**Final Project Report**

Student Name: Nariman Abubakirov

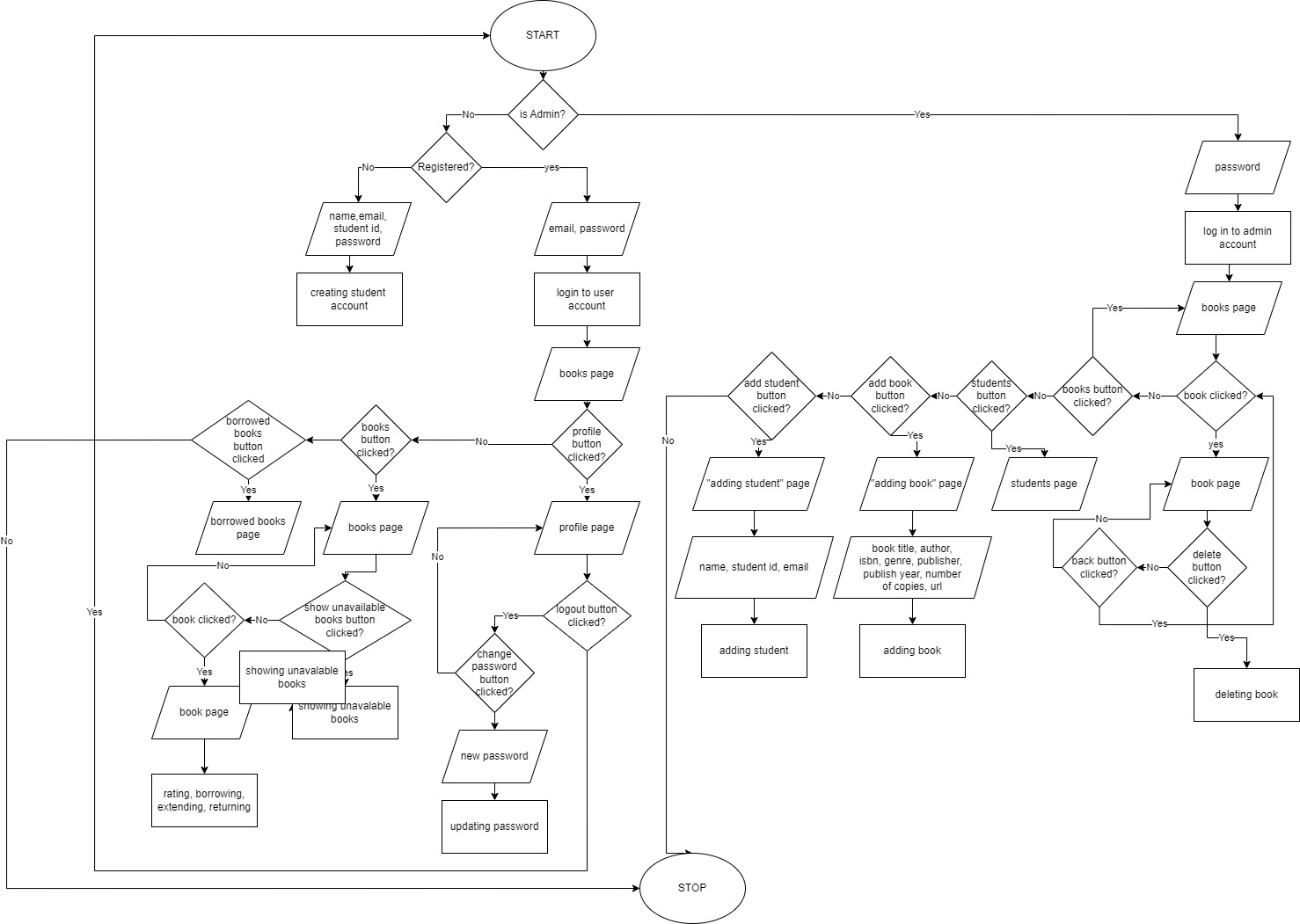
Sorry, I did not have time for comments, my project is too complex. If you want, I can send you the commented version 😊

The password for admin is “bigboss”

**1. Briefly describe the project purpose:**

The main purpose of my project is to provide a library management system. In the application students can create their accounts, log in to them, borrow and return books, rate books, extend the return date for the book, see available and unavailable books, see their borrowing history, see recommended books based on the last taken book, see book information and see their borrowed books. Therefore, one of the purposes of this project is to provide students with complex tool for borrowing and returning books from the library. On the other hand, admin can see the whole borrowing history of the library, add books, see added books, delete books, see book information, see students who borrowed the chosen book, see students, and add students to the system. Both student and admin can change their passwords in the profile page if they want. Therefore, another purpose of this project is to provide librarian with an environment to manage library if effective way. The application takes data only from the online database meaning that this system will work on any device that has internet connection.

**2. Draw the logic flow of the program (with flowchart):**



(I will include flow chart to zip file)

**3. Provide screenshots for each screen with brief description:**

Graphical user interface, application, website

Description automatically generated

This is a login page. Student can log in to his account or create account by pressing “create account”. Admin to log in should press “admin login” and enter his password.

Graphical user interface, text, application

Description automatically generated

This is a registration page. Student can create his account here. By pressing back button, user is returned to login page.

Graphical user interface, text, application, email

Description automatically generated

When student logs in to his account they can see table of books. Student can press “show unavailable books” button to show unavailable books. They can also search through the table. Student can return to this page by pressing “books” on the dashboard.

Graphical user interface

Description automatically generated

This is book’s page that students see. It is opened when student double clicks on the book. All necessary information about the books is shown here. Student can borrow the book. Here student can also rate the book when he borrows it at least once. Student can press back button to return to the table with books.

Text

Description automatically generated

This is how book’s page looks when it is borrowed and rated by students. Student can rate the book himself, return the book and extend the returning date, but it can only be done once.

Graphical user interface, application

Description automatically generated

This is the page with borrowed books. It can be accesses by pressing “borrowed books” on the dashboard. Here student can see his borrowed books, he can also double press on books to open their pages and search through the table.

Graphical user interface, timeline

Description automatically generated

This is profile page. It is accessible by pressing button with the profile icon on dashboard. Here student can see his information, recommended books based on the last borrowed book, and his borrowing history. By pressing log out button student will be returned to login page and he will have to re log in again. Changing password button allows to change the password.

Graphical user interface, text, application

Description automatically generated

This page opens by pressing “change password” button on the profile page. This page allows to change the password.

Graphical user interface, application

Description automatically generated

Now when admin logs in to his account, books list is shown but it is different from student’s. Here admin can see all books and their main information.

Graphical user interface, text, application, chat or text message

Description automatically generated

By double clicking on the book admin will open book’s page. Here admin can see book’s information and students who borrowed this book. Admin can also delete this book.

Graphical user interface, application

Description automatically generated

This page is accessible by pressing “add book” button on the dashboard. Here admin can add new book. There are many data checks involved to prevent admin’s most possible mistakes.

Graphical user interface, application, email

Description automatically generated

This Page is accessible by pressing “students” button on the dashboard. Here admin can see all students and their main information.

Graphical user interface, text, application

Description automatically generated

This page is accessible by pressing “add student” button on the dashboard. Here admin can create a new student so that student will not have to register himself if he does not want to. Temporary password is set to “bestlibrary”.

Graphical user interface, text, application, chat or text message

Description automatically generated

This is admin’s profile page. It is accessible by pressing button with the profile icon on dashboard. Here admin can see the list the borrowing history (it is also written to the file inside the project directory). Just like student admin can log out and change his password.

**4. Explain the code of the main functionalities**

**In my program there are too many main functionalities, so I will only explain some of them! Please notice that my project is too big to comment and explain everything. 😊**

1. @Override
2. **public** **void** addBook(String title, String author, String isbn, String publisher, String yearString, String genre, String numberString, String urlString, JPanel panel, JFrame frame) {
3. **if** (title.length() > 0 && author.length() > 0 && isbn.length() > 0 && publisher.length() > 0 && yearString.length() > 0 && numberString.length() > 0 && urlString.length() > 0 &&
4. !title.equals("Title") && !author.equals("Author") && !isbn.equals("ISBN") &&
5. **!publisher.equals("Publisher") && !yearString.equals("Publication Year") && !genre.equals("Choose a genre") && !numberString.equals("Number of Copies") && !urlString.equals("URL of Image")) {**
6. **if** (validateIsbn13(isbn)) {
7. **int** flag = 0;
8. **try** {
9. year = Integer.parseInt(yearString);
10. **if (year > 2022 || year < 1) {**
11. **throw** **new** Exception();
12. }
13. } **catch** (NumberFormatException ex) {
14. JOptionPane.showMessageDialog(**null**, "Enter a number in Publication Year", "Warning", JOptionPane.ERROR\_MESSAGE);
15. **flag++;**
16. } **catch** (Exception e1) {
18. JOptionPane.showMessageDialog(**null**, "Wrong Year", "Warning", JOptionPane.ERROR\_MESSAGE);
19. flag++;
20. **}**
21. **try** {
22. number = Integer.parseInt(numberString);
23. **if** (number < 1) {
24. **throw** **new** Exception();
25. **}**
26. } **catch** (NumberFormatException ex) {
27. JOptionPane.showMessageDialog(**null**, "Enter a number in Number of Copies", "Warning", JOptionPane.ERROR\_MESSAGE);
28. flag++;
29. } **catch** (Exception e1) {
30. **JOptionPane.showMessageDialog(null, "Wrong Number of Copies", "Warning", JOptionPane.ERROR\_MESSAGE);**
31. flag++;
32. }
33. **if** (flag == 0) {
34. **for** (Component cp: panel.getComponents()) {
35. **cp.setEnabled(false);**
36. }
38. SwingWorker < Boolean,
39. Void > urlWorker = **new** SwingWorker < Boolean,
40. **Void > () {**
41. **protected** Boolean doInBackground() {
42. **if** (testImage(urlString)) **return** **true**;
43. **else** **return** **false**;
45. **}**
47. **protected** **void** done() {
48. **try** {
49. Boolean checkBoolean = get();
50. **if (checkBoolean) {**
51. Book book = **new** Book(title, author, isbn, publisher, year, genre, number, urlString, 0, 0, 0);
52. DatabaseReference refBook = database.getReference("books");
54. **final** Query query1 = refBook.orderByChild("isbn").equalTo(isbn);
55. **query1.addListenerForSingleValueEvent(new ValueEventListener() {**
56. @Override
57. **public** **void** onDataChange(DataSnapshot dataSnapshot1) {
58. **if** (!dataSnapshot1.exists()) {
60. **refBook.child(isbn).setValueAsync(book);**
62. JOptionPane.showMessageDialog(**null**, "Book was successfully added", "Success", JOptionPane.INFORMATION\_MESSAGE);
64. } **else** {
65. **JOptionPane.showMessageDialog(null, "Book with such ISBN already exists", "Warning", JOptionPane.ERROR\_MESSAGE);**
66. }
67. **for** (Component cp: panel.getComponents()) {
68. cp.setEnabled(**true**);
69. }
71. }
73. @Override
74. **public** **void** onCancelled(DatabaseError databaseError1) {
75. **JOptionPane.showMessageDialog(null, "Canceled", "Warning", JOptionPane.ERROR\_MESSAGE);**
76. **for** (Component cp: panel.getComponents()) {
77. cp.setEnabled(**true**);
78. }
79. }
80. **});**
81. } **else** {
82. JOptionPane.showMessageDialog(**null**, "Wrong Image URL", "Warning", JOptionPane.ERROR\_MESSAGE);
83. **for** (Component cp: panel.getComponents()) {
84. cp.setEnabled(**true**);
85. **}**
86. }
88. } **catch** (Exception e) {
89. JOptionPane.showMessageDialog(**null**, "Error", "Error", JOptionPane.ERROR\_MESSAGE);
90. **for (Component cp: panel.getComponents()) {**
91. cp.setEnabled(**true**);
92. }
93. }
94. }
95. **};**
96. urlWorker.execute();
97. }
99. } **else** {
100. **JOptionPane.showMessageDialog(null, "ISBN is incorrect", "Warning", JOptionPane.ERROR\_MESSAGE);**
101. }
102. } **else** {
103. JOptionPane.showMessageDialog(**null**, "Fill all the fields", "Warning", JOptionPane.ERROR\_MESSAGE);
104. }
106. }

In this method which is an overridden method in admin class book gets added to the Firebase real time database. First it validates all the actual parameters it receives and if everything is right, swing worker validates the url of the image and if the url is right all of the data is passed to the Book class object. Then there is a query to check if there are other books with such ISBN in the database already. If there are no books with such ISBN, the book object will be used to set data to database.

1. registerButton.addActionListener(**new** ActionListener() {
2. **public** **void** actionPerformed(ActionEvent e) {
4. String email = textField.getText().trim();
5. **String name = textField\_1.getText().trim();**
6. String studentID = studentIDField.getText().trim();
7. String password = **new** String(passwordField.getPassword()).trim();
8. String password\_again = **new** String(passwordField\_1.getPassword()).trim();
9. **if** (email.length() > 0 && name.length() > 0 && studentID.length() > 0 && password.length() > 0 && password\_again.length() > 0) {
11. **if** (isEmailValid(email)) {
13. **if** (studentID.length() == 10 && onlyDigits(studentID)) {
14. **if** (password.length() > 5) {
16. **if** (password.equals(password\_again)) {
17. counter = 0;
18. **for** (Component cp: panel\_B.getComponents())
19. cp.setEnabled(**false**);
20. **for (Component cp: panel.getComponents())**
21. cp.setEnabled(**false**);
22. **final** Query query1 = ref.orderByChild("studentID").equalTo(studentID);
23. query1.addListenerForSingleValueEvent(**new** ValueEventListener() {
24. @Override
25. **public void onDataChange(DataSnapshot dataSnapshot1) {**
26. **if** (dataSnapshot1.exists()) {
28. counter++;
30. **}**
32. **if** (counter == 0) {
33. SwingWorker < Boolean, Void > registrationWorker = **new** SwingWorker < Boolean, Void > () {
34. **protected** Boolean doInBackground() {
35. **CreateRequest request = new CreateRequest()**
36. .setEmail(email)
37. .setEmailVerified(**false**)
38. .setPassword(password)
39. .setDisplayName(name)
40. **.setDisabled(false);**
42. UserRecord userRecord = **null**;
43. **try** {
44. flag = 0;
45. **userRecord = FirebaseAuth.getInstance().createUser(request);**
46. **if** (flag == 0) {
47. Student student = **new** Student(name, email, studentID, password);
49. ref.child(studentID).setValueAsync(student);
51. **return** **true**;
52. }
54. } **catch** (FirebaseAuthException e1) {
56. flag++;
57. JOptionPane.showMessageDialog(**null**, e1.getMessage(), "Error",
58. JOptionPane.ERROR\_MESSAGE);
59. **return** **false**;
61. }
62. **return** **false**;
63. }
65. **protected void done() {**
66. **try** {
67. **boolean** loginSuccessful = get();
68. **if** (loginSuccessful) {
69. JOptionPane.showMessageDialog(**null**, "Account was successfully created", "Sucess",
70. **JOptionPane.INFORMATION\_MESSAGE);**
71. LoginPage login = **new** LoginPage();
73. **if** (getExtendedState() != MAXIMIZED\_BOTH) {
74. login.setSize(getSize());
75. **login.setLocation(getX(), getY());**
76. } **else** {
77. login.setLocationRelativeTo(**null**);
78. }
79. login.setExtendedState(getExtendedState());
81. login.setVisible(**true**);
82. setVisible(**false**);
83. } **else** {
84. **for** (Component cp: panel\_B.getComponents()) {
85. **cp.setEnabled(true);**
86. }
87. **for** (Component cp: panel.getComponents()) {
88. cp.setEnabled(**true**);
89. }
90. **}**
91. } **catch** (Exception e) {
92. JOptionPane.showMessageDialog(**null**, e.getMessage(), "Error",
93. JOptionPane.ERROR\_MESSAGE);
94. **for** (Component cp: panel\_B.getComponents()) {
95. **cp.setEnabled(true);**
96. }
97. **for** (Component cp: panel.getComponents()) {
98. cp.setEnabled(**true**);
99. }
100. **}**
101. }
102. };
103. registrationWorker.execute();
104. } **else** {
105. **JOptionPane.showMessageDialog(null, "Student with such user ID is already registered", "Warning", JOptionPane.ERROR\_MESSAGE);**
106. **for** (Component cp: panel\_B.getComponents()) {
107. cp.setEnabled(**true**);
108. }
109. **for** (Component cp: panel.getComponents()) {
110. **cp.setEnabled(true);**
111. }
112. }
114. }
116. @Override
117. **public** **void** onCancelled(DatabaseError databaseError1) {
118. JOptionPane.showMessageDialog(**null**, "Canceled", "Warning", JOptionPane.ERROR\_MESSAGE);
119. **for** (Component cp: panel\_B.getComponents()) {
120. **cp.setEnabled(true);**
121. }
122. **for** (Component cp: panel.getComponents()) {
123. cp.setEnabled(**true**);
124. }
125. **}**
126. });
128. } **else** {
129. JOptionPane.showMessageDialog(**null**, "Passwords do not match", "Password",
130. **JOptionPane.ERROR\_MESSAGE);**
131. }
133. } **else** {
134. JOptionPane.showMessageDialog(**null**, "Password should have at least 6 characters", "Password",
135. **JOptionPane.ERROR\_MESSAGE);**
136. }
137. } **else** {
138. JOptionPane.showMessageDialog(**null**, "Enter correct Student ID number", "Student ID",
139. JOptionPane.ERROR\_MESSAGE);
140. **}**
142. } **else** {
143. JOptionPane.showMessageDialog(**null**, "Email address that you entered is not valid", "Email",
144. JOptionPane.ERROR\_MESSAGE);
145. **}**
147. } **else** {
149. JOptionPane.showMessageDialog(**null**, "Fill all the fields", "Warning",
150. **JOptionPane.ERROR\_MESSAGE);**
151. }
153. }
154. });

This is the code when student presses “sign up” button in the “create account” page. Here program first gets the values from text fields and then there are many input validations involved to ensure that account with the right information is created. Before doing a time consuming task, components on both panels are disabled so that user will not press anything while the program is trying to create an account. Query is made to check if there are any students with students ID that the student entered in the database, if there are – counter is increased. If counter is 0, meaning that there are no students with such student ID in the database, swing worker starts do its job. In the background its creating a request to set parameters and information to the user that will be added to Firebase Authentication System. Then it tries to add user record using the request object to the FirebaseAuth system, if it is successful student information is also added to the database. When Swing Worker is done, success or not success message is shown and if everything is alright user is sent to login page.

**5. Explain what is included in your project and why it is used (Polymorphism, Inheritance, File I/O, etc)**

In my program I have the **interface** Usable to specify a method prototype**. Abstract class User** implements Usable interface, since it does not override Usable’s method. Class User specifies fields that are shared by both student and admin and it is abstract because we want to leave the implementation to a class that extends it. Class Admin and class Student both **inherit** User class. Inheritance is used to extend the User class and override its abstract method.

**Polymorphism** is also included in my project since class Admin and class Student override addBook() method differently. Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance. It allows addBook() method to behave in multiple forms.

**Firebase Authentication (Online Authentication system)** used mostly for sign up, sign in, and additional input validation concerning signing in and signing up.

**Firebase Real Time Database (Online Database)** is the core of this program. It is used for everything in this program since this program’s functions involve managing data.

**Gradle** is used to include dependencies on libraries and plugins to the project.

**Plugins.** FlatLaf plugin is used for look and feel and shadowJar plugin is used to create a fat JAR, a jar file that includes libraries mentioned in the gradle file.

**Swing** is used to create a GUI in my program, this is the only GUI toolkit that is used in my program.

**Threads.** There are many Swing Workers used in my program. SwingWorker enables proper use of the event dispatching thread. When receiving and writing to database, threading is also used since onDataChange() method executes all the code inside of it before going further, its behavior is somewhat similar to SwingWorker’s doInBacground() and done().

**File I/O.** When admin goes to his profile page data about the borrowing history is written to the file borrowing\_history.txt. This is to make it possible for admin to get the text file that can be useful for further investigation of borrowing history. The content of the file is then written to the text area to not retrieve the data from database again.