**CS 3354 Software Engineering**

**Final Project Deliverable 2**

Online Library App

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

# Task Delegation

Mehdi Devjani

* Deliverable 1
  + Create use-case diagram
* Deliverable 2
  + Create test plan

Anna Ernst

* Deliverable 1
  + Create use-case diagram
* Deliverable 2
  + “Project Scheduling Estimate” section

Chunnu Ghimire

* Deliverable 1
  + Create sequence diagrams
    - Your Books
    - Read Book
    - See Chapters List
    - Search Book Contents
    - Change Appearance
    - Create Accounts
    - Modify Book Categories
* Deliverable 2
  + “Project Staffing Estimate” section

Jonathan Gruber

* Deliverable 1
  + 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
* Deliverable 2
  + “Effort Estimation” section

Kyle Keeton

* Deliverable 1
  + Commit “project\_scope” document to GitHub repository
  + Create class diagram
* Deliverable 2
  + “Estimated Cost of Personnel” section

Stefan Stojanovic

* Deliverable 1
  + Create first commit to the GitHub repository (README file with team name)
  + Create class diagram
* Deliverable 2
  + “Estimated Cost of Hardware Products” section

Hannah Swiney

* Deliverable 1
  + Create architectural design
* Deliverable 2
  + “Estimated Cost of Software Products” section
  + “Pricing Estimation” section

Micah Warner

* Deliverable 1
  + Set up GitHub repository and add all team members and TA as contributors
  + Create architectural design
* Deliverable 2
  + Comparison of our work to similar designs

# Group-Member Contributions

Disclaimer: since, for many of the presentation slides, writing those slides simply amounted to summarizing information already in this report document, there was no attempt to divide the work of doing those slides evenly amongst the group members. Group members instead worked on as many slides as they had time for, hence some group members worked on considerably more slides than did other group members.

Mehdi Devjani

* Deliverable 1
  + Deciding upon architectural design
  + Use-case diagram
* Deliverable 2
  + Test plan
  + Presentation slides
    - “Test Plan” and all “Test Plan (cont.)” slides

Anna Ernst

* Deliverable 1
  + Deciding upon architectural design
  + Use-case diagram
* Deliverable 2
  + “Project Scheduling Estimate” section
  + Presentation slides:
    - “Project Timeline”

Chunnu Ghimire

* Deliverable 1
  + Deciding upon architectural design
  + Sequence diagrams
    - Your Books
    - Read Book
    - See Chapters List
    - Search Book Contents
    - Change Appearance
    - Create Accounts
    - Modify Book Categories
* Deliverable 2
  + “Project Staffing Estimate” section
  + Presentation slides
    - “Staffing Estimation”

Jonathan Gruber

* Deliverable 1
  + 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
* Deliverable 2
  + “Effort Estimation” section
  + “Conclusion” section
  + Presentation slides
    - “Project Objective”
    - “Representative Functional Requirements”
    - “Representative Non-Functional Requirements”
    - “Use-Case Diagram”
    - “Sequence Diagram of Representative Operation”
    - “Sequence Diagram of Representative Operation (cont.)”
    - “Class Diagram”
    - “Architectural Design”

Kyle Keeton

* Deliverable 1
  + 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
* Deliverable 2
  + “Estimated Cost of Personnel” section
  + Presentation slides
    - “Cost Estimation – Staffing”
    - “Cost Estimation – Training”
    - “Cost Estimation – Personnel"

Stefan Stojanovic

* Deliverable 1
  + 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
  + Class diagram (collaborated with Kyle Keeton)
    - User
    - Librarian
    - Person
    - Authorization
    - Light/Dark mode and Chapter List functions/attributes in Book
* Deliverable 2
  + Report on estimated cost of hardware
  + “Estimated Cost of Hardware Products” section
  + “Conclusion” section
  + Presentation slides
    - “Cost Estimation – Hardware Products”
    - “Cost Estimation – Hardware Products (cont.)”

Hannah Swiney

* Deliverable 1
  + Deciding upon architectural design
  + Addressing feedback
  + “Architectural Design” section of this report
* Deliverable 2
  + “Estimated Cost of Software Products” section
  + Presentation slides
    - “Cost Estimation – Software Products”

Micah Warner

* Deliverable 1
  + Set up GitHub repository and added all team members and TA as contributors
  + Deciding upon architectural design
  + “Architectural Design” section of this report
* Deliverable 2
  + Comparison of our work to similar designs
  + Presentation slides
    - “Comparison to Similar Work”

# Project Deliverable 1 Content

Sections “Addressing of Feedback to Project Proposal” through “Architectural Design” constitute the content from the Project Deliverable 1 report.

# 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.

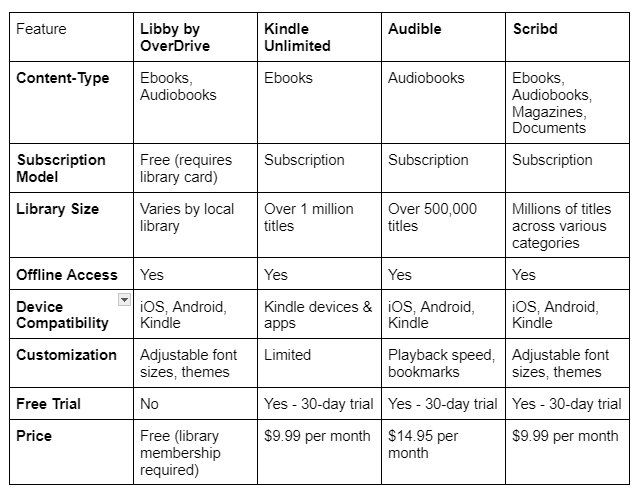
It is great to see a detailed breakdown of the tasks you have considered already. Good job.

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 (Deliverable 1)

Mehdi Devjani

* Deliverable 1
  + Create use-case diagram
* Deliverable 2
  + Create test plan

Anna Ernst

* Deliverable 1
  + Create use-case diagram
* Deliverable 2
  + “Project Scheduling Estimate” section

Chunnu Ghimire

* Deliverable 1
  + Create sequence diagrams
    - Your Books
    - Read Book
    - See Chapters List
    - Search Book Contents
    - Change Appearance
    - Create Accounts
    - Modify Book Categories
* Deliverable 2
  + “Project Staffing Estimate” section

Jonathan Gruber

* Deliverable 1
  + 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
* Deliverable 2
  + “Effort Estimation” section

Kyle Keeton

* Deliverable 1
  + Commit “project\_scope” document to GitHub repository
  + Create class diagram
* Deliverable 2
  + “Estimated Cost of Personnel” section

Stefan Stojanovic

* Deliverable 1
  + Create first commit to the GitHub repository (README file with team name)
  + Create class diagram
* Deliverable 2
  + “Estimated Cost of Hardware Products” section

Hannah Swiney

* Deliverable 1
  + Create architectural design
* Deliverable 2
  + “Estimated Cost of Software Products” section
  + “Pricing Estimation” section

Micah Warner

* Deliverable 1
  + Set up GitHub repository and add all team members and TA as contributors
  + Create architectural design
* Deliverable 2
  + Comparison of our work to similar designs

# Group-Member Contributions (Deliverable 1)

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
* Class diagram (collaborated with Kyle Keeton)
  + 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 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 log in with their username and password and log out of the application when finished. *(A necessary function of an account system, this also creates instances in the system and ensures none are taking up space and processing power when not in use.)*
3. 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.)*
4. 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.)*
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 they are preferred over physical books.)*
6. A customer shall be able 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 sizes 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.)*

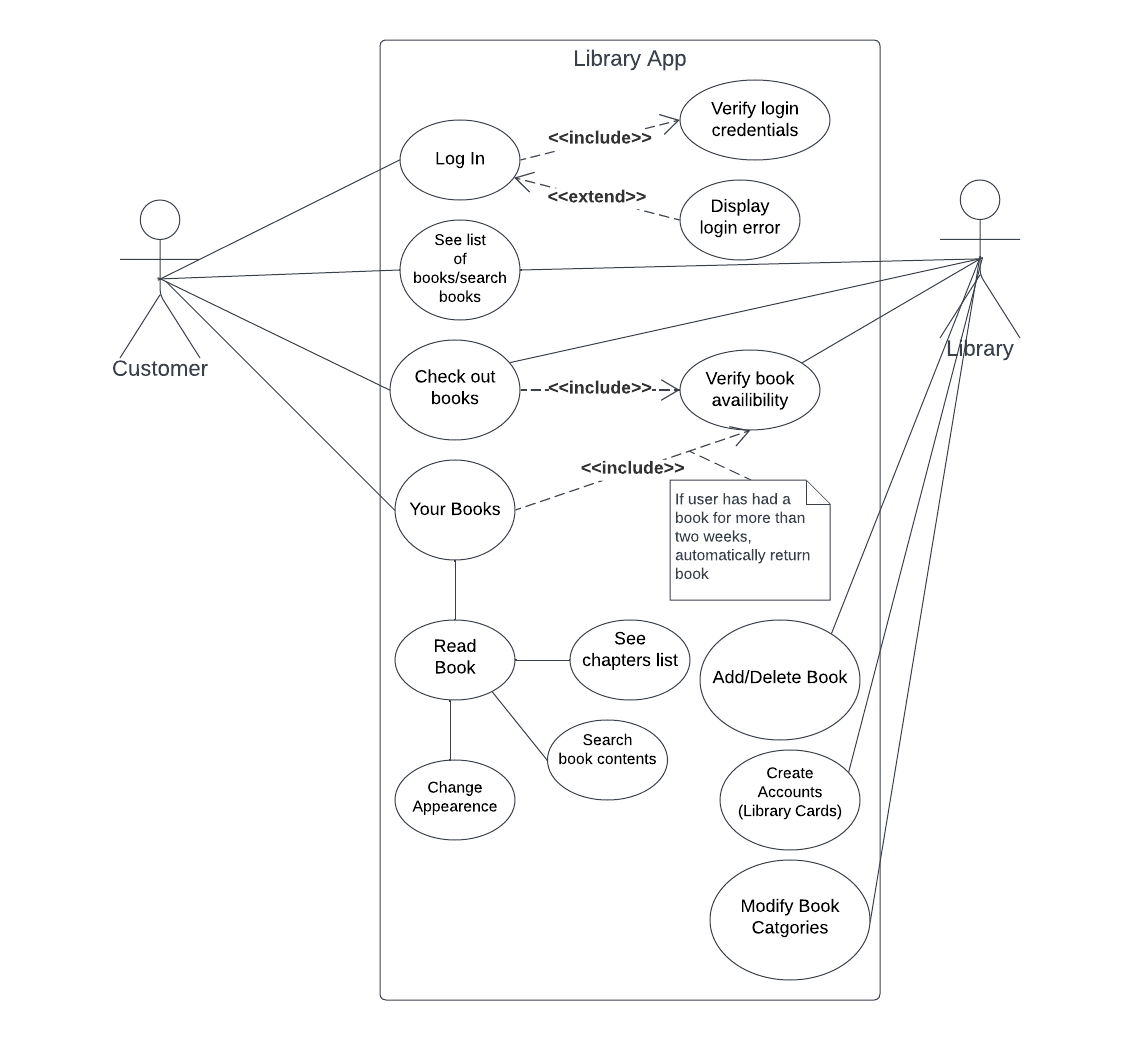
### Organizational requirements

* 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

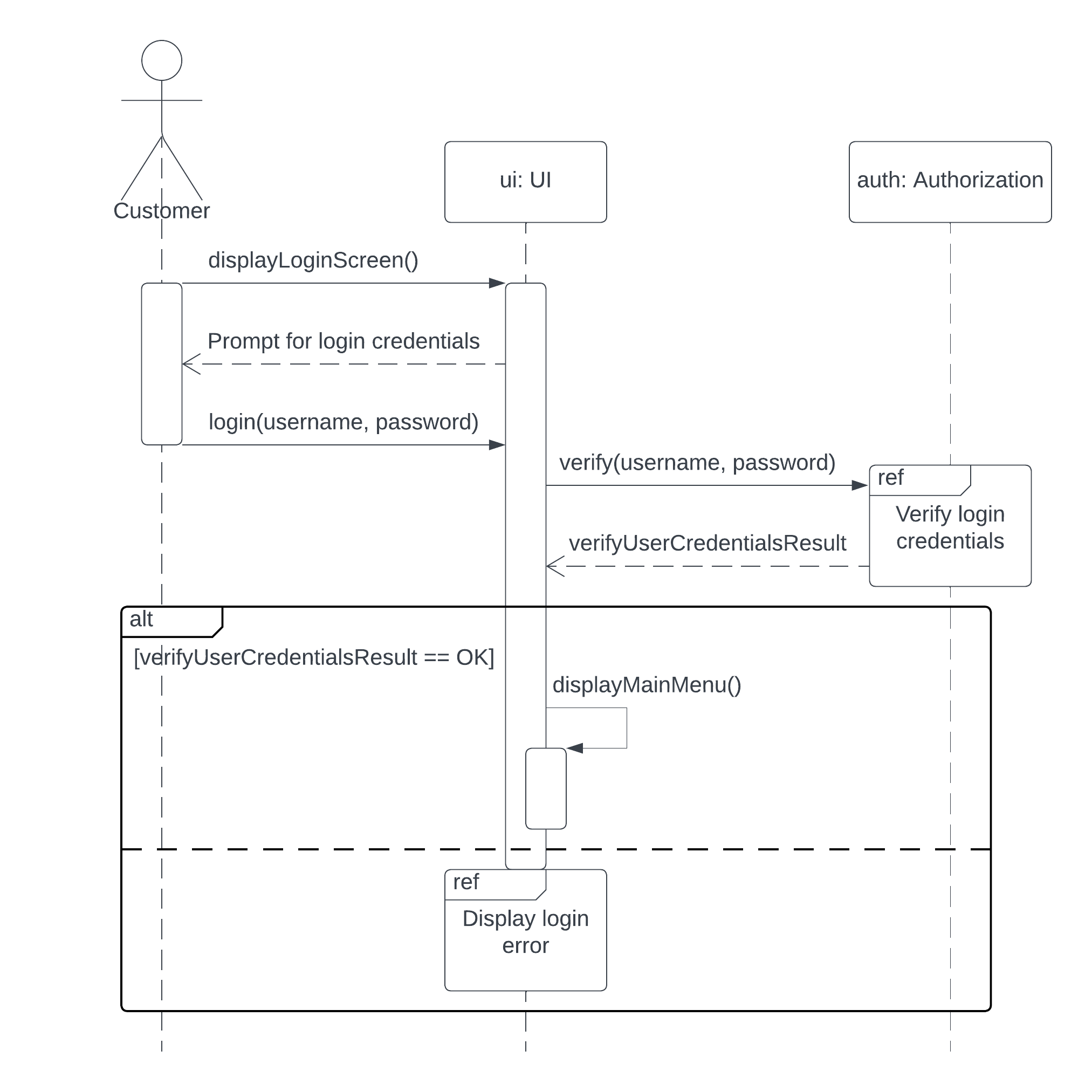
* Regulatory requirements
  + This app must comply with the California Consumer Privacy App (CCPA), Children’s Online Privacy App (COPA), and any similar US regulations that may arise during the development of the app. *(This allows the application to be used in any state in the United States. We do 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 similar US 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

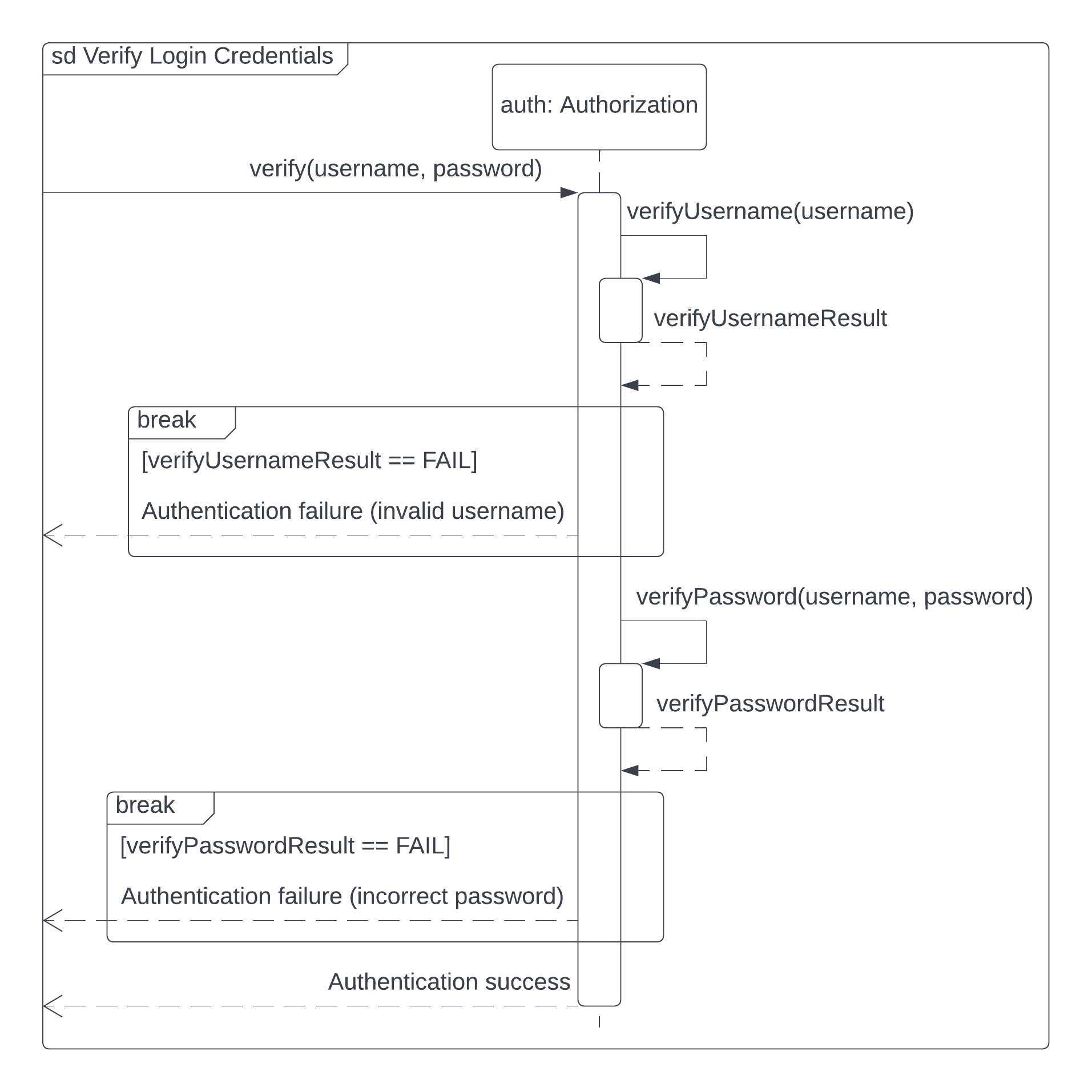


# Sequence Diagrams

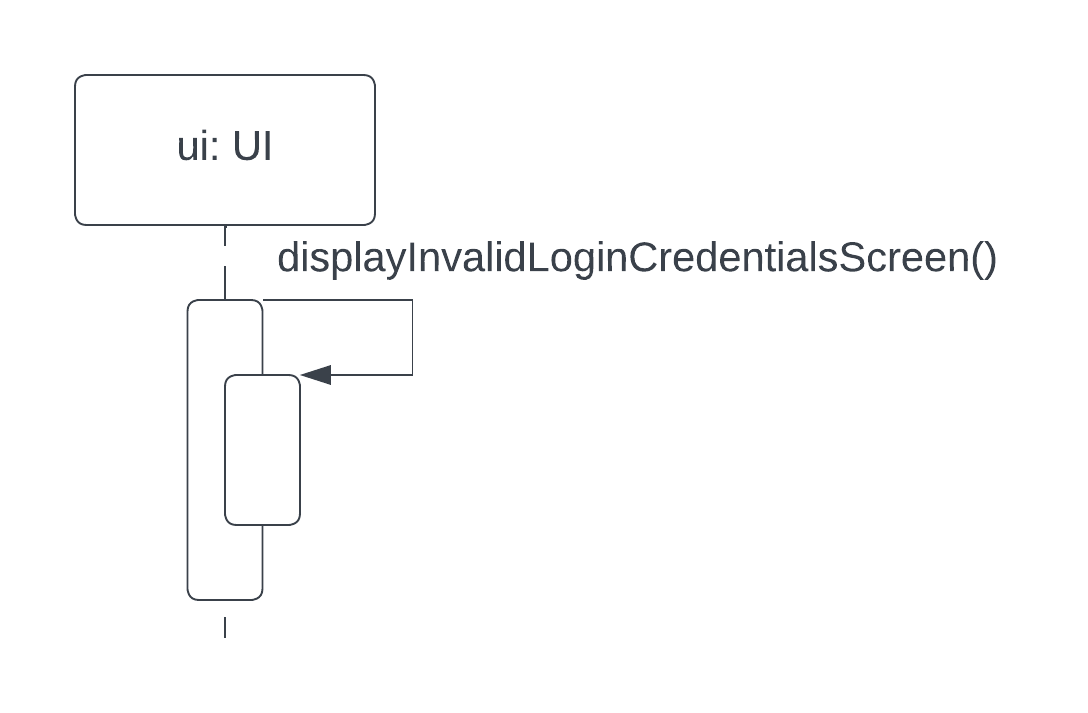
Log In



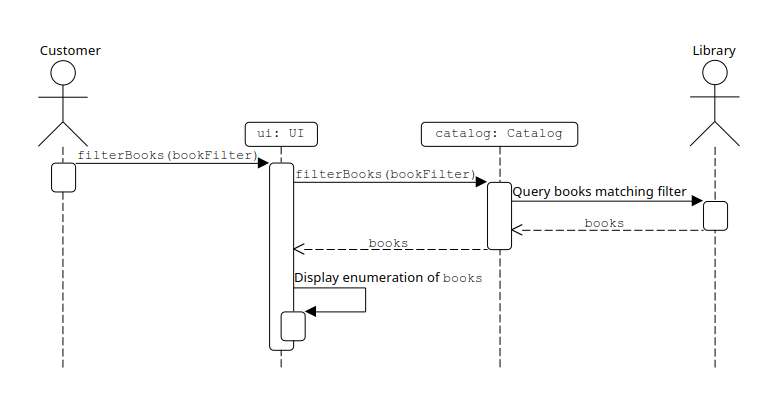
Verify Login Credentials



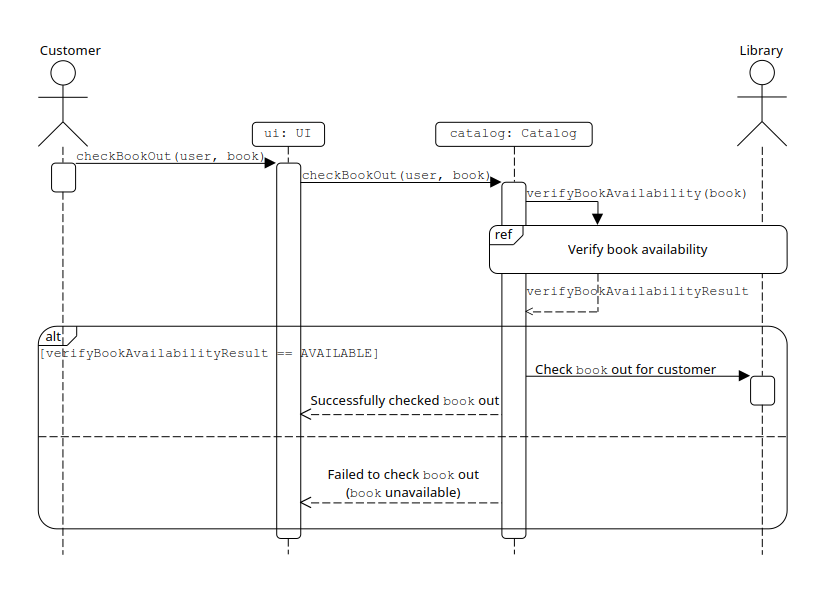
Display Login Error



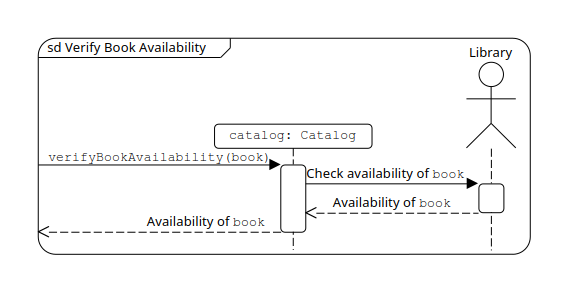
See List of Books/Search Books

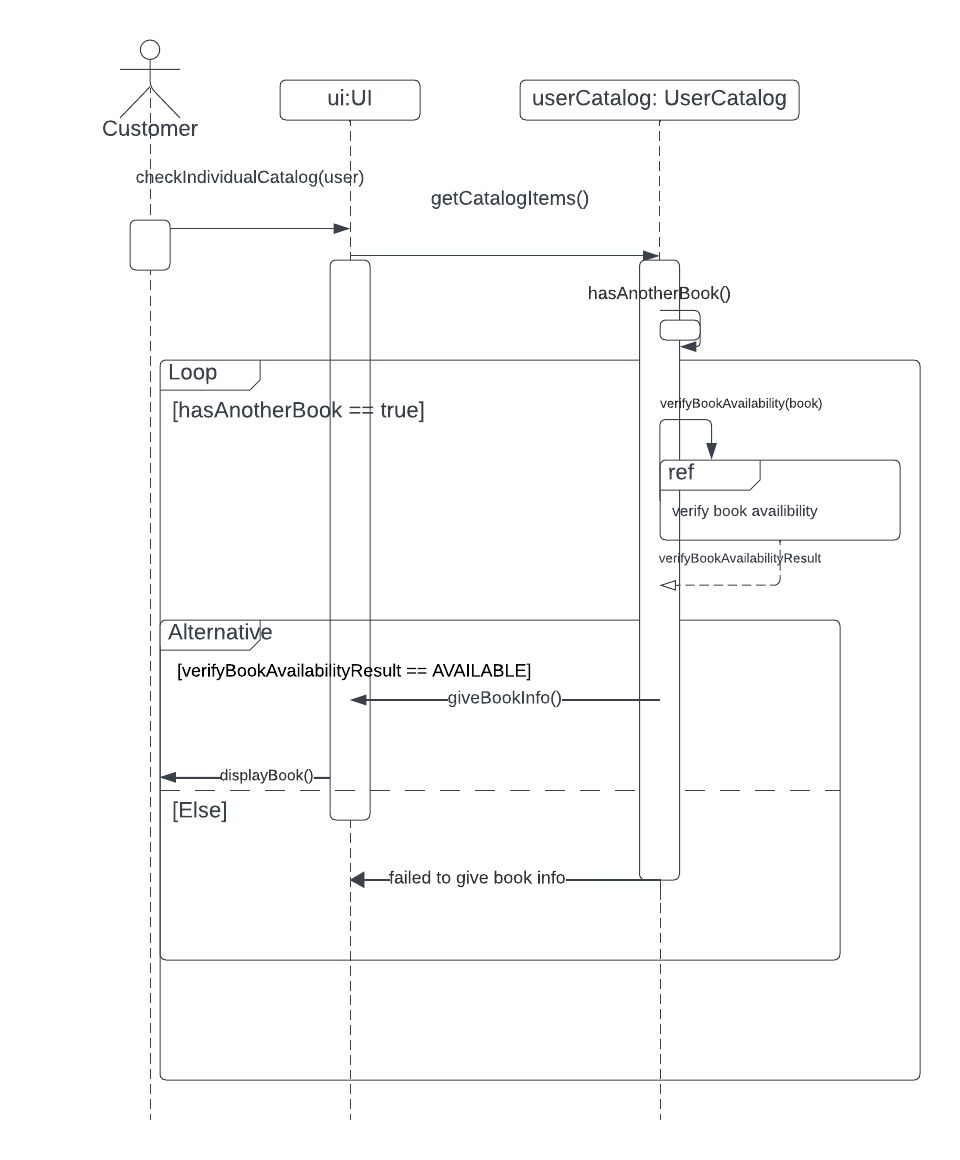


Check out Books

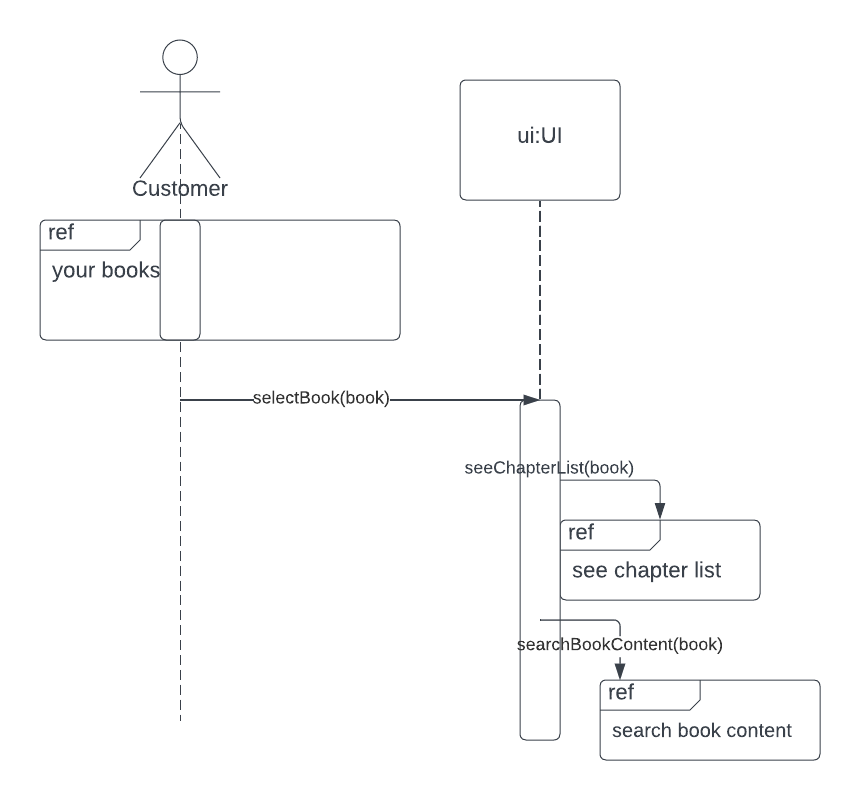


Verify Book Availability

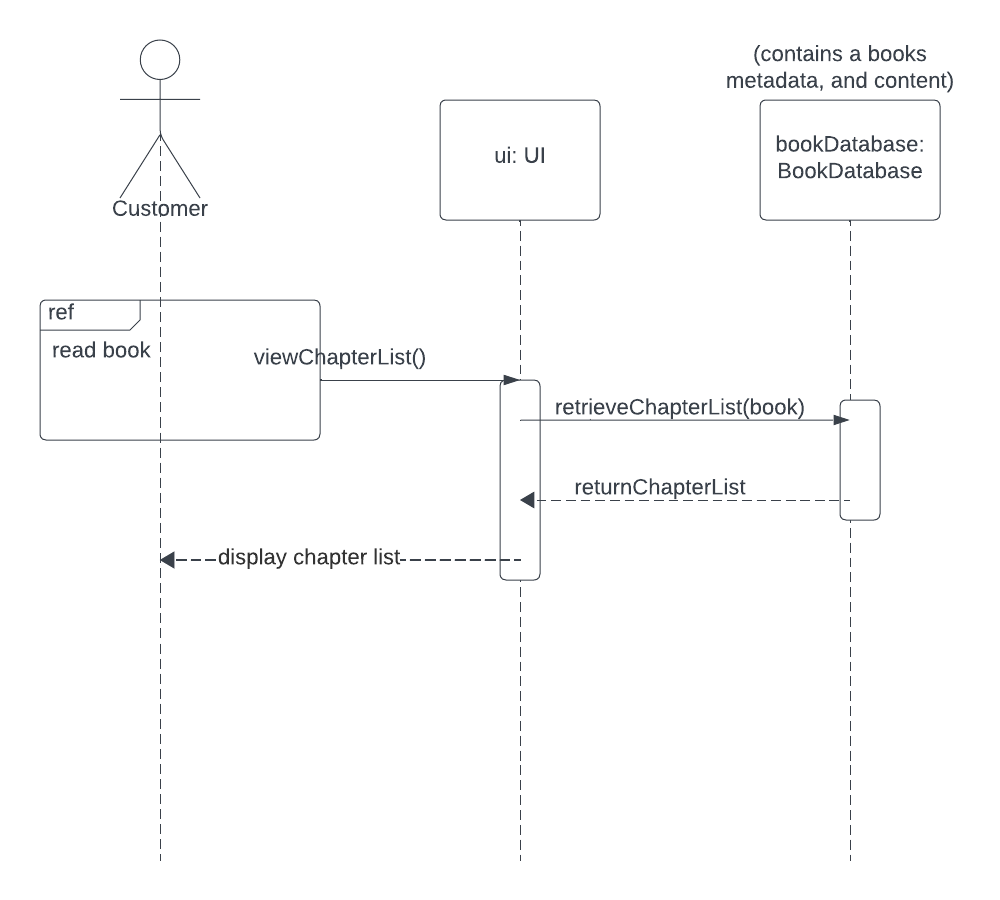
Your Books



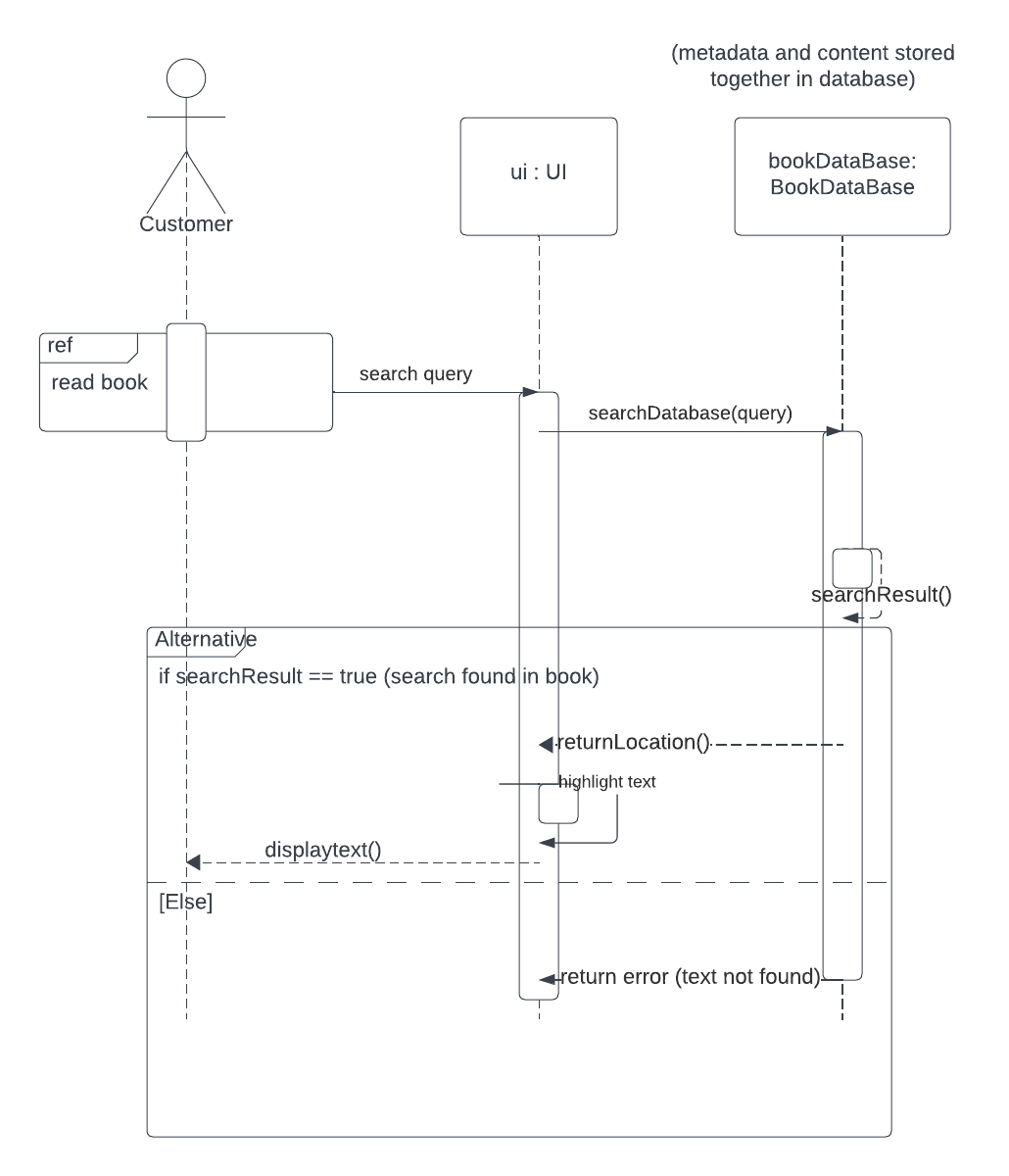
Read Book



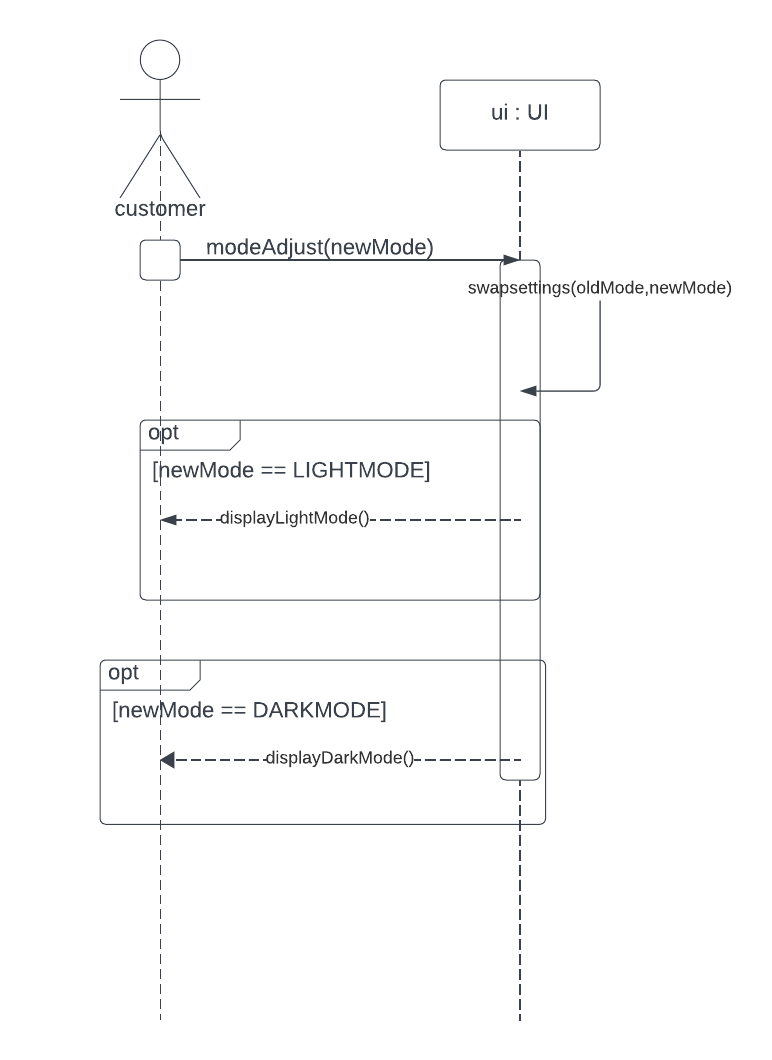
See Chapter List



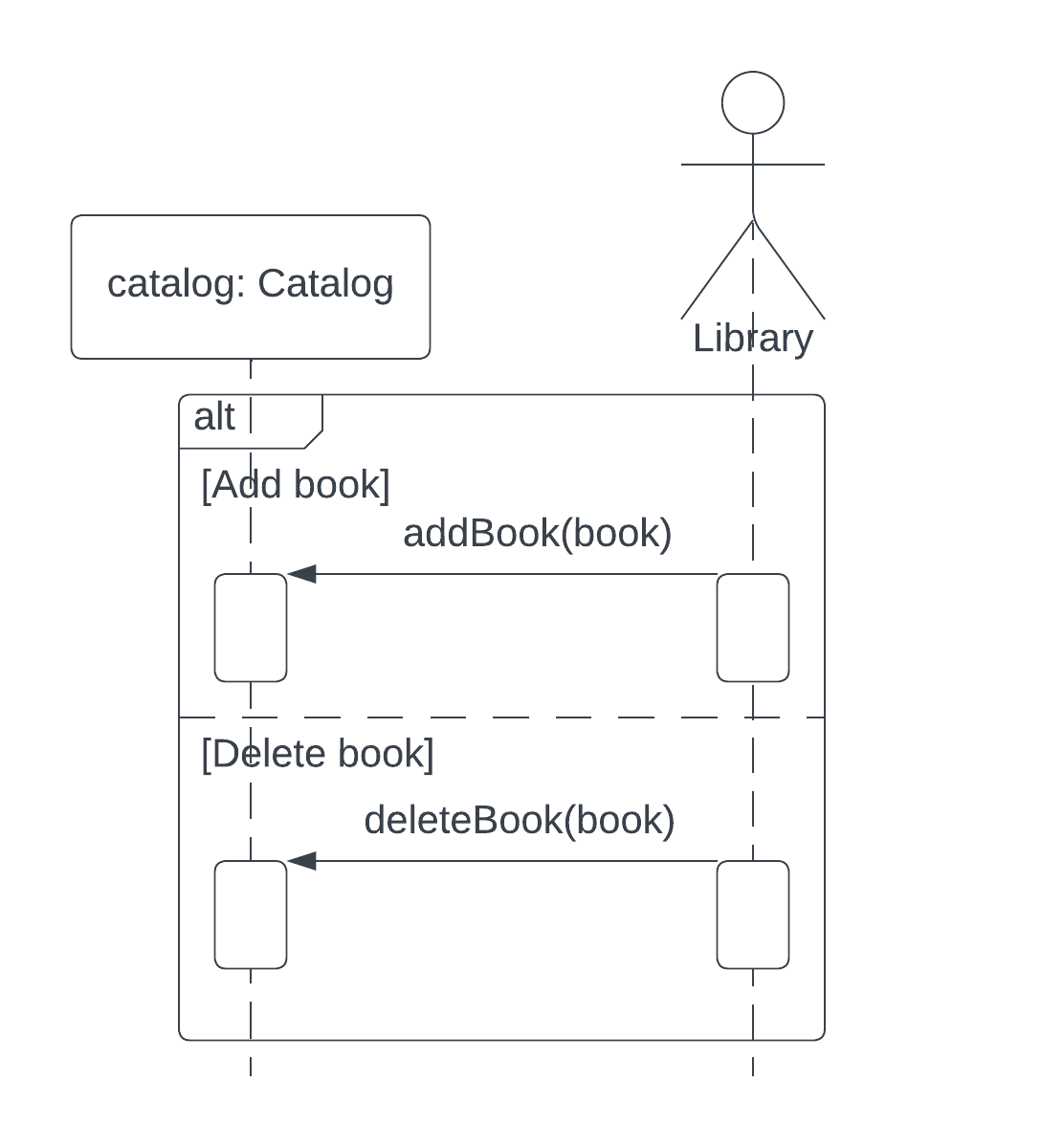
Search Book Content



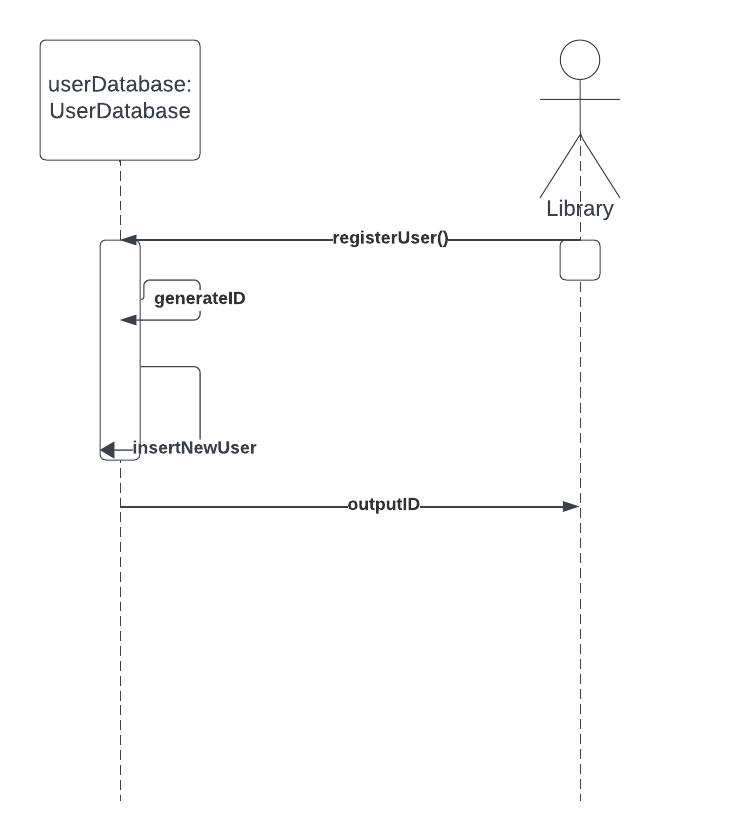
Appearance Options



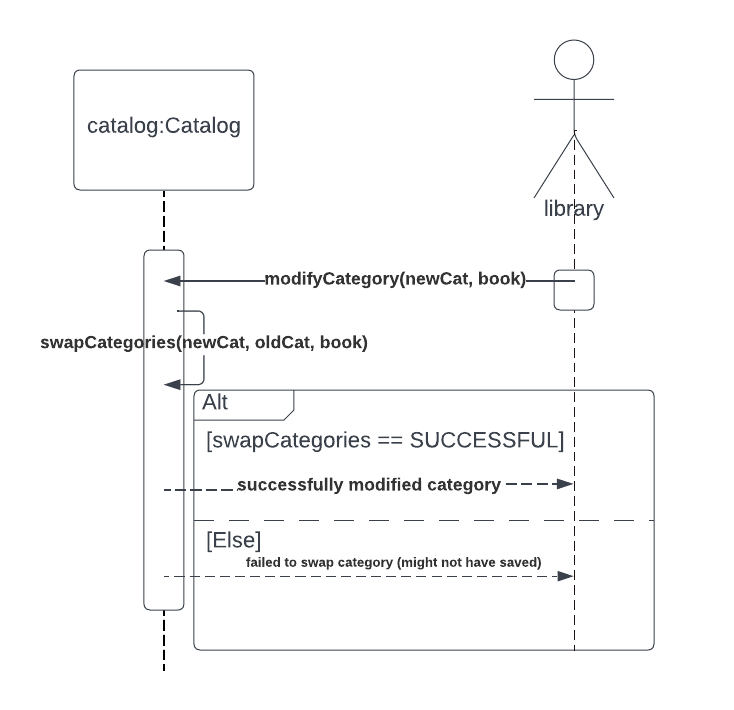
Add/Delete Book



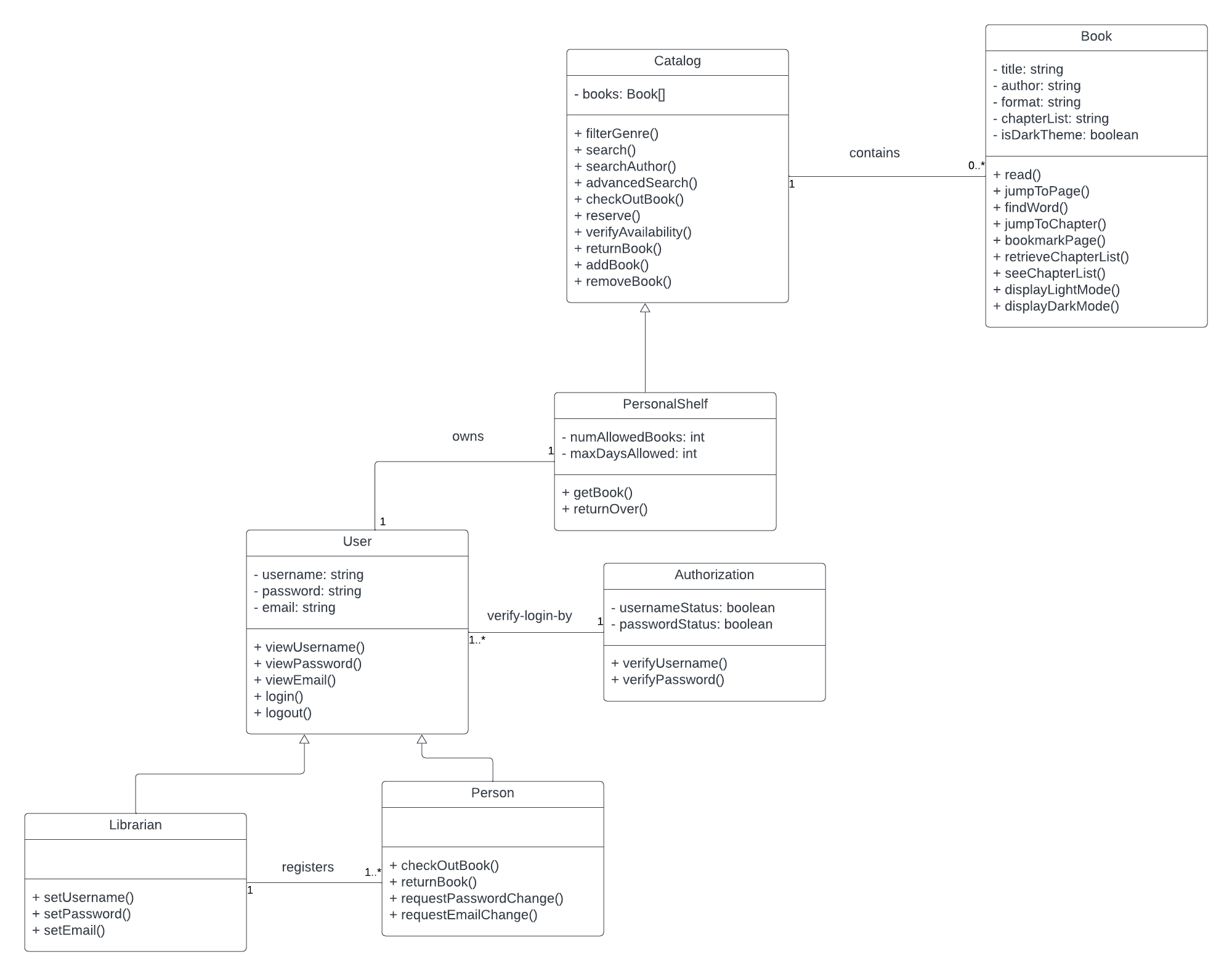
Create Accounts



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, browsing a library catalog, and searching for books in said 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.

# Project Scheduling, Staffing, Cost, Effort, and Pricing Estimation

## Effort Estimation

The Application Composition Model is intended for the early stages of the software development life cycle, for calculating an estimate of the effort that constructing a prototype would require. Were we to actually set about implementing our project, then we would currently be in the early stages of the software development life cycle, and constructing a working prototype might be a reasonable first step to developing the project in full. Additionally, the Application Composition Model is part of COCOMO II, which is more comprehensive than the Function Point Model. For all these reasons, we have chosen to use the Application Composition Model, rather than the Function Point Model, for estimating the effort required for this software project.

Based on the use-case and sequence diagrams and the functional and non-functional requirements for the project, we assume the following composition for the software.

10 screens

1. Customer login screen
   * The screen through which a customer logs in to their account
   * Corresponds to the “Log In”, “Verify Login Credentials”, and “Display Login Error” use-cases
   * “Verify Login Credentials” is not customer-facing and so does not constitute a separate screen or view, and, since it is only triggered by “Log In”, it is part of the same screen as “Log In”
   * “Display Login Error” is only triggered by “Log In” and is functionally inseparable from “Log In” (since it merely notifies the customer that their login credentials are invalid), and hence it is part of the same screen as “Log In”
   * “Display Login Error” does not constitute a separate usage mode of the screen from “Log In”, since it just notifies the customer that their login credentials are invalid, and so does not constitute a separate view of the screen
   * 1 view
   * 1 data table
     + Reads the table of user authentication information via the authentication server in order to verify the customer’s login credentials
   * Complexity level: simple
     + 1 view < 3 views
     + 1 data table < 4 data tables
2. Customer main menu
   * The application’s main menu for a customer
   * The screen displayed to a customer once they successfully log in to their account
   * This screen is functionally self-contained and does not have separate usage modes, so it has only one view
   * 1 view
   * 1 data table
     + Reads the table of the customer’s appearance settings in order to display the screen accordingly
   * Complexity level: simple
     + 1 view < 3 views
     + 1 data table < 4 data tables
3. See list of books/search books
   * The screen through which a customer browses the library catalog, possibly filtering for books that match certain criteria
   * Corresponds to the use-case of the same name
   * Browsing the library catalog without supplying criteria by which to filter the books is essentially browsing the catalog with an empty set of criteria that matches all books, so these two modes of browsing are not substantially different enough to constitute two separate usage modes or views of this screen
   * 1 view
   * 2 data tables
     + Reads the table of the customer’s appearance settings in order to display the screen accordingly
     + Reads the table of books in the library’s book database in order to determine which books are in the library’s catalog and which of those books match the customer-supplied search criteria
   * Complexity level: simple
     + 1 view < 3 views
     + 2 data tables < 4 data tables
4. Book transactions
   * The screen through which a customer checks out or reserves a book
   * Checking out and reserving a book are distinct functions, so each corresponds to a separate usage mode, or view, of this screen
   * 2 views
     + Check out book
       - Corresponds to the “Check out Books” use-case, together with the “Verify Book Availability” use-case
       - “Verify Book Availability” is not customer-facing and so is neither a separate screen nor view, and it is triggered by the “Check out Books” use-case and so is part of the same view
       - Note that “Verify Book Availability” is part of both this view and the “Your Books” screen below
     + Reserve book
       - Corresponds to Functional Requirement No. 6
   * 3 data tables
     + Reads the table of the customer’s appearance settings in order to display the screen accordingly
     + Reads the table of books in the library’s book database in order to verify a book’s availability for checkout or reservation, and possibly writes to that table in order to mark the book as checked out or reserved by the customer
     + Possibly writes to the table of the customer’s book-transaction history in order to record a book as newly checked out or reserved by the customer
   * Complexity level: simple
     + 2 view < 3 views
     + 3 data tables < 4 data tables
5. Your books
   * The screen through which a customer views a list of the books that they have currently checked out from the library, a list of the books that they have reserved, or a list of the books that were previously checked out by them but have been automatically returned to the library upon expiration of their checkout periods
   * Roughly corresponds to the use-case of the same name, together with the “Verify Book Availability” use-case
     + “Verify Book Availability” is not customer-facing and so is neither a separate screen nor view, and it is triggered by the “Your Books” use-case and so is part of the same screen
     + Note that “Verify Book Availability” is part of both this screen and the “Check out Book” view of the “Book Transactions” screen above
   * Viewing any of the three lists of books mentioned in this screen’s description is functionally distinct from, though related to, viewing any of the other two lists and so constitutes a separate usage mode, and hence view, of this screen
   * 3 views
     + View the list of books currently checked out by the customer
     + View the list of books reserved by the customer
     + View the list of books previously checked out by the customer that have been automatically returned to the library upon expiration of their checkout periods
   * 2 data tables
     + Reads the table of the customer’s appearance settings in order to display the screen accordingly
     + Reads the table of the customer’s book-transaction history in order to determine
       - What books they have checked out
       - What books they have reserved
       - What books previously checked out by them have been automatically returned to the library upon expiration of their checkout periods
   * Complexity level: simple
     + 3 views ≤ 3 views ≤ 7 views
     + 2 data tables < 4 data tables
6. Read book
   * The screen through which a customer reads the contents of a book, views the book’s chapter list, or searches the book’s contents
   * Corresponds to the use-case of the same name and the “See Chapter List” and “Search Book Contents” use-cases
   * The “Read Book”, “See Chapter List”, and “Search Book Contents” use-cases are functionally distinct, though related, and so each constitutes a separate usage mode, and hence view, of this screen
   * 3 views
     + Read the contents of the book
     + View the book’s chapter list
     + Search the book’s contents
   * 2 data tables
     + Reads the table of the customer’s appearance settings in order to display the screen accordingly
     + Reads the table of data for the book being read in order to perform the actions required by the above 3 views
   * Complexity level: simple
     + 3 views ≤ 3 views ≤ 7 views
     + 2 data tables < 4 data tables
7. Appearance options
   * The screen through which a customer sets appearance options for the application
   * Corresponds to the “Change Appearance” use-case
   * This screen is functionally self-contained and does not have separate usage modes, so it has only one view
   * 1 view
   * 1 data table
     + Reads the table of the customer’s appearance settings in order to both display the screen accordingly and display the current appearance settings, and writes to that table in order to update the customer’s appearance settings
   * Complexity level: simple
     + 1 view < 3 views
     + 1 data table < 4 data tables
8. Add/delete book
   * The screen through which a library administrator adds a book to or deletes a book from the library catalog
   * Corresponds to the “Add/Delete Book” use-case
   * Adding a book to and deleting a book from the library catalog are functionally distinct, so each corresponds to a separate usage mode or view of this screen
   * 2 views
     + Adding a book
     + Deleting a book
   * 1 data table
     + Writes to the table of books in library’s book database in order to perform the operations required by the above 2 views
   * Complexity level: simple
     + 2 views < 3 views
     + 1 data table < 4 data tables
9. Create customer account
   * The screen through which a library administrator creates an account for a customer
   * Corresponds to the “Create Accounts” use-case
   * This screen is functionally self-contained and does not have separate usage modes, so it has only one view
   * 1 view
   * 1 data table
     + Writes to the table of customer accounts in the database of customer accounts in order to add a newly created customer account
   * Complexity level: simple
     + 1 view < 3 views
     + 1 data table < 4 data tables
10. Modify book categories
    * The screen through which a library administrator modifies book categories
    * Corresponds to the use-case of the same name
    * This screen is functionally self-contained and does not have separate usage modes, so it has only one view
    * 1 view
    * 1 data table
      + Writes to the table of books in the library’s book database in order to modify book categories
    * Complexity level: simple
      + 1 view < 3 views
      + 1 data table < 4 data tables

11 reports

1. Result (successful/unsuccessful) of a customer’s login attempt
   * Output by the customer login screen
   * 1 section, since that is sufficient for expressing success or failure to log in
   * 1 data table
     + Reads the table of user authentication information via the authentication server in order to verify the customer’s login credentials
   * Complexity level: simple
     + 1 section < 2 sections
     + 1 data table < 4 data tables
2. Books in catalog, possibly filtered by customer-supplied search criteria
   * Output by the “See List of Books/Search Books” screen
   * 1 section, since that is sufficient for expressing a list of books
   * 1 data table
     + Reads the table of books in the library’s book database in order to determine which books are in the library’s catalog and which of those books match the customer-supplied search criteria
   * Complexity level: simple
     + 1 section < 2 sections
     + 1 data table < 4 data tables
3. Result (successful/unsuccessful) of a customer’s book checkout attempt
   * Output by the “Check out Book” view of the “Book Transactions” screen
   * 1 section, since that is sufficient for expressing success or failure to check out a book
   * 1 data table
     + Reads the table of books in the library’s book database in order to verify the book’s availability for checkout
   * Complexity level: simple
     + 1 section < 2 sections
     + 1 data table < 4 data tables
4. Result (successful/unsuccessful) of a customer’s book reservation attempt
   * Output by the “Reserve Book” view of the “Book Transactions” screen
   * If the book reservation was successful, then
     + If the duration until when the book will be automatically checked out to the customer can be determined, then that duration is reported to the customer
     + If that duration cannot be determined, then this fact is reported to the customer
   * 1 section, since that is sufficient for expressing both
     + Success or failure to reserve a book, and
     + In the case of success, either the duration until when a book shall be automatically checked out to the customer or the fact that such a duration is indefinite
   * 1 data table
     + Reads the table of books in the library’s book database in order to verify the book’s availability for reservation
   * Complexity level: simple
     + 1 section < 2 sections
     + 1 data table < 4 data tables
5. Books in a customer’s “Your Books” screen
   * Output by the “Your Books” screen
   * 3 sections
     + Books checked out by the customer
     + Books reserved by the customer
     + Books previously checked out by the customer that have been returned to the library upon expiration of their checkout periods
   * 1 data table
     + Reads the table of the customer’s book-transaction history in order to determine
       - What books they have checked out
       - What books they have reserved
       - What books previously checked out by them have been automatically returned to the library upon expiration of their checkout periods
   * Complexity level: simple
     + 2 sections ≤ 3 sections ≤ 3 sections
     + 1 data table < 4 data tables
6. Contents of a book
   * Output by the “Read the Contents of the Book” view of the “Read Book” screen
   * 1 section, since that is sufficient for expressing the contents of a book
   * 1 data table
     + Reads the table of data for the book in order to obtain the book’s contents
   * Complexity level: simple
     + 1 section < 2 sections
     + 1 data table < 4 data tables
7. List of a book’s chapters
   * Output by the “View the Book’s Chapter List” view of the “Read Book” screen
   * 1 section, since that is sufficient for expressing the chapter list of a book
   * 1 data table
     + Reads the table of data for the book in order to obtain a list of the book’s chapters
   * Complexity level: simple
     + 1 section < 2 sections
     + 1 data table < 4 data tables
8. Book contents matching customer-supplied search criteria
   * Output by the “Search the Book’s Contents” view of the “Read Book” screen
   * 1 section, since that is sufficient for expressing the contents of a book that match the search criteria
   * 1 data table
     + Reads the table of data for the book in order to determine the parts of its contents that match the search criteria
   * Complexity level: simple
     + 1 section < 2 sections
     + 1 data table < 4 data tables
9. Customer’s current appearance settings
   * Output by the “Appearance Options” screen
   * We assume the set of appearance options to be simple or small enough that it can be sufficiently expressed in one section
   * 1 section
   * 1 data table
     + Reads the table of the customer’s appearance settings in order to determine the current appearance settings
   * Complexity level: simple
     + 1 section < 2 sections
     + 1 data table < 4 data tables
10. User ID of a newly created customer account
    * Output by the “Create Accounts” screen
    * 1 section, since that is sufficient for expressing a user ID
    * 0 data tables, since the software already has the ID of a new customer account before it adds that account to the table of customer accounts in the database of customer accounts
    * Complexity level: simple
      + 1 section < 2 sections
      + 0 data tables < 4 data tables
11. Result (successful/unsuccessful) of attempting to modify a book category
    * Output by the “Modify Book Categories” screen
    * 1 section, since that is sufficient for expressing success or failure to modify a book category
    * 0 data tables, since the result (successful/unsuccessful) of attempting to modify a book category is not stored in any data table
    * Complexity level: simple
      + 1 section < 2 sections
      + 0 data tables < 4 data tables

8 components

1. Appearance
   * Logic for appearance of application to customer
   * Written in JavaScript and Kotlin
   * Complexity level: simple
     + Receiving a customer’s appearance settings and setting the application’s appearance accordingly should have negligible complexity, assuming the range of possible appearance settings is simple enough, which we do assume
2. Authentication-server logic
   * Server-facing side of customer authentication
   * Written in Java, since authentication is implemented via the Auth0 project, which is written in Java
   * Complexity level: difficult
     + Authentication likely involves various intricate steps, such as encryption/decryption or hashing/salting the user-supplied password
3. Customer main menu
   * Logic for the “Customer main menu” screen
   * Written in JavaScript and Kotlin
   * Complexity level: simple
     + A main menu is a user-interface component that ostensibly should have negligible complexity
4. Customer login
   * Customer-facing side of customer authentication, including the user interface for that
   * Written in JavaScript and Kotlin
   * Complexity level: simple
     + This simply involves receiving two fields from a customer (the username and password), forwarding them to the authentication-server logic component (which handles all the actual complexity of user authentication), and acting on the reply from that component by either redirecting the customer to their main menu or reporting to them that the supplied user credentials are invalid
5. Search
   * Customer-facing logic for browsing the library catalog
   * Written in JavaScript and Kotlin
   * Complexity level: simple
     + This component’s function amounts to displaying a list of books to the customer, which should have negligible complexity
6. Shelf
   * Customer-facing logic for such functions as
     + Managing and accessing the books that a customer has checked out
     + Reserving books
     + Checking books out from or returning them to the library
   * Written in JavaScript and Kotlin
   * Complexity level: medium
     + The customer-facing sides of the members of the general category of functions alluded to in this component’s description each would have a complexity level of simple since they themselves do not actually implement those functions and thus do not themselves bear those functions’ respective complexities. However, the variety and multitude of these customer-facing sides leads this component to have a complexity level of medium
7. Library administration
   * Library-administrator-facing side of such administrative functions as adding/deleting books, modifying book categories, and creating new customer accounts
   * Written in JavaScript and Kotlin
   * Complexity level: medium
     + The library-administrator-facing sides of the members of the general category of functions alluded to in this component’s description each would have a complexity level of simple since they themselves do not actually implement those functions and thus do not themselves bear those functions’ respective complexities. However, the variety and multitude of these library-administrator-facing sides leads this component to have a complexity level of medium
8. Library-server logic
   * Server-facing side of all interactions with the library server(s) (save for customer authentication), including browsing the library catalog, checking books out from or returning them to the library, creating customer accounts, modifying book categories, etc.
   * Written in Java and NoSQL
   * Complexity level: difficult
     + This component involves a variety of intricate server interactions

Each of the eight components listed above is written in at least one third-generation programming language, so all eight components of the software are 3GL components. In particular, Java and JavaScript are both third-generation programming languages. Kotlin is also arguably a third-generation programming language due to its similarity to other programming languages (e.g. Java) that are often considered to be third-generation.

Since each screen and report has a complexity level of “simple,” each of the 10 screens has a complexity weight of 1, and each of the 11 reports has a complexity weight of 2. Only 2 of the 8 3GL components have a complexity level of “difficult,” so these 2 3GL components each have a complexity weight of 10, and the remaining 6 3GL components each have a complexity weight of 0. The object-point count *OP* is the sum of the complexity weights of each of the screens, reports, and 3GL components, so *OP* = 10 \* 1 + 11 \* 2 + 2 \* 10 + 6 \* 0 = 52.

Since it simply uses Auth0, a preexisting piece of software, the authentication-server logic will be considered a reused component. The seven remaining components are more-or-less built from scratch, so the proportion *R* of reused components is *R* = 1 / 8. Therefore, the new-object-point count *NOP* is given by *NOP* = *OP* \* (100% – *R*) / 100% = 52 \* (1 – 1 / 8) / 1 = 91 / 2 = 45.5.

The object-point productivity *PROD* of the development team is given by the average of their experience/capability and ICASE maturity/capability. We intend to hire developers who have nominal experience/capability and nominal ICASE maturity/capability, so *PROD* = (13 + 13) / 2 = 13.

Since *PROD* = 13, the person-month effort *Effort PM* is given by *Effort PM* = *NOP* / *PROD* = (91 / 2) / 13 = 7 / 2 = 3.5 person-months. In other words, the person-month effort required for this software project is 7 / 2 = 3.5 person-months.

Project Staffing Estimate

With an estimated 3.5 person-months, we would be planning a small-scale project, which limits the scope for team sizing.

Team composition:

2 developers

* Front end developer will utilize ReactJS framework to build interfaces. This includes rendering screens, displaying the main menu, and providing the reading interface.
* Back-end developer will build the server-side logic, and work on data management. The developer must work in Java, as that is the platform we have outlined for user authentication in our application.

With two separate developers, there would be a division in labor, however the two developers would need to collaborate closely as the work of the front end and back end communicate through requests.

Part-time project manager

* Oversight is limited due to project scope. The hours can be as low as 25% of a full-time equivalent. Therefore, the project manager is able to work on other projects simultaneously and does not need to manage daily activities of tasks.
* Needed to facilitate scrum methodologies (organizing sprints, managing timelines, etc.).

UI/UX designer

* Consultation basis as design needs is minimal, but would be helpful in maintaining consistency in design, enhancing user experience
* Needed to optimize reading interface design to make it as intuitive as possible for the user.

QA tester

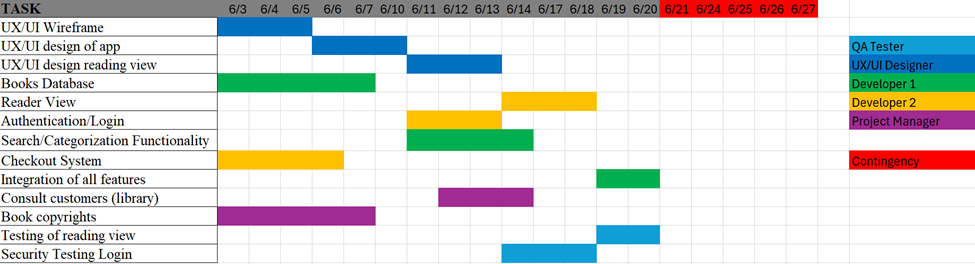
* Similar to UI, could be part-time or on a consultation basis
* The possibility of denial-of-service attacks would require this role to test not only functionality, but to ensure the application can handle security threats.
* Needed due to the requirement of security to properly manage permissions, access rights, as resources and services are based on distinct roles

## Project Scheduling Estimate

The scheduling estimate relies on the effort estimate of 3.5 person-months, and our estimated staffing of 5 team members (project manager, 2 developers, 1 UX/UI designer and 1 QA tester). Working days are considered Monday-Friday, 8-hour workdays. We take Monday, June 3rd, 2024, to be the start date of the project. 3.5 person-months of effort / 5 team members = 0.7 months, which, taking an average of 20 working days a month, is 14 working days. We add 30% contingency for a total of 18.2, rounded up to 19, working days. The estimated end date for the project is thus Thursday, June 27th, 2024. Following is a breakdown of the project by activities:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Task | Duration (days) | Dependencies | Assigned to |
| U1 | UX/UI Wireframe | 3 |  | UX/UI designer |
| U2 | UX/UI design of app | 3 | U1 | UX/UI designer |
| U3 | UX/UI design reading view | 3 | U1 | UX/UI designer |
| D1 | Books Database | 5 |  | Developer 1 |
| D2 | Reader View | 3 | U3 | Developer |
| D3 | Authentication/Login | 3 | U2 | Developer |
| D4 | Search/Categorization Functionality | 4 | U2 | Developer |
| D5 | Checkout System | 3 |  | Developer 2 |
| D6 | Integration of all features | 2 | D1, D2, D3, D4 | Developer |
| M1 | Consult customers (library) | 3 |  | Manager |
| M2 | Book copyrights | 5 |  | Manager |
| T1 | Testing of reading view | 2 | D2 | QA Tester |
| T2 | Security/Login Testing | 3 | D3 | QA Tester |

Following is a Gantt chart depicting the project timeline:



## Estimated Cost of Personnel

When calculating salaries, all numbers were taken from either Indeed or Glassdoor, 2 different job aggregation sites to determine average salaries. The two developer roles were taken to be mid-level developers as this seems to be about what an “average” developer would be. All figures taken only used base pay of employees because Indeed did not take into account total compensation. The salary breakdowns are as follows:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Mid-level developer [1] | Project manager [2] | UI/UX designer [3] | QA tester [4] |
| Avg. salary/yr. | $76253 | $88344 | $68453 | $65025 |
| Avg. salary/day | $208.91 | $242.04 | $187.50 | $178.15 |
| Avg. salary/hour | $26.11 | - | - | - |
| Number | 2 | 1 | 1 | 1 |
| Days (average) | 10.5 | 8 | 9 | 5 |
| Amount paid | $44480.92 | $10306.80 | $17113.25 | $18423.75 |

All numbers for days worked are taken from the project plan

Furthermore, there are additional potential costs associated with training. While the mid-level developer should be familiar with these technologies, it is important for them to have the relevant certifications for the job. All data is aggregated from Amazon websites. The break downs are as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Certification test | Hours needed | Course cost | Number done | Price |
| AWS [5] | 100 | 0-15 | 0 | 1 | $100-$442.55 |
| DB/Aurora [6] | 300 | 0-20 | 0 | 1 | $300-$756.74 |

As a result, the overall cost breakdown has resulted in a total cost of personal ranging from $8902.10 to $10101.40 depending on if the backend developer already has obtained the certificates needed to work on the AWS platform.

## Estimated Cost of Hardware Products

We will compare the hardware cost for our Online Library Application which is actually hosted on the cloud. So, the options that will be considered are virtual private servers which include AWS Lightsail and DigitalOcean which are some of the biggest players in this area. Specifically, the cost of their resources and how much it takes to scale as the demand on our service increases. Starting with AWS Lightsail, we can take a look at their virtual private servers pricing which is split into two categories that include IPV4 and IPV6 servers. This is a common pricing configuration with most virtual private server providers. We will be focusing on IPV4, the pricing page data for AWS Lightsail [11] shows:

Windows / Linux

* $8 / $3.50 per month
  + 512 MB Memory
  + 2 vCPUs
  + 30 GB SSD Disk
  + 1 TB Transfer
* $12 / $5 per month
  + 1 GB Memory
  + 2 vCPUs
  + 40 GB SSD Disk
  + 2 TB Transfer
* $20 / $10 per month
  + 2 GB Memory
  + 2 vCPUs
  + 60 GB SSD Disk
  + 3 TB Transfer
* $40 / $20 per month
  + 4 GB Memory
  + 2 vCPUs
  + 80 GB SSD Disk
  + 4 TB Transfer
* $70 / $40 per month
  + 8 GB Memory
  + 2 vCPUs
  + 160 GB SSD Disk
  + 5 TB Transfer
* $120 / $80 per month
  + 16 GB Memory
  + 4 vCPUs
  + 320 GB SSD Disk
  + 6 TB Transfer
* $240 / $160 per month
  + 32 GB Memory
  + 8 vCPUs
  + 640 GB SSD Disk
  + 7 TB Transfer

Managed Databases Standard Plan / High Availability Plan

* $15 / $30 per month
  + 1 GB Memory
  + 1 Core Processor
  + 40 GB SSD Disk
  + 100 GB Transfer
  + No Data encryption
* $30 / $60 per month
  + 2 GB Memory
  + 1 Core Processor
  + 80 GB SSD Disk
  + 100 GB Transfer
  + Data encrypted
* $60 / $120 per month
  + 4 GB Memory
  + 2 Core Processor
  + 120 GB SSD Disk
  + 100 GB Transfer
  + Data encrypted
* $115 / $230 per month
  + 8 GB Memory
  + 2 Core Processor
  + 240 GB SSD Disk
  + 200 GB Transfer
  + Data encrypted

Clearly Linux is a consideration as a less expensive alternative to going with Windows as it is tied to the same exact hardware for the virtual private servers which is affecting its cost. For the managed databases, since performance is a must, the high availability plan is a consideration worth taking. Also, data encryption for security reasons is essential.  
  
Taking a look at what we can scale we see with AWS Lightsail [11] we see taking a snapshot is 5 cents per GB per month and if we need more hard drive space the price per GB added is $0.10:

* $0.80 for 8 GB
* $3.20 for 32 GB
* $6.40 for 64 GB
* $12.80 for 128 GB
* $25.60 for 256 GB

As a migration process from Lightsail to EC2 example [12] shows, if we require scaling everything else such as CPUs and even more memory, we can use AWS EC2 or AWS lambda via a snapshot in order to expand our operations without too much delay.

Now we will analyze what DigitalOcean offers. It has many pricing options but the main one we will focus on is its application platform and the addons that are relevant for our application to work. This is because the infrastructure and dependencies are handled for us but we still are able to choose certain specifications such as the number of CPUS and RAM. It also specifies the ability to scale which is important. Looking at the pricing page data [13] it shows:

Basic Tier

* $5 per month
  + 1 CPU
  + 512 MiB Memory
* $10 per month
  + 1 CPU
  + 1 GiB Memory
* $20 per month
  + 1 CPU
  + 2 GiB Memory
* $40 per month
  + 2 CPUs
  + 4 GiB Memory

Professional Tier

* $12 per month
  + 1 CPU
  + 1 GiB Memory
* $25 per month
  + 1 CPU
  + 2 GiB Memory
* $50 per month
  + 2 CPUs
  + 4 GiB Memory
* $75 per month
  + 1 CPU (dedicated)
  + 4 GiB Memory
* $150 per month
  + 2 CPUs (dedicated)
  + 8 GiB Memory
* $300 per month
  + 4 CPUs (dedicated)
  + 16 GiB Memory

Since our online library application prioritizes performance, the professional tier and choosing the dedicated CPUs is a wise choice as shared hardware does not offer comparably good performance.

For the managed databases we decided to go with NoSQL which is offered by DigitalOcean in the form of MongoDB. We will be focusing on storage and according to the pricing [14] the optimized storage is:

* $240 per month
  + 2 vCPUs
  + 16 GiB Memory
  + 440 GiB Disk
* $480 per month
  + 4 vCPUs
  + 32 GiB Memory
  + 890 GiB Disk
* $960 per month
  + 8 vCPUs
  + 64 GiB Memory
  + 1.709 TiB Disk
* $1,920 per month
  + 16 vCPUs
  + 128 GiB Memory
  + 3.428 TiB Disk
* $2,880 per month
  + 24 vCPUs
  + 192 GiB Memory
  + 5.137 TiB Disk
* $3,830 per month
  + 32 vCPUs
  + 256 GiB Memory
  + 6.855 TiB Disk

DigitalOcean offers better managed databases than AWS Lightsail and the professional tier has dedicated hardware options while AWS Lightsail does not specify which may point to it being shared due to the lower price. Overall DigitalOcean offers many options in terms of various addons taking into account all sorts of use cases which includes an impressive ability to scale. Therefore, we will go with DigitalOcean instead of AWS Lightsail.

## Estimated Cost of Software Products

The bulk of the software for the Online Library app is to be produced in-house. Due to the dynamic nature of online security, it would be more prudent to outsource the authentication aspect of the Online Library app to a dedicated service, such as Auth0 by Okta. An encryption algorithm is estimated to have an average lifespan of less than two years [15]. To prevent data breaches, user authentication must be updated regularly.

The benefits of utilizing an authentication service are most evident in the schedule and budget, as providers such as Okta have created systems that are easily integrated into their clients’ projects. Auth0 can be configured to suit the needs of their clients, so the price of the software is variable [16]. There are three pricing levels suited to the software requirements as detailed in this report.

Free Plan

* Supports up to 7,500 active users
* Unlimited logins
* Community support

Essentials Plan

* Options range from 500 to 30,000+ active monthly users
* Average cost of $700 per month for every 10,000 users
* Unlimited logins
* Community and standard support
* Account Linking

Startup Plan

* 100,000 active monthly users
* First year free
* Includes greater customization than the essentials plan
* Enhanced password protection
* Community and standard support

It is essential to note that the Startup plan is a program that must be applied for by candidates that meet certain eligibility criteria. The Online Library Application is unlikely to immediately require support for more than 10, 000 active users, nor is it as business oriented as the Startup plan is intended for, so either the Free or Essentials plan is the more likely candidate for authentication. It is recommended to utilize the Free Plan at the outset of the project and, as the number of users increases, consider switching to the Essentials plan.

## Pricing Estimation

The cost of staffing during development of the Online Library Application is approximately $10,200. When determining the price of cloud-based software, the overhead cost of development is not the only factor. Regarding hardware costs, the average per-month cost among the choices detailed in the “Hardware Cost” section of this report is around $1,720 per month. If we plan for approximately 10,000 active users per month, the cost of authentication software is approximately $700 per month, as determined in the “Software Cost” section of this report.

In addition to the cost of outsourced hardware and software, the software created by developers must be maintained. For a software support specialist, the average salary is $64,000

per calendar year or approximately $5,400 per month [17]. Payroll costs, which are about 15% of the salary, would be $810 per month for such an employee.

The total monthly cost for hardware, software, and maintenance is $8,630.

Another expenditure that must be considered for the Online Library Application is the Customer Acquisition Cost, also known as CAC, which is the money spent on acquiring new customers. A good CAC ratio, the value a customer brings versus the cost of acquiring them, is 3:1 [18[. If the value a customer brings is $9, no more than $3 should be spent acquiring that customer.

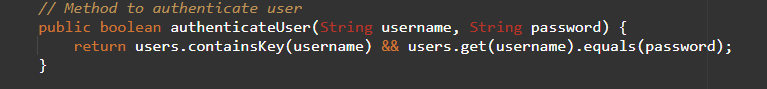
Analysis will start off with a base price of $10 per month, which is offered by two similar applications [listed under the Comparison to Similar Designs section]. If 30% of that value is reserved for customer acquisitions, each customer is worth $7 per month. To break-even on monthly expenditures at $7, the Online Library Application would need a minimum of 1,300 subscribers. Assuming we, as the company, absorb the cost of development prevents the incurrence of interest on the $10,200. Recouping the cost of development over a nine-month period would be $1,134 per month, or approximately 162 subscribers.

With a total break-even requirement of 1,462 subscribers, profit margins can determine the goal number of customers. To have a 40% profit margin, 2,437 customers would be ideal. Making the calculations with a base price of $6 per month results in 2,442 subscribers per month required to break even and 4,070 customers required to have a 40% profit margin.

Considering the substantial success of similar applications, it is not difficult to envision the Online Library Application having the subscriber base it needs to be successful, especially at a competitive price of $6 per month.

# Test Plan

In order to test out our library app, we implemented a code in java that will test a couple of different things. The first step of the app is for the user to login to the app. So, the first step of it will be to authenticate the user. For this we will run two different tests, one will be a valid user and one will be an invalid user. The method for this will take in a username and password which are both strings and will authenticate the username and password will match up to what is in the system. Below is the method we used in order to test this:



We initialized two different users in order to test this. We set their usernames to user1 and user2, and both of their passwords to password1 and password2, all of which are strings.



After that, we ran these test methods inside of JUnit to be able to test the function. In order to test the above “authenticateUser” method, I created two JUnit tests. The first deals with entering the right username and password which will allow the user to be able to access the app and his account. It will test the method using “user1” as the username and “password1” as the password which matches up with what we initialized above. We will use assertTrue for this one as it will return true since it is a valid username and password. We will also run a second test in which we will use “user1” as the username and “wrong\_password” as the password which is the incorrect password and does not match up with the username we initialized it with. For this reason, we will use assertFalse to test this, which will return true as it is an invalid user.

## Test Case 1

Input: username = user1, Password = password1

Expected output = True

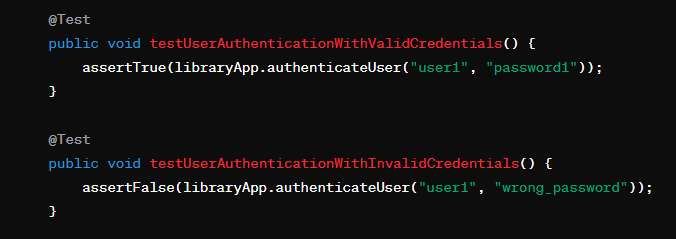
Actual output = True

## Test Case 2

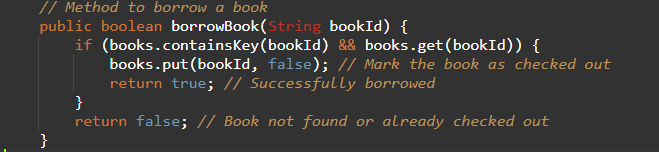
Input: username = user1, Password = wrong\_password

Expected output = False

Actual output = False



The other method that we will be testing for this will be regarding checking out books. For this, we will run two different tests as well. The first of them will be checking out a book that is available for checkout and the second will be an unavailable book. The method for this will take in a string which is the “bookId” and will return true or false based on whether the book is available for checkout. If the book is available, it will allow you to checkout and will make the book no longer available for anyone else to check out. If it has already been checked out, it will return false. Below is the method we used to test this:



We initialized two different books in order to test this. We set the first book's ID to “12345” and set it to true, which means that the book is available. We also created a second book in which we set the ID to “54321” and set the book to false, which means that it has already been checked out.



After that, we ran these test methods inside of JUnit to be able to test the function. In order to test the above “borrowBook” method, I created two JUnit tests. The first deals with entering the first book’s Id which is “12345” and trying to check out the book which will allow the user to be able to do as the book is available. We will use assertTrue for this one as the book is available for checkout. We will also run a second test in which we will use the second book’s Id which is “54321” and try to check out the book which will not allow the user to do so as the book has already been checked out. We will use assertFalse for this one as the book is not available.

## Test Case 1

Input: bookId = 12345, book is available

Expected output = True

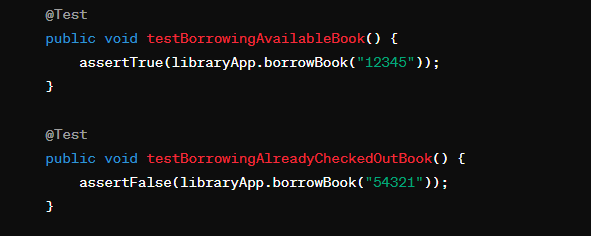
Actual output = True

## Test Case 2

Input: book Id = 54321, book is not available

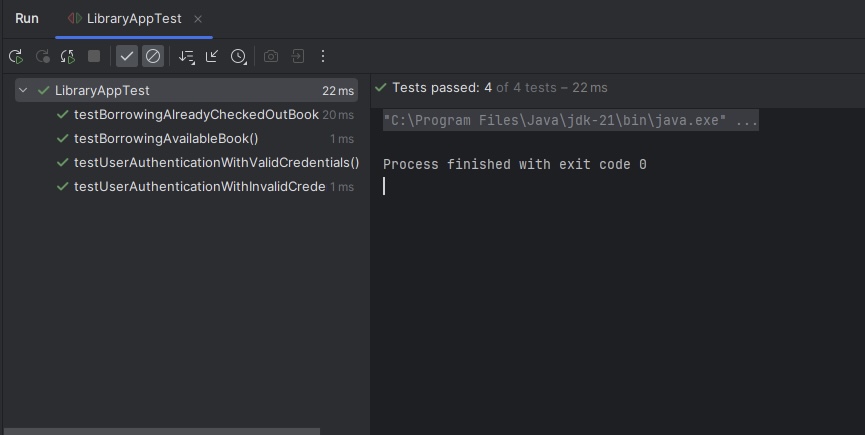
Expected output = False

Actual output = False



## Test Results

The picture below shows the output that was given on JUnit after running the above programs. It shows that we ran four tests, and all of them passed as we expected.



# Comparison to Similar Designs

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Feature | **Libby by OverDrive [7]** | **Kindle Unlimited [8]** | **Audible**  **[9]** | **Scribd**  **[10]** | **Our Library App** |
| **Content-Type** | eBooks, audiobooks | eBooks, audiobooks, magazines | Audiobooks | Documents  (academic, legal, DIY, manuals, and more) | eBooks, journals, research materials |
| **Subscription Model** | Free (library membership required) | Subscription | Subscription | Subscription | Subscription |
| **Library Size** | Varies by local library | Over 1 million titles | Over 500,000 titles | Over 1 million titles | Varies by local library |
| **Offline Access** | Yes | Yes | Yes | Yes | Yes |
| **Device Compatibility** | iOS, Android, Kindle, Web Browsers | Kindle devices and apps, Web Browsers | iOS, Android, Kindle, Web Browsers | iOS, Android, Kindle, Web Browsers | iOS, Android, Web Browsers |
| **Customization** | Adjustable font sizes, themes | Limited | Playback speed, bookmarks | Adjustable font sizes, themes | Themes, fonts, font sizes and more |
| **Accessibility** | Screen reader support, dyslexia-friendly fonts, customizable display options | Limited accessibility features | Screen reader support, adjustable playback speed, captions available for some titles | Dyslexia-friendly fonts, adjustable font sizes, screen reader support | Screen reader support, adjustable fonts, adjustable font sizes, customizable display options |
| **Price** | Free | $9.99 per month | $14.95 per month | $11.99 per month | $6 per month |

Libby by OverDrive allows readers to borrow eBooks and audiobooks from their local libraries at no cost. This makes it an exceptional choice for those who prioritize budget-friendly reading options and prefer to borrow rather than purchase books outright. Moreover, Libby's commitment to inclusivity shines through its accessibility features, including screen reader support, dyslexia-friendly fonts, and customizable display options. These features ensure that users with diverse needs can enjoy a seamless and comfortable reading experience. Additionally, Libby offers a range of customization options such as adjustable font sizes, themes, and more, allowing readers to tailor their reading environment to suit their preferences.

Kindle Unlimited, with its vast library of over one million eBooks, is a subscription-based service priced at $9.99 per month. While it offers a substantial collection of titles, its accessibility features and customization options are somewhat limited compared to platforms like Libby and Scribd. This may pose challenges for users with specific accessibility requirements or those seeking extensive customization capabilities to enhance their reading experience.

Audible, priced at $14.95 per month, caters only to audiobook enthusiasts with its extensive collection of audiobooks. While it offers basic customization features such as playback speed adjustments and bookmarks, its accessibility features are somewhat limited compared to eBook platforms like Libby and Scribd. However, Audible does provide screen reader support and adjustable playback speed, making it accessible to visually impaired users and those who prefer customizable listening experiences.

Scribd, another subscription-based service priced at $9.99 per month, distinguishes itself with its diverse library encompassing eBooks, audiobooks, magazines, and documents. This comprehensive content offering provides excellent value for readers seeking a multifaceted digital library experience. Scribd's commitment to accessibility is evident through its features like dyslexia-friendly fonts, adjustable font sizes, and screen reader support, ensuring that users with varying needs can enjoy their reading experience to the fullest. Furthermore, Scribd offers a wide range of customization options, including adjustable font sizes, themes, and more, empowering users to personalize their reading environment according to their preferences.

Our Library App offers a comprehensive library experience that includes eBooks, journals, research materials, and more. Users can access and borrow all of these different materials through their local library at $6 per month. The app's robust accessibility features, such as screen reader support, adjustable fonts, and customizable display options, ensure inclusivity for all users, regardless of their reading preferences or accessibility needs. Moreover, this application provides extensive customization options, including themes, fonts, font sizes, and more, allowing users to tailor their reading experience to their liking and create a personalized reading environment that enhances their enjoyment and engagement with the content. Overall, our app is a cost-effective, inclusive, and customizable solution for readers, researchers, and learners, offering a comprehensive library experience that caters to diverse interests and accessibility needs.

# Conclusion

Our online library application, despite having no implementation, has progressed into a comprehensive idea. At the onset of the project, we were all ignorant of how to accomplish the many parts of it that then lay before us. However, as we advanced through the semester and became exposed to increasingly more software engineering concepts, we applied our newly accrued knowledge to one part of the project after another, gradually taking us to the project’s present, completed form. In short, the project is the culmination of what we have learned over the semester.

As we progressed through the project and developed a clearer vision of what the final product ought to be, we made countless revisions to our work. However, there are a few noteworthy changes that we made at various points in the project to our early plans. These changes are detailed below.

One set of aspects that changed during the project from our original plans was the delegation of certain tasks, and these changes were in response to updated directions from the professor. We initially delegated the task of deciding upon the software’s architectural design to specific people within our group, but Assignment 4 required us to perform this task as a group. As a result, we indeed decided upon the architectural design as a group but delegated the task of writing the Architectural Design section to specific group members. We also initially delegated the task, of estimating the effort that would hypothetically be required for implementing our software, to a specific group member, but, in a repeat of the situation for the architectural design, Assignment 8 required us to instead conduct the effort estimation as a group. However, the group member already delegated with performing the effort estimation had worked extensively on that task, and we did not wish for their work to go to waste. Consequently, while we indeed decided upon the effort estimation as a group, we gave a degree of deference on the matter to the aforementioned group member.

Another aspect that changed was that we initially intended to contact local libraries in order to determine what software, if any, they use for online content distribution. However, we also wanted our design to be original, and, considering that we were not actually going to implement our design, we decided that we would be okay if our design deviated from that of existing software of the same type. For these reasons, we ultimately did not deem it necessary to contact local libraries about the aforementioned issue and so abandoned that idea.

One change that was made rather late in the project was the company whose cloud services our software would utilize. In Deliverable 1, we chose for our software to use the cloud services of Amazon Web Services (AWS), but, in Deliverable 2, we chose for our software to instead use the cloud services of DigitalOcean. The motivation behind this change was that DigitalOcean would give our application dedicated CPUs and would manage most of our application’s operation for us, and that we judged DigitalOcean’s services to be more scalable than those of AWS.

# References

[1] “Salary: Mid Level Programmer in United States 2024.” Glassdoor.com. Accessed: Apr. 16, 2024. [Online.] Available: https://www.glassdoor.com/Salaries/mid-level-programmer-salary-SRCH\_KO0,20.htm

[2] “Project manager salary in Texas.” Indeed.com. Accessed: Apr. 16, 2024. [Online.] Available: https://www.indeed.com/career/project-manager/salaries/TX

[3] “User interface designer salary in Texas.” Indeed.com. Accessed: Apr.16, 2024. [Online.] Available: https://www.indeed.com/career/user-interface-designer/salaries/TX

[4] “Quality assurance tester salary in Texas.” Indeed.com. Accessed: Apr. 16, 2024. [Online.] Available: www.indeed.com/career/quality-assurance-tester/salaries/TX

[5] “AWS Certified Cloud Practitioner Certification.” AWS.Amazon.com. Accessed: Apr. 15, 2024. [Online.] Available: https://aws.amazon.com/certification/certified-cloud-practitioner/

[6] “AWS Certified Database - Specialty Certification.” AWS.Amazon.com. Accessed: Apr. 15, 2024. [Online.] Available: https://aws.amazon.com/certification/certified-database-specialty/

[7] “Read with Libby.” Overdrive.com. Accessed: Apr. 16, 2024. [Online.] Available: https://www.overdrive.com/apps/libby#Features

[8] “Join Kindle Unlimited.” Amazon.com. Accessed: Apr. 16, 2024. [Online.] Available: https://www.amazon.com/kindle-dbs/hz/subscribe/ku

[9] “About Audible.” Audible.com. Accessed: Apr. 16, 2024. [Online.] Available: https://www.audible.com/

[10] “What is Scribd?” Scribd.com. Accessed: Apr. 16, 2024. [Online.] Available: https://www.scribd.com/what-is-scribd

[11] “VPS, web hosting pricing—Amazon Lightsail—amazon web services.” AWS.Amazon.com. Accessed: Apr. 17, 2024. [Online.] Available: https://aws.amazon.com/lightsail/pricing/

[12] N. Kashyap. “Migrating web services from Amazon Lightsail to EC2.” AWS.Amazon.com. Accessed: Apr. 17, 2024. [Online.] Available: https://aws.amazon.com/blogs/startups/migrating-web-services-from-amazon-lightsail-to-ec2/

[13] “App Platform Pricing.” DigitalOcean.com. Accessed: Apr. 18, 2024. [Online.] Available: https://www.digitalocean.com/pricing/app-platform

[14] “Managed database pricing.” DigitalOcean.com. Accessed: Apr. 18, 2024. [Online.] Available: https://www.digitalocean.com/pricing/managed-databases

[15] “Build vs. Buy.” Okta. Accessed: Apr. 18, 2024. [Online.] Available: https://www.okta.com/resources/whitepaper-build-vs-buy

[16] “Pricing.” Auth0. Accessed: Apr. 18, 2024. [Online.] Available: https://auth0.com/pricing

[17] “Software Support Specialist Salaries.” Glassdoor. Accessed: Apr.18, 2024. [Online.] Available: https://www.glassdoor.com/Salaries/software-support-specialist-salary-SRCH\_KO0,27.htm

[18] “Average Customer Acquisition Cost by Industry: Tracking CAC Benchmarks.” Vena Solutions. Accessed: Apr. 18, 2024. [Online.] Available: https://www.venasolutions.com/blog/average-cac-by-industry