



Assignment Cover Letter

(Individual Work)

Student Information:	1.	Surname Vanessa	Given Names Vicky	Student ID Number 2201807791
----------------------	----	--------------------	----------------------	---------------------------------

Course Code	: COMP6510	Course Name	: Programming Language
-------------	------------	-------------	------------------------

Class	: L2BC	Name of Lecturer(s)	: Minaldi Loeis
-------	--------	---------------------	-----------------

Major	: CS
-------	------

Title of Assignment (if any)	: Library Storage System
---------------------------------	-----------------------------

Type of Assignment	: Final Project
--------------------	-----------------

Submission Pattern

Due Date	: 01 - 07 - 2019	Submission Date	: 01 - 07 - 2019
----------	------------------	-----------------	------------------

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:
Vicky Vanessa

(Name of Student)

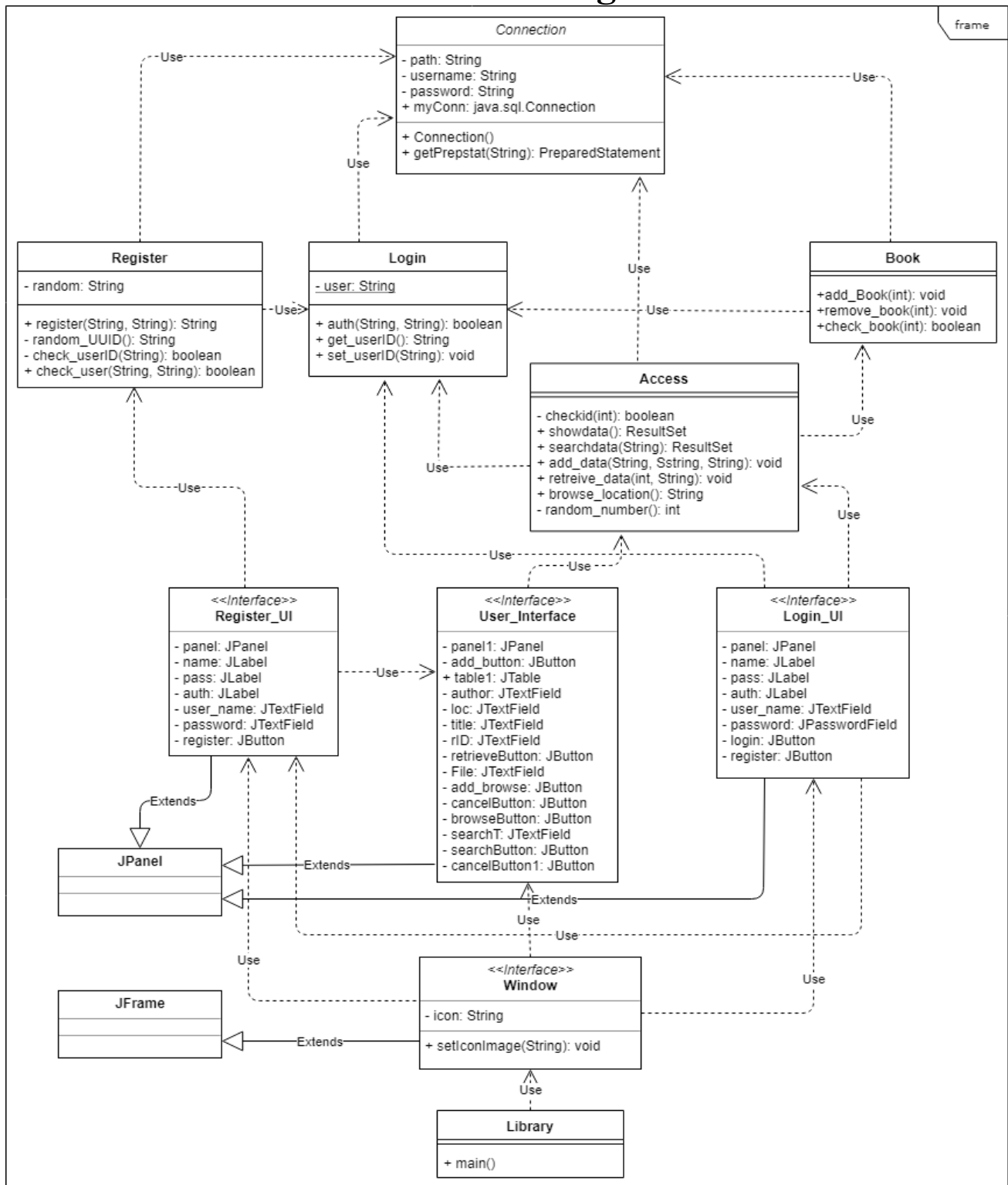
Table of Contents

I.	Cover.....	1
II.	Table of Contents.....	2
III.	Description.....	3
IV.	Solution Design.....	3
V.	Discussion	
	i. Implementation.....	4
	ii. How it works.....	4
	iii. Code explanation.....	5
VI.	Evidence.....	9
VII.	References.....	11

Description

This project is about a Library Storage System, as the name imply, it is a storage system which store a small sized file like a pdf file in the program, and it can be inserted or retrieved like a library would. This system use database, and graphical user interface. The database is used as the storage system to store the data and information of the file, and GUI is used to make it easier for the user to view and use the program.

Solution Design



Discussion

Implementation:

- java.sql – to help process the data from and to the database
- java.swing – to make the GUI
- java.awt – to help the layout management and the event happening on the GUI
- java.util.UUID – to generate random unique ID for each of the user
- java.util.Random – to generate random unique ID for the book
- java.io – to retrieve the blob file in the database table and download it to the selected file location

How it works:

- The registering part of the program is for new user, they enter their username and password, and the program will generate a new random ID for every new user that will be stored in the database table.
- When the user logged in, the program will check the inputted username and password whether it is in the database table or not. If yes, the program will store the ID of the user and direct the user to their library and load the saved/stored data, else, it will send a warning to them.
- The inserted information considered as book in the program will store a blob file from the data selected. While the data is processed, it will generate random number id for the book to use as identification of the book as book(s) can have the same title, but they will have different identification.
- One of the database table specifically used to store the book(s) under the logged user by their unique ID and the book ID. The program will check which book(s) are connected to the logged user using their ID and will show the user which book(s) belong to them in the program.
- It will delete/retrieve book based on the book ID and check whether the logged user own the specified book in the database or not. Then, it will write the file to the selected file location and specified file name and extension.
- The search function is to search the book(s) by title and update the table with the resulted data every time the user clicked search or cancel.

Code explanation:

- Connection Class

This class is specifically made to connect the java code with the database and to make easier access of it in the other classes.

```
myConn = DriverManager.getConnection(path, username, password);
```

This line of code is to connect the code with the database. The path is the name of the database in the localhost and the username and password are the one used to open and access the database.

```
PreparedStatement prepStat = myConn.prepareStatement(query);  
return prepStat;
```

This part of the code in the getPrepStat method is for the other class have easier access to use the Prepared Statement without having to repeatedly having to return the connection.

- Login Class

This class handles the user authentication of whether their account already registered or not in the login table.

```

public boolean auth(String name, String password){
    try {
        PreparedStatement prepStat = connect.getPrepstat( query: "SELECT user_id, user_name, pass FROM login " +
            "WHERE user_name = ? AND pass = ?");
        prepStat.setString( parameterIndex: 1, name);
        prepStat.setString( parameterIndex: 2, password);
        ResultSet myRs = prepStat.executeQuery();
        if(myRs.next()){
            user = myRs.getString( columnLabel: "user_id");
            System.out.println("User " + myRs.getString( columnLabel: "user_name") + " Successfully Logged In!");
            return true;
        }
    }
}

```

It will check the username and password inputted by the user and set it to the parameter in the query. myRs.next() is to check whether there is someone with that data or not. If there is a next, it will set the user string to the user ID from the ResultSet and it will return true to get access in the login panel in the code.

```

private static String user;

```

This string is set to be static so that even though this class is instantiated by many classes, this string will not change and can be used. If it is not set to be static, by the many instance of this class, it will be set to null and therefore cannot be used.

The setter and getter for the user string is used to set the string with the randomized user ID when the user registered and for the other class to get the string to check the data owned by the user.

- Register Class

This class it used for registering purposes for the new user.

```

public void register(String user_name, String password){
    try {
        PreparedStatement prepStat = connect.getPrepstat( query: "INSERT INTO login " +
            "VALUES(?, ?, ?)");
        prepStat.setString( parameterIndex: 1, random);
        prepStat.setString( parameterIndex: 2, user_name);
        prepStat.setString( parameterIndex: 3, password);
        int i = prepStat.executeUpdate();
        if(i > 0){
            login.set_userID(random);
            System.out.println( "Successfully Registered!");
        }
    }
}

```

This method enters the username, password, and the randomized ID into the login table in the database. If(i > 0) is to check whether the data is successfully entered into the database or not, if it is successfully registered, it will set the randomized ID into the user string in the login class.

```

public boolean check_user(String user, String password){
    try {
        PreparedStatement prepStat = connect.getPrepstat( query: "SELECT user_name, pass FROM login " +
            "WHERE user_name = ? AND pass = ?");
        prepStat.setString( parameterIndex: 1, user);
        prepStat.setString( parameterIndex: 2, password);

        ResultSet myRs = prepStat.executeQuery();
        if(myRs.next()){
            return true;
        }
        else{
            return false;
        }
    }
}

```

This method is to check whether the user and password entered is already registered or not. If there is a next in the ResultSet means that there is already a user under that same username and password, and it will be checked before the register method is called and will be asked to reregister again.

```
private String random_UUID(){
    UUID uuid = UUID.randomUUID();
    String random = uuid.toString();
    while(true){
        if(check_userID(random)){
            random = uuid.toString();
            continue;
        }
    }
}
```

This method will be the one to generate the random unique user ID. However, before it is registered as it is, it will first check whether the generated ID is already used or not using the check_userid method, if there is already a user with the same as the randomized ID, it will loop until the program get a different ID that is not registered yet. (it might not happen, but just in case).

- Book Class

This class is to write and edit the data necessarily in the storage table.

```
public void add_Book(int book_id){
    try {
        PreparedStatement prepStat = connect.getPrepstat( query: "INSERT INTO storage " +
            "VALUES (?,?)" );
        prepStat.setString( parameterIndex: 1, login.get_userID());
        prepStat.setInt( parameterIndex: 2, book_id);
    }
```

This method is to add the book data based on the user ID to the table along with the logged user who inputted the data. It is so that when the program loads the data when it will only display the book under their ID and will not display other user(s) data.

```
public void remove_book(int book_id){
    try {
        PreparedStatement prepStat = connect.getPrepstat( query: "DELETE FROM storage " +
            "WHERE book_id_store = ?" );
        prepStat.setInt( parameterIndex: 1, book_id);
        int i = prepStat.executeUpdate();
        //if the book from the storage table is deleted, it will delete the data in the book table as well
        if(i > 0){
            PreparedStatement pdstmt = connect.getPrepstat( query: "DELETE FROM books WHERE book_id = ?" );
            pdstmt.setInt( parameterIndex: 1, book_id);
            pdstmt.executeUpdate();
            System.out.println("Data have been removed from storage");
        }
    }
```

This method is to remove the book in both the storage and book tables. First, it will delete the book of the user in the storage table, after it is successfully deleted from the storage table, the program will delete the book information under the book table as well.

```
public boolean check_book(int book_id){
    try {
        PreparedStatement prepStat = connect.getPrepstat( query: "SELECT * FROM storage " +
            "WHERE user_id = ? AND book_id_store = ?" );
        prepStat.setString( parameterIndex: 1, login.get_userID());
        prepStat.setInt( parameterIndex: 2, book_id);

        ResultSet myRs = prepStat.executeQuery();
        if(myRs.next()){ //to check whether the book id is in the table or not
            return true;
        }
    }
```

This will be used in the retrieve_data method in the Access class to first check whether the book ID inputted is listed under the user ID or not, if yes then proceed, else, it will ask the user to confirm the inputted book ID to be retrieved.

- Access Class

This class is mostly used to handle the access of books table in the database which have the usual function such as viewing, insert and delete.

```

PreparedStatement prepStat = connect.getPrepstat( query: "SELECT book_id, Title, Author FROM books " +
    "INNER JOIN storage ON books.book_id = storage.book_id_store " +
    "WHERE user_id = ?");
prepStat.setString( parameterIndex: 1, login.get_userID());
ResultSet myRs = prepStat.executeQuery();
System.out.println("Data Showed!");
return myRs;

```

The lines of code are used for showdata and searchdata methods since the use of this code is similar, which is to display the ResultSet to the table in the main user interface. The difference is the searchdata need the title parameter since usually people search the data by title. The parameter user_id is needed so it will only show data which belong to the logged user.

This add_data method is for adding the data into the books table. The parameter is from the information inputted by the user.

```

File file = new File(file_loc);
InputStream inputStream = new FileInputStream(file);

```

This line of code is needed since the program need to store blob file selected by the user. The file and input stream are used to read the bytes from the selected file so the program can store it as a blob file in the table.

```
int random = random_number();
```

The random int is to store the newly randomized book ID. After the data is successfully added to the books table, it will call the add_Book method from the Book class and add the data in to the storage table as well.

This random_number method is like the random_UUID method from Register class, however, this only randomized the number from 1 to the maximum value of an integer which is 2147483647 instead of using a UUID, to make sure that the user does not have to enter a long number into the text field.

And like the check_userID method from the Register class, the checkid will check whether the ID had been taken or not, if not it will proceed, else, it will rerandomized again. The difference with the check_userID is this method does not need to check whether the ID is used under the user or not, it checks all the book ID regardless of the user(s).

```

public String browse_location(){
    JFileChooser chooser = new JFileChooser();
    chooser.showOpenDialog( parent: null);
    File f = chooser.getSelectedFile();
    String file = f.getAbsolutePath();
}

```

This function will be used in the main user interface later for the file chooser, so the user can easily choose the file location instead of having to write the file path by themselves. The JFileChooser is implemented from java swing.

The retrieve_data method is similar to any delete function. First, it will check whether the book ID is owned by the user or not, if yes, then it will proceed to the delete code. However, before it proceeds to deleting, it will select the stored blob file and the OutputStream is used to write the streams of bytes into the data of the selected file path.

```

File file = new File(file_loc);
FileOutputStream output = new FileOutputStream(file);

if (result.next()) {
    InputStream input = result.getBinaryStream( columnLabel: "Loc");
    byte[] buffer = new byte[1024];
    while (input.read(buffer) > 0) {
        output.write(buffer);
    }
}

```

The InputStream is to read the bytes(blob) file in stream, and the byte is to set the minimum file size, and then write it to the designated file path.

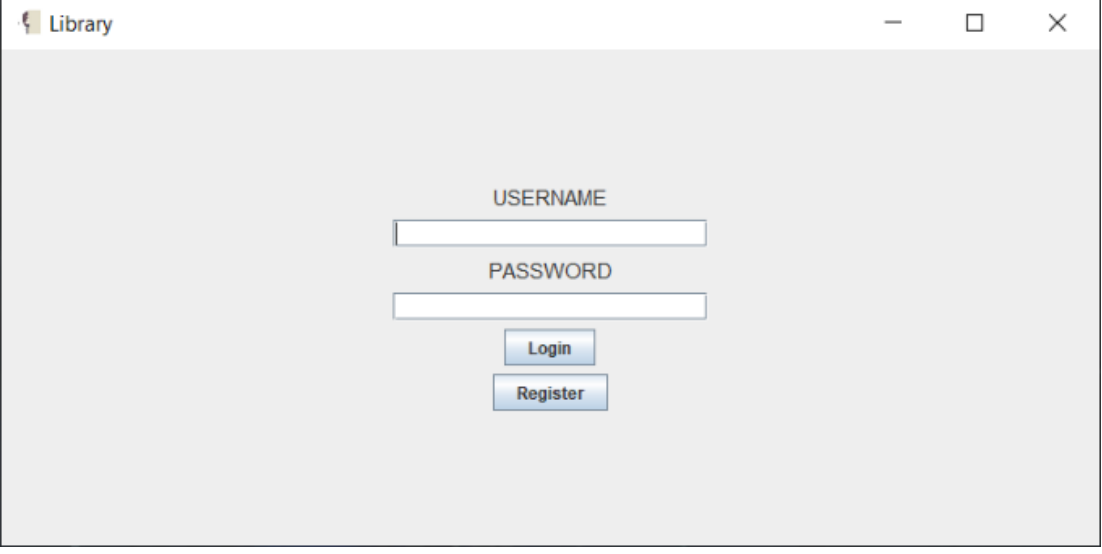
- Window Class

This class is to handle the frame/window of the GUI. The purpose is so that it will have constant location, size, icon of the window. So instead of using multiple frame for different use like the login, register and the main GUI, in this code, it uses multiple panel.

```
private void setIconImage(String s) {  
    setIconImage(Toolkit.getDefaultToolkit().getImage(getClass().getResource(s)));  
}
```

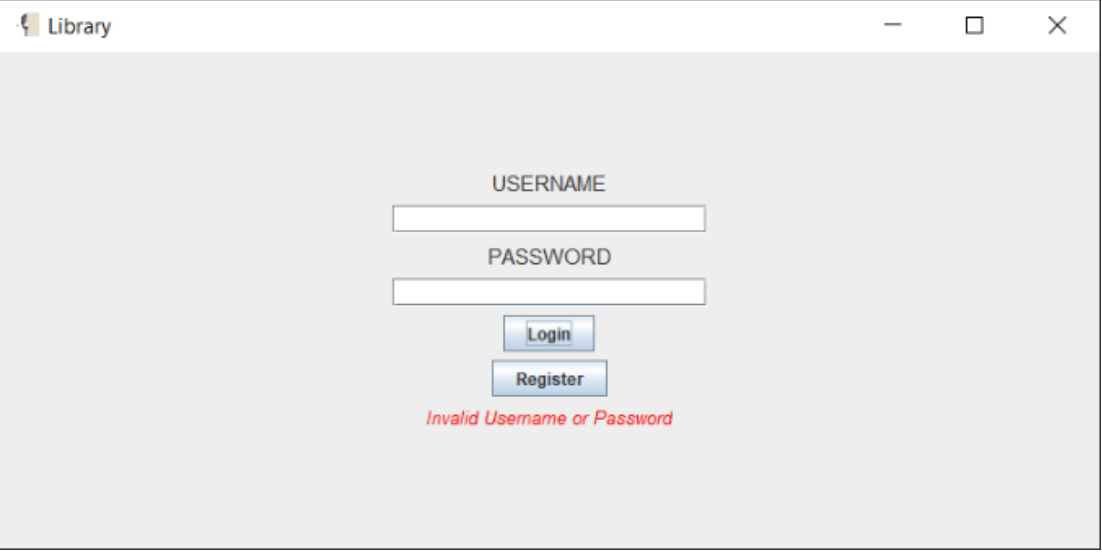
This line of the code is to change the icon picture of the application.

Evidence



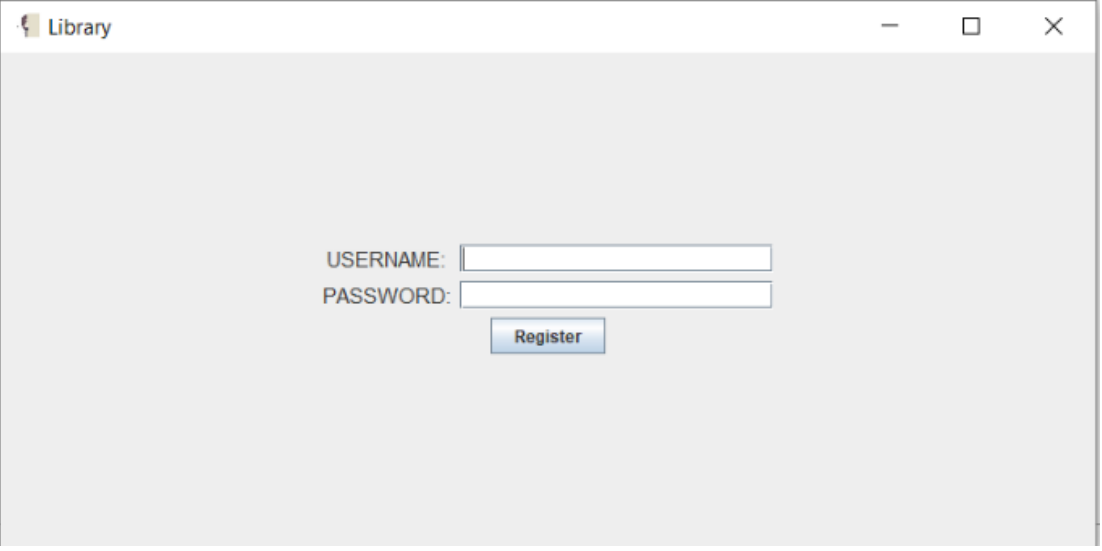
A screenshot of a web application window titled "Library". The window has a light gray background. In the center, there are two text input fields. The first field is labeled "USERNAME" and the second is labeled "PASSWORD". Below the password field, there are two buttons: "Login" and "Register".

Fig. 1.1 Login panel



A screenshot of the same "Library" web application window. The "USERNAME" and "PASSWORD" input fields are present. Below the "Register" button, there is a red error message that reads "Invalid Username or Password".

Fig. 1.2 Login panel, when user typed the wrong username or password



A screenshot of the "Library" web application window. The "USERNAME:" and "PASSWORD:" labels are now followed by input fields. Below the password field, there is a single "Register" button.

Fig. 2.1 Register panel

Library

USERNAME:

PASSWORD:

The user with this ID is already listed!

Fig. 2.2 Register panel, when the ID is already registered

Library

ADD BOOK

Title:

Author:

File Location:

RETRIEVE BOOK

Book ID:

File Location:

VIEW BOOKS

Search:

ID	Title	Author
314643486	book	maio
888821356	hello	who is this
1907823646	hello	something

Fig. 3 Main Panel

References

- Making the database and connecting the program with JDBC: <https://youtu.be/2i4t-SL1VsU>
- Connecting the GUI with the code for accessing the database: <https://www.youtube.com/watch?v=q8Z3CmruGzI>
- Inserting data into Jtable: <https://www.youtube.com/watch?v=7GZppdcccFfs>
- Searching and updating data in Jtable: <https://www.youtube.com/watch?v=ncOxOPRBUgM>
- Manual coding for the GUI: <https://www.youtube.com/watch?v=IMjfmWVxd2E&t=472s>
- Writing and retrieving blob file to and from database: <http://www.mysqltutorial.org/mysql-jdbc-blob>
- Using JFileChooser: <https://www.youtube.com/watch?v=nVWXJ3qqi0o>

Source code and program file can be downloaded from <https://github.com/vynsss/Final-Project-Java>