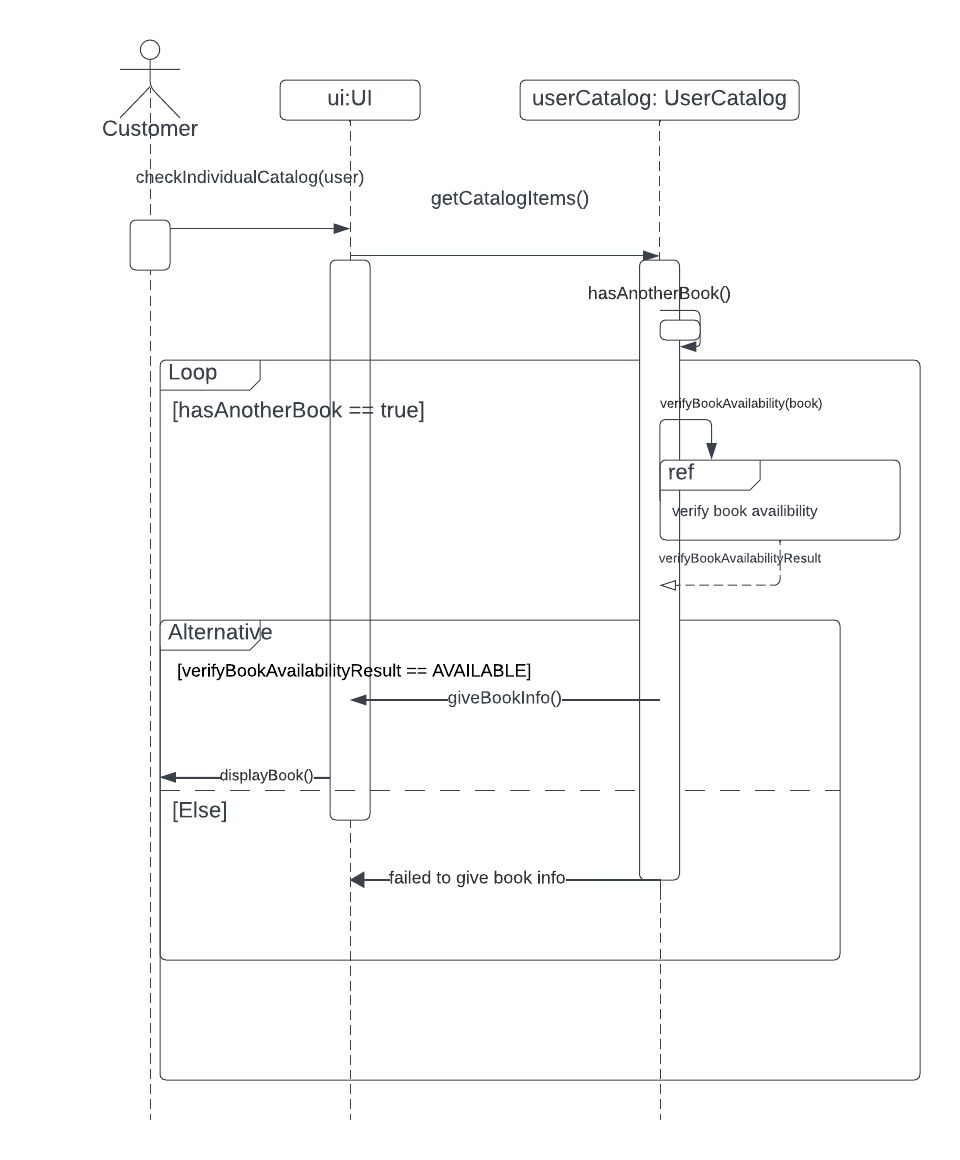


Check out Books

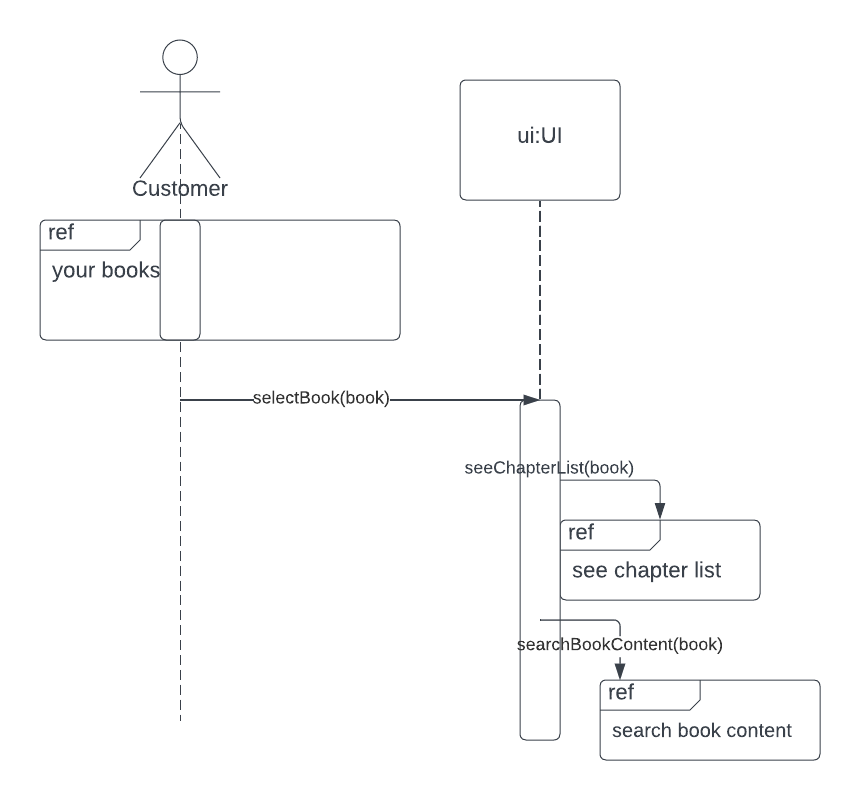


Verify Book Availability

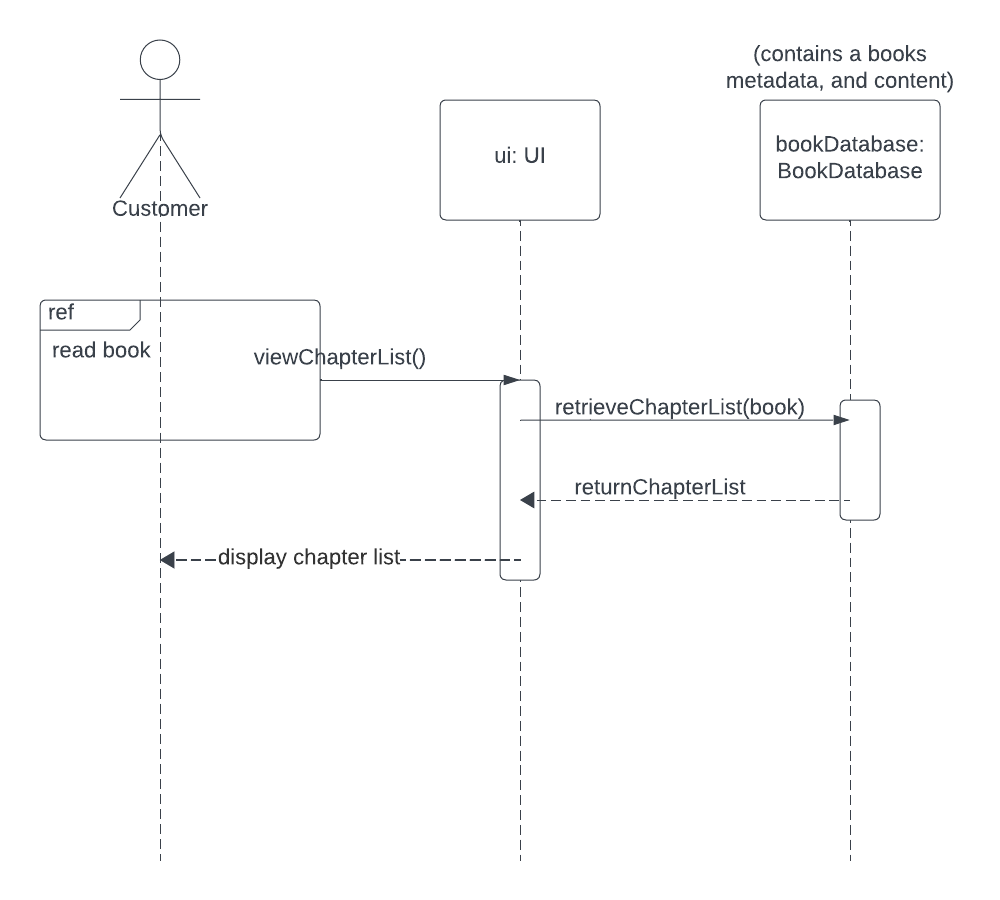
Your Books



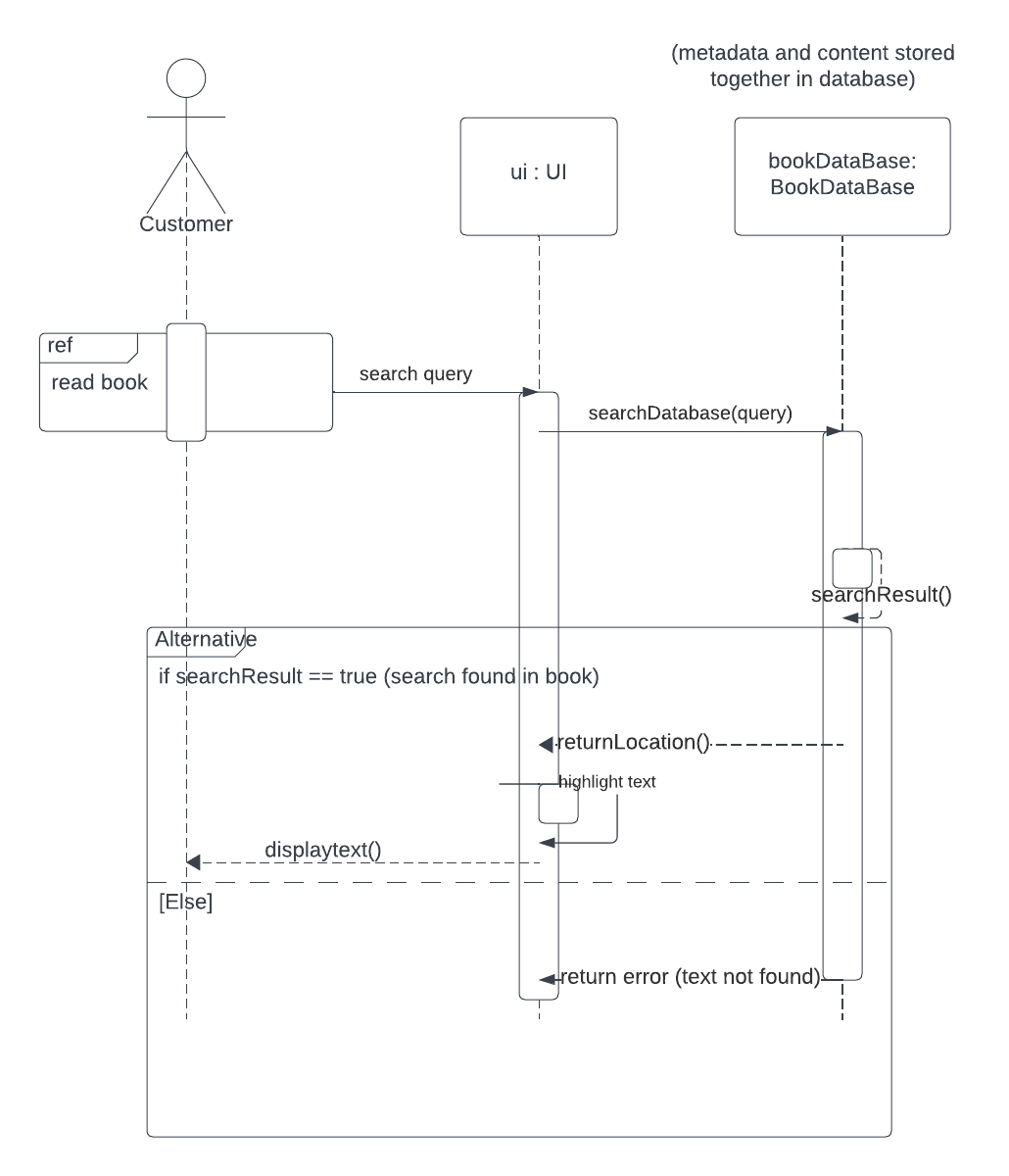
Read Book



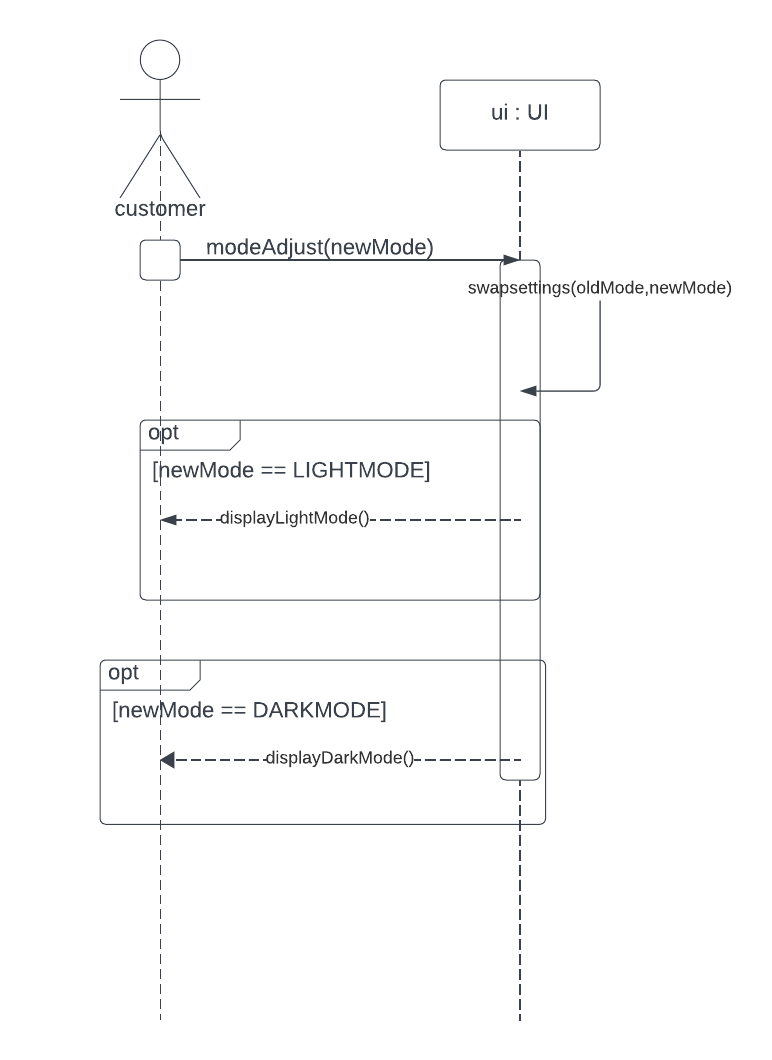
See Chapter List



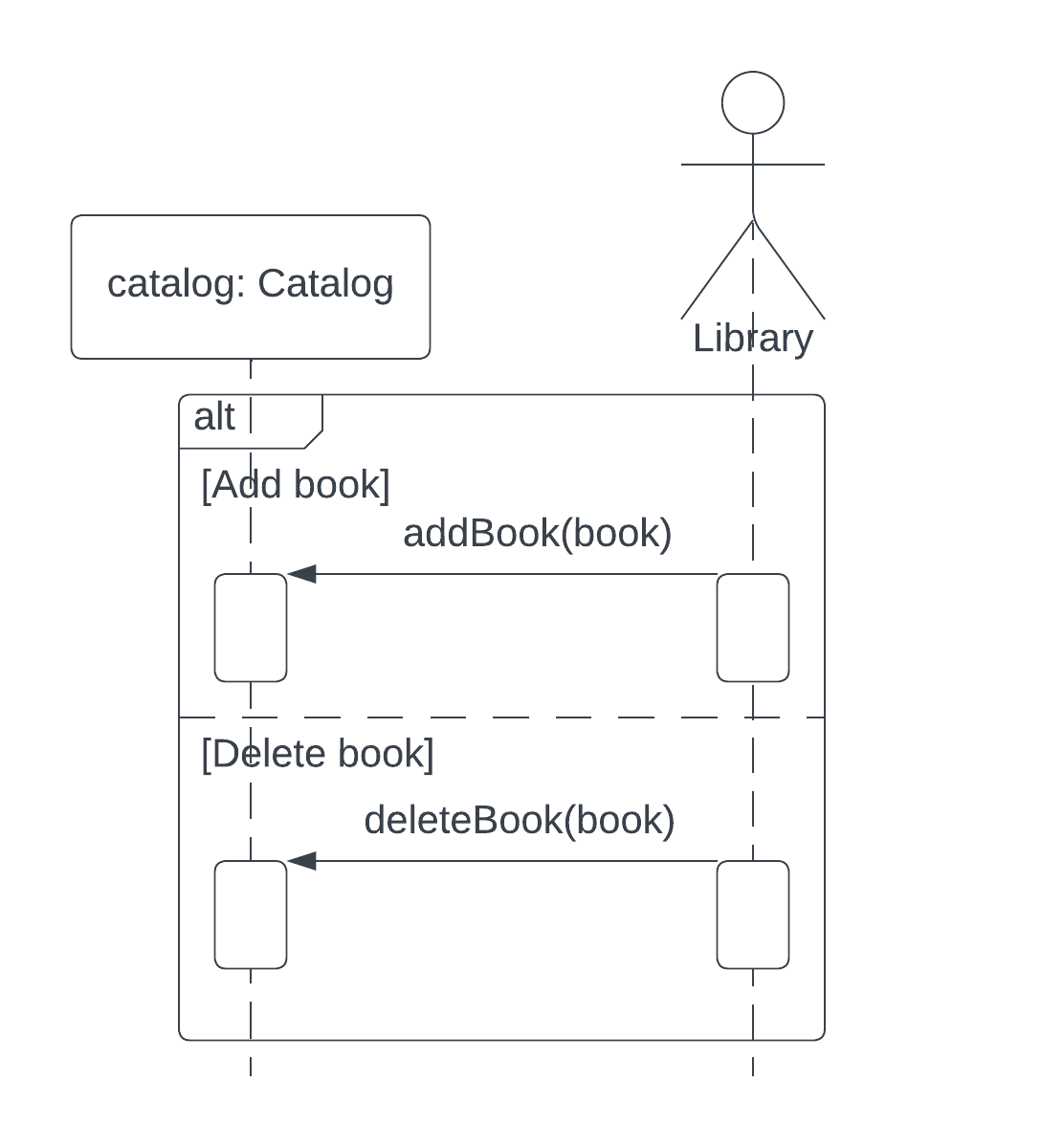
Search Book Content



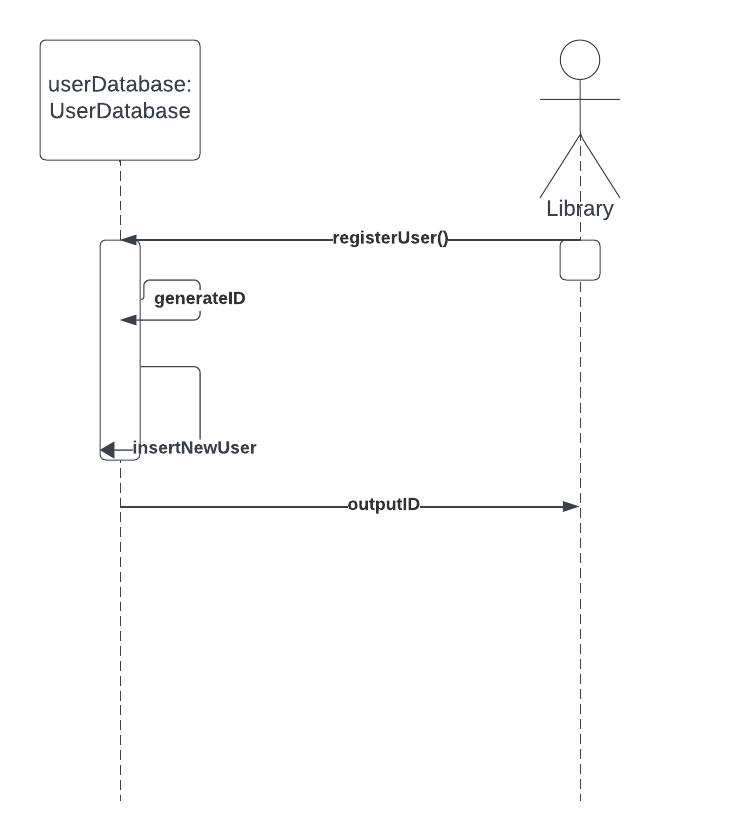
Appearance Options



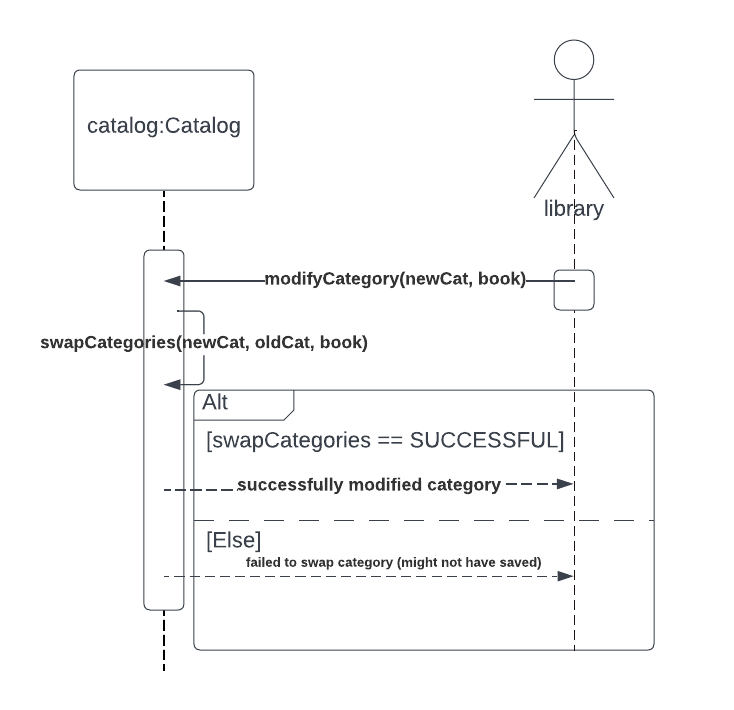
Add/Delete Book



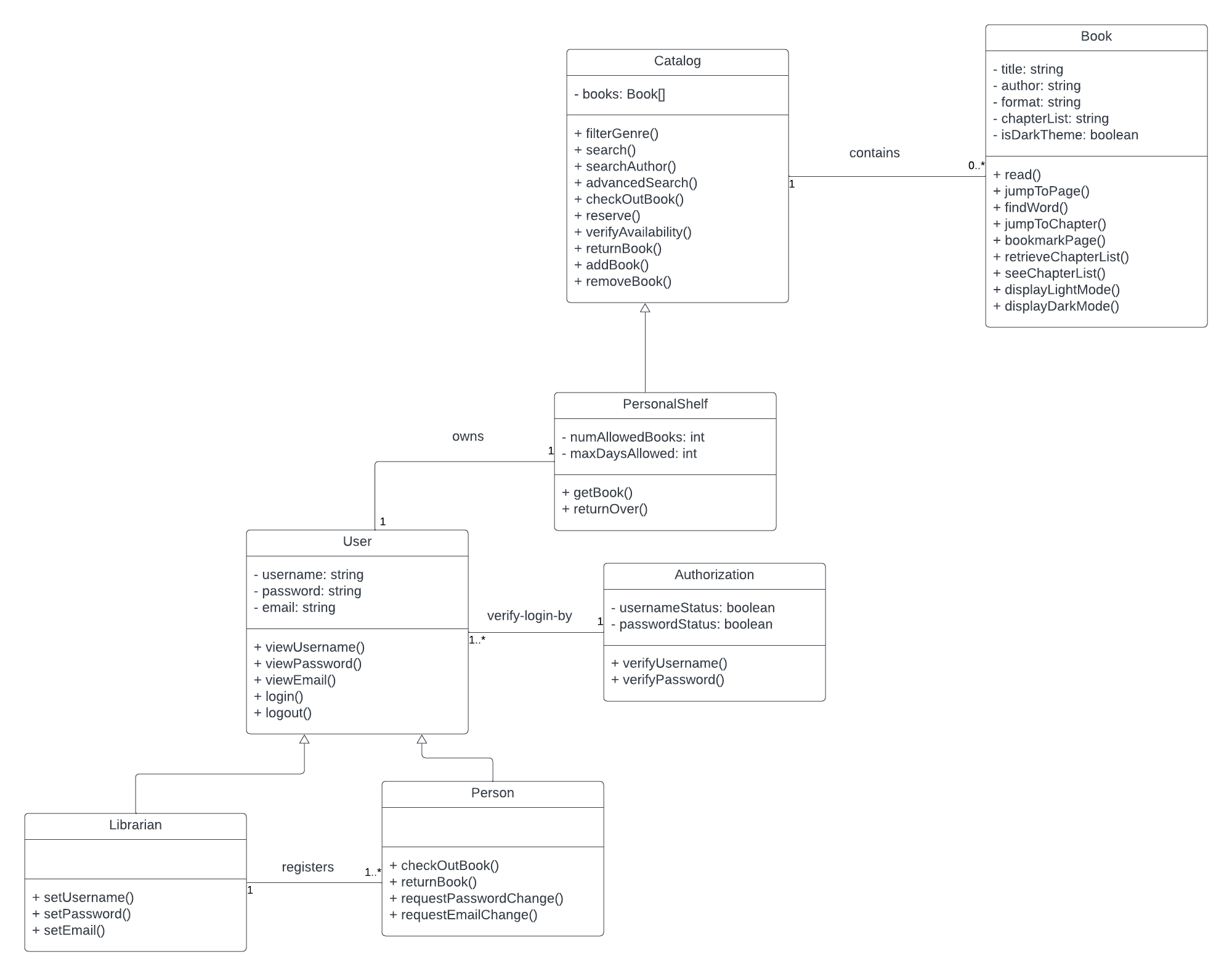
Create Account



Modify Book Categories



**Class Diagram**



# Architectural Design

The Online Library App follows the Client-Server architecture with the different functionalities being provided by multiple servers. In order to provide service from anywhere, the system would need to store data in the cloud as well as locally, accessed through user accounts. Authentication, Search, and Catalog, some of the functions that this app would implement, are considered general services that would be used by any number of clients at the same time and can each be maintained by separate servers as needed.

A few key examples of the importance of this architectural design within this app can be seen with content delivery, user authentication, and other backend processes. This app maintains a digital catalog containing different types of books, such as research materials, journals, and more. This catalog would be hosted on a server, allowing many users access to the app's resources anywhere without having to download everything. Clients (library users) send requests to the server for specific information, such as book availability, article search, etc., and the server processes these requests and sends back the relevant data. In this setup, clients connect to the library's server over the internet to access these services. Additionally, a big part of our app is user access and privileges, based on being either a librarian or just a member, and a client-server architecture is crucial for managing user authentication and access control. Our servers handle user login credentials, permissions, and access rights, ensuring that users can only access resources and services based on their roles and privileges. While a Client-Server architecture may be susceptible to denial-of-service attacks, an Online Library App is an unlikely target for attackers as it does not provide any vital or controversial services.

An alternate suggestion to this architecture was the layered architecture design. The different functionalities of the software would be separated into layers, such as layers for user interface, authentication, database communication logic, and the database itself. While this design might be feasible for the software project, a layered design presents issues with performance in addition to failures between layers as the service requests are interpreted.