*CS3354 Software Engineering Final Project*

*Deliverable 2*

**Readary**

(An App for Book Lovers)

**The Group Members**

Aisha Ashfaque

Dante Moreno

Emily Wojciechowski

Joel Francis

Junaid Hashmi

Priscilla Adomako

**Date**

March 15, 2019



# 

# Feedback on Project Draft comments:

Our proposal was approved without a recommendation for any change. However, our plan will be adjusted based on the fact that one of our team members, Michael Elizalde, has joined another project team and is no longer part of our team.

Based on the comments we received, we surveyed and found several similar apps already in use. The most surprising discovery was to find an app named *Bookish*, our proposed app name. Consequently, we have renamed our product to *Readary.* Other than reshuffling of responsibilities due to loss of a team member, this is the biggest change we incorporated.

Since we found similar products already in use, we will do a detailed comparison as part of our final project report. For now, we would like to mention an additional feature to be included in our project in order to distinguish it from other similar products in the market.

* 1. Addition to previously stated purpose

We will incorporate a MUTE function in the app to allow users to silence notifications as follows:

* + 1. Audio notification may be turned on/off in application setup
    2. Notification messages may be muted (turned on/off in application setup)
    3. Notification messages may be muted until a future date provided by user

***NOTE:*** URL for our project GitHub is: <https://github.com/CS3354-Readary>

# Task Delegation:

|  |  |
| --- | --- |
| Name | Task Delegation |
| Dante Moreno | Team Proposal, Create GitHub Repository and Add Team Members, Software Requirements, Addressing Feedback, Presentation |
| Aisha Ashfaque | Team Proposal, Software Requirements, Cost Effort and Pricing Estimation, Software Test Plan, Presentation |
| Junaid Hashmi | Commit README, Architectural Design, Project scheduling, Citing references, Presentation |
| Joel Francis | Team Proposal, Class diagram, Project Scheduling, Software Requirements, Software Test Plan, Presentation |
| Priscilla Adomako | Team Proposal, Sequence Diagram, Cost Effort and Pricing Estimation, Comparison and Conclusion, Presentation |
| Emily Wojciechowski | Project Scope file, Use Case Diagram, Citing References, Comparison and Conclusion, Presentation |

# Software Process Model:

The software process model that will be employed with this project is the spiral model. The spiral model contains development cycles, in which risk management is considered before employing an addition to a prototype. Since Readary’s architecture features multiple systems interacting with a database, it would make sense any risks are considered before implementing to ensure that no additional feature negatively affects the database.

# Software Requirements:

Functional Requirements

1. A user holding an author account shall be able to submit book titles along with one summary and cover of that book to be added to the repository.
2. A user shall be able to search for books solely on genre, author, reviews and ratings, or other tags or a combination of the four.
3. A user shall be able to add and remove genres, authors, or other tags and interests from their book preferences.
4. The system shall generate each day, for each user, a new list of books based on each user’s preferences and the books each user has tagged.
5. Each user shall be uniquely identified by their member ID.
6. A user shall be able mute notifications until a specified date or indefinitely.

Non-functional

1. **Product Requirements**
   1. **Usability**
      1. Users shall be able to access all functions from the main menu in less than 3 clicks.
      2. 95% of users will be able to search for a book and add a new interest tag by the third attempt without requiring assistance.
      3. The system shall be accessible to users with vision needs, specifically, users shall be able to increase the font size for the entire user interface.
   2. **Efficiency**
      1. **Performance**
         1. Responses to actions shall take no longer than 5 seconds

to load on to the screen.

* + - 1. The system shall be able to process a single notification in

less than 1 second.

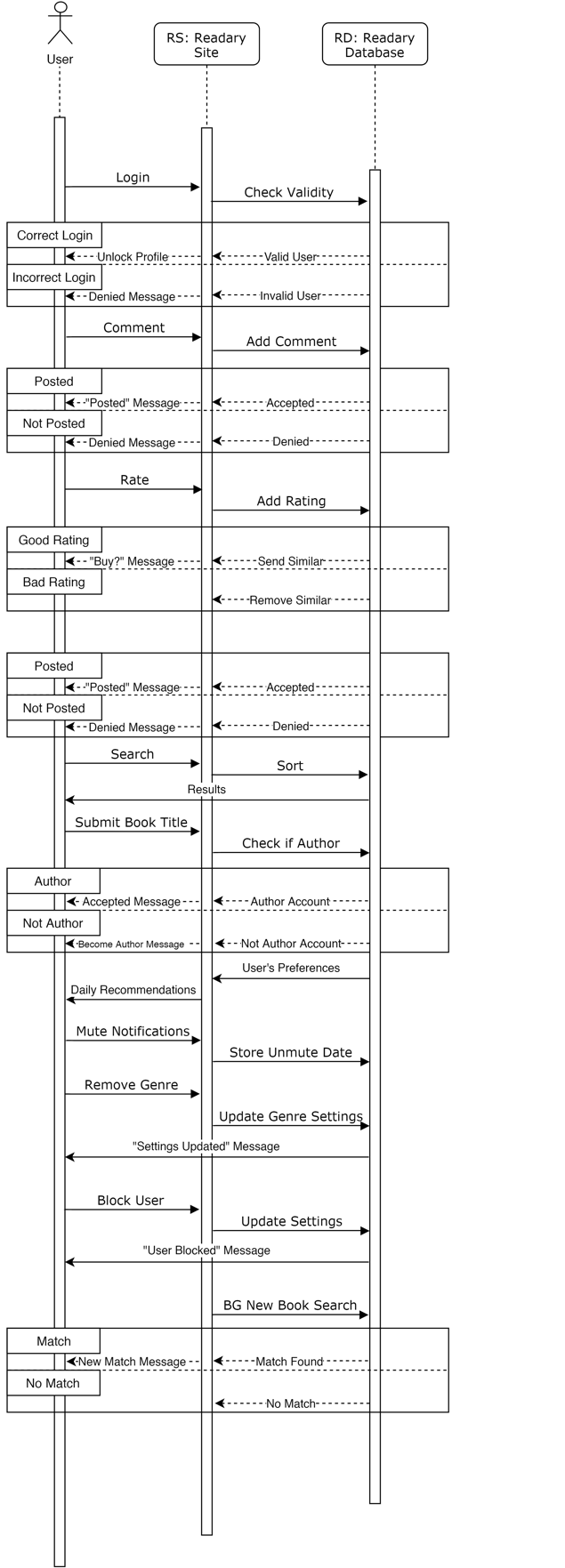
* + 1. **Space**
       1. The system shall not require more than 1 GB of space when downloaded on the users device.
  1. **Dependability**
     1. In the case of system failure, less than 0.1% of data shall be lost.
     2. The system shall be operational 95% of the time
     3. In case of system failure, down time shall be 2 hours or less.
  2. **Security**
     1. User account passwords must be at least 8 characters long.

1. **Organizational Requirements**
   1. **Environmental**
      1. The system shall be compatible with both iOS and Android.
   2. **Development**
      1. The system shall be developed using the programming language Java.
2. **External Requirements**
   1. **Legislative**
      1. The system shall implement and abide by copyright laws for each book that is requested to be submitted to the repository.

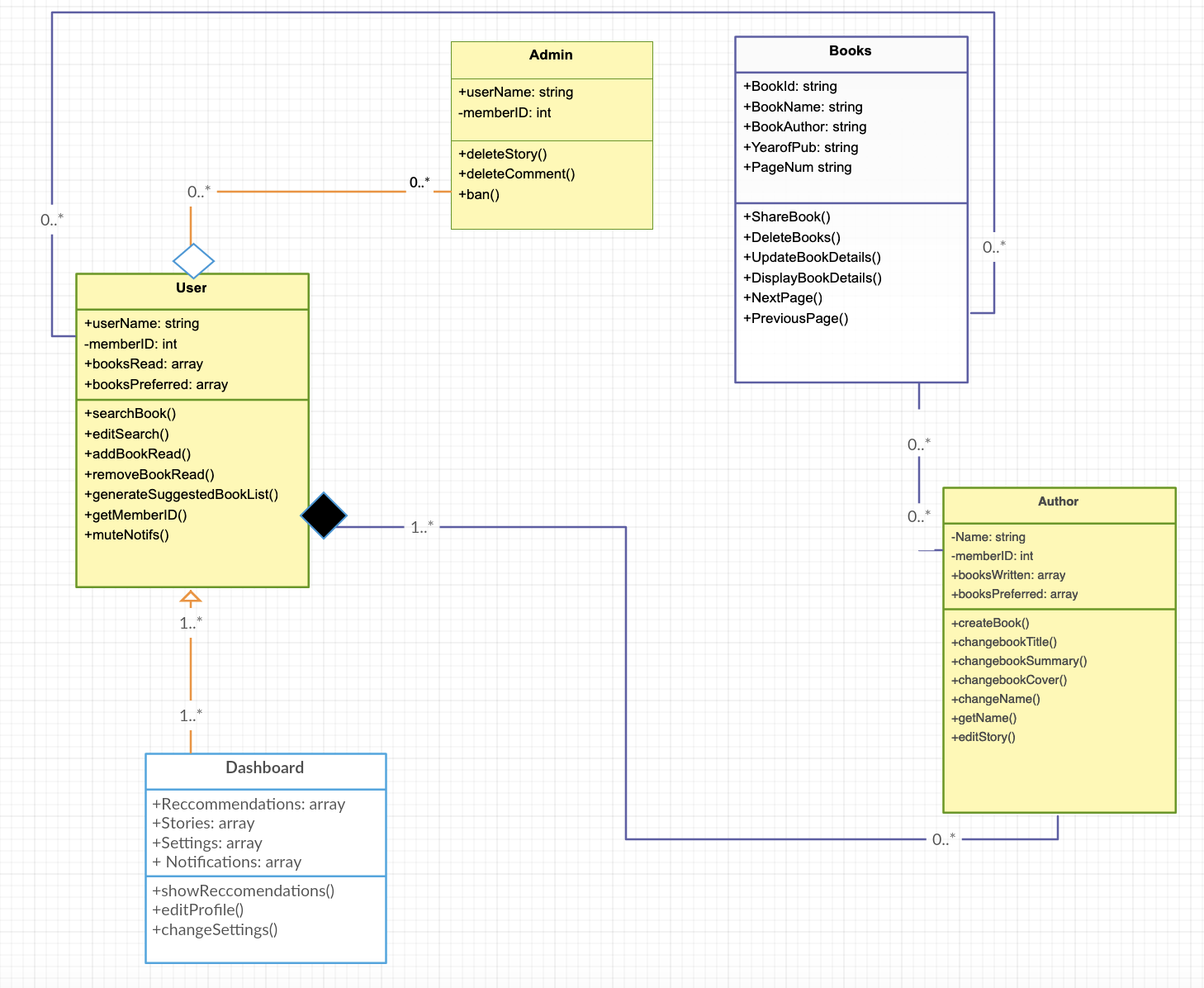
# Use Case Diagram:



# Sequence Diagram:

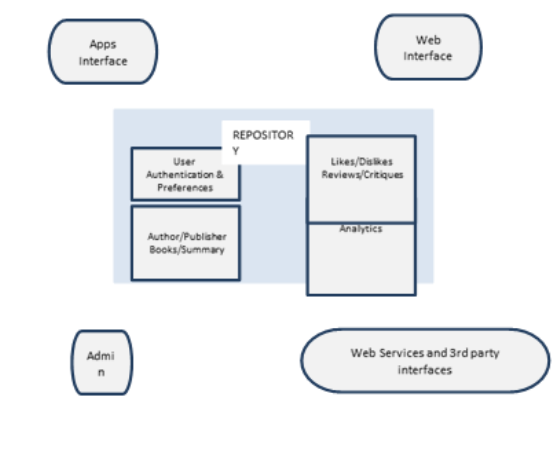


# Class Diagram:



# Architectural Design:

The central part of Readary is a centralized DBMS. Therefore, our architectural choice is a central repository pattern. A high level architecture is shown below:



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

# Project Duration and Staffing:

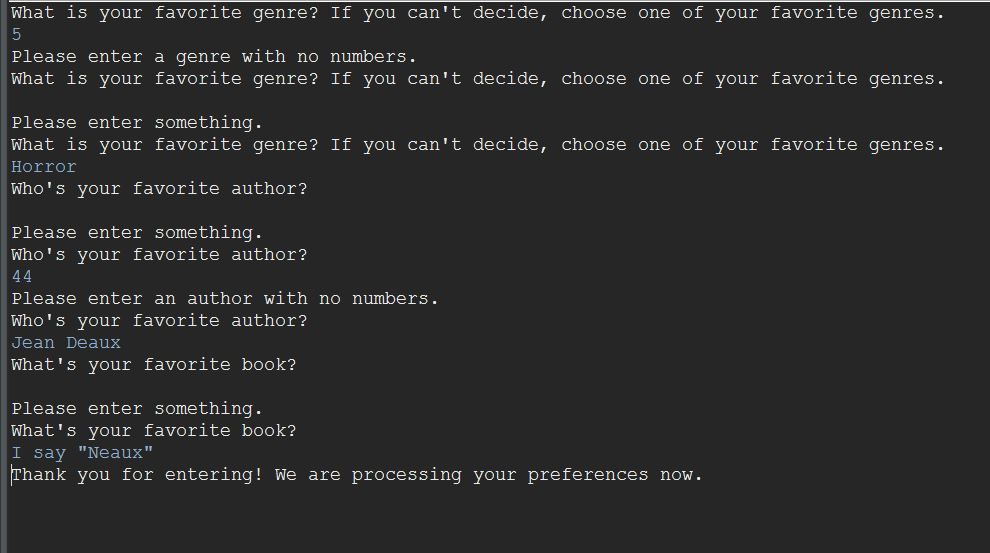
# Test Plan:

The method userInput() asks and takes user input and stores it into three variables (there can be more, but for the sake of testing purposes we will be using just three). For the plan, we are using the equivalence partitioning method that is described in black box testing. The variable userGenre is the user’s input for their favorite genre, userAuth stores the user’s favorite author, and userBook is the user’s favorite book. All these variables will undergo a type of validation to see if the user left these entries empty (**userGenre.isEmpty()**, **userAuth.isEmpty()** and **userBook.isEmpty()**), or if the user input numbers where there should not be (**userGenre.matches(“[0-9]+”)** and **userAuth.matches(“[0-9]+”)**), or, in some cases, both. Once the userInput() method is done, the program should have the user’s favorite book, author, and genre for the program to use for matchmaking. JUnit was used to test the unit. The unit came back with no errors, a setup time of 0.000 seconds, and runtime of .094 seconds. The output and JUnit results are pictured below. A copy of the code is located in the zip file.

**Cases:**

1. **userGenre.matches(“[0-9]+”)** 
   1. **Partition 1:** If the user input has a number, repeat until the user does not.
   2. **Partition 2:** If the user does not input a number, then continue.
2. **userGenre.isEmpty()** 
   1. **Partition 1:** If the user does not input anything, repeat until the user does.
   2. **Partition 2:** If the user does input something, then continue.
3. **userGenre.matches(“[0-9]+”)|| userGenre.isEmpty()**
   1. **Partition 1:** If the user does not input anything, repeat until the user does.
   2. **Partition 2:** If the user does input something, but it’s a number, repeat until the user does.
   3. **Partition 3:** If the user does input something, and it does not have a number, then continue.
4. **userAuth.matches(“[0-9]+”)**
   1. **Partition 1:** If the user input has a number, repeat until the user does not.
   2. **Partition 2:** If the user does not input a number, then continue.
5. **userAuth.isEmpty()**
   1. **Partition 1:** If the user does not input anything, repeat until the user does.
   2. **Partition 2:** If the user does input something, then continue.
6. **userAuth.matches(“[0-9]+”) || userAuth.isEmpty()**
   1. **Partition 1:** If the user does not input anything, repeat until the user does.
   2. **Partition 2:** If the user does input something, but it’s a number, repeat until the user does.
   3. **Partition 3:** If the user does input something, and it does not have a number, then continue.

# 



# Comparison

Following is an overview of comparison between similar products found:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FEATURES** | **Readary** | **Goodreads** | **iBookworm** | **Wattpad** |
| Social App features and fan-base |  |  |  |  |
| Rate and comment |  |  |  |  |
| Preferences, including genres, ratings, etc. |  |  |  |  |
| Smart auto-suggest list |  |  |  |  |
| **Notification mute controls** | **** | **** | **** | **** |
| **Select library integration** | **** | **** | **** | **** |
| **Direct-connect with authors** | **** | **** | **** | **** |
| Publication, sampling, promotion |  |  |  |  |
| List of sellers, price, availability |  |  |  |  |

# Conclusion

One of the first issues to be encountered is understanding how Git handles word documents. Initially there were concerns that Git might not track changes to Word documents, so we created a sort of naming convention to keep track of updated versions of a Word document which ended up being unnecessary. Overall the use of a decentralized version control system was a fairly simple process for the team to pick up. Other issues arose as a result of individual members attempting to complete parts of the first deliverable at the same time, before the requirements had really been clearly thought out. This lack of clarity in the requirements was partially a result of the fact that our initial requirements for the project had too much overlap with existing products so we had to quickly come up with ideas to help differentiate our product from the existing products. The lack of clarity also led to some initial discrepancies between the works of individual members. However, these discrepancies were more apparent in the diagrams created to show how the software would functions and how users would interact with the software. This led to the need for alterations to certain portions of the first deliverable. These alterations attempted to reuse as much of the existing work possible which led to a trade-off between efficiency and cohesiveness. There were some benefits from the fact that team members began their portions of the deliverable at the first time since it allowed some members to finish very quickly so they were able to help other members with their potions as well. For the second deliverable we managed to avoid some of issues we faced when completing the first deliverable because we clearly defined the estimation model we intended to use along with the parameters we would take into account for making the estimations. When defining these parameters and assigning some initial counts for each category we realized that we had underestimated these counts since the design of our app proved to be more sophisticated than we had initially anticipated. This led to some changes having to be made to our estimations of the effort and cost needed for the completion of the project. Overall there was a much smoother flow for the second deliverable.

# References: