VIETNAM NATIONAL UNIVERSITY – HCM

INTERNATIONAL UNIVERSITY

SEMESTER 2 (2023-2024)
PRINCIPLE OF DATABASE MANAGEMENT

LIBRARY MANAGEMENT SYSTEM

Members:
Nguyễn Đức Hiền - ITITIU22053
Hoàng Ngọc Quỳnh Anh - ITCSIU22256
Âu Nguyễn Nhật Thư - ITITIU22152
Nguyễn Thị Hồng Anh - ITCSIU22010
Phạm Vũ Hoàng Bảo - ITCSIU22250

TABLE OF CONTENT

I INTRODUCTION

1. Abstract

The Library Management System is a complex software system that enables library staff to efficiently handle student data, categorize and track books, monitor return dates, manage book issuance, and maintain accurate records of defaulters in the university library.

The project's user-centric approach is exemplified by its emphasis on user-friendly interfaces, role-based access control, and effective database administration. This ensures that all library personnel can proficiently use this application, as well as that users only have access to see the data and information if they are library personnel. This application also claims that it can handle a large amount of data, making it very easy for library personnel to retrieve data and find information.

Database design is very important to our library management project, and we closely adhere to the B.C. normalization form. Designing entails careful consideration, as well as establishing the entity, qualities, relationship, and limitations. This initiative aims to protect accuracy and prevent errors in addition to helping libraries become more efficient at discovering information. As the system changes over time, its logical architecture plays a critical role in ensuring its effectiveness, dependability, and flexibility.

This project uses simulated data during testing sessions to ensure that the system will function properly and is error-free. Should the system function well, actual data may be utilized for this system.

The user interface is very easy to use; after a short introduction, the user will be able to operate the system independently without help from other users who have used it before. The login feature enables users to authenticate themselves and gain access to system data.

In general, The development of the library management system facilitates the administration of student data, book issuance, return dates, classification and tracking, and correct documentation of defaulters in the university library by the library staff. This system makes it easier for library employees to enter, retrieve, and manage all the data from a vast number of books and students in the library by offering a user-friendly interface, role-based access control, and efficient database management.

2. System overview

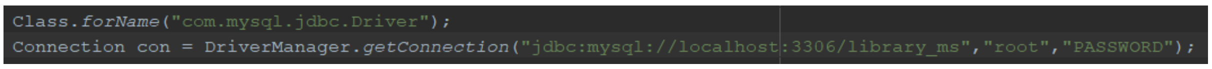
The university's library staff can regulate pupils owing to technology. They can now track which students are borrowing what books, when they should be returning them, and which students will be placed on a defaulter list if they do not return their books on time because they have access to all student data from various faculty members. Furthermore, this strategy helps library personnel keep track of which books are checked out, how many volumes are available, and when each book is due back. The staff library can also manage any book issues and students on the defaulter list.

3. Goal

- B.C. normalization was certified by the design database.
- Link and connect the database between the application's front end and the database's back end.
- Provide the features that a library needs, such login, book management, student management, issue book management, and default list management.

4. The technique and tools use

- In this project, Netbeans was used to create the user interface, and we also used it to create the Java structure and make database queries using mySQL server.
- The database was connect by using JDBC



```
Class.forName("com.mysql.jdbc.Driver");
Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/library_ms","root","PASSWORD");
```

II PROJECT PLANNING

1. Project timeline and miles

Stage	*Action*	*Week*
PLANNING	Giving ideas on tool and project structure	**1**

	Research for an supporting document	
	Determine goal, learning objective	
	Building timeline	
	Consider the tool use	
	Setup a meeting, to determine the tool use, timeline and process	
CONCEPTUAL DESIGN	Identify target user and function for the system	**2**
	Decide for which type of information store in database	
	Design visual for UI	
	Prepare for ERD diagram	
IMPLEMENTATION	Determine the constraint, relationship, cardinality between database	**3-5**
	Draw ERD and relational schema	
	Connect the database to the code.	

	Code to accomplish all the necessary functions	
DEMO	Run and test the whole project	**6-7**
	Detection and bug fixes	
	Modify the database, add or delete function if needed	
	Start writing the report	
PREPARE FOR PRESENTATION	Finish the report	**8**
	Making powerpoint	

2. Roles and responsibility of team members

Name	*Role*	*Contribution*
Hoàng Ngọc Quỳnh Anh	Project leader	20%
Nguyễn Thị Hồng Anh	member	20%
Âu Nguyễn Nhật Thư	member	20%
Nguyễn Đức Hiền	member	20%
Phạm Vũ Hoàng Bảo	member	20%

III PROJECT ANALYSIS

1. Requirements Analysis

- Authorization:
  - Secure user registration and login processes with appropriate authorization levels.
  - Implement functionalities for users to access and manage the library database.

- Information Management:
  - Enable users to modify and manage details related to books and student borrowers.
  - Show user comprehensive information on books, students, borrowed books, issue dates, due dates, and book status.

- User Interface Design:
  - Develop user-friendly interfaces that facilitate effortless interaction with the system.
  - Design specific data entry fields to streamline information tracking.

- Database Design:
  - Adhere to Boyce-Codd Normal Form (BCNF) in database design.
  - Define and design data entities, relationships, attributes, and constraints to ensure data integrity and eliminate redundancies.

- Data Management:
  - Optimize data retrieval and storage mechanisms for system efficiency, reliability, and scalability.

- Personalized Access:
  - Design a personalized signin and login system that grants individual user accounts for interacting with the library database.

2. Approach Analysis

2.1. *Reviewed Materials*

- MySQL

(URL: MySQL :: MySQL Documentation)

- Apache NetBeans

(URL: Java SE Learning Trail (apache.org))

2.2. *Research Analysis*

The team works by prioritizing task planning, making necessary adjustments, and allocating work to team members in order to guarantee progress. We hold meetings to assess the status of the project and talk about any necessary modifications. We are able to come to agreements during these meetings and promptly modify and enhance the project without compromising its organizational structure.

This approach allows us to keep our working process flexible while making sure that everyone in the team is aware of all developments and changes. Consensus-building through weekly meetings helps to avoid conflicts in the project structure and creates an environment conducive to effective revision implementation. This approach maximizes project performance and quality by preserving team collaboration's consistency and flexibility.

3. System Analysis

3.1. *Database Design*

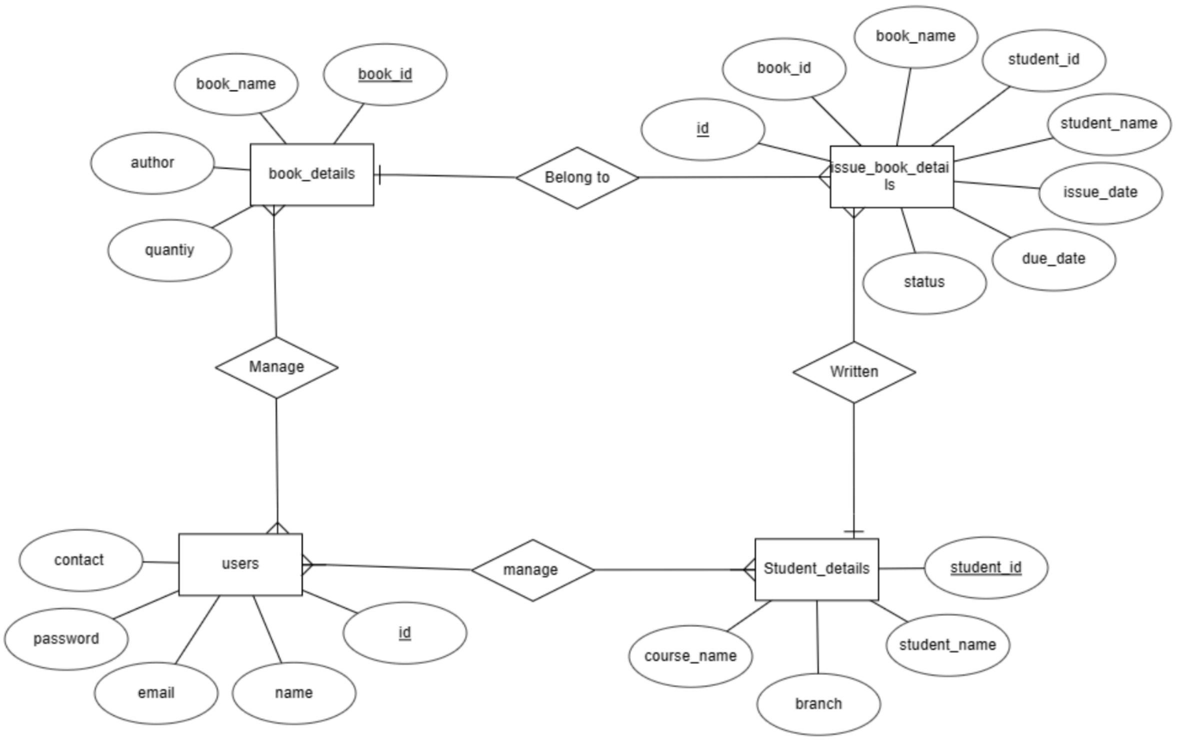


Figure 1: Library Management System ERD Diagram

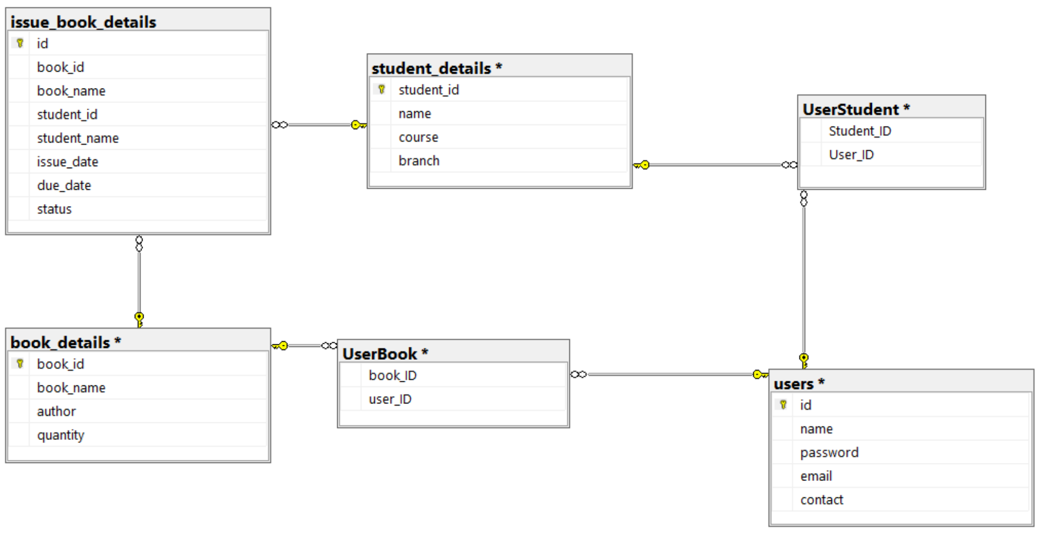


Figure 2: Schema design

Based on the two figures above, our database achieved BC normal form since it meets the demand for normal form requirements. In the first normal form (1NF), there is one value in each table cell, each column has a unique name, rows and columns are not repeated within a table. In the second normal form (2NF) and third normal form (3NF), all non-key attributes must depend on the primary key and are not related to other non-key attributes in the same table. Boyce-Codd Normal Form (BCNF) allows every non-key attribute to rely exclusively on the candidate key. Four tables in our database satisfy all the requirements of normal forms.

3.2. *Database and Table Creation*

a. *Database Creation*

```
CREATE DATABASE library_ms;
USE library_ms;
```

b. *Tables creation*

Table: Entities and attributes

Table	Attributes
users	id INT PRIMARY KEY

	name VARCHAR(50) password VARCHAR(50) email VARCHAR(100) contact VARCHAR(20)
book_details	book_id INT PRIMARY KEY book_name VARCHAR(250) author VARCHAR(100) quantity INT
student_details	student_id INT PRIMARY KEY student_name VARCHAR(30) course_name VARCHAR(50) branch VARCHAR(50)
issue_book_details	id INT PRIMARY KEY book_id INT book_name VARCHAR(150) student_id INT student_name VARCHAR(50) issue_date DATE due_date DATE status VARCHAR(20)

Foreign Key Relationships:

To create relationships between tables and guarantee data consistency and integrity, foreign key constraints are added. The book_id in the issue_book_details table references the book_id in the book_details table. The student_id in the issue_book_details table references the student_id in the student_details table.

Create tables:

- **users table**

```
CREATE TABLE users (
        id  INT PRIMARY KEY not null AUTO_INCREMENT,
        name VARCHAR(50),
        password VARCHAR(50),
        email VARCHAR(100),
        contact VARCHAR(20)
);
```

- **book_details table**

```
CREATE TABLE book_details (
        book_id INT PRIMARY KEY not null AUTO_INCREMENT NOT
NULL,
        book_name VARCHAR(250),
        author VARCHAR(100),
        quantity INT
);
```

- **student_details table**

```
CREATE TABLE student_details (
        student_id INT PRIMARY KEY NOT null AUTO_INCREMENT,
        name VARCHAR(30),
        course_name VARCHAR(50),
        branch VARCHAR(50)
);
```

- **issue_book_details**

```
CREATE TABLE issue_book_details (
        id INT PRIMARY KEY not null AUTO_INCREMENT,
        book_id INT,
        book_name VARCHAR(150),
        student_id INT,
        name VARCHAR(50),
        issue_date DATE,
        due_date DATE,
        status VARCHAR(20)
);
```

- **Create Foreign Key:**

Adding foreign key constraints to the [issue_book_details] table

```
ALTER TABLE issue_book_details
ADD CONSTRAINT FK_issue_book_details_book_details
FOREIGN KEY (book_id) REFERENCES book_details (book_id) ;


ALTER TABLE issue_book_details
ADD CONSTRAINT FK_issue_book_details_student_details
FOREIGN KEY (student_id) REFERENCES student_details (student_id) ;
```

3.3. *Database Data Insertion*

- **users table**

```
INSERT INTO users ( id, name, password, email, contact)
VALUES
        ( 1, 'sofia', 'mpwd', sofia@gmail.com, 123456789),
        (2,  'josh123', '098' , josh123@gmail.com, 012345678);
```

- **book_details table**

```
INSERT INTO book_details(book_id, book_name, author, quantity)
VALUES
        (1, 'Java: How to program', 'Deitel', 5),
        (2, 'Python Programming', 'Clive Campbell', 2),
        (3, 'Problem solving with C++', 'Walter Savitch', 5),
        (4, 'Data structures & algorithms in java', 'Robert Lafore', 3) ;
```

- **student_details table**

```
INSERT INTO student_details (student_id,name,course,branch)
VALUES
        (120, 'Sofia' , 'BSC',  'IT'),
        (173, 'Josh', 'PHD', 'CS');
```

- **issue_book_details table**

```
INSERT INTO issue_book_details (id, book_id, book_name ,student_id, name,
issue_date, due_date, status)
VALUES
```

(1, 1, 'Java: How to program', 120, 'Sofia', '2023-05-06', '2023-05-11', 'pending')
(2, 3, 'Problem solving with C++', 173, 'Josh', '2024-02-11', '2024-02-18', 'pending');

3.3. *Database Queries:*

a. Queries analysis

Table: SQL keywords

Keyword	**Function**
SELECT	Retrieves data from one or more tables in a database.
DELETE	Removes one or more rows from a table.
INSERT INTO VALUES	Adds one or more records to a table.
AND	Combines multiple conditions in a WHERE clause, ensuring that all conditions must be true.
FROM	Specifies the source table or tables for a query.
WHERE	Filters the result set based on a specified condition.
AS	Renames a column or table using an alias.
UPDATE	Modifies data in a table by updating existing records.
BETWEEN	Used to select values within a given range. The values can be numbers, text, or dates

b. All Queries used

MODEL	ACTIONS	SQL QUERIES
users	Get All from users	SELECT * FROM users;
	Get users by ID	SELECT * FROM users WHERE id = @ID;
	Create users	INSERT INTO users (id, name, password, email, contact) VALUES(@id, @name, @password, @email, @contact)

	Update users by id	UPDATE users
		SET id =@id ,
		name=@name,
		password=@password,
		email=@email,
		contact=@contact,
		
		WHERE id =@id

	Select name and password from user for login	SELECT * FROM users WHERE name = ? AND password = ?
	Delete users by id	DELETE FROM users WHERE id = @id;
book_details	Get All from book_details	SELECT * FROM book_details;
	Get book_details by ID	SELECT * FROM book_details WHERE book_id=@id ;

	Create book_details	INSERT INTO book_details (book_id, book_name, author, quantity,) VALUES(@book_id, @ book_name, @author, @quantity);

	Update book_details by id	UPDATE book_details SET book_id=@book_id, book_name=@ book_name, author= @author, quantity=@quantity,
		
		 WHEREbook_id=@book _id;

	Update book quantity by book_id	UPDATE book_details set quantity = quantity - 1 WHERE book_id = ?
	Delete book_details by id	DELETE FROM book_details WHERE book_id=@book_id;

student_details	Get All from student_details	SELECT * FROM student_details;
	Get student_details by ID	SELECT * FROM student_details WHERE student_id=@id;
	Create student_details	INSERT INTO student_details (student_id,student_name, course_name, branch) VALUES(@student_id, @student_name, @course_name, @branch);
	Update student_details by id	UPDATE book_details SET(student_id,student_name,course_name, branch) WHERE student_id=@student_id;

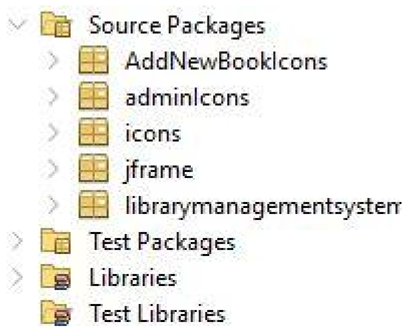
	Delete users by id	DELETE FROM student_details WHERE student_id=@student_id;
issue_book_details	Get All from issue_book_details	SELECT * FROM issue_book_details;
	Get issue_book_details by ID	SELECT * FROM issue_book_details WHERE id=@id ;
	Get issue_book_details by due_date and status	SELECT * FROM issue_book_details WHERE due_date < ? AND status = ?

	Create issue_book_details	INSERT INTO issue_book_details (id,book_id, book_name,student_id, student_id, issue_date, due_date, status) VALUES(id=@id,book_id =@book_id, book_name=@book_name,student_id=@student_id ,student_name=@student_name, issue_date=@issue_date, due_date=@due_date, status=@status);
	Select from issue_book_details by due date and status	SELECT * FROM issue_book_details WHERE due_date < '"+todaysDate+"' and status = '"+"pending"+"'
	Select issue_book_details by status	SELECT * FROM issue_book_details WHERE status = '"+"pending"+"'
	Select issue_book_details by issue_date	SELECT * FROM issue_book_details WHERE issue_date BETWEEN ? to ?

	Update issue_book_details by id	UPDATE issue_book_details SET(id,book_id, book_name,student_id, student_id, issue_date, due_date, status) WHERE id=@id;
	Delete issue_book_details by id	DELETE FROM issue_book_details WHERE id=@id;

4. Application Java Structure

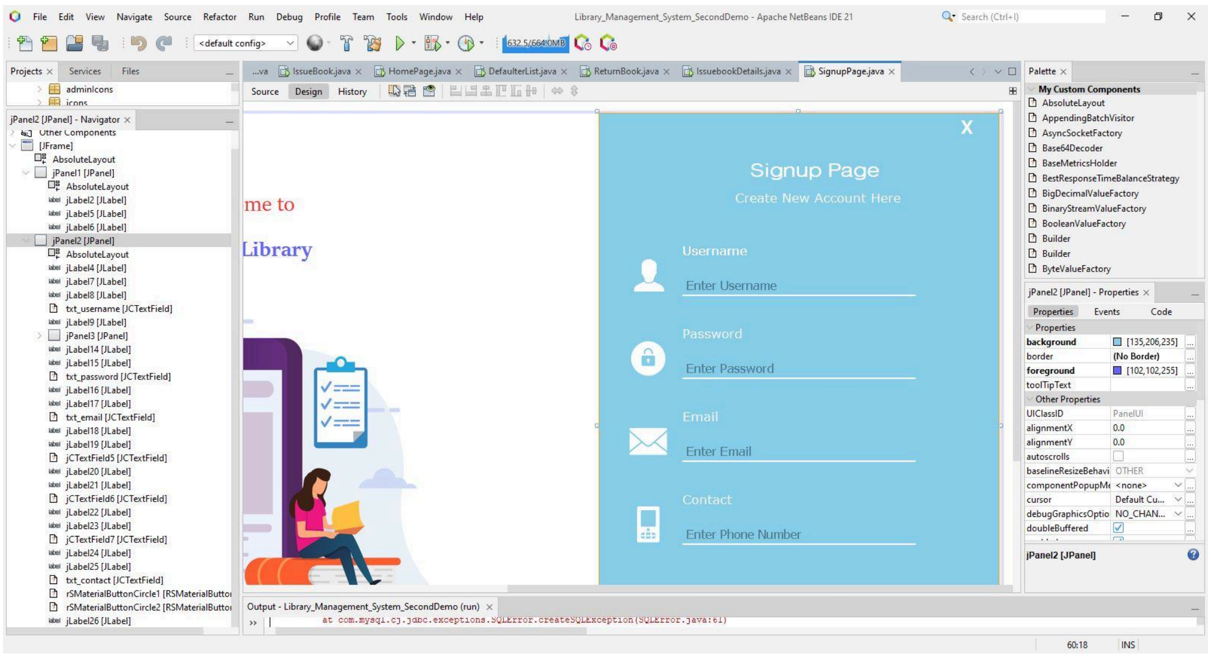
4.1 *Project Structure*



- *Source Packages:* Include all of the icons and Frames for the project.
    - *AddNewBookIcons, adminIcons, icons:* Include all of the icons.
    - *jframe:* Include all of the Frames and the Database Connection.
- *Libraries: Include all of the additional component (calendar, chart, .etc) for the project*
- *Test Packages, Test Libraries:* NetBeans' premade folder

4.1.1 Front-end

- SignupPage.java: Frame for users to Register.

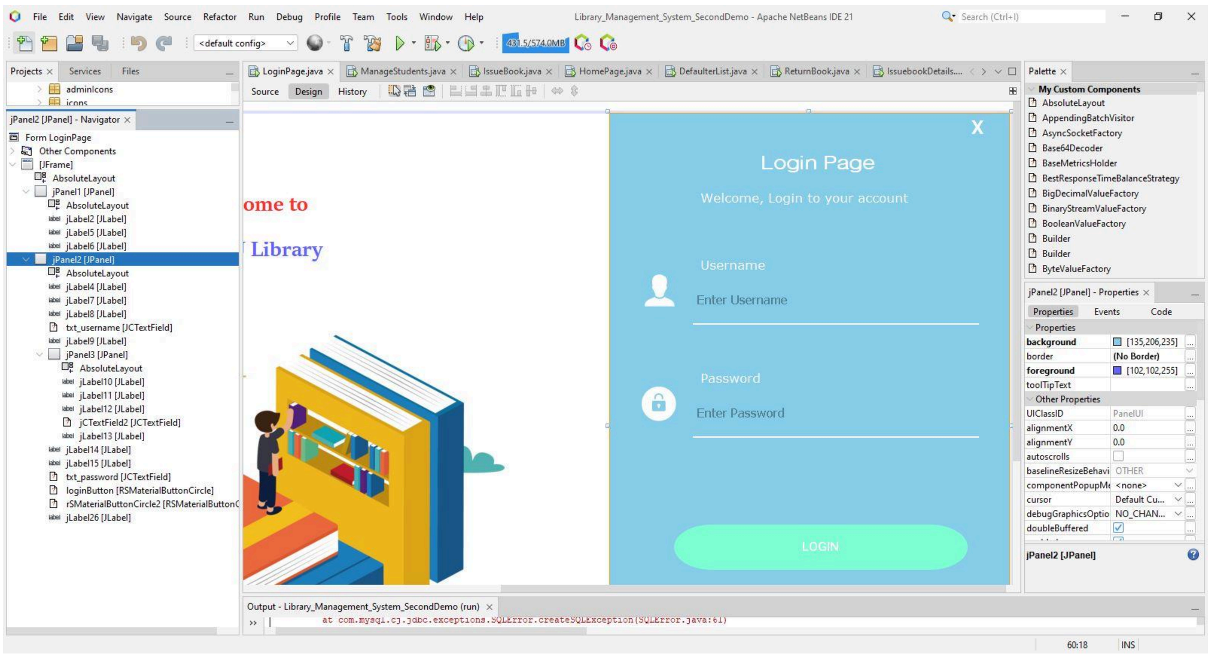


jPanel2 [JPanel] - Properties

background	[135,206,235]
border	(No Border)
foreground	[102,102,255]
UIClassID	PanelUI
alignmentX	0.0
alignmentY	0.0
baselineResizeBehavi	OTHER
componentPopupMe	<none>
cursor	Default Cu...
debugGraphicsOptio	NO_CHAN...

```
at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:61)
```

- LoginPage.java: Frame for users to Login.

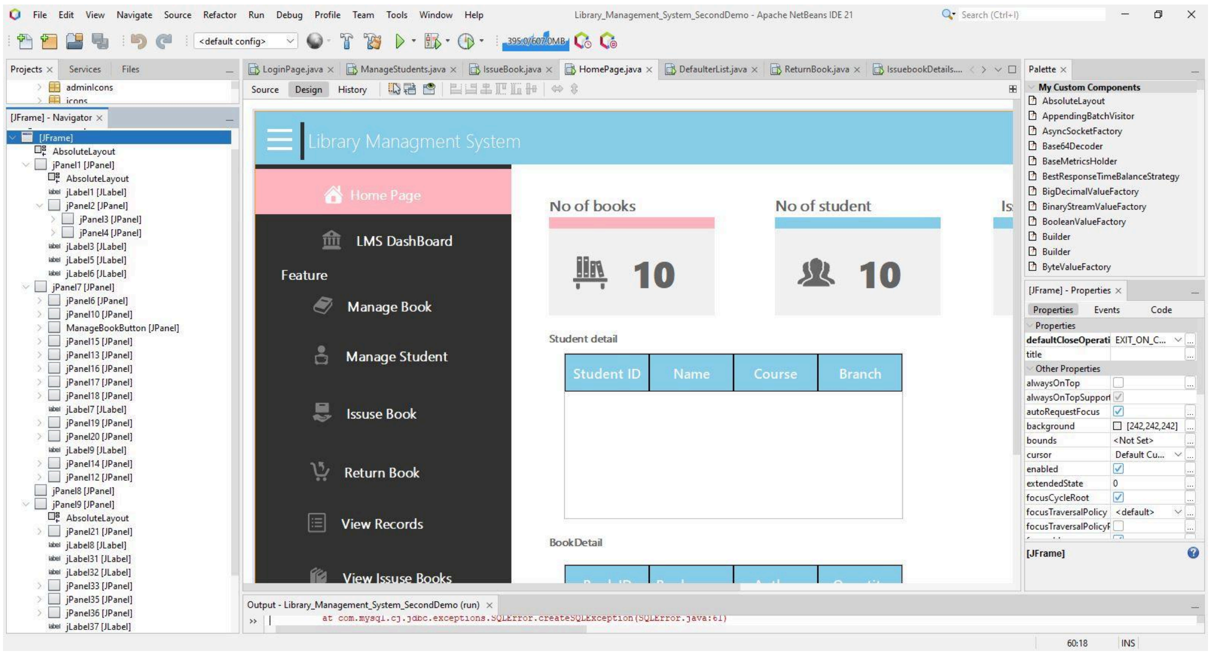


jPanel2 [JPanel] - Properties

background	[135,206,235]
border	(No Border)
foreground	[102,102,255]
UIClassID	PanelUI
alignmentX	0.0
alignmentY	0.0
baselineResizeBehavi	OTHER
componentPopupMe	<none>
cursor	Default Cu...
debugGraphicsOptio	NO_CHAN...

```
at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:61)
```

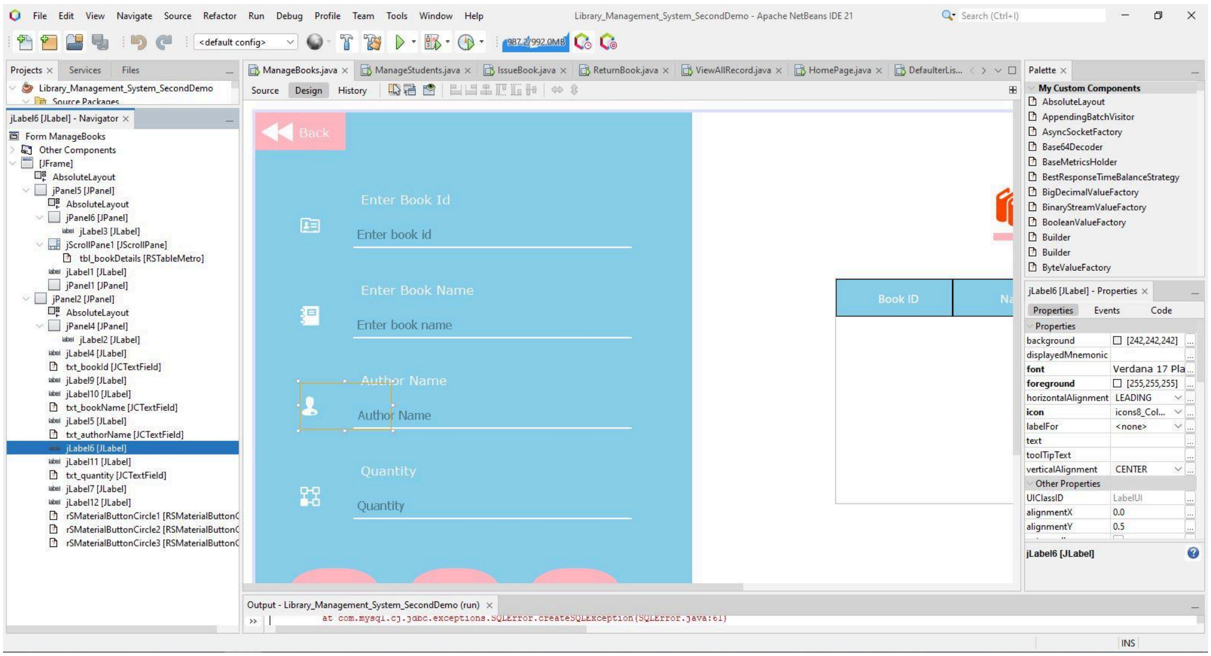
- HomePage.java: The main page to access other Frames



```
Output - Library_Management_System_SecondDemo (run)
        at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:61)
```

[JFrame] - Properties	
defaultCloseOperati	EXIT_ON_C...
title	
alwaysOnTop	☐
alwaysOnTopSupport	✓
autoRequestFocus	✓
background	[242,242,242]
bounds	<Not Set>
cursor	Default Cu...
enabled	✓
extendedState	0
focusCycleRoot	✓
focusTraversalPolicy	<default>
focusTraversalPolicyF	☐

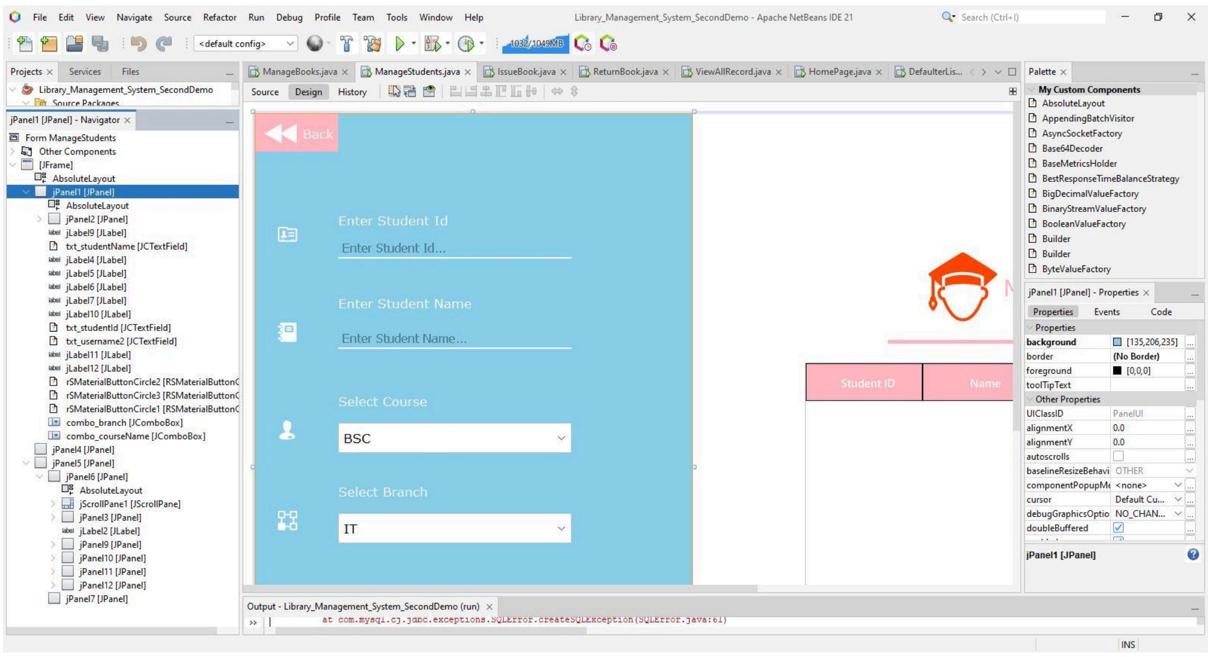
- ManageBook.java: Frame for users to update, add, and delete some books in the list



```
Output - Library_Management_System_SecondDemo (run)
        at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:61)
```

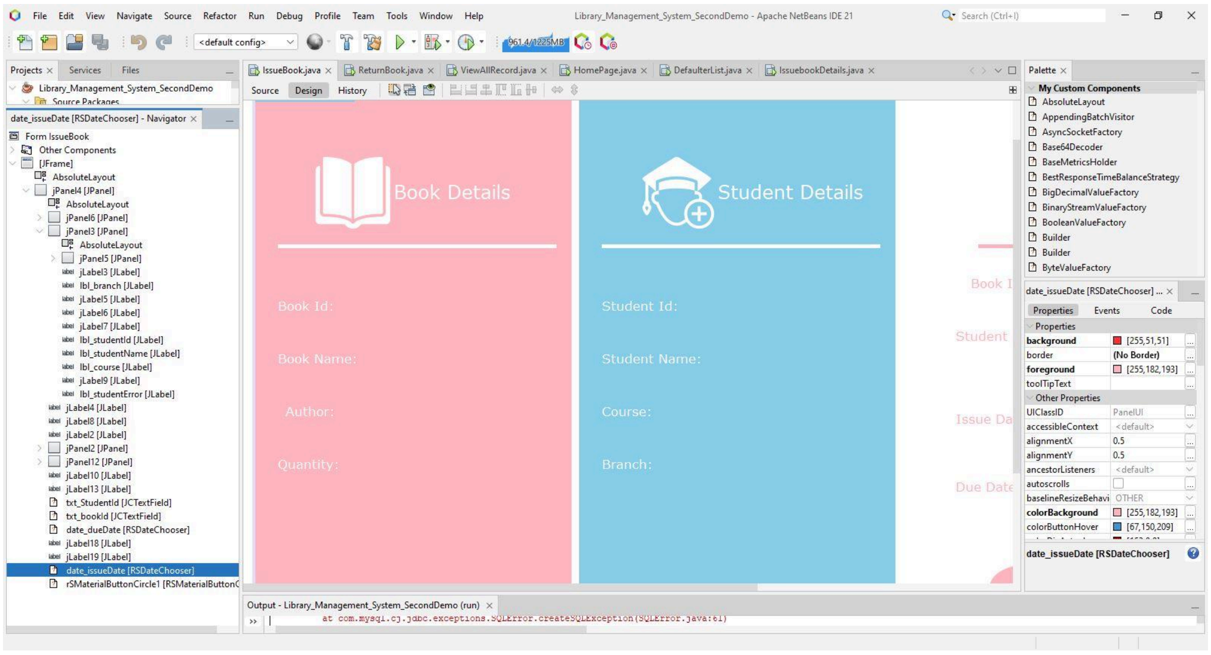
jLabel6 [JLabel] - Properties	
background	[242,242,242]
displayedMnemonic	
font	Verdana 17 Pla...
foreground	[255,255,255]
horizontalAlignment	LEADING
icon	icons8_Col...
labelFor	<none>
text	
toolTipText	
verticalAlignment	CENTER
UIClassID	LabelUI
alignmentX	0.0
alignmentY	0.5

- ManageStudents.java: Frame for users to manage all of the students who have register



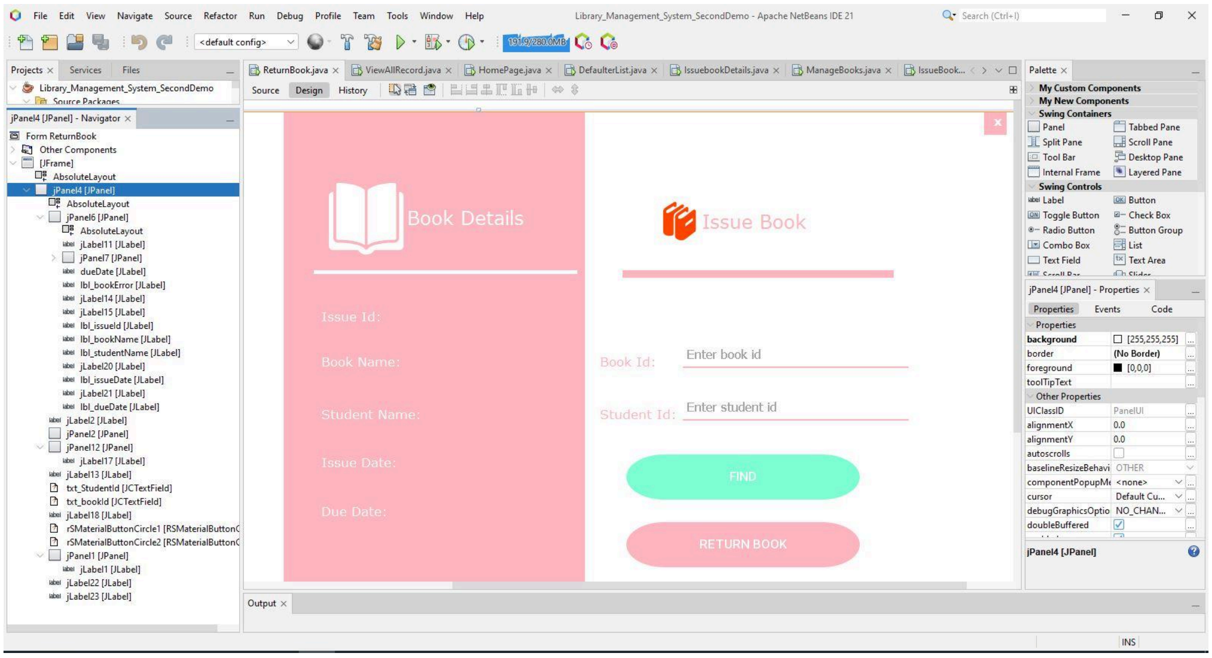
```
at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:61)
```

- IssueBook.java: Frame for users to issue books to student

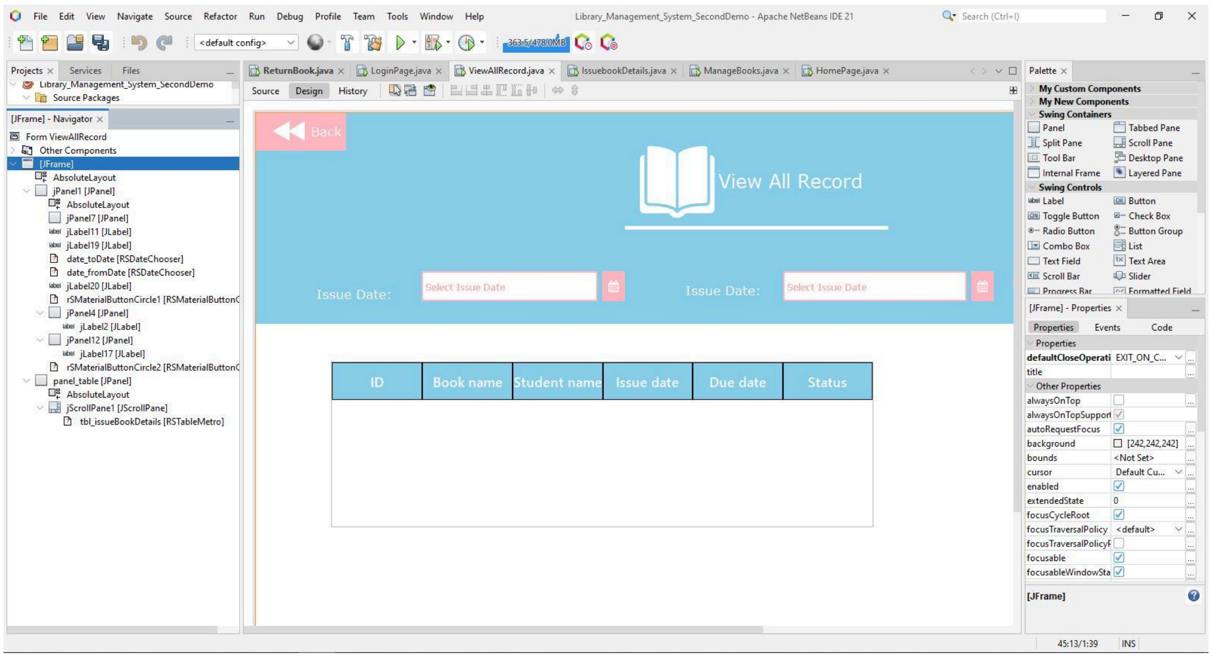


```
at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:61)
```

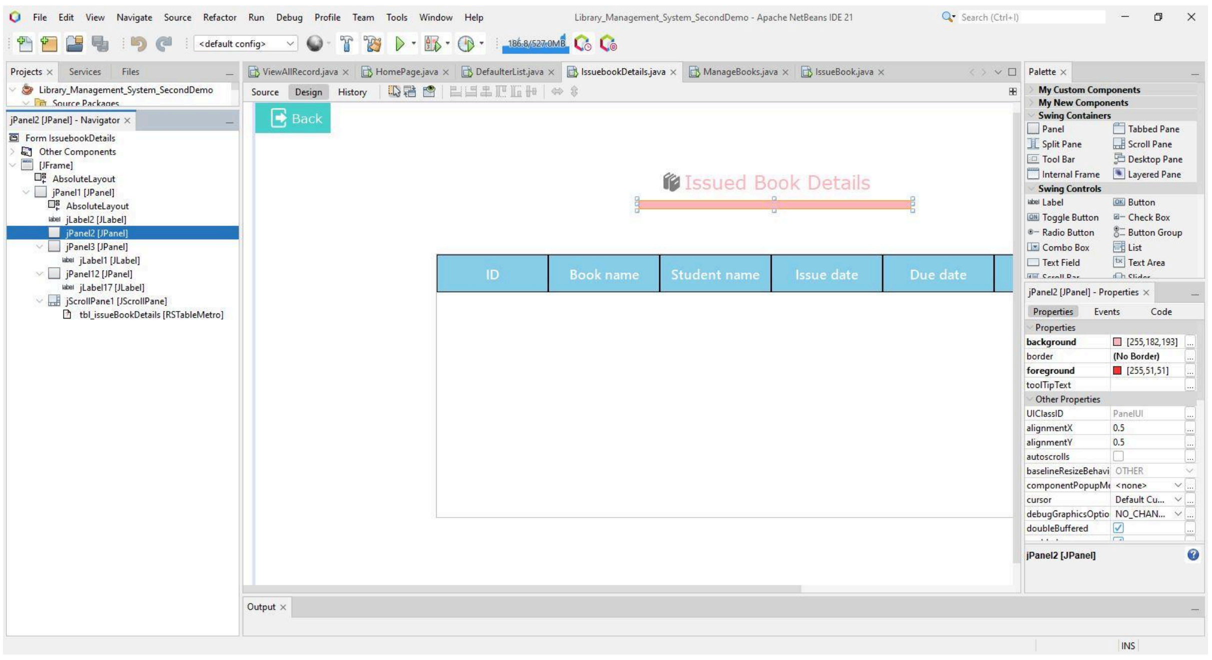
- ReturnBook.java: Frame to return the books which the student has returned.



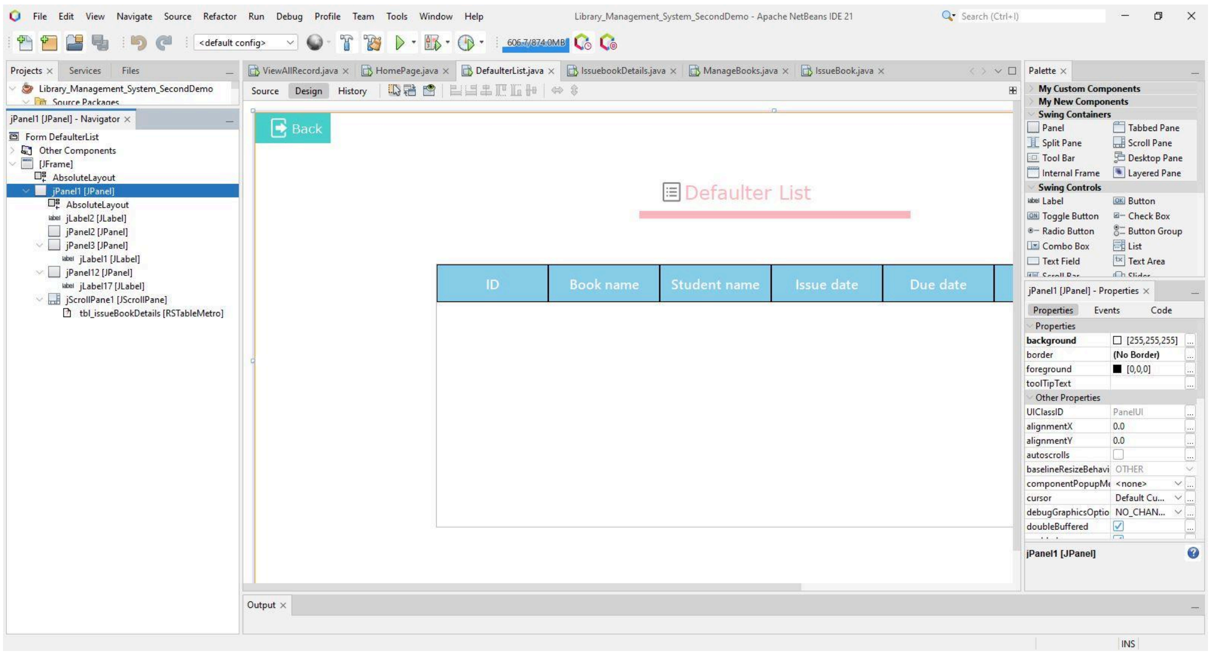
- ViewAllRecord.java: Show all of the books has issued, returned and the student who issued or returned the books.



- IssuebookDetails.java: Show all of the issued books



- DefaulterList.java: Show student exceeds the Expected Return Date for book and do not return the book



4.1.2 Back-end:

- SignupPage.java:

Method name	Code

insertSignupDetails():
The system will retrieve the username, password, email, and contact, then it will connect to the database to insert into users table.
	public void insertSignupDetails() {
	String name = txt_username.getText();
	String pwd = txt_password.getText();
	String email = txt_email.getText();
	String contact = txt_contact.getText();
	
	try {
	Connection con =
	DBConnection.getConnection();
	String sql = "insert into users(name, password,
	email, contact) values(?, ?, ?, ?)";
	PreparedStatement pst =
	con.prepareStatement(sql);
	
	pst.setString(1, name);
	pst.setString(2, pwd);
	pst.setString(3, email);
	pst.setString(4, contact);
	
	int updatedRowCount = pst.executeUpdate();
	if (updatedRowCount > 0) {
	JOptionPane.showMessageDialog(this,
	"Recorded Inserted Successfully");
	LoginPage page = new LoginPage();
	page.setVisible(true);
	dispose();
	
	}
	else {
	JOptionPane.showMessageDialog(this,
	"Recorded Inserted Failure");
	}
	
	} catch (Exception e) {
	e.printStackTrace();
	}
	}

validateSignup():
Check if the user has entered the username, password, email, and phone number correctly. If not, the system will show an error message.

```
public boolean validateSignup() {
        String name = txt_username.getText();
        String pwd = txt_password.getText();
        String email = txt_email.getText();
        String contact = txt_contact.getText();

        if (name.equals("")) {
           JOptionPane.showMessageDialog(this, "Please
enter username!");
           return false;
        }

        if (pwd.equals("")) {
           JOptionPane.showMessageDialog(this, "Please
enter password!");
           return false;
        }

        if (email.equals("") ||
!email.matches("^.+@.+\\..+$")) {
           JOptionPane.showMessageDialog(this, "Please
enter valid email!");
           return false;
        }

        if (contact.equals("")) {
           JOptionPane.showMessageDialog(this, "Please
enter your phone number!");
           return false;
        }
        return true;
    }
```

checkDuplicateUsers():
Check if the username exists before. If yes, then when the user click Signup button, the system will show the error message

```
public boolean checkDuplicateUser() {
        String name = txt_username.getText();
        boolean isExits = false;

        try {
           Class.forName("com.mysql.jdbc.Driver");
           Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:33
06/library_ms","root","PASSWORD!");

           PreparedStatement pst =
```

	con.prepareStatement("SELECT * FROM users where name = ?");
	pst.setString(1, name);
	ResultSet rs = pst.executeQuery();
	if (rs.next()) {
	isExits = true;
	} else {
	isExits = false;
	}
	} catch (Exception e) {
	e.printStackTrace();
	}
	return isExits;
	}

- LoginPage.java:

Method name	Code

validateLogin():
Check if the user has entered the username and password. If not, the system will display the message that asks the users to enter the username or password.
	public boolean validateLogin(){
	String name = txt_username.getText();
	String pwd = txt_password.getText();
	
	if(name.equals("")){
	JOptionPane.showMessageDialog(this," Please enter username");
	return false;
	}
	if(pwd.equals("")){
	JOptionPane.showMessageDialog(this," Please enter password");
	return false;
	}
	return true;
	}

login():
First the system will retrieve the username and password, then it will try to connect to the database. If the system
	public void login (){
	String name = txt_username.getText();
	String pwd = txt_password.getText();
	try{
	Class.forName("com.mysql.jdbc.Driver");

succeeds, it will prepare the sql statement and set username and password in the query. After that, the system will execute the query and check if the username and password is matched. If yes, then the home page appears. If not, the system will indicate incorrect username or password.	
	```
	        Connection con =
	DriverManager.getConnection("jdbc:mysql://localhost:33
	06/library_ms","root","PASSWORD");
	        PreparedStatement pst =
	con.prepareStatement("SELECT * FROM users WHERE
	name = ? AND password = ?");

	        pst.setString(1, name);
	        pst.setString(2,pwd);

	        ResultSet rs = pst.executeQuery();

	        if(rs.next()){
	           JOptionPane.showMessageDialog(this,"Login
	Successful");
	           HomePage home = new HomePage();
	           home.setVisible(true);
	           this.dispose();
	        }
	        else{

	JOptionPane.showMessageDialog(this,"Incorrect
	username or password");
	        }
	     }catch(Exception e){

	     }
	   }
	```

- HomePage.java:

Method name	Code

setStudentDetailsToTable():
Fetch data from the "library_ms" database for students. They extract details like student ID, name, course and populate them into separate tables within the application.
	public void setStudentDetailsToTable(){
	    try{
	        Class.forName("com.mysql.cj.jdbc.Driver");
	        java.sql.Connection con =
	DriverManager.getConnection("jdbc:mysql://localhost:3306/library_ms","root","PASSWORD");
	        Statement st = con.createStatement();
	
	        ResultSet rs = st.executeQuery("select * from
	student_details");
	
	        while(rs.next()){
	           String StudentId = rs.getString("student_id");
	           String StudentName = rs.getString("name");
	           String course = rs.getString("course");
	           String branch = rs.getString("branch");
	
	           Object[] obj =
	{StudentId,StudentName,course,branch};
	           model = (DefaultTableModel)
	tbl_studentDetails.getModel();
	           model.addRow(obj);
	        }
	
	    }catch(Exception e){
	        e.printStackTrace();
	       }
	    }

setBookDetailsToTable():
Fetch data from the "library_ms" database for books. They extract details like book ID, title, author, and populate them into separate tables within the application.

```
 public void setBookDetailsToTable(){
       try{
          Class.forName("com.mysql.cj.jdbc.Driver");
          java.sql.Connection con =
DriverManager.getConnection("jdbc:mysql://localhost:33
06/library_ms","root","PASSWORD");
          Statement st = con.createStatement();
          ResultSet rs = st.executeQuery("select * from
book_details");

          while(rs.next()){
             String bookId = rs.getString("book_id");
             String bookName = rs.getString("book_name");
             String author = rs.getString("author");
             int quantity = rs.getInt("quantity");

             Object[] obj =
{bookId,bookName,author,quantity};
             model = (DefaultTableModel)
tbl_bookDetails.getModel();
             model.addRow(obj);
          }

       }catch(Exception e){
          e.printStackTrace();
   }}
```

setDataToCards():
Retrieves counts for various aspects of the library, likely for display on the interface. It counts the total number of books, students, issued books, and overdue books using queries to the database.

```
public void setDataToCards() {

       Statement st = null;
       ResultSet rs = null;

       long l = System.currentTimeMillis();
       java.sql.Date todaysDate = new java.sql.Date(l);

       try {
          java.sql.Connection con =
DBConnection.getConnection();
          st = con.createStatement();
          rs = st.executeQuery("Select * from
book_details");
          rs.last();

lbl_noOfBooks.setText(Integer.toString(rs.getRow()));
```

	rs = st.executeQuery("Select * from student_details");
	rs.last();
	
	lbl_noOfStudents.setText(Integer.toString(rs.getRow()));
	
	rs = st.executeQuery("Select * from issue_book_details");
	rs.last();
	
	lbl_issueBooks.setText(Integer.toString(rs.getRow()));
	
	rs = st.executeQuery("select * from issue_book_details where due_date < '"+todaysDate+"' and status = '"+"pending"+"'");
	rs.last();
	
	lbl_defaulterList.setText(Integer.toString(rs.getRow()));
	
	} catch (Exception e) {
	e.printStackTrace();
	}
	}

showPieChart():
Creates a pie chart to visualize book issuance data. It connects to the database and retrieves book names along with how many times each book has been issued. This data is then used to generate pie chart slices, and the chart is displayed within the application.
	public void showPieChart(){
	
	//create dataset
	DefaultPieDataset barDataset = new DefaultPieDataset( )
	try{
	Class.forName("com.mysql.cj.jdbc.Driver");
	java.sql.Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/library_ms","root","PASSWORD");
	String sql = "SELECT book_name, COUNT(*) AS issue_count FROM issue_book_details GROUP BY book_id, book_name";
	Statement st = con.createStatement();
	ResultSet rs = st.executeQuery(sql);
	while(rs.next()){
	barDataset.setValue(rs.getString("book_name") , new Double(rs.getDouble("issue_count")));
	}
	} catch (Exception e){

	e.printStackTrace();
	}
	
	//create chart
	JFreeChart piechart = ChartFactory.createPieChart("isssue book details",barDataset, true,true,false);//explain
	
	PiePlot piePlot =(PiePlot) piechart.getPlot();
	
	piePlot.setBackgroundPaint(Color.white);
	
	//create chartPanel to display chart(graph)
	ChartPanel barChartPanel = new ChartPanel(piechart);
	panelBarChart.removeAll();
	panelBarChart.add(barChartPanel, BorderLayout.CENTER);
	panelBarChart.validate();
	}

- ManageBook.java

Method name	**Code**

setBookDetailsToTable():
Fetches book information from the library database. It connects to the database, retrieves all data from the "book_details" table, and iterates through each row. For each book, it extracts details like ID, name, author, and quantity. These details are then stored in an array and added as a new row to a table likely displayed within the application's interface.
	public void setBookDetailsToTable(){
	try{
	Class.forName("com.mysql.jdbc.Driver");
	java.sql.Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/library_ms","root","PASSWORD");
	Statement st = con.createStatement();
	ResultSet rs = st.executeQuery("select * from book_details");
	
	while(rs.next()){
	String bookId = rs.getString("book_id");
	String bookName = rs.getString("book_name");
	String author = rs.getString("author");
	int quantity = rs.getInt("quantity");
	
	Object[] obj =

	{bookId,bookName,author,quantity};
	model = (DefaultTableModel)
	tbl_bookDetails.getModel();
	model.addRow(obj);
	}
	
	}catch(Exception e){
	e.printStackTrace();
	}}

addBook():
Attempts to add a new book record to the database. It retrieves book details (ID, name, author, quantity) likely from text fields in the application's interface. It establishes a connection, prepares a SQL INSERT statement with placeholders for the values, sets the placeholders with the retrieved data, and executes the statement. The method returns a boolean value (isAdded) indicating success (if at least one row was affected by the update).
	public boolean addBook(){
	boolean isAdded = false;
	bookId = Integer.parseInt(txt_bookId.getText());
	bookName = txt_bookName.getText();
	author = txt_authorName.getText();
	quantity = Integer.parseInt(txt_quantity.getText());
	
	try {
	Connection con =
	DBConnection.getConnection();
	String sql = "Insert into book_details
	values(?,?,?,?)";
	PreparedStatement pst =
	con.prepareStatement(sql);
	pst.setInt(1, bookId);
	pst.setString(2, bookName);
	pst.setString(3, author);
	pst.setInt(4, quantity);
	
	int rowCount = pst.executeUpdate();
	if (rowCount > 0) {
	isAdded = true;
	}else{
	isAdded = false;
	}
	}catch (Exception e) {
	e.printStackTrace();
	}
	return isAdded;
	}

updateBook():
Updates an existing book record. It retrieves details, establishes a
	public boolean updateBook(){
	boolean isUpdated = false;
	bookId = Integer.parseInt(txt_bookId.getText());

connection, prepares a SQL UPDATE statement targeting specific columns based on the book ID, sets the placeholders with new values, and executes the statement. It returns a boolean value (isUpdated) based on whether the update affected at least one row.

```
        bookName = txt_bookName.getText();
        author = txt_authorName.getText();
        quantity = Integer.parseInt(txt_quantity.getText());

        try {
            Connection con =
DBConnection.getConnection();
            String sql = "update book_details set book_name
= ?, author = ?, quantity = ? where book_id = ?";
            PreparedStatement pst =
con.prepareStatement(sql);
            pst.setString(1, bookName);
            pst.setString(2, author);
            pst.setInt(3, quantity);
            pst.setInt(4, bookId);


            int rowCount = pst.executeUpdate();
            if (rowCount > 0) {
                isUpdated = true;
            }else{
                isUpdated = false;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return isUpdated;
    }
```

deleteBook():
Deletes a book record based on the provided book ID. It establishes a connection, prepares a SQL DELETE statement filtering by book ID, sets the placeholder with the ID, and executes the statement. It returns a boolean value (isDeleted) indicating success (if at least one row was affected by the deletion).

```
public boolean deleteBook(){
        boolean isDeleted = false;
        bookId = Integer.parseInt(txt_bookId.getText());

        try{
            Connection con =
DBConnection.getConnection();
            String sql = "delete from book_details where
book_id = ? ";
            PreparedStatement pst =
con.prepareStatement(sql);
            pst.setInt(1, bookId);

            int rowCount = pst.executeUpdate();
            if(rowCount > 0) {
```

	`isDeleted = true;`
	`}else{`
	`isDeleted = false;`
	`}`
	`} catch (Exception e) {`
	`e.printStackTrace();`
	`}`
	`return isDeleted;`
	`}`

clearTable():
Clears all existing data from the table displayed in the application. It retrieves the table model (DefaultTableModel) and sets the row count to zero, effectively removing all rows from the table view.
	`public void clearTable(){`
	`DefaultTableModel model = (DefaultTableModel) tbl_bookDetails.getModel();`
	`model.setRowCount(0);`
	`}`

- ManageStudents.java:

Method name	**Code**

setStudentDetailsToTable():
Retrieves student information from the database and populates a table in the application. It establishes a connection, retrieves all data from the "student_details" table, and iterates through each row. Extracted details like student ID, name, course, and branch are stored in an array and added as a new row to a table, likely displayed within the interface.
	`public void setStudentDetailsToTable(){`
	`try{`
	`Class.forName("com.mysql.jdbc.Driver");`
	`java.sql.Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/library_ms","root","PASSWORD");`
	`Statement st = con.createStatement();`
	
	`ResultSet rs = st.executeQuery("select * from student_details");`
	
	`while(rs.next()){`
	`String StudentId = rs.getString("student_id");`
	`String StudentName = rs.getString("student_name");`
	`String course = rs.getString("course_name");`
	`String branch = rs.getString("branch");`
	
	`Object[] obj = {StudentId,StudentName,course,branch};`

	`            model = (DefaultTableModel)`
	`tbl_bookDetails.getModel();`
	`            model.addRow(obj);`
	`         }`
	
	`      }catch(Exception e){`
	`         e.printStackTrace();`
	`   }`
	`  }`

addStudent():
Attempts to add a new student record. It retrieves student information (ID, name, course, branch) likely from text fields or combo boxes. It connects to the database, prepares a SQL INSERT statement with placeholders, sets the placeholders with the retrieved data, and executes the statement. The method returns a boolean value (isAdded) indicating success (if at least one row was affected by the update).
	`public boolean addStudent(){`
	`      boolean isAdded = false;`
	`      studentId = Integer.parseInt(txt_studentId.getText());`
	`      studentName = txt_studentName.getText();`
	`      course =`
	`combo_courseName.getSelectedItem().toString();`
	`      branch =`
	`combo_branch.getSelectedItem().toString();`
	
	`      try {`
	`         Connection con =`
	`DBConnection.getConnection();`
	`         String sql = "Insert into student_details`
	`values(?,?,?,?)";`
	`         PreparedStatement pst =`
	`con.prepareStatement(sql);`
	`         pst.setInt(1, studentId);`
	`         pst.setString(2, studentName);`
	`         pst.setString(3, course);`
	`         pst.setString(4, branch);`
	
	`         int rowCount = pst.executeUpdate();`
	`         if (rowCount > 0) {`
	`            isAdded = true;`
	`         }else{`
	`            isAdded = false;`
	`         }`
	`      }catch (Exception e) {`
	`         e.printStackTrace();`
	`      }`
	`      return isAdded;`
	`   }`

updateStudent():
Updates an existing student record. It retrieves details,
	`public boolean updateStudent(){`
	`      boolean isUpdated = false;`
	`      studentId = Integer.parseInt(txt_studentId.getText());`

establishes a connection, prepares a SQL UPDATE statement targeting specific columns based on the student ID, sets the placeholders with new values, and executes the statement. It returns a boolean value (isUpdated) based on whether the update affected at least one row.	
	```
	    studentName = txt_studentName.getText();
	    course =
	combo_courseName.getSelectedItem().toString();
	    branch =
	combo_branch.getSelectedItem().toString();

	    try {
	      Connection con =
	DBConnection.getConnection();
	      String sql = "update student_details set name = ?,
	course = ?, branch = ? where student_id = ?";
	      PreparedStatement pst =
	con.prepareStatement(sql);
	      pst.setString(1, studentName);
	      pst.setString(2, course);
	      pst.setString(3, branch);
	      pst.setInt(4, studentId);

	      int rowCount = pst.executeUpdate();
	      if (rowCount > 0) {
	        isUpdated = true;
	      }else{
	        isUpdated = false;
	      }
	    } catch (Exception e) {
	      e.printStackTrace();
	    }
	    return isUpdated;
	  }
	```

deleteStudent():
Deletes a student record based on the provided student ID. It establishes a connection, prepares a SQL DELETE statement filtering by student ID, sets the placeholder with the ID, and executes the statement. It returns a boolean value (isDeleted) indicating success (if at least one row was affected by the deletion).
	```
	public boolean deleteStudent(){
	    boolean isDeleted = false;
	    studentId = Integer.parseInt(txt_studentId.getText());

	    try{
	      Connection con =
	DBConnection.getConnection();
	      String sql = "delete from student_details where
	student_id = ?, ";
	      PreparedStatement pst =
	con.prepareStatement(sql);
	      pst.setInt(1, studentId);

	      int rowCount = pst.executeUpdate();
	```

	if(rowCount > 0) {
	    isDeleted = true;
	}else{
	    isDeleted = false;
	}
	} catch (Exception e) {
	    e.printStackTrace();
	}
	return isDeleted;
	}

clearTable():
Clears all existing data from the table displayed in the application. It retrieves the table model (DefaultTableModel) and sets the row count to zero, effectively removing all rows from the table view.
	public void clearTable(){
	    DefaultTableModel model = (DefaultTableModel) tbl_bookDetails.getModel();
	    model.setRowCount(0);
	}

- IssueBook.java:

Method name	Code

getBookDetails():
Retrieves details of a book based on its ID entered by the user. It connects to the database, executes a query to find the book, and populates labels in the interface with extracted information like title, author, and quantity. If the ID is invalid, it displays an error message.
	public void getBookDetails(){
	    int bookId = Integer.parseInt(txt_bookId.getText());
	
	    try{
	        Connection con = DBConnection.getConnection();
	        PreparedStatement pst = con.prepareStatement("select * from book_details where book_id = ?");
	        pst.setInt(1, bookId);
	        ResultSet rs = pst.executeQuery();
	
	        if(rs.next()){
	            lbl_bookId.setText(rs.getString("book_id"));
	
	lbl_bookName.setText(rs.getString("book_name"));
	            lbl_author.setText(rs.getString("author"));
	            lbl_quantity.setText(rs.getString("quantity"));
	
	        } else {
	            lbl_bookError.setText("Invalid book id");

	`}`
	`} catch (Exception e){`
	`e.printStackTrace();`
	`}`
	`}`

getStudentDetails():
Retrieves details of a student based on their ID. It connects to the database, searches for the student using the ID, and populates labels on the interface with details like name, course, and branch. An error message is displayed if the student ID is invalid.
	`public void getStudentDetails(){`
	`int studentId =`
	`Integer.parseInt(txt_StudentId.getText());`
	
	`try{`
	`Connection con = DBConnection.getConnection();`
	`PreparedStatement pst =`
	`con.prepareStatement("select * from student_details where student_id = ?");`
	`pst.setInt(1, studentId);`
	`ResultSet rs = pst.executeQuery();`
	
	`if(rs.next()){`
	
	`lbl_studentId.setText(rs.getString("student_id"));`
	`lbl_studentName.setText(rs.getString("name"));`
	`lbl_course.setText(rs.getString("course"));`
	`lbl_branch.setText(rs.getString("branch"));`
	`} else {`
	`lbl_studentError.setText("Invalid student id");`
	`}`
	
	`} catch (Exception e){`
	`e.printStackTrace();`
	`}`
	`}`

issueBook():
Handles issuing a book to a student. It gathers details like book and student IDs, retrieves issues and due dates, and establishes a database connection. It then prepares an INSERT statement to add a new record to the "issue_book_details" table, containing details about the issued book. The method checks the success of the insertion and
	`public boolean issueBook() {`
	`boolean isIssued = false;`
	`int bookId = Integer.parseInt(txt_bookId.getText());`
	`int studentId =`
	`Integer.parseInt(txt_StudentId.getText());`
	`String bookName = lbl_bookName.getText();`
	`String studentName = lbl_studentName.getText();`
	
	`Date uIssueDate = date_issueDate.getDatoFecha();`
	`Date uDueDate = date_dueDate.getDatoFecha();`
	
	`Long l1 = uIssueDate.getTime();`
	`Long l2 = uDueDate.getTime();`

returns a boolean value indicating whether the book was issued successfully.	java.sql.Date sIssueDate = new java.sql.Date(l1);
	java.sql.Date sDueDate = new java.sql.Date(l2);
	
	try {
	Connection con = DBConnection.getConnection();
	String sql = "Insert into issue_book_details(book_id,book_name,student_id,student_name" + "issue_date_date, due_date, status) values(?,?,?,?,?,?,?)";
	PreparedStatement pst = con.prepareStatement(sql);
	pst.setInt(1,bookId);
	pst.setString(2,bookName);
	pst.setInt(3,studentId);
	pst.setString(4,studentName);
	pst.setDate(5, sIssueDate);
	pst.setDate(6,sDueDate);
	pst.setString(7,"pending");
	
	int rowCount = pst.executeUpdate();
	if(rowCount > 0){
	isIssued = true;
	} else {
	isIssued = false;
	}
	}catch (Exception e){
	e.printStackTrace();
	}
	
	return isIssued;
	}

updateBookCount():
Updates the book quantity after a book is issued. It retrieves the book ID, connects to the database, and prepares an UPDATE statement to decrement the quantity for the book in the "book_details" table. The method checks the update's success and updates the book quantity displayed on the interface if successful. Otherwise, it displays an error
	public void updateBookCount() {
	int bookId = Integer.parseInt(txt_bookId.getText());
	try {
	Connection con = DBConnection.getConnection();
	String sql = "update book details set quantity = quantity - 1 where book_id = ?";
	PreparedStatement pst = con.prepareStatement(sql);
	pst.setInt(1,bookId);
	
	int rowCount = pst.executeUpdate();
	if (rowCount > 0) {
	JOptionPane.showMessageDialog(this,"book

message.	count update");
	            int initialCount =
	Integer.parseInt(lbl_quantity.getText());
	            lbl_quantity.setText(Integer.toString(initialCount
	- 1));
	         } else {
	            JOptionPane.showMessageDialog(this,"can't
	update book count");
	         }
	      } catch (Exception e){
	         e.printStackTrace();
	      }
	   }

isAlreadyIssued():
Checks if a specific book is already issued to a particular student. It retrieves book and student IDs, connects to the database, and executes a query targeting the "issue_book_details" table. The query searches for a record matching the IDs with a "pending" status (indicating an unreturned book). The method returns a boolean value indicating whether the book is already issued.
	public boolean isAlreadyIssued() {
	      boolean isAlreadyIssued = false;
	      int bookId = Integer.parseInt(txt_bookId.getText());
	      int studentId =
	Integer.parseInt(txt_StudentId.getText());
	
	      try {
	         Connection con = DBConnection.getConnection();
	         String sql = "select * from issue_book_details
	where book_id = ? and student_id = ? and status = ?";
	         PreparedStatement pst =
	con.prepareStatement(sql);
	         pst.setInt(1,bookId);
	         pst.setInt(2,studentId);
	         pst.setString(3,"pending");
	
	         ResultSet rs = pst.executeQuery();
	         if(rs.next()) {
	            isAlreadyIssued = true;
	         } else {
	            isAlreadyIssued = false;
	         }
	      } catch(Exception e) {
	         e.printStackTrace();
	      }
	      return isAlreadyIssued;
	   }

- ReturnBook.java:

Method name	Code

getIssueBookDetails():
Retrieves details of an issued book. It retrieves book and student IDs from text fields, connects to the database, and executes a query targeting the "issue_book_details" table. The query searches for a record matching the IDs with a "pending" status (indicating an unreturned book). If a record is found, it extracts details like issue ID, book name, student name, issue date, and due date, and populates corresponding labels on the interface. If no record is found, it displays an error message and clears all the detail labels.
	public void getIssueBookDetails() {
	
	        int bookId = Integer.parseInt(txt_bookId.getText());
	        int studentId =
	Integer.parseInt(txt_StudentId.getText());
	
	        try{
	            Connection con = DBConnection.getConnection();
	            String sql = "Select * from issue_book_details
	where book_id = ? and student_id = ? and status = ?";
	
	            PreparedStatement pst =
	con.prepareStatement(sql);
	            pst.setInt(1,bookId);
	            pst.setInt(2, studentId);
	            pst.setString(3, "pending");
	            ResultSet rs = pst.executeQuery();
	            if(rs.next()) {
	
	                lbl_issueId.setText(rs.getString("id"));
	
	lbl_bookName.setText(rs.getString("book_name"));
	
	lbl_studentName.setText(rs.getString("student_name"));
	
	lbl_issueDate.setText(rs.getString("issue_date"));
	                lbl_dueDate.setText(rs.getString("due_date"));
	                lbl_bookError.setText(" ");
	
	            } else {
	                lbl_bookError.setText("No record found.");
	                lbl_issueId.setText("");
	                lbl_bookName.setText("");
	                lbl_studentName.setText("");
	                lbl_issueDate.setText("");
	                lbl_dueDate.setText("");
	            }
	        } catch(Exception e) {
	            e.printStackTrace();
	        }
	    }

returnBook():	public boolean returnBook() {

This method handles returning a book. It retrieves book and student IDs, connects to the database, and prepares an UPDATE statement to modify the status of the corresponding record in the "issue_book_details" table. The status is changed from "pending" (issued) to "returned". The method checks the update's success and returns a boolean value indicating whether the book was successfully returned.	
	```
	        boolean isReturned = false;
	        int bookId = Integer.parseInt(txt_bookId.getText());
	        int studentId =
	Integer.parseInt(txt_StudentId.getText());

	        try{
	            Connection con = DBConnection.getConnection();
	            String sql = "update issue_book_details set status
	= ? where student_id = ? and book_id = ? and status = ?";
	            PreparedStatement pst =
	con.prepareStatement(sql);
	            pst.setString(1,"returned");
	            pst.setInt(2,studentId);
	            pst.setInt(3, bookId);
	            pst.setString(4,"pending");

	            int rowCount = pst.executeUpdate();
	            if(rowCount > 0) {
	              isReturned = true;
	            } else {
	               isReturned = false;
	            }
	        } catch(Exception e) {
	            e.printStackTrace();
	        }
	        return isReturned;
	    }
	```

updateBookCount():
Similar to the previous implementation, this method updates the book quantity after a book is returned. It retrieves the book ID, connects to the database, and prepares an UPDATE statement to increment the quantity for the book in the "book_details" table. The method checks the update's success and displays a message using JOptionPane depending on the outcome.
	```
	public void updateBookCount() {
	        int bookId = Integer.parseInt(txt_bookId.getText());
	        try {
	            Connection con = DBConnection.getConnection();
	            String sql = "update book details set quantity =
	quantity + 1 where book_id = ?";
	            PreparedStatement pst =
	con.prepareStatement(sql);
	            pst.setInt(1,bookId);

	            int rowCount = pst.executeUpdate();
	            if (rowCount > 0) {
	               JOptionPane.showMessageDialog(this,"book
	count update");
	            } else {
	               JOptionPane.showMessageDialog(this,"can't
	update book count");
	```

	```
	        }
	    } catch (Exception e){
	        e.printStackTrace();
	    }
	}
	```

- ViewAllRecord.java:

Method name	Code

setIssueBookDetailsToTable(): Populates a table (likely tbl_issueBookDetails) in the interface with details of all issued books. It establishes a connection to the database, executes a query to retrieve all records from the "issue_book_details" table, and iterates through the results. For each record, it extracts details like ID, student name, book name, issue date, due date (seems to be an integer here), and status. This data is then stored in an array (obj) and added as a new row to the table model (model). In case of any exceptions during database operations, it includes error handling (try-catch).	
	```
	public void setIssueBookDetailsToTable(){
	    try{
	        Class.forName("com.mysql.jdbc.Driver");
	        java.sql.Connection con =
	DriverManager.getConnection("jdbc:mysql://localhost:3306/library_ms","root","PASSWORD");
	        Statement st = con.createStatement();
	        ResultSet rs = st.executeQuery("select * from
	issue_book_details");

	        while(rs.next()){
	            String id = rs.getString("id");
	            String studentName =
	rs.getString("student_name");
	            String bookName = rs.getString("book_name");
	            String issueDate = rs.getString("issue_date");
	            int dueDate = rs.getInt("due_date");
	            String status = rs.getString("status");
	            Object[] obj = {id,bookName,studentName,
	issueDate,dueDate,status};
	            model = (DefaultTableModel)
	tbl_issueBookDetails.getModel();
	            model.addRow(obj);
	        }

	    } catch(Exception e){
	        e.printStackTrace();
	    }
	}
	```

clearTable(): Clears the existing data from the table (tbl_issueBookDetails). It retrieves the table model	
	```
	public void clearTable() {
	    DefaultTableModel model = (DefaultTableModel)
	tbl_issueBookDetails.getModel();
	    model.setRowCount(0);
	```

(model) and sets the row count to zero, effectively removing all previously added rows.	}

search():
enables searching for issued books based on a date range. It retrieves dates from the interface's date pickers (date_fromDate and date_toDate), converts them to compatible java.sql.Date objects, and establishes a connection to the database. It then prepares a SELECT statement targeting the "issue_book_details" table with a condition that the issue_date falls between the provided fromDate and toDate (the syntax seems to have a typo, using to instead of AND). The statement uses placeholders for the dates, which are then set with the converted fromDate and toDate objects. The method executes the query and checks for results. If no records are found (rs.next() returns false), it displays a message using JOptionPane. Otherwise, it iterates through the results, similar to setIssueBookDetailsToTable, extracting details and adding them as rows to the table model. Error handling (try-catch) is included for database operations.
	public void search() {
	    Date uFromDate = date_fromDate.getDatoFecha();
	    Date uToDate = date_toDate.getDatoFecha();
	
	    long l1 = uFromDate.getTime();
	    long l2 = uToDate.getTime();
	
	    java.sql.Date fromDate = new java.sql.Date(l1);
	    java.sql.Date toDate = new java.sql.Date(l2);
	
	    try {
	      Connection con = DBConnection.getConnection();
	      String sql = "select * from issue_book_details where issue_date BETWEEN ? to ?";
	      PreparedStatement pst = con.prepareStatement(sql);
	      pst.setDate(1,fromDate);
	      pst.setDate(2,toDate);
	
	      ResultSet rs = pst.executeQuery();
	
	      if(rs.next()== false) {
	        JOptionPane.showMessageDialog(this,"No record found");
	      } else {
	
	        while(rs.next()){
	          String id = rs.getString("id");
	          String studentName = rs.getString("student_name");
	          String bookName = rs.getString("book_name");
	          String issueDate = rs.getString("issue_date");
	          int dueDate = rs.getInt("due_date");
	          String status = rs.getString("status");
	
	          Object[] obj = {id,bookName,studentName, issueDate,dueDate,status};
	          model = (DefaultTableModel) tbl_issueBookDetails.getModel();
	          model.addRow(obj);

	`}`
	`}`
	
	`} catch(Exception e) {`
	`e.printStackTrace();`
	`}`
	`}`

- IssuebookDetails.java:

Method name	**Code**

setIssueBookDetailsToTable(): Retrieves and displays details of issued books that are still pending return. It connects to the database, executes a query to fetch records with a status of "pending" from the "issue_book_details" table, and iterates through the results. Extracted details like ID, student name, book name, issue date, due date, and status are then stored in an array and added as a new row to a table likely displayed within the application's interface.	`public void setIssueBookDetailsToTable(){`
	`try{`
	`Class.forName("com.mysql.cj.jdbc.Driver");`
	`java.sql.Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/library_ms","root","PASSWORD");`
	`Statement st = con.createStatement();`
	`ResultSet rs = st.executeQuery("select * from issue_book_details where status = '"+"pending"+"'");`
	
	`while(rs.next()){`
	`String id = rs.getString("id");`
	`String studentName = rs.getString("student_name");`
	`String bookName = rs.getString("book_name");`
	`String issueDate = rs.getString("issue_date");`
	`java.sql.Date dueDate = rs.getDate("due_date");`
	`String status = rs.getString("status");`
	`Object[] obj = {id,bookName,studentName, issueDate,dueDate,status};`
	`model = (DefaultTableModel) tbl_issueBookDetails.getModel();`
	`model.addRow(obj);`
	`}`
	
	`} catch(Exception e){`
	`e.printStackTrace();`
	`}`
	`}`

- DefaulterList.java:

Method name	Code

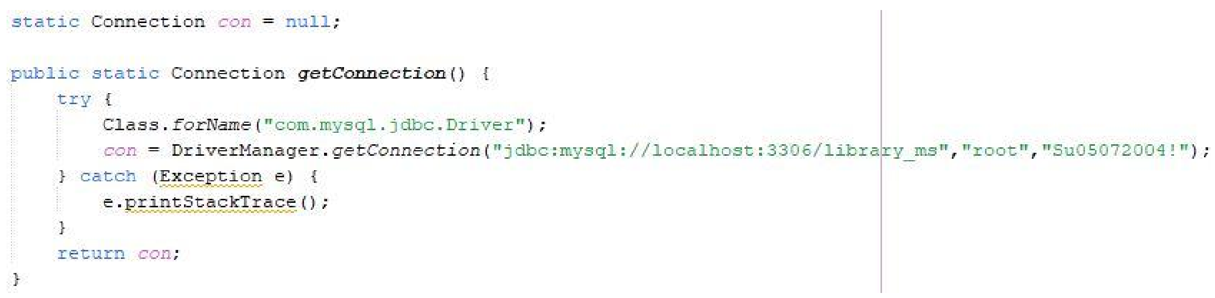
setIssueBookDetailsToTable(): Displays a list of overdue issued books in the interface's table. It gets the current date, converts it to a java.sql.Date object, and connects to the database. It then prepares a query targeting the "issue_book_details" table. The query searches for records where the "due_date" is earlier than the current date (indicating overdue books) and the status is still "pending" (not returned). The retrieved data (ID, student name, book details, issue/due date, and status) is presented in a table by iterating through results, storing them in an array, and adding the array as a new row to the table model. The method includes error handling (try-catch) for database operations.	
	```
	public void setIssueBookDetailsToTable(){
	    long l = System.currentTimeMillis();
	    java.sql.Date todaysDate = new java.sql.Date(l);
	    try{
	        Class.forName("com.mysql.cj.jdbc.Driver");
	        java.sql.Connection con =
	DriverManager.getConnection("jdbc:mysql://localhost:33
	06/library_ms","root","PASSWORD");
	        java.sql.PreparedStatement pst =
	con.prepareStatement("select * from issue_book_details
	where due_date < ? and status = ?");
	        pst.setDate(1, todaysDate);
	        pst.setString(2, "pending");
	        ResultSet rs = pst.executeQuery();

	        while(rs.next()){
	            String id = rs.getString("id");
	            String studentName = rs.getString("name");
	            String bookName = rs.getString("book_name");
	            String issueDate = rs.getString("issue_date");
	            java.sql.Date dueDate = rs.getDate("due_date");
	            String status = rs.getString("status");
	            Object[] obj = {id,bookName,studentName,
	issueDate,dueDate,status};
	            model = (DefaultTableModel)
	tbl_issueBookDetails.getModel();
	            model.addRow(obj);
	        }

	    } catch(Exception e){
	        e.printStackTrace();
	    }
	}
	```

4.2 Connection implementation

Database Connection Configuration:



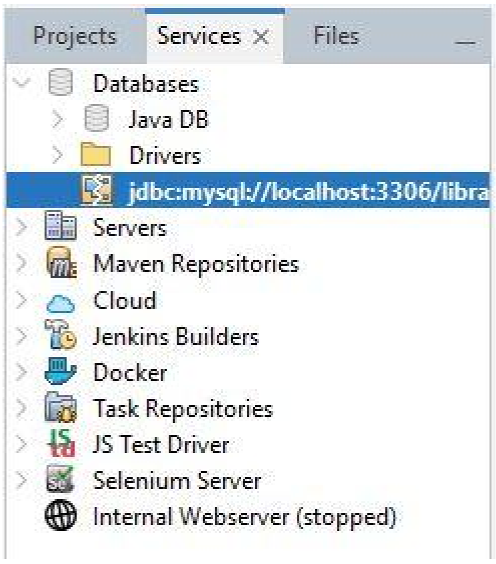
```
static Connection con = null;

public static Connection getConnection() {
    try {
        Class.forName("com.mysql.jdbc.Driver");
        con = DriverManager.getConnection("jdbc:mysql://localhost:3306/library_ms","root","Su05072004!");
    } catch (Exception e) {
        e.printStackTrace();
    }
    return con;
}
```

This section defines the configuration for the database connection. It includes the server address, database name, and users (with password) to connect to the database.

Server Setup:

We set up the server by adding a new database connection in the Services tab so that we can have the access to the database.

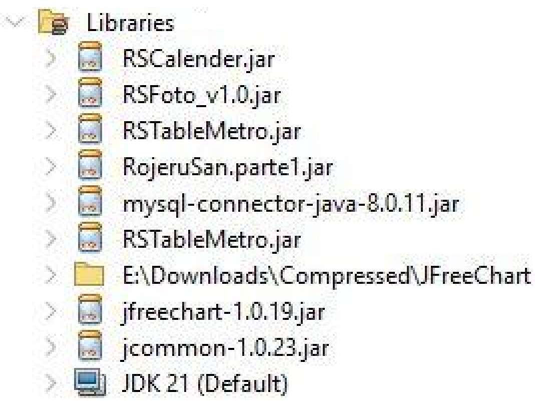


```
Projects   Services ×   Files
Databases
    Java DB
    Drivers
    jdbc:mysql://localhost:3306/libra
Servers
Maven Repositories
Cloud
Jenkins Builders
Docker
Task Repositories
JS Test Driver
Selenium Server
Internal Webserver (stopped)
```

Figure 4.1: Adding Database Connection

- ***Import Components:***

Apart from the database connection component, we also used some other components made by RojeruSan (RSCalendar, RSTableMetro, .etc) and JFree (JFreeChart and JCommon)



```
Libraries
    RSCalender.jar
    RSFoto_v1.0.jar
    RSTableMetro.jar
    RojeruSan.parte1.jar
    mysql-connector-java-8.0.11.jar
    RSTableMetro.jar
    E:\Downloads\Compressed\JFreeChart
    jfreechart-1.0.19.jar
    jcommon-1.0.23.jar
    JDK 21 (Default)
```

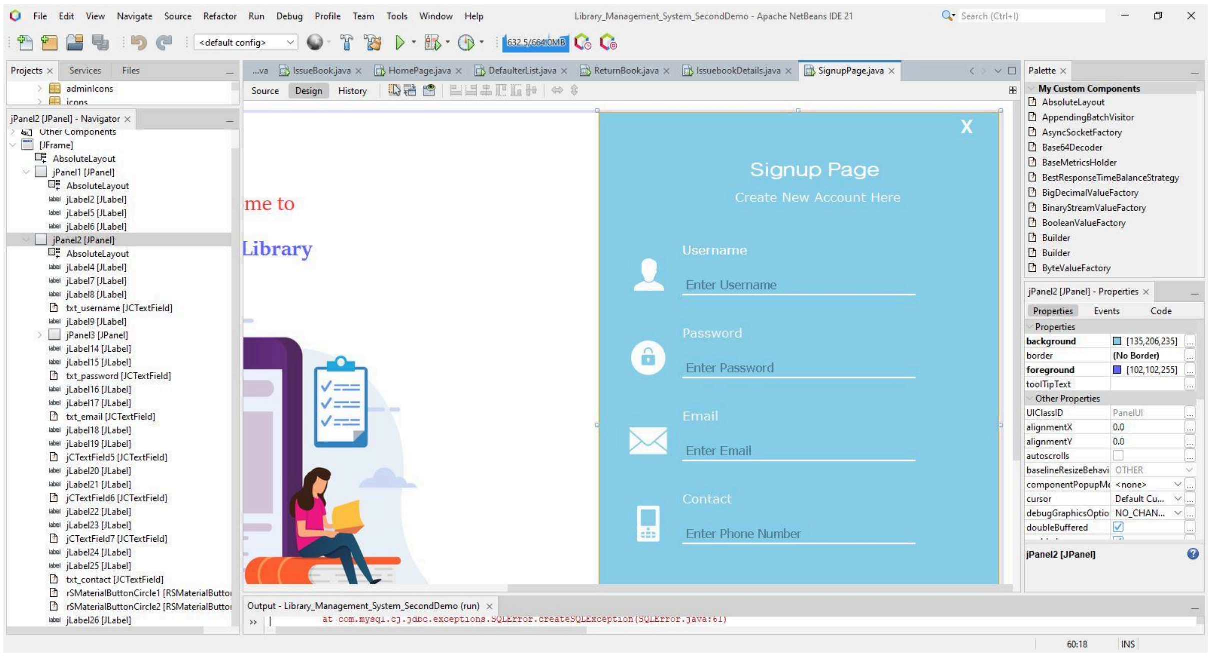
Figure 4.2: Some of the Components

4.3 GUI Design - Frames Design.

We've designed the frames directly on Netbeans GUI Designer, then we just need to drag and drop the element to the JFrame created.

To initialized the frames, we used the constructor provided by both the Netbeans's GUI Designer and RojeruSan's Components (JPanel, JTextField, RSDateChooser, .etc):

Element type	**Description**
JPanel	to add a frame for other elements, it must be extended from our frame classes
RSMaterialButtonCircle	add a button to perform logical code when clicking (discuss more in the later part)
JScrollPane	add a frame that can be scrollable, especially for long content
JLabel	to add title and instruction on the interface
JTextField	to receive user input on the interface
RSTableMetro	to display the query result in tabular form
RSDateChooser	to receive user input on the interface and display the calendar when clicking the calendar icon



background	[135,206,235]
border	(No Border)
foreground	[102,102,255]
toolTipText	
UIClassID	PanelUI
alignmentX	0.0
alignmentY	0.0
autoscrolls	☐
baselineResizeBehavi	OTHER
componentPopupMe	<none>
cursor	Default Cu...
debugGraphicsOptio	NO_CHAN...
doubleBuffered	✓

```
at com.mysql.cj.jdbc.exceptions.SQLError.createSQLException(SQLError.java:61)
```

Figure 4.3: Example of Using Elements to Create SignupPage.java

4.4 Button implementation

Buttons play a significant role in facilitating user interactions on websites. To streamline the administration and consistency of all buttons throughout the program, we begin by developing a universal button component. This method enhances code organization and cleanliness while implementing UI design. It provides an effective way to call and manage buttons across the system.

As there were many buttons that needed to be implemented, we only showed some common patterns for the button:

Button name	**Action description**
Log in, sign up buttons	Use user input (account, password) for authentication and perform the subsequent actions
Add buttons	To insert the new data into the query
Update buttons	To update the data into the correct one

Delete buttons	Delete the books or student that is no longer needs
"X" buttons	To exit the program
Find Button	To return the result of the query
Logout button	To log out of the application, then return to Login page
Back button	To go back to the previous frame
Issue Book button	To issue the book to the student

4.5 Application Demo - Screenshots:

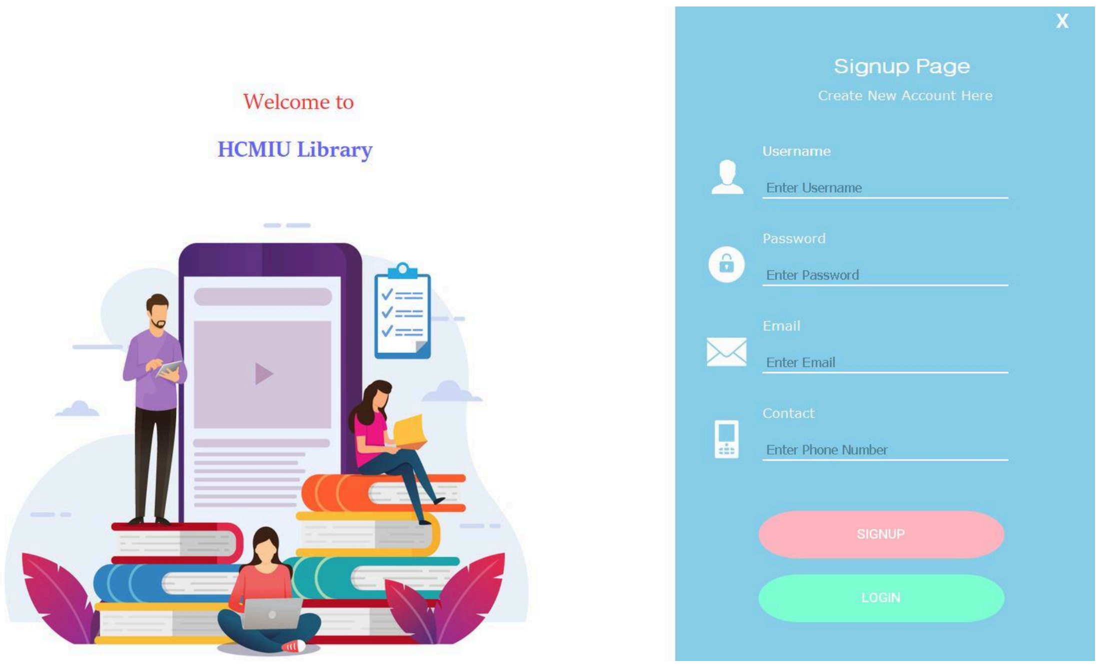


Figure 4.4: Signup Page

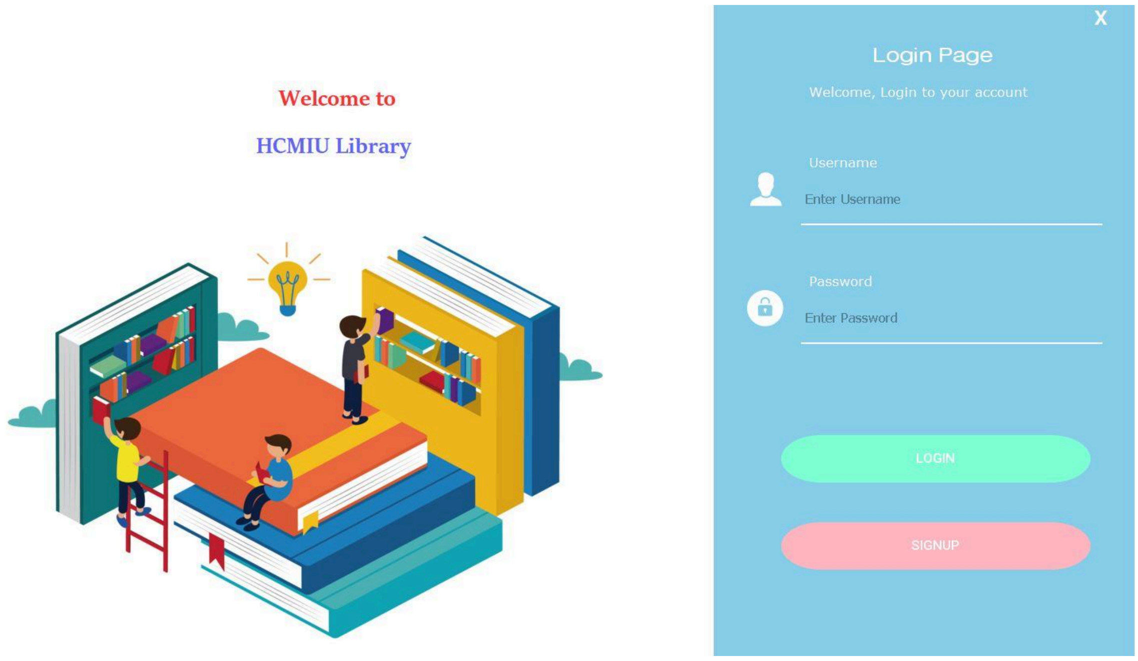
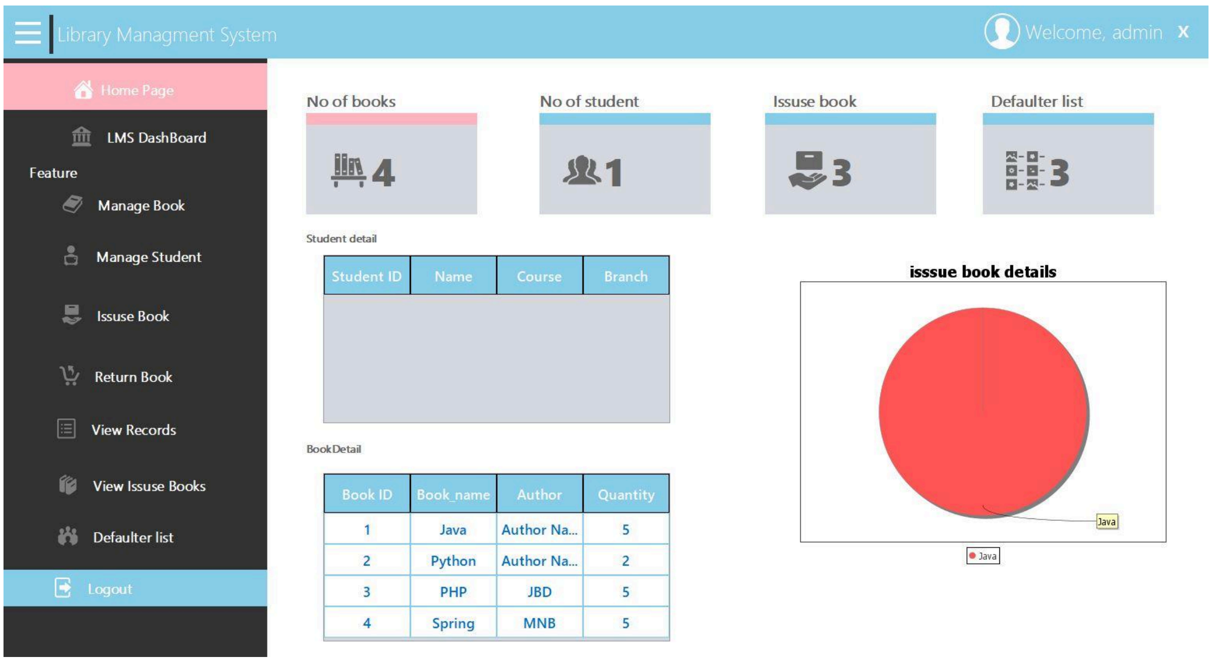


Figure 4.5: Login Page



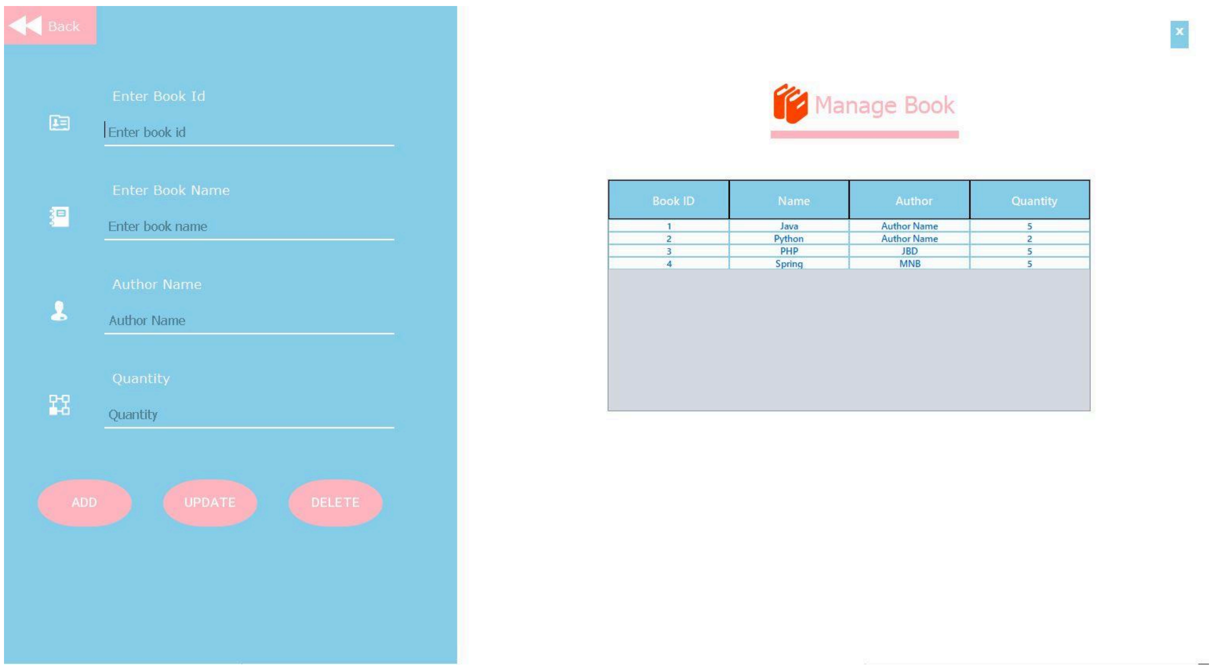
Student detail

Student ID	Name	Course	Branch

BookDetail

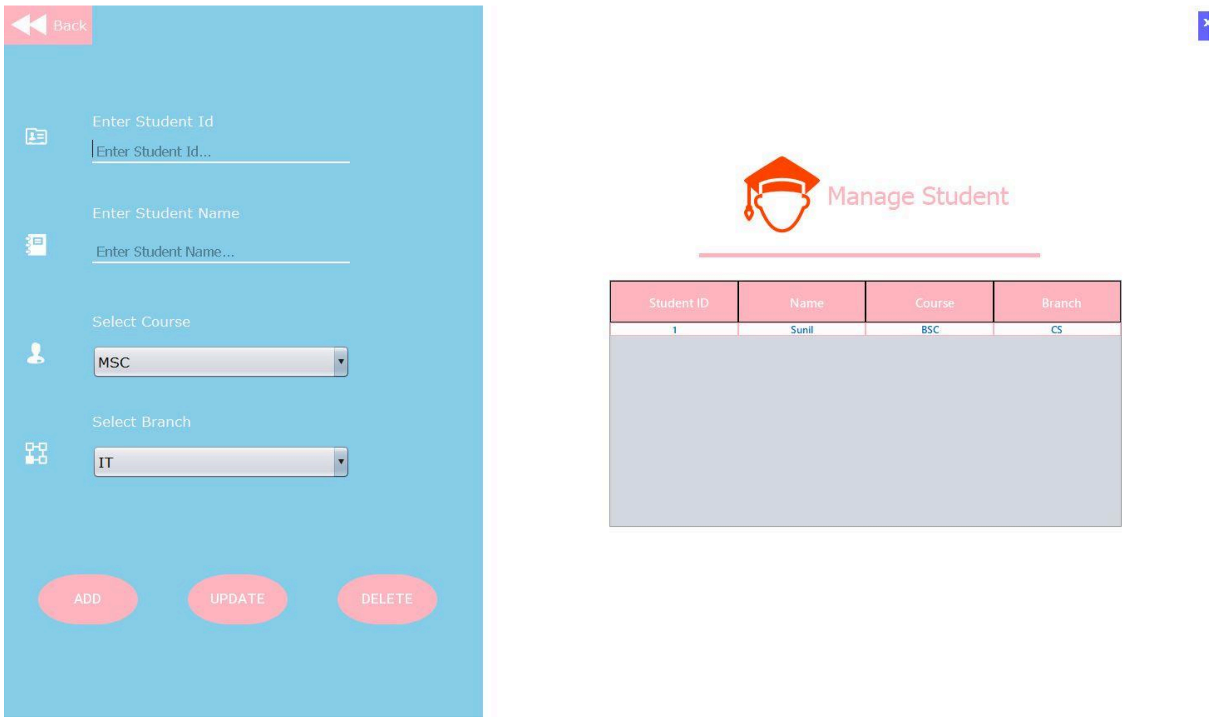
Book ID	Book_name	Author	Quantity
1	Java	Author Na...	5
2	Python	Author Na...	2
3	PHP	JBD	5
4	Spring	MNB	5

Figure 4.6: Home Page



Book ID	Name	Author	Quantity
1	Java	Author Name	5
2	Python	Author Name	2
3	PHP	JBD	5
4	Spring	MNB	5

Figure 4.7: Manage Book Page



Student ID	Name	Course	Branch
1	Sunil	BSC	CS

Figure 4.8: Manage Student Page

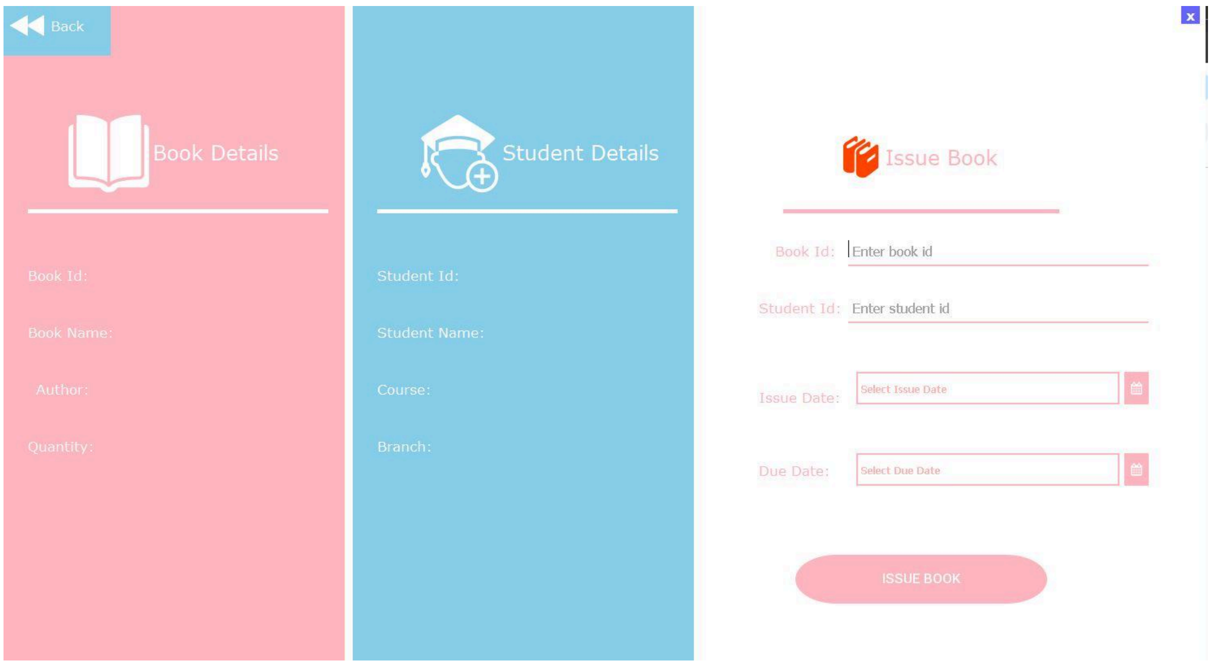
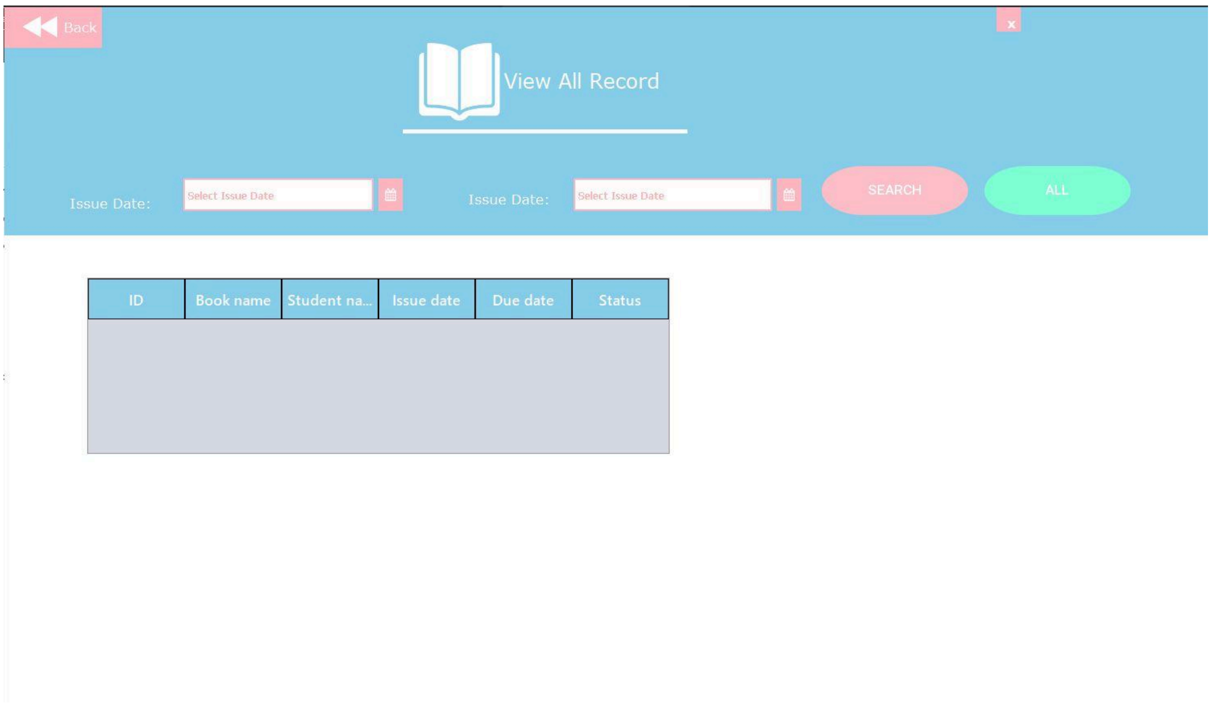
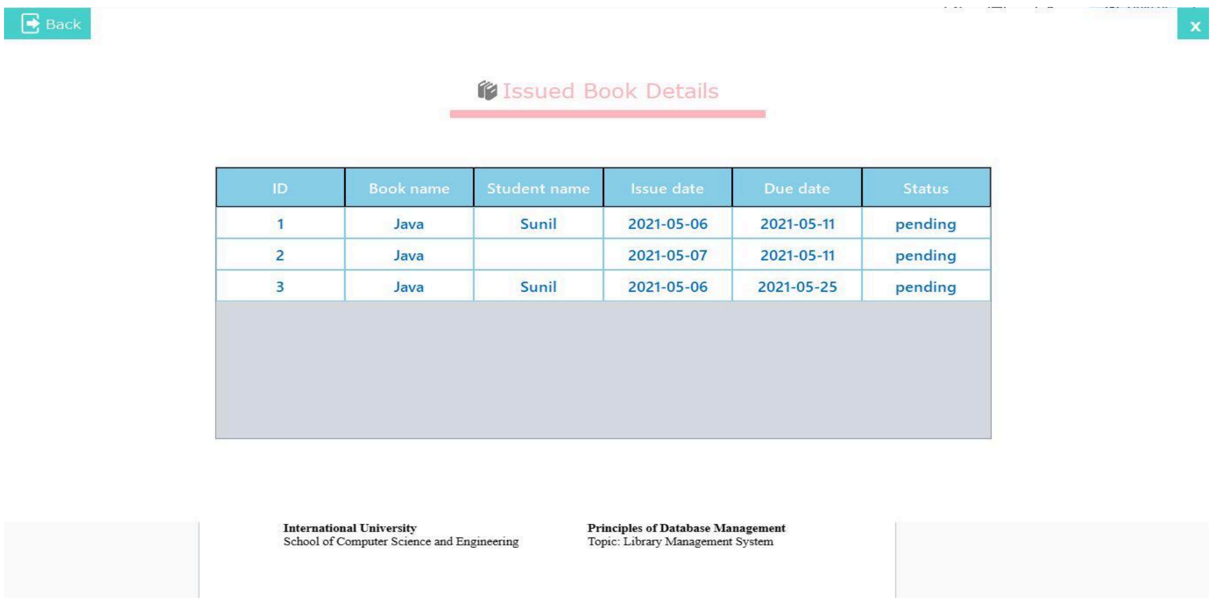


Figure 4.9: Issue Book Page



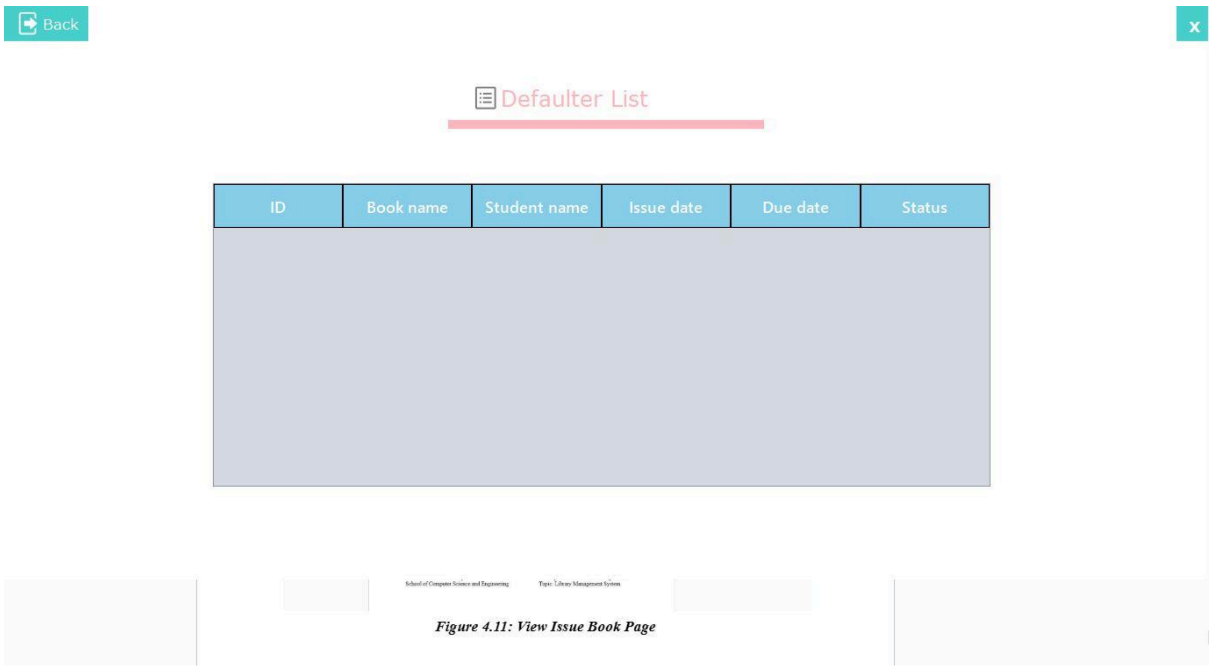
ID	Book name	Student na...	Issue date	Due date	Status

Figure 4.10: View Record Page



ID	Book name	Student name	Issue date	Due date	Status
1	Java	Sunil	2021-05-06	2021-05-11	pending
2	Java		2021-05-07	2021-05-11	pending
3	Java	Sunil	2021-05-06	2021-05-25	pending

Figure 4.11: View Issue Book Page



ID	Book name	Student name	Issue date	Due date	Status

Figure 4.12: Defaulter List Page

IV CONCLUSION

1. Achieved goals

During the project, our team successfully achieved the goal of developing the library management system. Firstly, we constructed a database adhering to the principles of Boyce-Codd Normal Form (BCNF). This optimized data storage and retrieval while minimizing redundancy and data anomalies. Secondly, we established the system using NetBeans for the user interface and MySQL for the backend database, utilizing JDBC for a secure connection between the front-end and back-end. Finally, we developed essential functionalities like user login, account creation, and diverse management tools. In addition, complex queries were effectively employed to handle various library scenarios.

These accomplishments resulted in a robust and user-friendly library management system that aligns with contemporary standards in database management, security, and user experience. This system streamlines library operations and facilitates efficient information access.

2. Future work

With these goals achieved, the project holds immense potential to evolve into a robust and practical application for real-world use. Moving forward, we aim to further enhance the library management system by prioritizing multi-user functionality. To be specific, we will introduce functionalities that cater to students as users. This will empower them with the ability to manage their accounts, viewing borrowed books, placing holds on desired titles, and initiating new borrowing requests. This personalized approach will streamline library operations and empower students with greater self-service capabilities. Additionally, we plan to introduce search functionalities, allowing students to locate books by title, author, category or publisher, significantly improving the efficiency and user-friendliness of navigating the library's collection.

3. Concluding thoughts

In summary, this project has been a valuable learning experience, fostering collaboration, communication, and problem-solving skills that will undoubtedly benefit us in future attempts. We are grateful to our instructors for their guidance, acknowledging their crucial role in the project's success. Moreover, we envision further development of the library management system, prioritizing user-friendliness and practicality to create a seamless experience for all users. With a commitment to continuous learning and innovation, we remain enthusiastic about the system's potential to streamline library processes and enhance user experience.