APPLICATION PROGRAMMING IN JAVA PROJECT REPORT

Library management system

“COMPILEMILE” TEAM

Github URL:

https://github.com/iuthub/group-project-compilemile

Team:

Yarkinov Ulugbek U1910140 sec 003

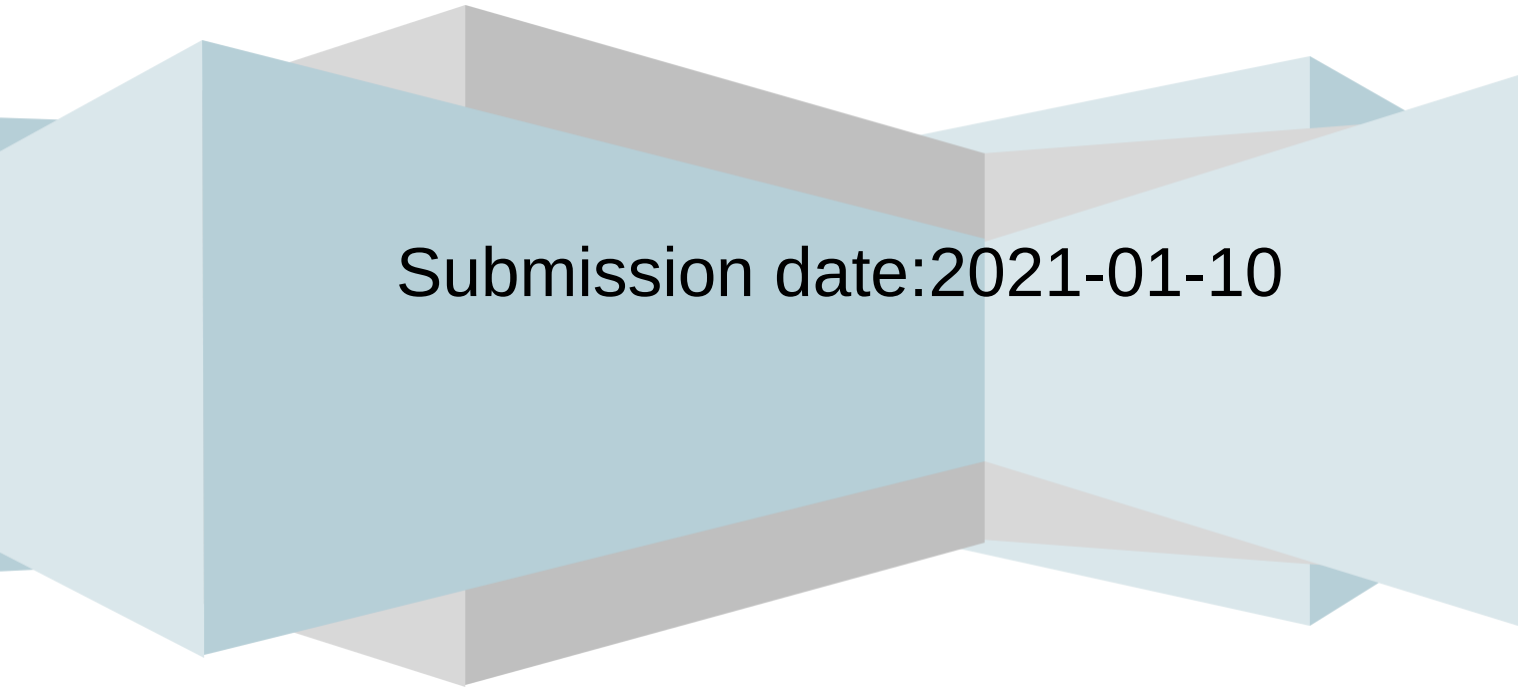
Kattakhodjaev Boiskhon U1910171 sec 004

Keldiyarov Bekzod U1910142 sec 004

Musaev Azamat U1910177 sec 004

Zaynutdinov Zufarbek U1910139 sec 003

INHA UNIVERSITY IN TASHKENT



Submission date:2021-01-10

Table of contents

User Accounts

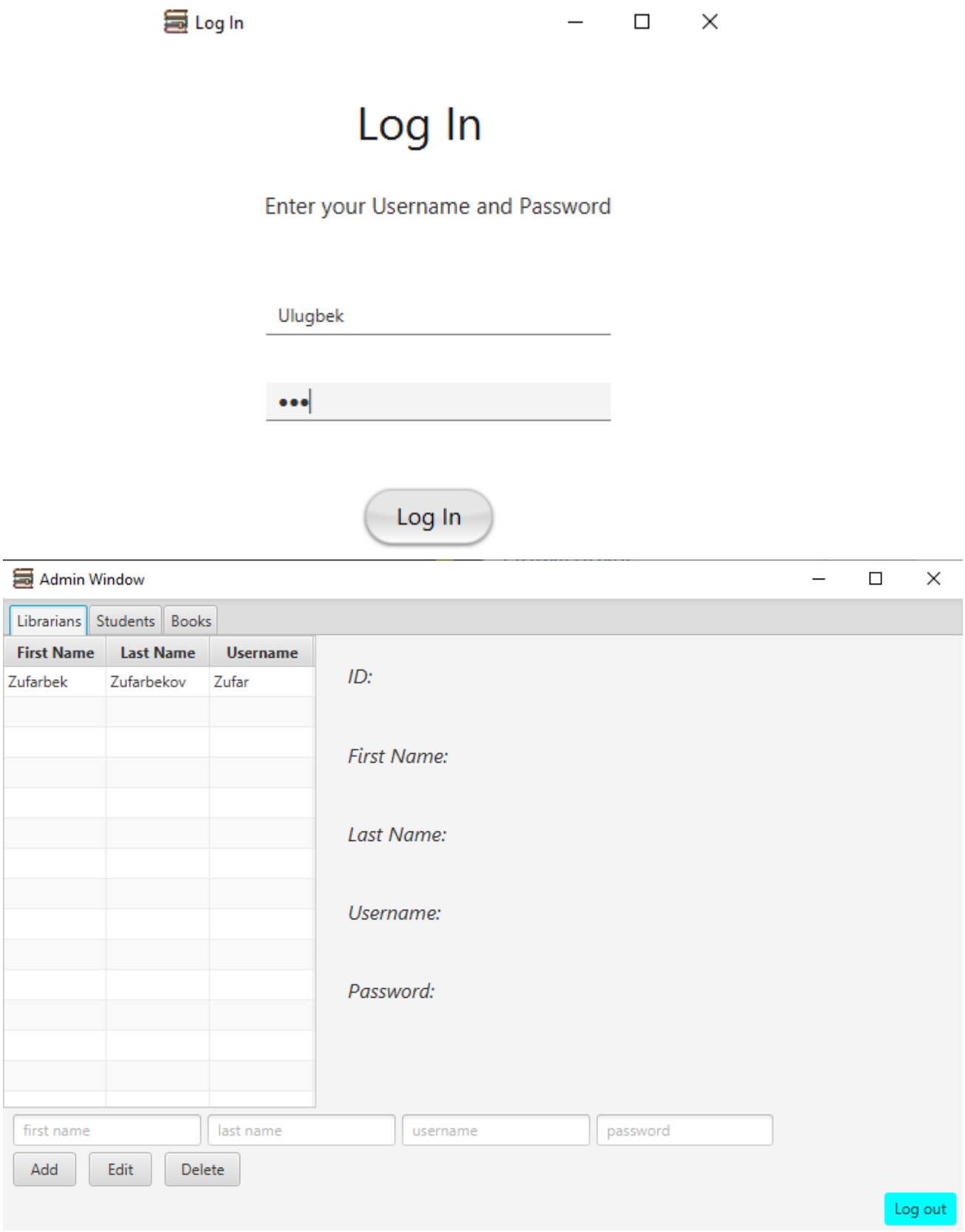
Account	Username	Password
Admin	Ulugbek	123
Librarian	Zufar	456
Student	Bois	789

Description of Implemented Functionality

For our project we implemented JavaFX and Derby
We have three active roles that could use our product: admin, librarian and student

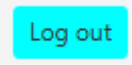
Login package – Boiskhon, Ulugbek contributed

1. In class LogInRepository we connected to our database using method connectDB()
2. In class LoginController we implemented the method logIn() which checks correctness and existing of typed data.
3. Then, depending on the state of method logIn() we introduce function handleLogIn(), which will navigate us through the windows of admin, librarian and student (getting the role string).

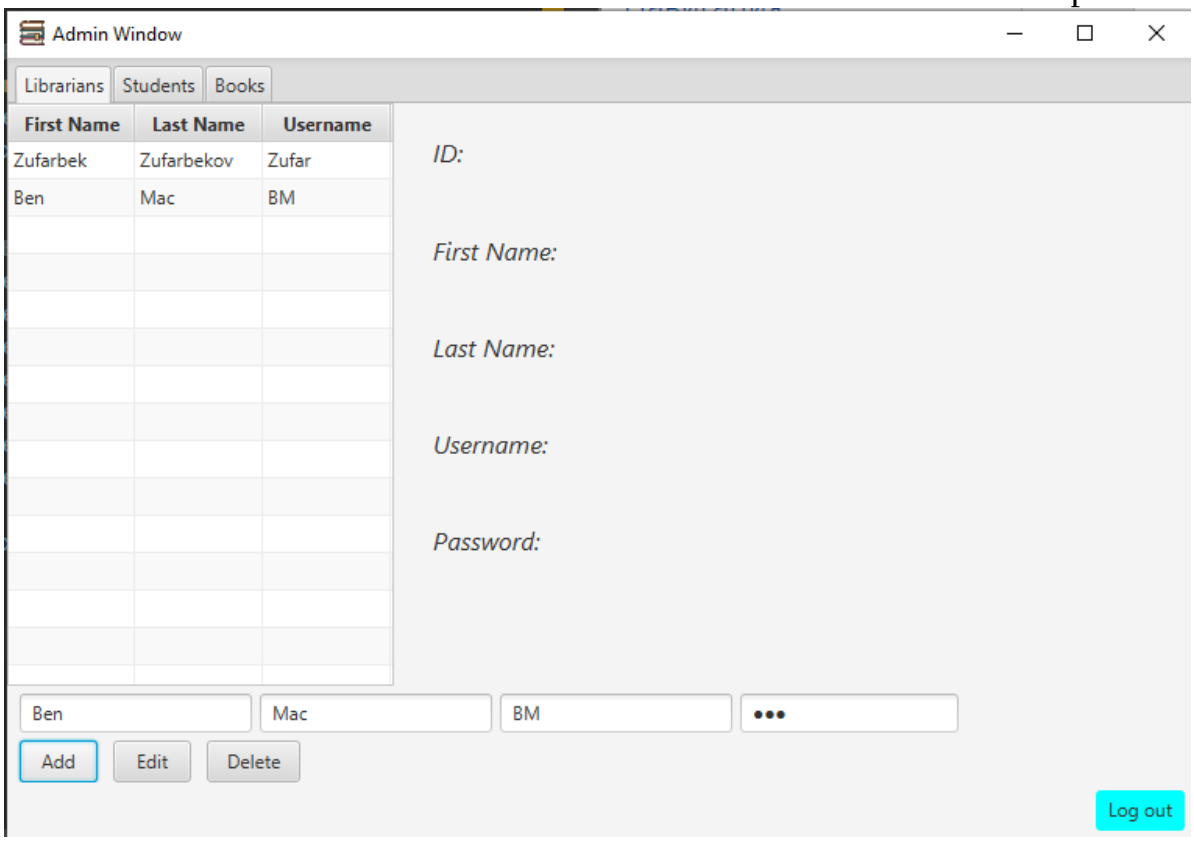


Admin package – Bekzod and Zufarbek collaborated
Objections of administrators

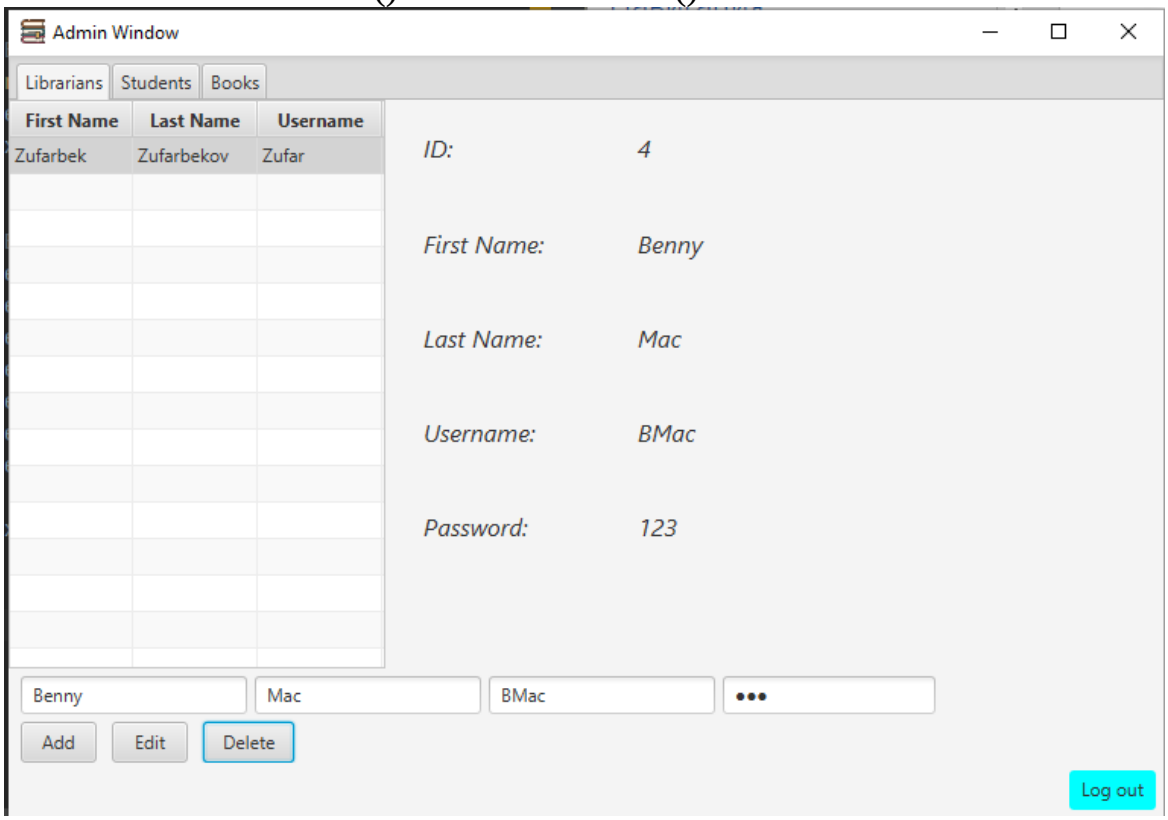
1. In class Controller, method handleLogOut() is responsible for sending you on login page



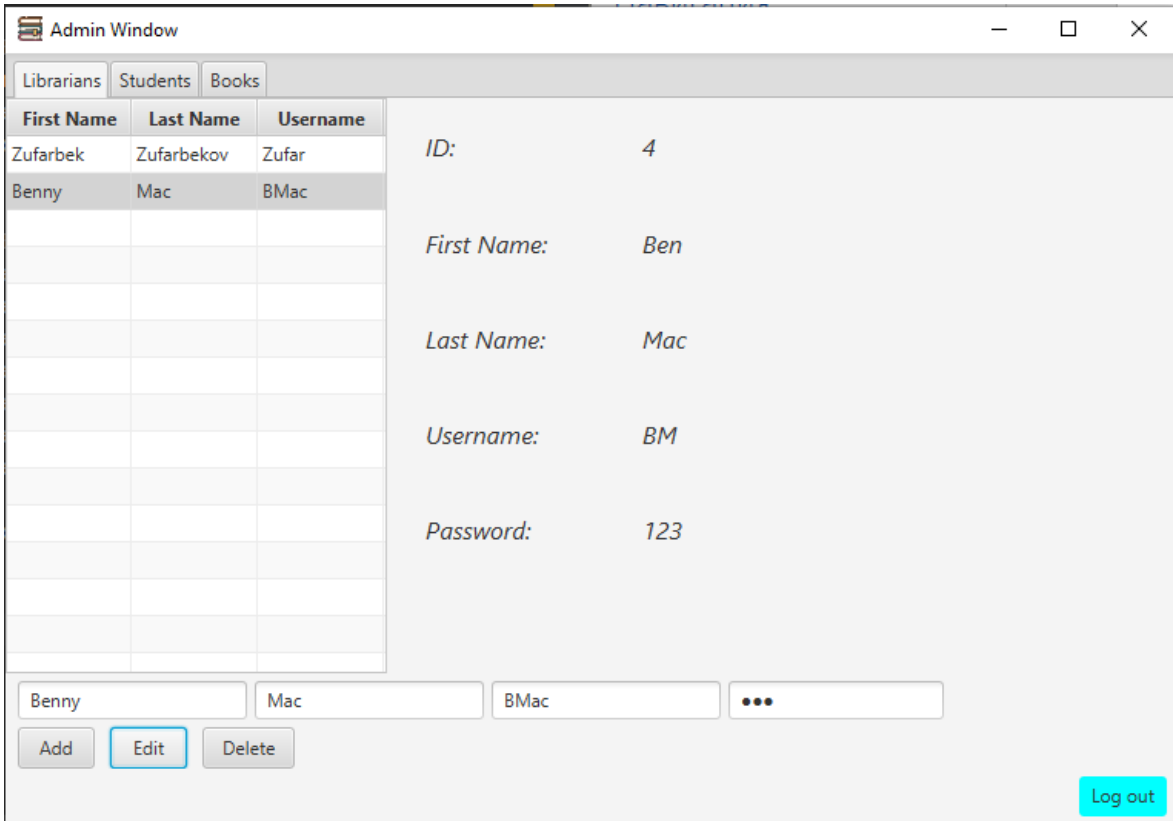
2. Methods addlib(), addStud(), are created to generate new librarians/students and also there included a verification for distinct data that is controlled and prevented



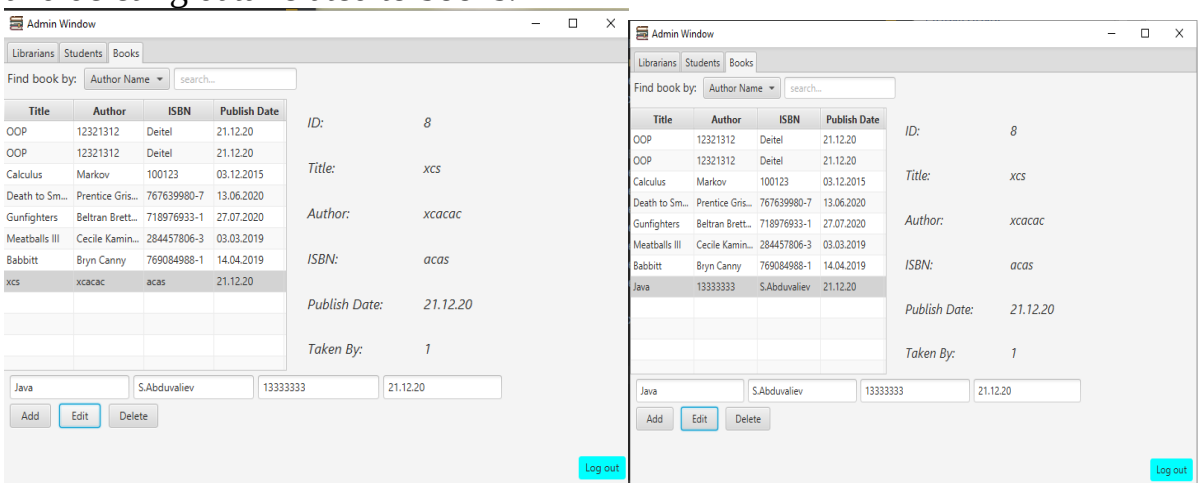
3. Methods deleteLibrarian() and deleteStudent() remove users.



4. Methods updateLibrarian() and updateStudent() alter primarily given data to the modified ones and make a verification for distinct elements.

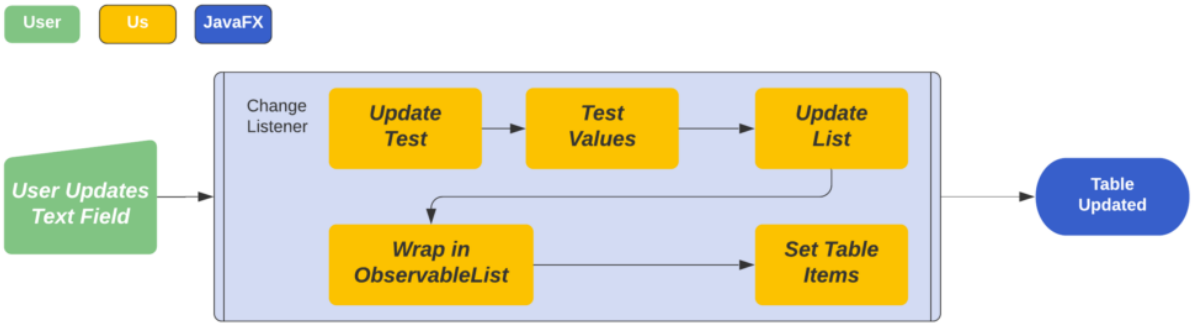


5. Methods addBook(), updateBook() and deleteBook() are responsible for editing and deleting data related to books.

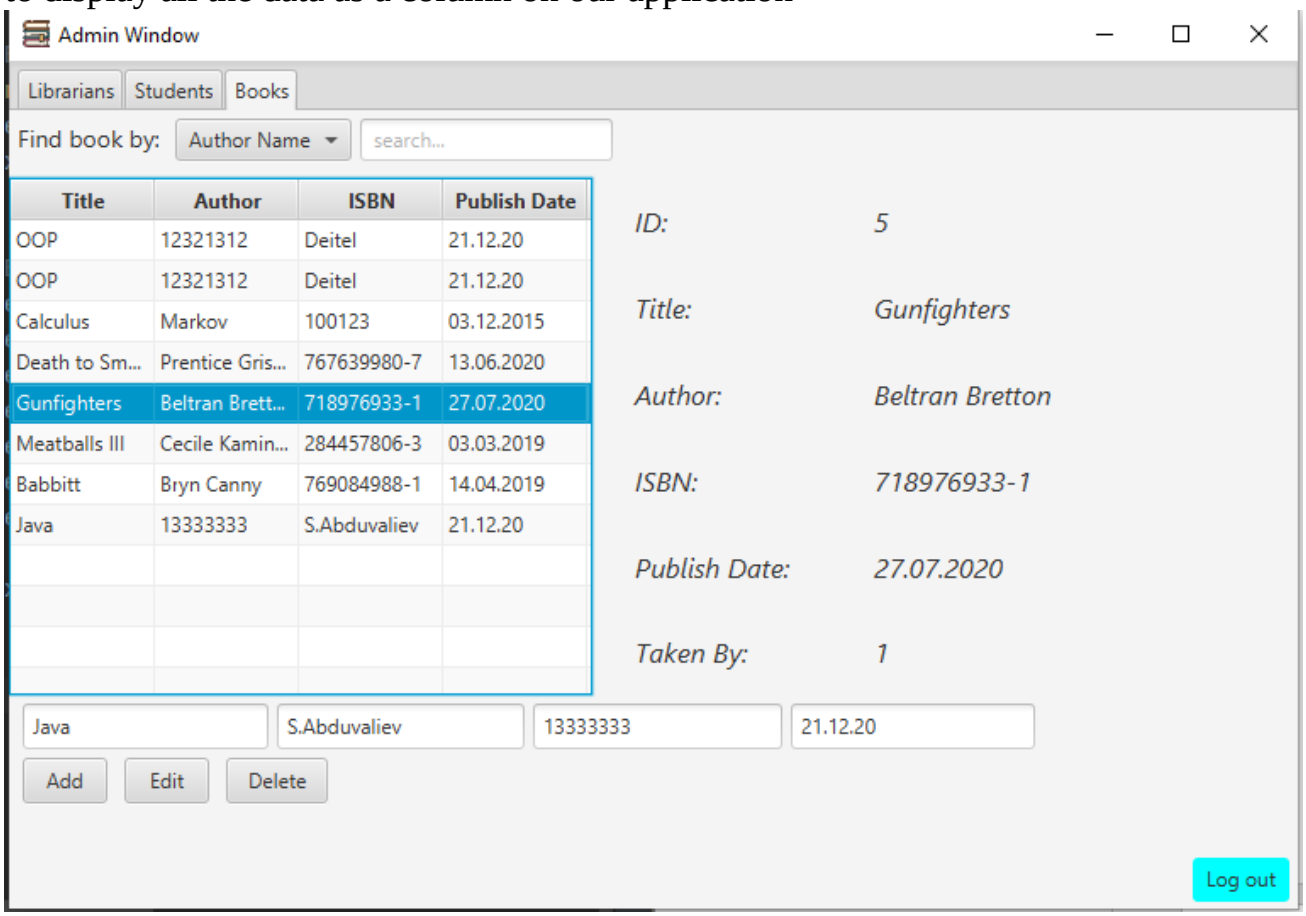


6. Method initialize fills the tables of librarians, students, and books by data taken form data base and adding them to ObservableLists of librarians, students, and books

7. Method filter; dynamically filtering content can be achieved by setting a Predicate on a FilteredList. This predicate should be updated as user input changes the search criteria. Placing a listener on the search box TextField is a common way to achieve this.

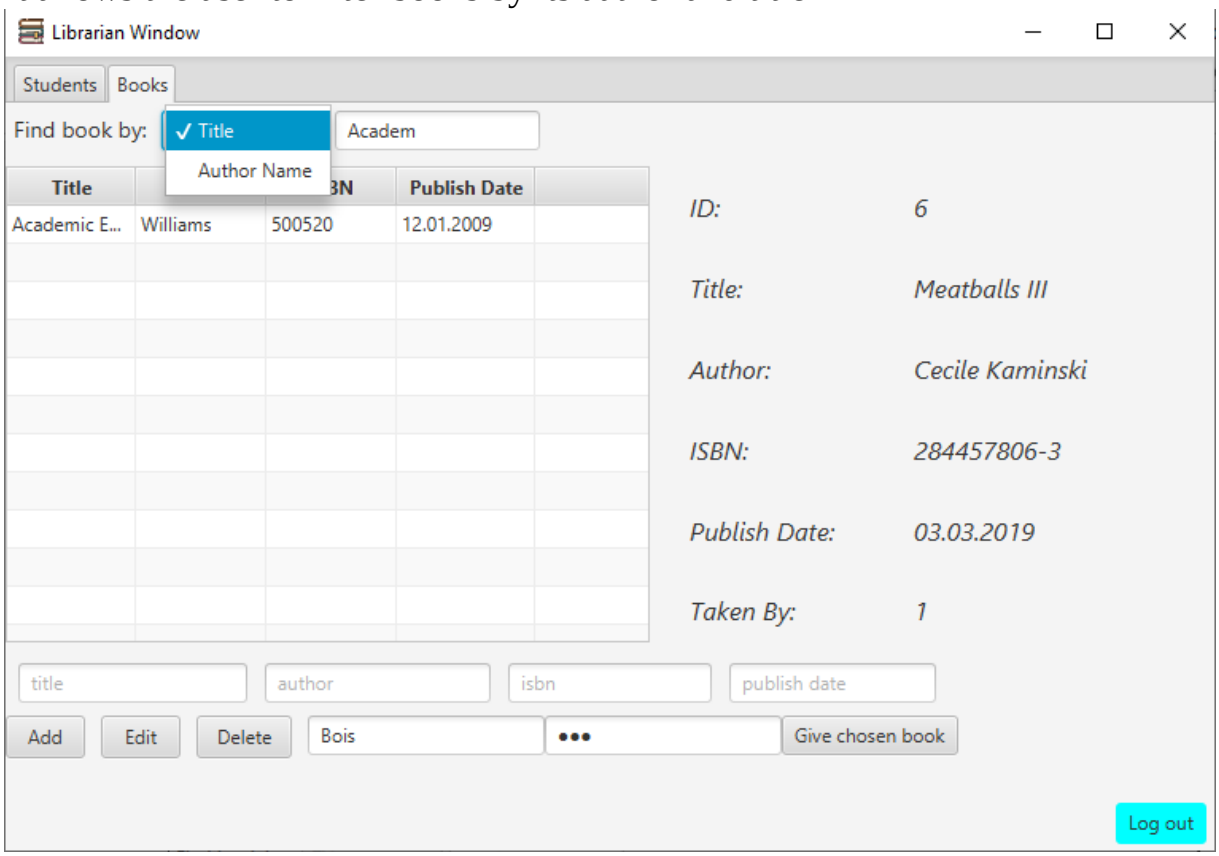


8. Methods displayLibrarian/Student/BookDetails() made for interactive purposes to display all the data as a column on our application

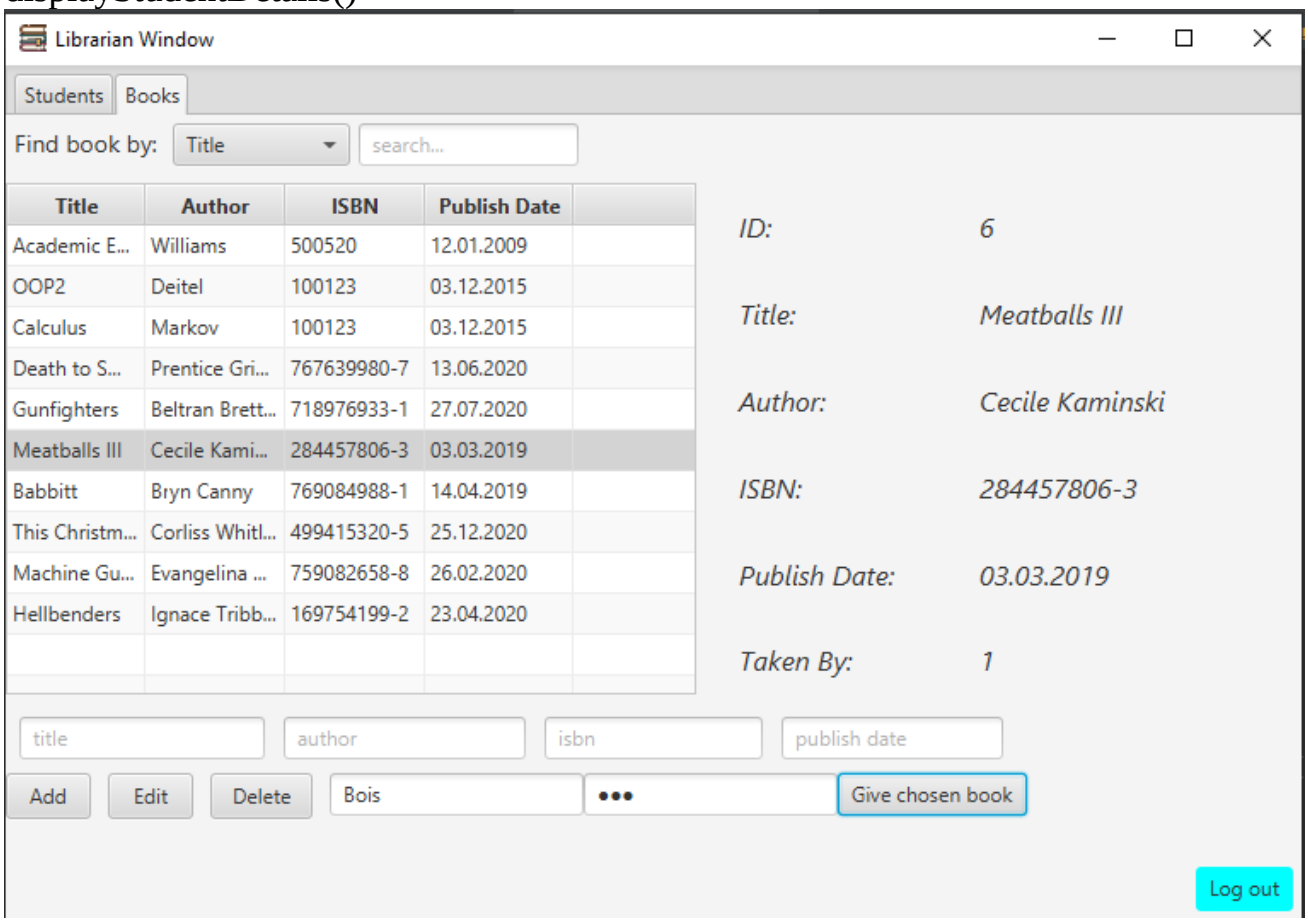


Librarian package – Ulugbek, Azamat collaborated

1. In class Controller of Librarian package, method handleLogOut() is responsible for sending you on login page
2. Method addStud(), is created to generate new students and also there included a verification for distinct data that is controlled and prevented. Screenshot is the same as in admin because they share the same functionality.
3. Method deleteStudent() removes students from local database. Screenshot is the same as in admin because they share the same fucntionality
4. Methods updateStudent() alters primarily given data to the modified one and makes a verification for distinct elements. Screenshot is the same as in admin because they share the same functionality.
5. Methods addBook(), updateBook() and deleteBook() are responsible for editing and deleting data related to books. Screenshot is the same as in admin because they share the same functionality.
6. Method initialize fills the tables of students and books by data taken form data base and adding them to ObservableLists of students and books
7. Method filter; dynamically filtering content can be achieved by setting a Predicate on a FilteredList. This predicate should be updated as user input changes the search criteria. Placing a listener on the search box TextField is a common way to achieve this.
   It allows the user to filter books by its author and title



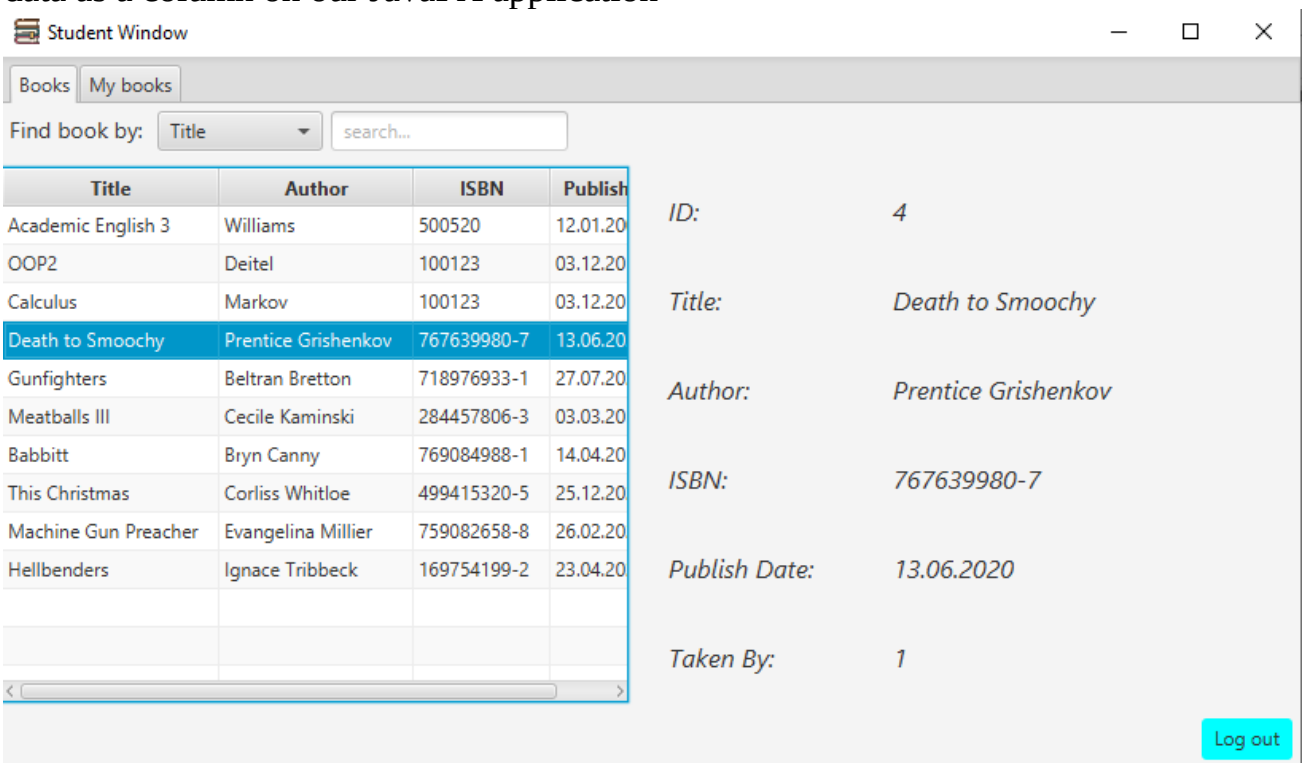
8. Method displayBookDetails() made for interactive purposes to display all the data as a column on our JavaFX application. Screenshot is the same as in admin because they share the same functionality.
9. Method displayStudentDetails() allows the librarian to see all the details of a particular student. Screenshot is the same as in admin because they share the same functionality.
10. Method handleGiveBook() allows the librarian issue a particular book to a Student by inserting his username and password that could be checked in displayStudentDetails()



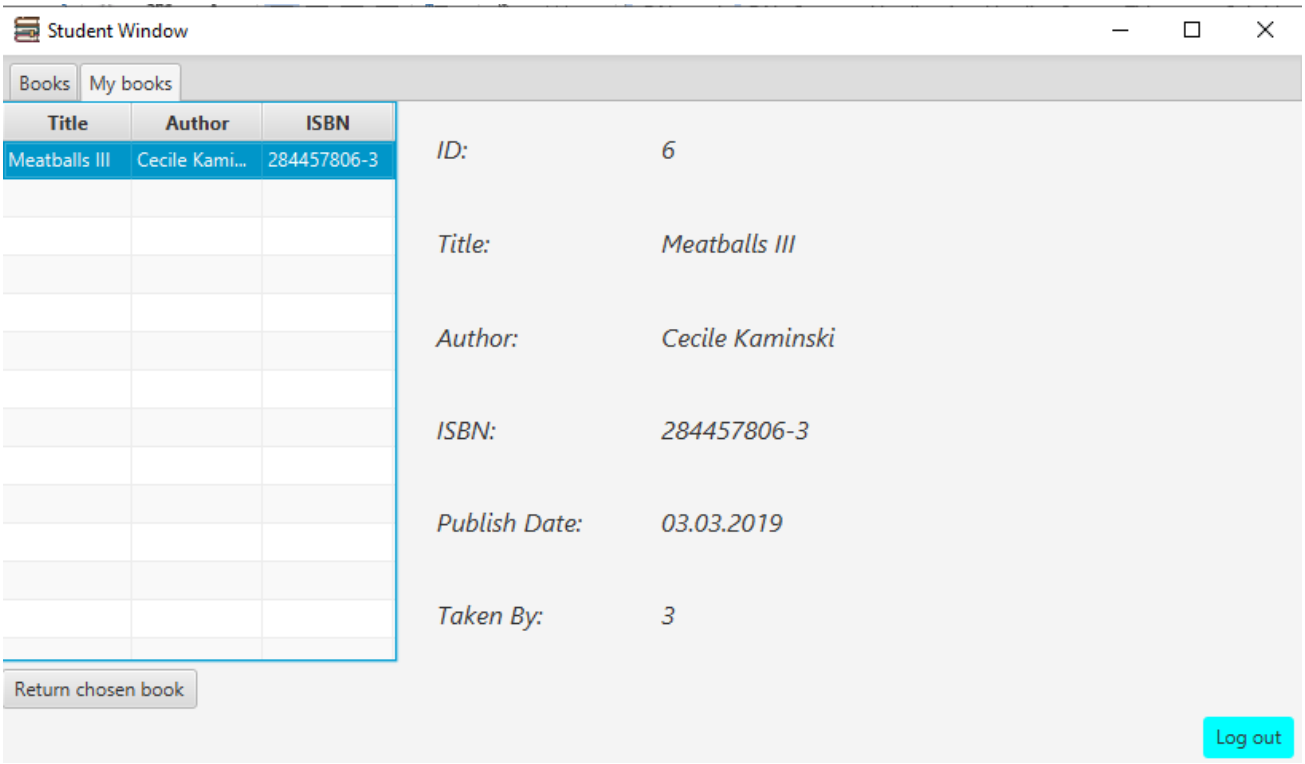
Student package – Bekzod and Zufarbek contributed

1. In class Controller of Student package, method handleLogOut() is responsible for sending you on login page
2. Method initialize() fills the tables of books and myBooks(represents the books taken by a particular student) and adding them to ObservableLists of books and myBooks

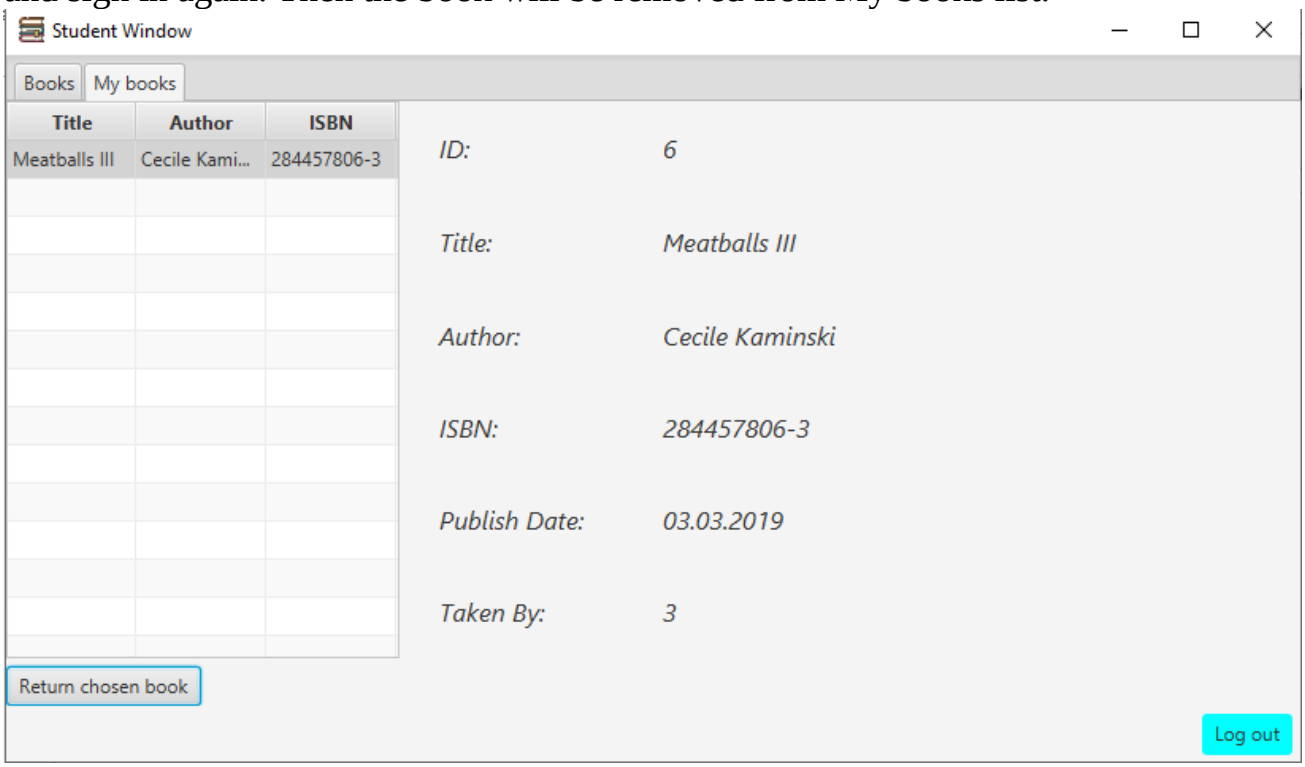
3. Method displayBookDetails() made for interactive purposes to display all the data as a column on our JavaFX application

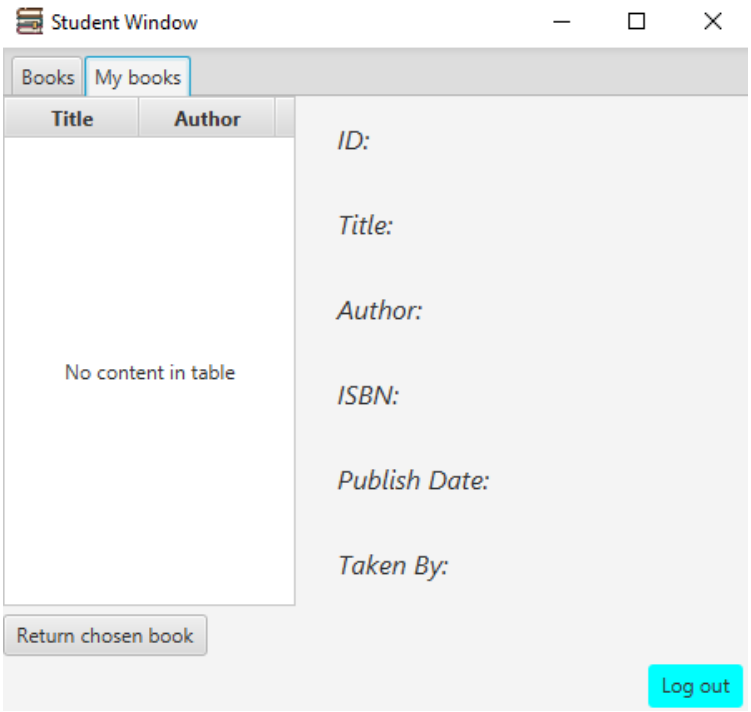


4. Method DisplayMyBookDetails() is designed to display particular Student's borrowed books.



5. Method <u>filter;</u> dynamically filtering content can be achieved by setting a Predicate on a FilteredList. This predicate should be updated as user input changes the search criteria. Placing a listener on the search box TextField is a common way to achieve this. It allows the user to filter books by its author and title. Screenshot is the same as in admin because they share the same functionality.
6. Method handleReturnBook() allows the Student to return issued books. However, books are not removed from the table instantly. You should log out and sign in again. Then the book will be removed from My books list.





ExtraClasses package - Ulugbek, Azamat, Boiskhon, Bekzod and Zufarbek collaborated

Books class

1. Setters and getters are created to initialize the data

User class

1. Setters and getters are created to initialize the data

BookRepository class

This class is used to connect with Data Base and set all necessary Queries such as retrieving the data by getBook() and getAllBooks methods, adding, and deleting books and also updating. In this class we used SIGNL TONE design pattern to not allow to create the object of this class two times, because the data base reference must be created only one time

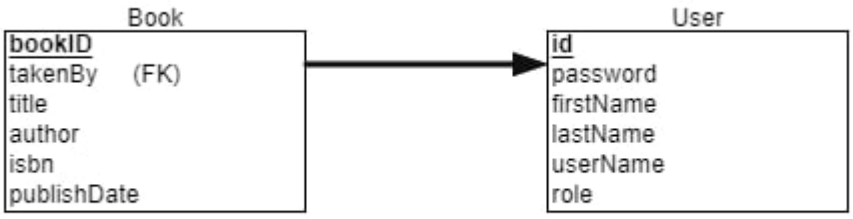
UserRepository class

This class is used to connect with Data Base and set all necessary Queries such as retrieving the data by getAllLivrarians() and getAllLibrarians methods, adding, and deleting users and also updating. In this class we used SIGNL TONE design pattern to not allow to create the object of this class two times, because the data base reference must be created only one time

Overall Experience

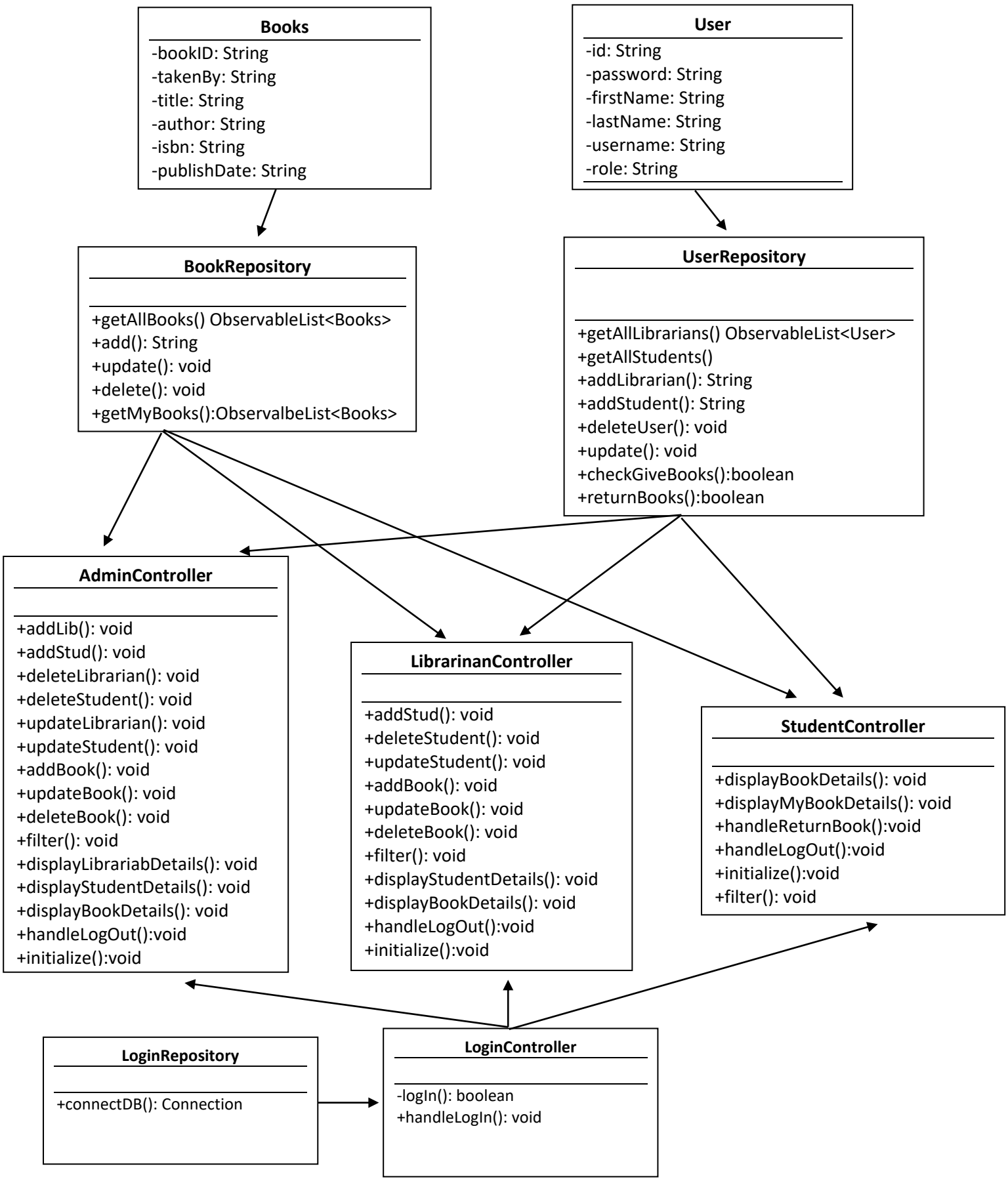
Developing such program was very challenging experience for us because of number of reasons, starting from inability to collaborate together face-to-face and finishing by the lack of experience and knowledge in this sphere. Also it was difficult to organize all colleagues as there were winter holidays and nobody wanted to work these times. Also, very poor knowledge of the sphere of databases also impacted on us, as we were struggling to manage our own methods due to lack of necessary details.

Entity Relationship diagram



In our database we have used only to tables, “User” and “Book” for storing data about users and books respectively. The table “User” is responsible for all the registered users in our application (Administrators, Librarians, Students). One of the benefits of designing single table for Users is that it allows us to add as many “roles” as we want. For example, we can add new role “System Administrator” in the row “role” and sign in to application via logIn() method. We do not need to change our database by creating new table for the new “roles”.
There are also some drawbacks. For example, if we want to add new fields in “User” table we should make changes for all users in our database.
In Book table the takenBy has the value of student’s id as a foreign key to scan the book that are assigned(borrowed) by a particular student. However, takenBy column has value ‘1’ by default and shows that the book is in a library, not taken by a student

UML diagram


Books
-bookID: String
-takenBy: String
-title: String
-author: String
-isbn: String
-publishDate: String
User
-id: String
-password: String
-firstName: String
-lastName: String
-username: String
-role: String
BookRepository
+getAllBooks() ObservableList<Books>
+add(): String
+update(): void
+delete(): void
+getMyBooks():ObservalbeList<Books>
UserRepository
+getAllLibrarians() ObservableList<User>
+getAllStudents()
+addLibrarian(): String
+addStudent(): String
+deleteUser(): void
+update(): void
+checkGiveBooks():boolean
+returnBooks():boolean
AdminController
+addLib(): void
+addStud(): void
+deleteLibrarian(): void
+deleteStudent(): void
+updateLibrarian(): void
+updateStudent(): void
+addBook(): void
+updateBook(): void
+deleteBook(): void
+filter(): void
+displayLibrariabDetails(): void
+displayStudentDetails(): void
+displayBookDetails(): void
+handleLogOut():void
+initialize():void
LibrarinanController
+addStud(): void
+deleteStudent(): void
+updateStudent(): void
+addBook(): void
+updateBook(): void
+deleteBook(): void
+filter(): void
+displayStudentDetails(): void
+displayBookDetails(): void
+handleLogOut():void
+initialize():void
StudentController
+displayBookDetails(): void
+displayMyBookDetails(): void
+handleReturnBook():void
+handleLogOut():void
+initialize():void
+filter(): void
LoginRepository
+connectDB(): Connection
LoginController
-logIn(): boolean
+handleLogIn(): void