AMRITSAR GROUP OF COLLEGES

Java Project Report

On

Library Management System

Submitted in the partial fulfillment of the requirement for the award of degree of

Bachelor of Technology

In

COMPUTER SCIENCE & ENGINEERING

Batch (2023-2027)

Submitted to

Er. Jaspreet Singh

Submitted by

Kaushlendra Kumar (2333338)

Kundan Kumar (2333345)

Kundan Kumar Sah (2333346)

Kundan Kumar Singh (2333347)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

TABLE OF CONTENTS

Declaration

We, **Kundan Kumar, Kaushlendra Kumar, Kundan Kumar Sah, Kundan Kumar Singh**, student of COMPUTER SCIENCE & ENGINEERING at AMRITSAR GROUP OF COLLEGES, hereby declare that the project titled **"**LIBRARY MANAGEMENT SYSTEM**"** submitted by me is my own original work.

We affirm that this project has been carried out by me under the guidance of **ER. JASPREET SINGH**, and all the information, ideas, and content used in this project have been appropriately acknowledged. we further declare that this work has not been submitted previously, either partially or fully, for the award of any degree, diploma, or certificate in any institution.

I take full responsibility for the authenticity of this project and confirm that the contents reflect my own efforts and learning.

Kaushlendra Kumar (2333338)

Kundan Kumar (2333345)

Kundan Kumar Sah (2333436)

Kundan Kumar Singh (2333347)

Acknowledgement

We would like to express our heartfelt gratitude to all those who have guided and supported us throughout the development of this Library Management System project.

First and foremost, we extend our sincere thanks to **Er. Jaspreet Singh**, our project supervisor, for their invaluable guidance, constant encouragement, and insightful suggestions. Their expertise and mentorship were crucial in enabling us to complete this project successfully.

We are also grateful to the faculty members of **Computer Science &Engineering** for providing the necessary resources, technical support, and motivation, which greatly facilitated our learning and project development process. Their support created an environment that fostered both innovation and practical learning.

Our sincere appreciation goes to our friends and colleagues who assisted us with ideas, feedback, and collaborative efforts. Their support and cooperation made the project journey more efficient and enjoyable.

Finally, we express our profound gratitude to our families for their unwavering support, understanding, and encouragement throughout this project. Their confidence in us inspired perseverance and dedication during challenging times.

This project has been an invaluable learning experience, enhancing our technical skills, problem-solving abilities, and teamwork. We are truly grateful to everyone who contributed, directly or indirectly, to the successful completion of this project.

Abstract

The Library Management System is a software application designed to automate and streamline the daily operations of a library. The main objective of this system is to efficiently manage book records, member information, and the process of issuing and returning books. This project aims to reduce manual work, minimize errors, and improve the overall management of library resources.

Through this system, library staff can easily add, update, delete, and track books, while members can search for and borrow books efficiently. The system also calculates fines automatically for late returns and generates various reports, such as issued books, available books, and fines collected.

Developed using **Java** for the graphical user interface and **MySQL** for the backend database, this system provides a user-friendly and secure environment for both staff and members. The Library Management System ensures organized record-keeping, time efficiency, and enhanced accessibility, making library operations more reliable and effective.

This project not only improves the operational efficiency of a library but also offers a practical solution for managing books and members in an organized and systematic manner.

Introduction

A library is a vital institution in any educational or research environment, providing access to knowledge, information, and learning resources. Traditionally, library operations, such as managing books, recording issued and returned items, and maintaining inventories, were handled manually. Manual methods, however, are time-consuming, prone to human errors, and inefficient when dealing with a large number of books and members.

With the increasing volume of information and the growing need for efficiency, Library Management Systems (LMS) have become essential. An LMS is a software application designed to automate and streamline library operations, making the management of books, members, and transactions more accurate and efficient. It reduces manual work, ensures proper recordkeeping, and enhances both staff and member experiences.

The Library Management System allows library staff to manage book inventories, track issued and returned books, calculate fines, and generate reports. Members can search for books, check availability, and monitor their borrowed items. By providing a user-friendly interface and secure access, the system ensures smooth operations and improves overall library productivity.

Modern LMS solutions are built using technologies such as Java for the graphical user interface (GUI) and MySQL for database management, ensuring reliability, security, and scalability. Such systems not only simplify day-to-day operations but also support future enhancements, including digital book management, e-book issuance, and integration with online catalogs.

Additionally, the Library Management System contributes to better decision-making by providing detailed insights into library usage patterns. Administrators can monitor book circulation, identify popular titles, manage member activity, and forecast inventory needs effectively. This analytical capability helps optimize library resources, improve service quality, and ensure that the library continues to meet the evolving needs of its users.

Overall, the Library Management System is an effective solution that modernizes library operations, minimizes errors, saves time, and enhances the learning experience for both staff and library members.

Scope Of Project

The Library Management System (LMS) is designed to modernize and automate the operations of a library, addressing the challenges of traditional manual systems. The scope of this project covers multiple aspects of library management, ensuring efficiency, accuracy, and enhanced user experience for both staff and members.

1. **Automation of Library Operations**

   The LMS eliminates the need for manual record-keeping, reducing human errors and saving significant time. Tasks such as adding new books, updating records, tracking borrowed and returned books, and calculating fines are fully automated. This allows library staff to focus on more productive activities, such as assisting users and maintaining resources effectively.

2. **Comprehensive Book Management**

   The system facilitates efficient management of books, including addition, deletion, updating, and searching. Books are categorized by title, author, genre, or publication year, making it easier for users to locate resources. It also tracks multiple editions, copies, and availability, ensuring accurate inventory management.

3. **Efficient Member Management**

   Library members—students, faculty, and staff—can be registered and managed seamlessly. The system records member activity, borrowing history, fines, and members status, ensuring transparency and accountability while improving user experience.

4. **Streamlined Issue and Return Process**

   The LMS automates book issuance and return processes, including due date tracking and reminders for overdue books. Fines for late returns are calculated automatically, ensuring timely returns and optimal resource utilization.

5. **Reporting and Analytics**

   The system can generate detailed reports on issued and available books, overdue items, fines collected, and member activity. These analytical insights help administrators make informed decisions about acquiring new titles, identifying popular books, and optimizing library operations.

6. **User-Friendly Interface**

   A simple, interactive, and intuitive interface ensures that users of all technical skill levels can easily navigate the system. Minimal training is required, making the system practical for everyday use by staff and members.

7. **Data Security and Integrity**

   All records are securely stored in a MySQL relational database, ensuring protection against unauthorized access and maintaining data consistency. Role-based access control provides appropriate permissions to staff and members, enhancing overall system security.

8. **Scalability and Future Expansion**

The system is designed to support future enhancements and growth. Possible extensions include:

- Integration with digital libraries and e-book platforms.
- Online portals for book reservations and remote access.
- Mobile applications for on-the-go library access.
- Advanced reporting tools for predictive analysis of usage trends.
- Email or SMS notifications for overdue books, new arrivals, and reminders.

9. **Institutional and Educational Impact**

   Implementing this system improves operational efficiency, reduces administrative workload, and enhances the learning environment. Students and faculty benefit from faster access to resources, accurate tracking, and modernized library services.

10. **Overall Scope**

    The Library Management System is more than a tool for managing books—it serves as a complete digital framework for modern library operations. It ensures:

- Efficient management of library resources.
- Transparency in all transactions.
- Enhanced user satisfaction.
- Future readiness for digital and online integration.

Objectives Of Project

The primary objective of the Library Management System (LMS) is to automate and streamline the operations of a library to improve efficiency, accuracy, and service quality. The system aims to reduce manual work, ensure smooth data handling, and provide a user-friendly interface for both library staff and members.

The specific objectives of this project include:

1. **Efficient Book Management:**

   To maintain a comprehensive and well-organized database of all books available in the library. This includes detailed information such as book title, author, publisher, edition, category, publication year, and current availability status. The system should also allow easy addition, deletion, and modification of book records, ensuring that inventory is always up to date and accurately reflects the current collection.

2. **Member Management:**

   To register and manage all library members—students, faculty, and staff—by maintaining detailed profiles that include contact information, borrowing history, membership validity, and outstanding fines (if any). The system ensures accountability by keeping a complete record of each member's activities and facilitates quick access to member information when needed.

3. **Issue and Return Automation:**

   To simplify and automate the process of issuing and returning books. The system should automatically update inventory records, track due dates, and calculate fines for overdue returns. It will also generate receipts or confirmation messages for transactions, ensuring transparency and accuracy in all library operations.

4. **Search and Accessibility:**

   To provide users with a powerful and user-friendly search feature that allows quick access to book information. Members and staff can search for books by title, author, subject, or category, and instantly view details such as availability and location within the library. This feature reduces search time, enhances user satisfaction, and improves overall library accessibility.

5. **Report Generation:**

   To generate comprehensive and detailed reports for administrators and staff. These reports may include data on issued books, overdue items, popular titles, fine collections,

and member activity. Such analytical reports help in effective decision-making, resource allocation, and performance evaluation of the library's operations.

6. **Reduce Manual Effort:**

   To minimize dependency on manual recordkeeping and paperwork by automating all major library tasks. This significantly reduces the chances of human error, saves valuable time, and allows library staff to focus on more meaningful and productive activities such as user support, catalog expansion, and collection improvement.

7. **Secure Multi-user Access:**

   To allow multiple users—such as librarians, staff, and members—to access the system simultaneously with role-based permissions. This ensures secure data handling, maintaining confidentiality and integrity of library records.

8. **Scalability:**

   To develop a flexible and scalable system that can be enhanced in the future with advanced features such as digital book management, e-book issuance, mobile app integration, and online catalog access.

The overall goal of the project is to create a reliable, efficient, and user-friendly system that enhances library management, reduces manual workload, and provides an improved experience for both staff and members.

Methodology

The methodology of the Library Management System (LMS) defines the approach and techniques used to develop and implement the system effectively. The project follows a systematic software development process to ensure a robust, efficient, and user-friendly application. The key steps in the methodology are as follows:

1. **Requirement Analysis:**

   The first step involves understanding the requirements of the library, including book management, member registration, issue and return of books, fine calculation, and report generation. Interactions with library staff and study of existing manual processes help identify the system's functional and non-functional requirements.

2. **System Design:**

   In this phase, the architecture of the system is designed. The database structure, user interface, and system modules are planned. UML diagrams, flowcharts, and ER diagrams are created to visualize the system's workflow and ensure clarity in design.

3. **Database Development:**

   MySQL database is designed to store information about books, members, transactions, and fines. Tables, relationships, and constraints are defined to maintain data integrity and support efficient retrieval of information.

4. **GUI Development:**

   The graphical user interface (GUI) is developed using Java Swing. User-friendly forms, dropdowns, buttons, and tables are created for staff and members to interact with the system efficiently.

5. **Implementation of Functional Modules:**

   Core modules such as book management, member management, issue and return of books, fine calculation, search functionality, and report generation are implemented. The system is developed using Java with JDBC for database connectivity.

6. **Testing:**

   The system undergoes rigorous testing, including unit testing, integration testing, and system testing. Errors and bugs are identified and corrected to ensure smooth functionality.

7. **Deployment:**

   The final system is deployed in the library environment. Training is provided to staff for using the system, and user manuals are prepared for reference.

8. **Maintenance and Future Enhancement:**

   After deployment, the system is monitored for performance, and necessary updates are made. Future enhancements may include features like digital book management, e-book issuance, and online catalog integration.

This systematic methodology ensures that the Library Management System is reliable, efficient, and meets the operational requirements of modern libraries.

Technology Used

The Library Management System (LMS) is developed using a combination of modern technologies, programming languages, and database management tools to ensure a robust, efficient, and user-friendly application. Each technology has been carefully chosen to optimize the system's performance, security, and scalability. Below is a detailed description of the technologies used:

1. Programming Language – Java

- Purpose: Java is used to develop the main application logic and the graphical user interface (GUI).
- Advantages:
    - Platform-independent, allowing the system to run on any operating system (Windows, Linux, Mac).
    - Object-oriented, which helps in modular programming and easy maintenance.
    - Provides a rich set of libraries and tools for GUI development (Java Swing) and database connectivity (JDBC).
- Application in LMS: Java handles all system functionalities such as book management, member management, issue/return processes, fine calculation, and report generation.

2. Graphical User Interface – Java Swing

- Purpose: To create a user-friendly and interactive interface for both library staff and members.
- Advantages:
    - Provides ready-to-use components like buttons, tables, text fields, and panels.
    - Allows customization of the interface for better usability.
    - Enhances user experience by making the system easy to navigate.
- Application in LMS: Swing components are used to design forms for adding/updating books, registering members, issuing/returning books, and viewing reports.

3. Database Management – MySQL

- Purpose: To store, manage, and retrieve all data related to books, members, and transactions securely.

- Advantages:
    - Open-source and widely used relational database management system.
    - Supports complex queries, data integrity, and multi-user access.
    - Provides secure storage of sensitive information with features like access control and backup support.
- Application in LMS:
    - Stores tables for Books, Members, Transactions, and Fines.
    - Maintains relationships between tables using primary keys and foreign keys.
    - Enables efficient data retrieval for searching, reporting, and updating records.

4. JDBC (Java Database Connectivity)

- Purpose: To connect the Java application with the MySQL database seamlessly.
- Advantages:
    - Facilitates communication between the application and the database.
    - Supports executing SQL queries and retrieving results efficiently.
    - Provides error handling mechanisms for database operations.
- Application in LMS: JDBC is used for executing queries related to adding, updating, deleting, and searching records in the database.

5. IDE – NetBeans / Eclipse

- Purpose: To write, compile, and debug Java code effectively.
- Advantages:
    - Provides an integrated environment for coding, GUI design, and database connectivity.
    - Offers debugging tools, code suggestions, and error highlighting to improve productivity.
- Application in LMS: The IDE is used to develop the entire application, design GUI forms, and integrate database operations.

6. Operating System – Cross-Platform Support

- The system can run on Windows, Linux, or Mac operating systems, thanks to Java's platform independence.
- Ensures that the LMS can be implemented in different institutional environments without compatibility issues.

7. Reporting and Analytics Tools (Optional Future Use)

- Reports can be generated using Java inbuilt features or integrated with external tools for better visualization.
- Enables library staff to analyze book circulation, popular books, overdue trends, and member activity patterns.

Source Code

LoginGUI.java

```
import javax.swing.*;
import java.awt.event.*;
public class LoginGUI extends JFrame implements ActionListener {
   JTextField txtUsername = new JTextField();
   JPasswordField txtPassword = new JPasswordField();
   JButton btnLogin = new JButton("Login");
   public LoginGUI() {
      setTitle("Login");
      setSize(500, 350);
      setLayout(null);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      setLocationRelativeTo(null);
      JLabel l1 = new JLabel("Username:");
      JLabel l2 = new JLabel("Password:");
      l1.setBounds(30, 30, 80, 30);
      l2.setBounds(30, 70, 80, 30);
      txtUsername.setBounds(120, 30, 180, 30);
      txtPassword.setBounds(120, 70, 180, 30);
      btnLogin.setBounds(120, 120, 100, 30);
      add(l1); add(l2); add(txtUsername); add(txtPassword); add(btnLogin);
      btnLogin.addActionListener(this);    }
   @Override
   public void actionPerformed(ActionEvent e) {
      if (e.getSource() == btnLogin) {
         String username = txtUsername.getText().trim();
         String password = new String(txtPassword.getPassword());
         if (username.equals("admin") && password.equals("admin123")) {
            JOptionPane.showMessageDialog(this, "Login Successful!");
            this.dispose(); // close login window
            new MainGUI().setVisible(true); // open dashboard
```

```
        } else {
            JOptionPane.showMessageDialog(this, "Invalid Username or Password!");
        } } }}
```

StudentGUI.java

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.sql.*;
public class StudentGUI extends JFrame {
    JTextField txtId = new JTextField(), txtName = new JTextField(), txtClass = new
JTextField(), txtEmail = new JTextField(), txtMobile = new JTextField();
    JButton btnAdd = new JButton("Add"), btnUpdate = new JButton("Update"), btnDelete =
new JButton("Delete"), btnSearch = new JButton("Search");
    JTable table;
    DefaultTableModel tableModel;
    public StudentGUI() {
        setTitle("Manage Students");
        setSize(700, 500);
        setLayout(null);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        JLabel l1 = new JLabel("ID:"), l2 = new JLabel("Name:"), l3 = new JLabel("Class:"), l4
= new JLabel("Email:"), l5 = new JLabel("Mobile:");
        l1.setBounds(50, 30, 100, 30); txtId.setBounds(150, 30, 200, 30);
        l2.setBounds(50, 70, 100, 30); txtName.setBounds(150, 70, 200, 30);
        l3.setBounds(50, 110, 100, 30); txtClass.setBounds(150, 110, 200, 30);
        l4.setBounds(50, 150, 100, 30); txtEmail.setBounds(150, 150, 200, 30);
        l5.setBounds(50, 190, 100, 30); txtMobile.setBounds(150, 190, 200, 30);
        btnAdd.setBounds(50, 230, 80, 30);
        btnUpdate.setBounds(150, 230, 80, 30);
        btnDelete.setBounds(250, 230, 80, 30);
        btnSearch.setBounds(350, 30, 80, 30);
        add(l1); add(txtId); add(l2); add(txtName); add(l3); add(txtClass);
        add(l4); add(txtEmail); add(l5); add(txtMobile);
        add(btnAdd); add(btnUpdate); add(btnDelete); add(btnSearch);
```

```
        tableModel = new DefaultTableModel(new String[]{"ID", "Name", "Class", "Email",
"Mobile"}, 0);
        table = new JTable(tableModel);
        JScrollPane scroll = new JScrollPane(table);
        scroll.setBounds(50, 280, 600, 160);
        add(scroll);
        loadTable();
        btnAdd.addActionListener(_ -> { addStudent(); loadTable(); });
        btnUpdate.addActionListener(_ -> { updateStudent(); loadTable(); });
        btnDelete.addActionListener(_ -> { deleteStudent(); loadTable(); });
        btnSearch.addActionListener(_ -> searchStudent());  }
    private void addStudent() {
        try {
            // Email ko bhi required field banaya gaya hai
            if (txtId.getText().trim().isEmpty() || txtName.getText().trim().isEmpty()
                    || txtClass.getText().trim().isEmpty() || txtEmail.getText().trim().isEmpty()
                    || txtMobile.getText().trim().isEmpty()) {
                JOptionPane.showMessageDialog(this, "ID, Name, Class, Email, and Mobile are
required!");
                return;            }
            int id = Integer.parseInt(txtId.getText().trim());
            String email = txtEmail.getText().trim();
            String mobile = txtMobile.getText().trim();
            if (!mobile.matches("\\d{10}")) {
                JOptionPane.showMessageDialog(this, "Mobile number must be 10 digits!");
                return;            }
            // Simple email validation
            if (!email.matches("^[\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$")) {
                JOptionPane.showMessageDialog(this, "Enter a valid email address!");
                return;            }
            Connection c = Connect.ConnectToDB();
     PreparedStatement check = c.prepareStatement("SELECT id FROM student WHERE id=?");
            check.setInt(1, id);
```

```
            ResultSet rs = check.executeQuery();
            if (rs.next()) {
                JOptionPane.showMessageDialog(this, "ID already exists!");
                return;            }
            PreparedStatement pst = c.prepareStatement("INSERT INTO
student(id,name,class,email,mobile) VALUES(?,?,?,?,?)");
            pst.setInt(1, id);
            pst.setString(2, txtName.getText().trim());
            pst.setString(3, txtClass.getText().trim());
            pst.setString(4, email);
            pst.setString(5, mobile);
            pst.executeUpdate();
            JOptionPane.showMessageDialog(this, "Student Added Successfully!");
            clearFields();
        } catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
        }    }
    private void updateStudent() {
        try {
            if (txtId.getText().trim().isEmpty() || txtName.getText().trim().isEmpty()
                    || txtClass.getText().trim().isEmpty() || txtEmail.getText().trim().isEmpty()
                    || txtMobile.getText().trim().isEmpty()) {
                JOptionPane.showMessageDialog(this, "ID, Name, Class, Email, and Mobile are
required!");
                return;            }
            int id = Integer.parseInt(txtId.getText().trim());
            String email = txtEmail.getText().trim();
            String mobile = txtMobile.getText().trim();
            if (!mobile.matches("\\d{10}")) {
                JOptionPane.showMessageDialog(this, "Mobile number must be 10 digits!");
                return;            }
            if (!email.matches("^[\\w-\\.]+@([\\w-]+\\.)+[\\w-]{2,4}$")) {
```

```
            JOptionPane.showMessageDialog(this, "Enter a valid email address!");
            return;          }
        Connection c = Connect.ConnectToDB();
        PreparedStatement pst = c.prepareStatement("UPDATE student SET name=?, class=?,
email=?, mobile=? WHERE id=?");
        pst.setString(1, txtName.getText().trim());
        pst.setString(2, txtClass.getText().trim());
        pst.setString(3, email);
        pst.setString(4, mobile);
        pst.setInt(5, id);
        int updated = pst.executeUpdate();
        if (updated > 0) {
            JOptionPane.showMessageDialog(this, "Student Updated Successfully!");
            clearFields();
        } else {
            JOptionPane.showMessageDialog(this, "No student found with that ID!");     }
    } catch (Exception ex) {
        ex.printStackTrace();
        JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
    }   }
  private void deleteStudent() {
    try {
        int id = Integer.parseInt(txtId.getText().trim());
        Connection c = Connect.ConnectToDB();
     PreparedStatement pst = c.prepareStatement("DELETE FROM student WHERE id=?");
        pst.setInt(1, id);
        int deleted = pst.executeUpdate();
        if (deleted > 0) {
            JOptionPane.showMessageDialog(this, "Student Deleted Successfully!");
            clearFields();
        } else {
            JOptionPane.showMessageDialog(this, "No student found with that ID!");
        }
```

```
    } catch (Exception ex) {
        ex.printStackTrace();
        JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
    }    }
private void searchStudent() {
    try {
        String idText = txtId.getText().trim();
        String nameText = txtName.getText().trim();
        if (idText.isEmpty() && nameText.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Enter Student ID or Name to search!");
            return;        }
        Connection c = Connect.ConnectToDB();
        PreparedStatement pst;
        if (!idText.isEmpty()) {
            pst = c.prepareStatement("SELECT * FROM student WHERE id=?");
            pst.setInt(1, Integer.parseInt(idText));
        } else {
            pst = c.prepareStatement("SELECT * FROM student WHERE name LIKE ?");
            pst.setString(1, "%" + nameText + "%");
        }
        ResultSet rs = pst.executeQuery();
        if (rs.next()) {
            txtId.setText(rs.getString("id"));
            txtName.setText(rs.getString("name"));
            txtClass.setText(rs.getString("class"));
            txtEmail.setText(rs.getString("email"));
            txtMobile.setText(rs.getString("mobile"));
        } else {
            JOptionPane.showMessageDialog(this, "Student not found!");         }
    } catch (Exception ex) {
        ex.printStackTrace();
        JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
    }    }
```

```
    private void loadTable() {
        try {
            tableModel.setRowCount(0);
            Connection c = Connect.ConnectToDB();
            PreparedStatement pst = c.prepareStatement("SELECT * FROM student");
            ResultSet rs = pst.executeQuery();
            while (rs.next()) {
                tableModel.addRow(new Object[]{
                        rs.getInt("id"),
                        rs.getString("name"),
                        rs.getString("class"),
                        rs.getString("email"),
                        rs.getString("mobile")
                });          }
        } catch (Exception ex) {
            ex.printStackTrace();
        }    }
    private void clearFields() {
        txtId.setText("");
        txtName.setText("");
        txtClass.setText("");
        txtEmail.setText("");
        txtMobile.setText("");
    }
public static void main(String[] args) {
        new StudentGUI().setVisible(true);
    }}
```

BookingGUI.java

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.sql.*;
public class BookGUI extends JFrame {
```

```
    JTextField txtId = new JTextField(), txtTitle = new JTextField(), txtAuthor = new
JTextField(), txtYear = new JTextField(), txtCopies = new JTextField();

    JButton btnAdd = new JButton("Add"), btnUpdate = new JButton("Update"), btnDelete =
new JButton("Delete"), btnSearch = new JButton("Search"), btnRefresh = new
JButton("Refresh");

    JTable table;

    DefaultTableModel model;

    public BookGUI() {

        setTitle("Manage Books");

        setSize(700, 600);

        setLayout(null);

        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        JLabel l1 = new JLabel("Book ID:"), l2 = new JLabel("Title:"), l3 = new
JLabel("Author:"), l4 = new JLabel("Year:"), l5 = new JLabel("Copies:");

        l1.setBounds(50, 30, 100, 30); txtId.setBounds(150, 30, 200, 30);

        l2.setBounds(50, 70, 100, 30); txtTitle.setBounds(150, 70, 200, 30);

        l3.setBounds(50, 110, 100, 30); txtAuthor.setBounds(150, 110, 200, 30);

        l4.setBounds(50, 150, 100, 30); txtYear.setBounds(150, 150, 200, 30);

        l5.setBounds(50, 190, 100, 30); txtCopies.setBounds(150, 190, 200, 30);

        btnAdd.setBounds(50, 240, 80, 30);

        btnUpdate.setBounds(150, 240, 80, 30);

        btnDelete.setBounds(250, 240, 80, 30);

        btnSearch.setBounds(370, 30, 80, 30);

        btnRefresh.setBounds(480, 30, 100, 30);

        add(l1); add(txtId); add(l2); add(txtTitle); add(l3); add(txtAuthor);

        add(l4); add(txtYear); add(l5); add(txtCopies);

        add(btnAdd); add(btnUpdate); add(btnDelete); add(btnSearch); add(btnRefresh);

        // Table for live data

        model = new DefaultTableModel(new String[]{"Book ID", "Title", "Author", "Year",
"Copies", "Status"}, 0);

        table = new JTable(model);

        JScrollPane sp = new JScrollPane(table);

        sp.setBounds(50, 300, 600, 220);

        add(sp);

        // Load initial data
```

```
    loadBooks();
    // Button Actions
    btnAdd.addActionListener(_-> addBook());
    btnUpdate.addActionListener(_-> updateBook());
    btnDelete.addActionListener(_ -> deleteBook());
    btnSearch.addActionListener(_-> searchBook());
    btnRefresh.addActionListener(_-> loadBooks());
    // Table click to fill text fields
    table.addMouseListener(new java.awt.event.MouseAdapter() {
        public void mouseClicked(java.awt.event.MouseEvent evt) {
            int i = table.getSelectedRow();
            txtId.setText(model.getValueAt(i, 0).toString());
            txtTitle.setText(model.getValueAt(i, 1).toString());
            txtAuthor.setText(model.getValueAt(i, 2).toString());
            txtYear.setText(model.getValueAt(i, 3).toString());
            txtCopies.setText(model.getValueAt(i, 4).toString());
        }     });
    setVisible(true);    }
// Load all books into the JTable
private void loadBooks() {
    try {
        Connection c = Connect.ConnectToDB();
        Statement st = c.createStatement();
        ResultSet rs = st.executeQuery("SELECT * FROM book");
        model.setRowCount(0); // Clear previous data
        while (rs.next()) {
            model.addRow(new Object[]{
                    rs.getString("id"),
                    rs.getString("title"),
                    rs.getString("author"),
                    rs.getString("year"),
                    rs.getString("copies"),
                    rs.getString("status")
```

```
            });          }
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Error loading data: " +
ex.getMessage());   }    }
    private void addBook() {
        try {
            if (txtId.getText().trim().isEmpty() || txtTitle.getText().trim().isEmpty()
                || txtAuthor.getText().trim().isEmpty() || txtYear.getText().trim().isEmpty()
                || txtCopies.getText().trim().isEmpty()) {
                JOptionPane.showMessageDialog(this, "All fields are required!");
                return;      }
            Connection c = Connect.ConnectToDB();
            PreparedStatement check = c.prepareStatement("SELECT * FROM book WHERE
id=?");
            check.setString(1, txtId.getText().trim());
            ResultSet rs = check.executeQuery();
            if (rs.next()) {
                JOptionPane.showMessageDialog(this, "Book ID already exists!");
                return;      }
            PreparedStatement pst = c.prepareStatement("INSERT INTO book(id, title, author,
year, copies, status) VALUES(?,?,?,?,?, 'Available')");
            pst.setString(1, txtId.getText().trim());
            pst.setString(2, txtTitle.getText().trim());
            pst.setString(3, txtAuthor.getText().trim());
            pst.setInt(4, Integer.parseInt(txtYear.getText().trim()));
            pst.setInt(5, Integer.parseInt(txtCopies.getText().trim()));
            pst.executeUpdate();
            JOptionPane.showMessageDialog(this, "Book Added Successfully!");
            clearFields();
            loadBooks(); // Auto refresh
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
        } }
    private void updateBook() {
```

```
        try {
            if (txtId.getText().trim().isEmpty()) {
                JOptionPane.showMessageDialog(this, "Enter Book ID to update!");
                return;   }
            // ☑ Validate all fields before updating
            if (txtTitle.getText().trim().isEmpty() || txtAuthor.getText().trim().isEmpty() ||
                txtYear.getText().trim().isEmpty() || txtCopies.getText().trim().isEmpty()) {
                JOptionPane.showMessageDialog(this, "Please fill all input fields!");
                return;      }
            Connection c = Connect.ConnectToDB();
            PreparedStatement pst = c.prepareStatement("UPDATE book SET title=?, author=?,
year=?, copies=? WHERE id=?");
            pst.setString(1, txtTitle.getText().trim());
            pst.setString(2, txtAuthor.getText().trim());
            pst.setInt(3, Integer.parseInt(txtYear.getText().trim()));
            pst.setInt(4, Integer.parseInt(txtCopies.getText().trim()));
            pst.setString(5, txtId.getText().trim());
            int updated = pst.executeUpdate();
            if (updated > 0)
                JOptionPane.showMessageDialog(this, "Book Updated Successfully!");
            else
                JOptionPane.showMessageDialog(this, "Book ID not found!");
            clearFields();
            loadBooks(); //  Auto refresh after update
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
        } }
    private void deleteBook() {
        try {
            if (txtId.getText().trim().isEmpty()) {
                JOptionPane.showMessageDialog(this, "Enter Book ID to delete!");
                return;    }
            Connection c = Connect.ConnectToDB();
```

```
PreparedStatement pst = c.prepareStatement("DELETE FROM book WHERE id=?");
        pst.setString(1, txtId.getText().trim());
        int deleted = pst.executeUpdate();
        if (deleted > 0)
            JOptionPane.showMessageDialog(this, "Book Deleted Successfully!");
        else
            JOptionPane.showMessageDialog(this, "Book ID not found!");
        clearFields();
        loadBooks(); //  Auto refresh after delete
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
    }}
  private void searchBook() {
    try {
        String id = txtId.getText().trim();
        String title = txtTitle.getText().trim();
        if (id.isEmpty() && title.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Enter Book ID or Title to search!");
            return;    }
        Connection c = Connect.ConnectToDB();
        PreparedStatement pst;
        if (!id.isEmpty()) {
            pst = c.prepareStatement("SELECT * FROM book WHERE id=?");
            pst.setString(1, id);
        } else {
            pst = c.prepareStatement("SELECT * FROM book WHERE title LIKE ?");
            pst.setString(1, "%" + title + "%");      }
        ResultSet rs = pst.executeQuery();
        model.setRowCount(0);
        while (rs.next()) {
            model.addRow(new Object[]{
                    rs.getString("id"),
                    rs.getString("title"),
```

```
                rs.getString("author"),
                rs.getString("year"),
                rs.getString("copies"),
                rs.getString("status")
            });   }
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Error: " + ex.getMessage());
        }    }
    private void clearFields() {
        txtId.setText("");
        txtTitle.setText("");
        txtAuthor.setText("");
        txtYear.setText("");
        txtCopies.setText("");  }
    public static void main(String[] args) {
        new BookGUI(); }}
```

IssueReturnGUI.java

```
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.sql.*;
import java.text.SimpleDateFormat;
import java.util.Date;
public class IssueReturnGUI extends JFrame {
    JComboBox<String> comboBookId = new JComboBox<>();
    JComboBox<String> comboStudentId = new JComboBox<>();
    JTextField txtTitle = new JTextField(), txtAuthor = new JTextField(), txtYear = new
JTextField();
    JTextField txtCopies = new JTextField(), txtAvailable = new JTextField();
    JTextField txtStudentName = new JTextField(), txtStudentClass = new JTextField(),
txtStudentMobile = new JTextField();
    JTextField txtIssueDate = new JTextField(), txtDueDate = new JTextField();
    JButton btnIssue = new JButton("Issue Book");
    JButton btnReturn = new JButton("Return Book");
```

```
    JTable table;
    DefaultTableModel tableModel;
    public IssueReturnGUI() {
        setTitle("Issue / Return Books");
        setSize(950, 700);
        setLayout(null);
        // ===== BOOK SECTION =====
        JLabel lbBook = new JLabel("BOOK DETAILS");
        lbBook.setBounds(50, 10, 200, 25);
        add(lbBook);
        JLabel l1 = new JLabel("Book ID:"), l2 = new JLabel("Title:"), l3 = new
JLabel("Author:"),
        l4 = new JLabel("Year:"), l5 = new JLabel("Total Copies:"), l6 = new
JLabel("Available:");
        l1.setBounds(50,40,100,25); comboBookId.setBounds(150,40,150,25);
        l2.setBounds(50,70,100,25); txtTitle.setBounds(150,70,200,25);
        l3.setBounds(50,100,100,25); txtAuthor.setBounds(150,100,200,25);
        l4.setBounds(50,130,100,25); txtYear.setBounds(150,130,200,25);
        l5.setBounds(50,160,100,25); txtCopies.setBounds(150,160,200,25);
        l6.setBounds(50,190,100,25); txtAvailable.setBounds(150,190,200,25);
        add(l1); add(comboBookId); add(l2); add(txtTitle); add(l3); add(txtAuthor);
        add(l4); add(txtYear); add(l5); add(txtCopies); add(l6); add(txtAvailable);
        txtTitle.setEditable(false); txtAuthor.setEditable(false); txtYear.setEditable(false);
        txtCopies.setEditable(false); txtAvailable.setEditable(false);
        // ===== STUDENT SECTION =====
        JLabel lbStudent = new JLabel("STUDENT DETAILS");
        lbStudent.setBounds(50,240,200,25); add(lbStudent);
        JLabel s1 = new JLabel("Student ID:"), s2 = new JLabel("Name:"), s3 = new
JLabel("Class:"), s4 = new JLabel("Mobile No:");
        s1.setBounds(50,270,100,25); comboStudentId.setBounds(150,270,150,25);
        s2.setBounds(50,300,100,25); txtStudentName.setBounds(150,300,200,25);
        s3.setBounds(50,330,100,25); txtStudentClass.setBounds(150,330,200,25);
        s4.setBounds(50,360,100,25); txtStudentMobile.setBounds(150,360,200,25);
```

```
        add(s1); add(comboStudentId); add(s2); add(txtStudentName); add(s3);
add(txtStudentClass); add(s4); add(txtStudentMobile);

        txtStudentName.setEditable(false); txtStudentClass.setEditable(false);
txtStudentMobile.setEditable(false);

        // ===== ISSUE SECTION =====

        JLabel lbIssue = new JLabel("ISSUE DETAILS");

        lbIssue.setBounds(50,410,200,25); add(lbIssue);

        JLabel l3a = new JLabel("Issue Date:"), l4a = new JLabel("Due Date:");

        l3a.setBounds(50,440,100,25); txtIssueDate.setBounds(150,440,150,25);

        l4a.setBounds(50,470,100,25); txtDueDate.setBounds(150,470,150,25);

        add(l3a); add(txtIssueDate); add(l4a); add(txtDueDate);

        txtIssueDate.setEditable(false);

        btnIssue.setBounds(100,520,150,30);

        btnReturn.setBounds(300,520,150,30);

        add(btnIssue); add(btnReturn);

        SimpleDateFormat inputFormat = new SimpleDateFormat("dd/MM/yyyy");

        txtIssueDate.setText(inputFormat.format(new Date()));

        Date due = new Date(System.currentTimeMillis() + 7L*24*60*60*1000);

        txtDueDate.setText(inputFormat.format(due));

        // ===== TABLE =====

        tableModel = new DefaultTableModel(new String[]{

              "Book ID","Title","Student ID","Student Name","Issue Date","Due Date","Return
Date","Status"},0);

        table = new JTable(tableModel);

        JScrollPane scroll = new JScrollPane(table);

        scroll.setBounds(400,40,520,500); add(scroll);

        loadBookIds();

        loadStudentIds();

        comboBookId.addActionListener(_ -> loadBookDetails());

        comboStudentId.addActionListener(_-> loadStudentDetails());

        btnIssue.addActionListener(_-> { issueBook(); loadIssuedTable(); loadBookDetails(); });

        btnReturn.addActionListener(_-> { returnBook(); loadIssuedTable(); loadBookDetails();
});

        loadIssuedTable();
```

```
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);    }
    private void loadBookIds() {
        try (Connection c = Connect.ConnectToDB()) {
            PreparedStatement pst = c.prepareStatement("SELECT id FROM book");
            ResultSet rs = pst.executeQuery();
            comboBookId.removeAllItems();
            while(rs.next()) comboBookId.addItem(rs.getString("id"));
        } catch(SQLException ex) { ex.printStackTrace(); }    }
    private void loadStudentIds() {
        try (Connection c = Connect.ConnectToDB()) {
            PreparedStatement pst = c.prepareStatement("SELECT id FROM student");
            ResultSet rs = pst.executeQuery();
            comboStudentId.removeAllItems();
            while(rs.next()) comboStudentId.addItem(rs.getString("id"));
        } catch(SQLException ex) { ex.printStackTrace(); }  }
    private void loadBookDetails() {
        if(comboBookId.getSelectedItem()==null) return;
        try (Connection c = Connect.ConnectToDB()) {
         PreparedStatement pst = c.prepareStatement("SELECT * FROM book WHERE id=?");
            pst.setString(1,comboBookId.getSelectedItem().toString());
            ResultSet rs = pst.executeQuery();
            if(rs.next()) {
                txtTitle.setText(rs.getString("title"));
                txtAuthor.setText(rs.getString("author"));
                txtYear.setText(rs.getString("year"));
                txtCopies.setText(rs.getString("copies"));
                // Calculate available copies
                PreparedStatement pst2 = c.prepareStatement("SELECT COUNT(*) AS
issuedCount FROM issued_books WHERE bookid=? AND status='Issued'");
                pst2.setString(1, comboBookId.getSelectedItem().toString());
                ResultSet rs2 = pst2.executeQuery();
                int issuedCount = 0;
```

```
            if(rs2.next()) issuedCount = rs2.getInt("issuedCount");
            int available = rs.getInt("copies") - issuedCount;
            txtAvailable.setText(String.valueOf(available));      }
        } catch(SQLException ex){ ex.printStackTrace(); }    }
    private void loadStudentDetails() {
        if(comboStudentId.getSelectedItem()==null) return;
        try (Connection c = Connect.ConnectToDB()) {
       PreparedStatement pst = c.prepareStatement("SELECT * FROM student WHERE id=?");
            pst.setString(1, comboStudentId.getSelectedItem().toString());
            ResultSet rs = pst.executeQuery();
            if(rs.next()){
                txtStudentName.setText(rs.getString("name"));
                txtStudentClass.setText(rs.getString("class"));
                txtStudentMobile.setText(rs.getString("mobile"));            }
        } catch(SQLException ex){ ex.printStackTrace(); }
    }
    private void issueBook() {
        if(comboBookId.getSelectedItem()==null || comboStudentId.getSelectedItem()==null){
            JOptionPane.showMessageDialog(this,"Select Book ID and Student ID first!");
            return;        }
        try (Connection c = Connect.ConnectToDB()) {
            // Check if copies are available
            PreparedStatement pst2 = c.prepareStatement("SELECT COUNT(*) AS issuedCount
FROM issued_books WHERE bookid=? AND status='Issued'");
            pst2.setString(1, comboBookId.getSelectedItem().toString());
            ResultSet rs2 = pst2.executeQuery();
            int issuedCount = 0;
            if(rs2.next()) issuedCount = rs2.getInt("issuedCount");
            PreparedStatement pstBook = c.prepareStatement("SELECT copies FROM book
WHERE id=?");
            pstBook.setString(1, comboBookId.getSelectedItem().toString());
            ResultSet rsBook = pstBook.executeQuery();
            int totalCopies = 0;
```

```
            if(rsBook.next()) totalCopies = rsBook.getInt("copies");
            if(issuedCount >= totalCopies) {
               JOptionPane.showMessageDialog(this,"No available copies to issue!");
               return;       }
            SimpleDateFormat inputFormat = new SimpleDateFormat("dd/MM/yyyy");
            SimpleDateFormat mysqlFormat = new SimpleDateFormat("yyyy-MM-dd");
            String issueDate = mysqlFormat.format(inputFormat.parse(txtIssueDate.getText()));
            String dueDate = mysqlFormat.format(inputFormat.parse(txtDueDate.getText()));
            PreparedStatement pst3 = c.prepareStatement(
               "INSERT INTO issued_books (bookid, studentid, issue_date, due_date, status)
VALUES (?, ?, ?, ?, 'Issued')");
            pst3.setInt(1, Integer.parseInt(comboBookId.getSelectedItem().toString()));
            pst3.setInt(2, Integer.parseInt(comboStudentId.getSelectedItem().toString()));
            pst3.setString(3, issueDate);
            pst3.setString(4, dueDate);
            pst3.executeUpdate();
            JOptionPane.showMessageDialog(this," ▧ Book Issued Successfully!");
            loadBookDetails();
            loadIssuedTable();
         } catch(Exception ex){ ex.printStackTrace(); }    }
      private void returnBook() {
      if(comboBookId.getSelectedItem() == null || comboStudentId.getSelectedItem() == null){
         JOptionPane.showMessageDialog(this,"Select Book ID and Student ID!");
         return;    }
      try (Connection c = Connect.ConnectToDB()) {
         PreparedStatement pst = c.prepareStatement(
            "DELETE FROM issued_books WHERE bookid=? AND studentid=? LIMIT 1"
         );
         pst.setInt(1, Integer.parseInt(comboBookId.getSelectedItem().toString()));
         pst.setInt(2, Integer.parseInt(comboStudentId.getSelectedItem().toString()));
         int rows = pst.executeUpdate();
         if(rows > 0)
```

```
            JOptionPane.showMessageDialog(this,"☑ Book Returned and removed from
database!");
        else
            JOptionPane.showMessageDialog(this,"⚠ No active issue found for this student!");
        loadBookDetails();
        loadIssuedTable();
    } catch(Exception ex){ ex.printStackTrace(); }}
    private void loadIssuedTable() {
        try (Connection c = Connect.ConnectToDB()) {
            tableModel.setRowCount(0);
            String query = """
    SELECT i.bookid, b.title, i.studentid, s.name, i.issue_date, i.due_date, i.return_date, i.status
                FROM issued_books i
                JOIN book b ON i.bookid = b.id
                JOIN student s ON i.studentid = s.id
                ORDER BY i.id DESC
            """;
            PreparedStatement pst = c.prepareStatement(query);
            ResultSet rs = pst.executeQuery();
            while(rs.next()){
                tableModel.addRow(new Object[]{
                    rs.getString("bookid"),
                    rs.getString("title"),
                    rs.getString("studentid"),
                    rs.getString("name"),
                    rs.getString("issue_date"),
                    rs.getString("due_date"),
                    rs.getString("return_date") == null ? "" : rs.getString("return_date"),
                    rs.getString("status")
                });     }
        } catch(Exception ex){ ex.printStackTrace(); }  }
    public static void main(String[] args){
        new IssueReturnGUI();  }}
```

Connect.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.util.logging.Level;
import java.util.logging.Logger;
public class Connect {
   private static Connection con = null;
   public static Connection ConnectToDB() {
      try {
         // Agar connection null ya closed hai, naya connection banao
         if (con == null || con.isClosed()) {
            String url =
"jdbc:mysql://localhost:3306/library?useSSL=false&allowPublicKeyRetrieval=true&serverTi
mezone=UTC";
            String user = "admin";
            String password = "1234";
            con = DriverManager.getConnection(url, user, password);       }
      } catch (SQLException ex) {
         Logger.getLogger(Connect.class.getName()).log(Level.SEVERE, null, ex);        }
      return con;  }}
```

MainGUI.java

```
import javax.swing.*;
public class MainGUI extends JFrame {
   JButton btnStudent = new JButton("Student");
   JButton btnBook = new JButton("Book");
   JButton btnIssue = new JButton("Issue/Return");
   public MainGUI() {
      setTitle("Library Dashboard");
      setSize(400, 300);
      setLayout(null);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      setLocationRelativeTo(null);
```

```
    btnStudent.setBounds(50, 50, 300, 40);
    btnBook.setBounds(50, 120, 300, 40);
    btnIssue.setBounds(50, 190, 300, 40);
    add(btnStudent);
    add(btnBook);
    add(btnIssue);
    btnStudent.addActionListener(_ -> new StudentGUI().setVisible(true));
    btnBook.addActionListener(_ -> new BookGUI().setVisible(true));
    btnIssue.addActionListener(_ -> new IssueReturnGUI().setVisible(true));  }
public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        // Show login page first
        new LoginGUI().setVisible(true);
    });   }}
```

Screenshorts

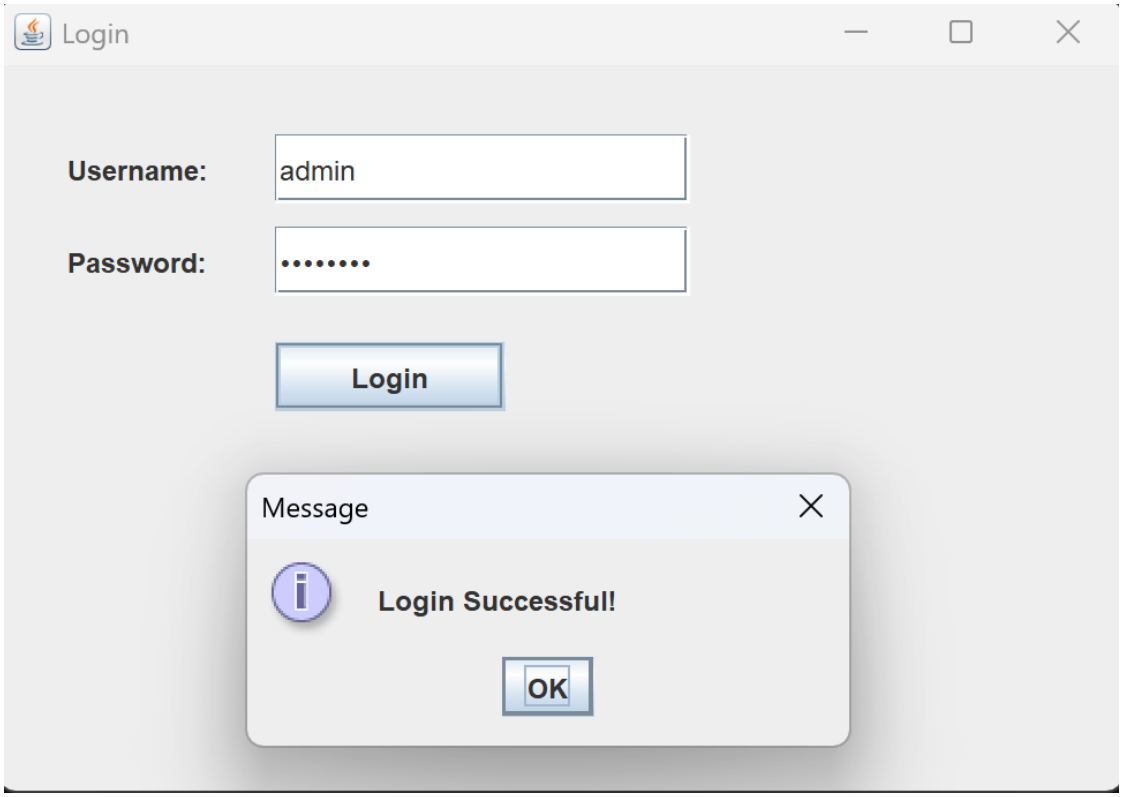

Fig no:-1 Login page

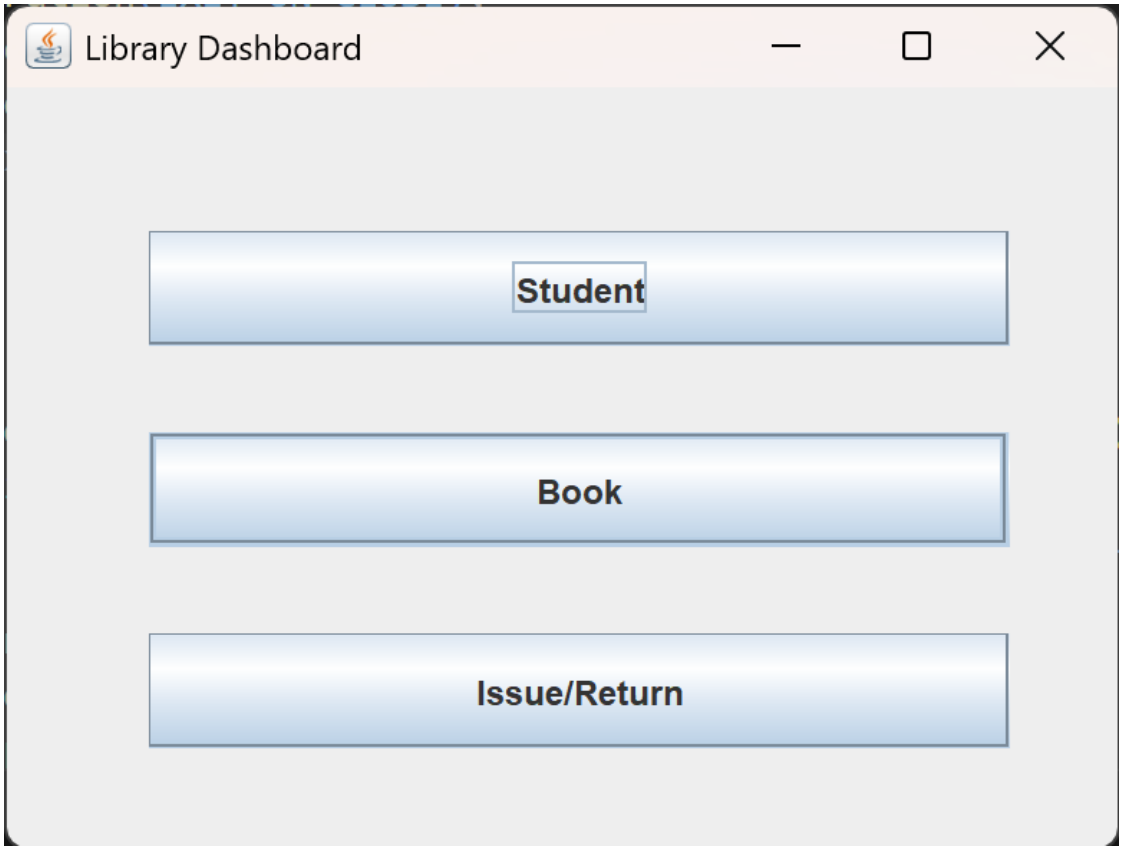

Fig no:-2 Home page

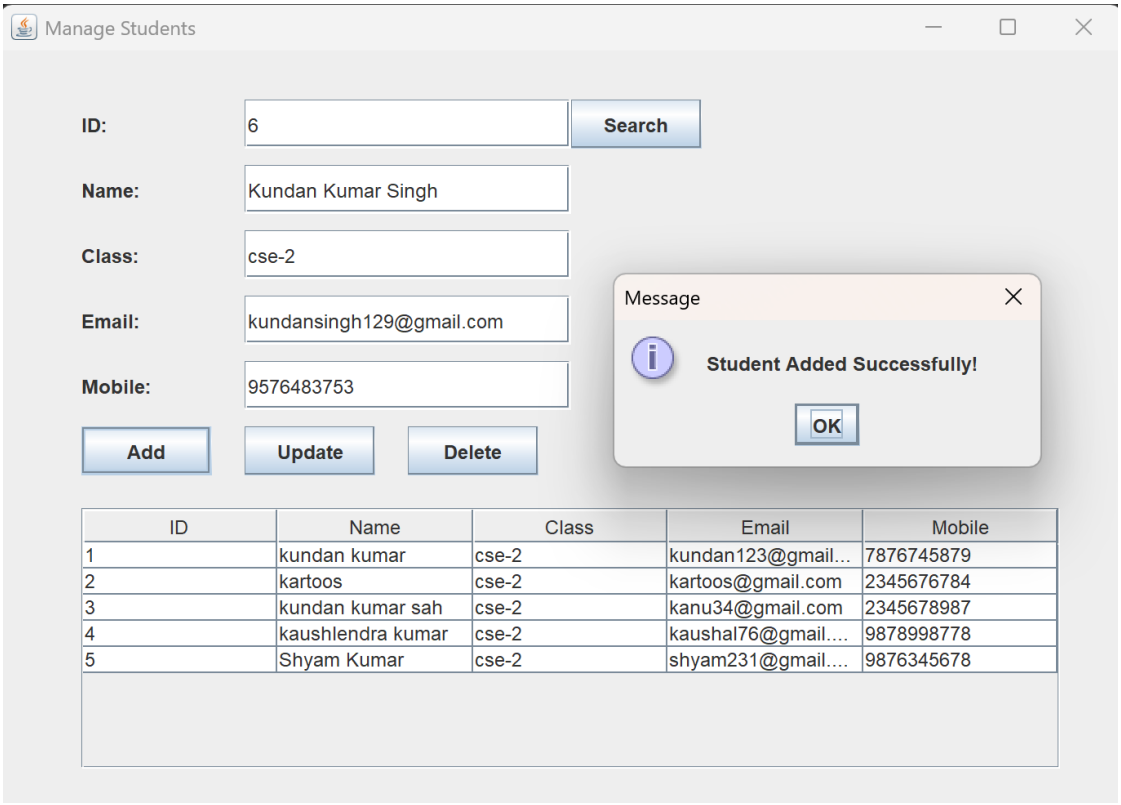

Fig:-3 Student Dashboard

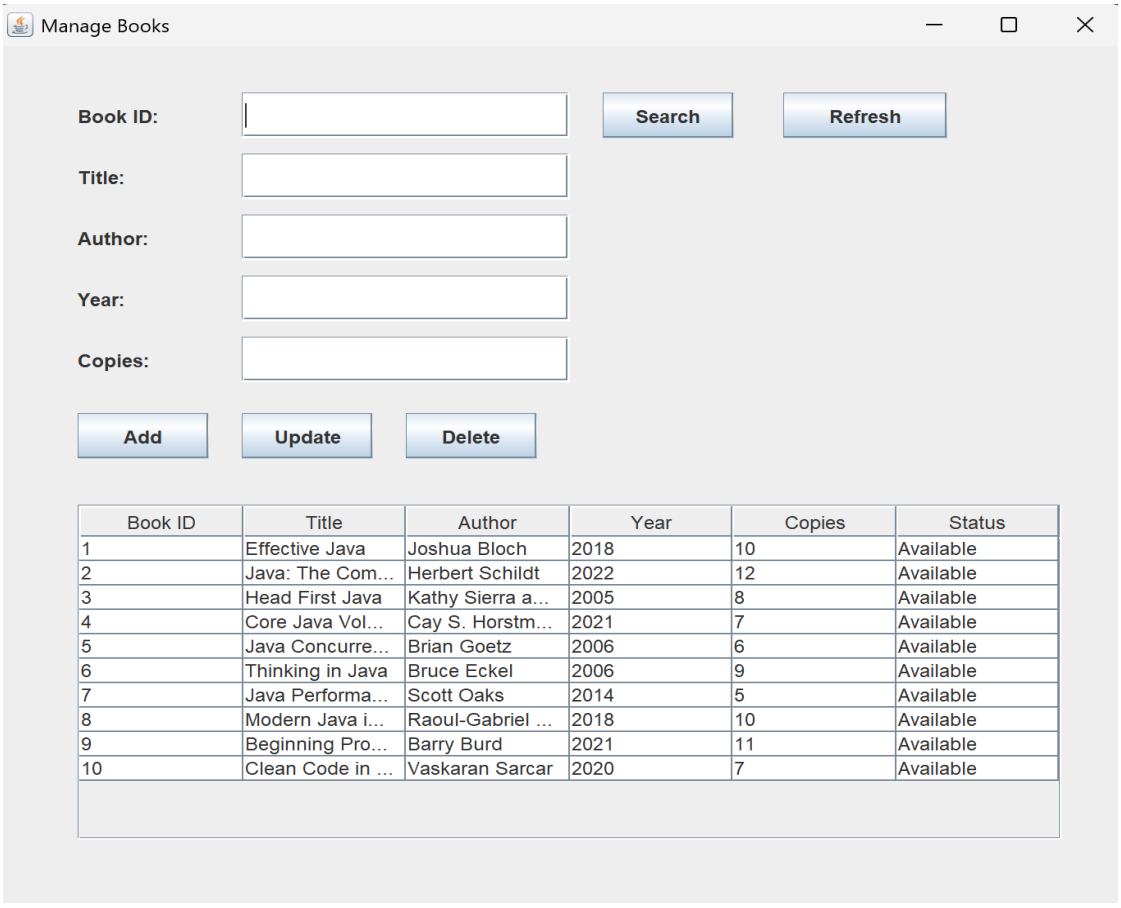

Fig no:-4 Books Dashboard

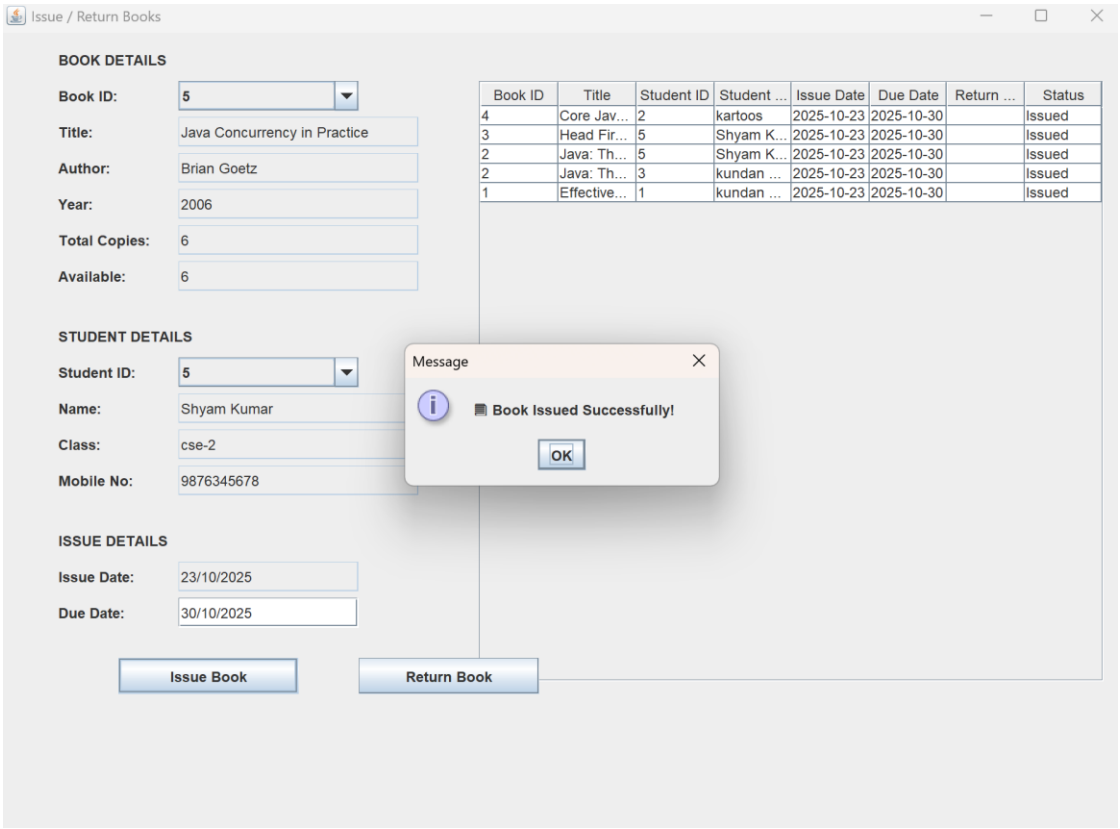

Fig no:-5 Book Issue Page

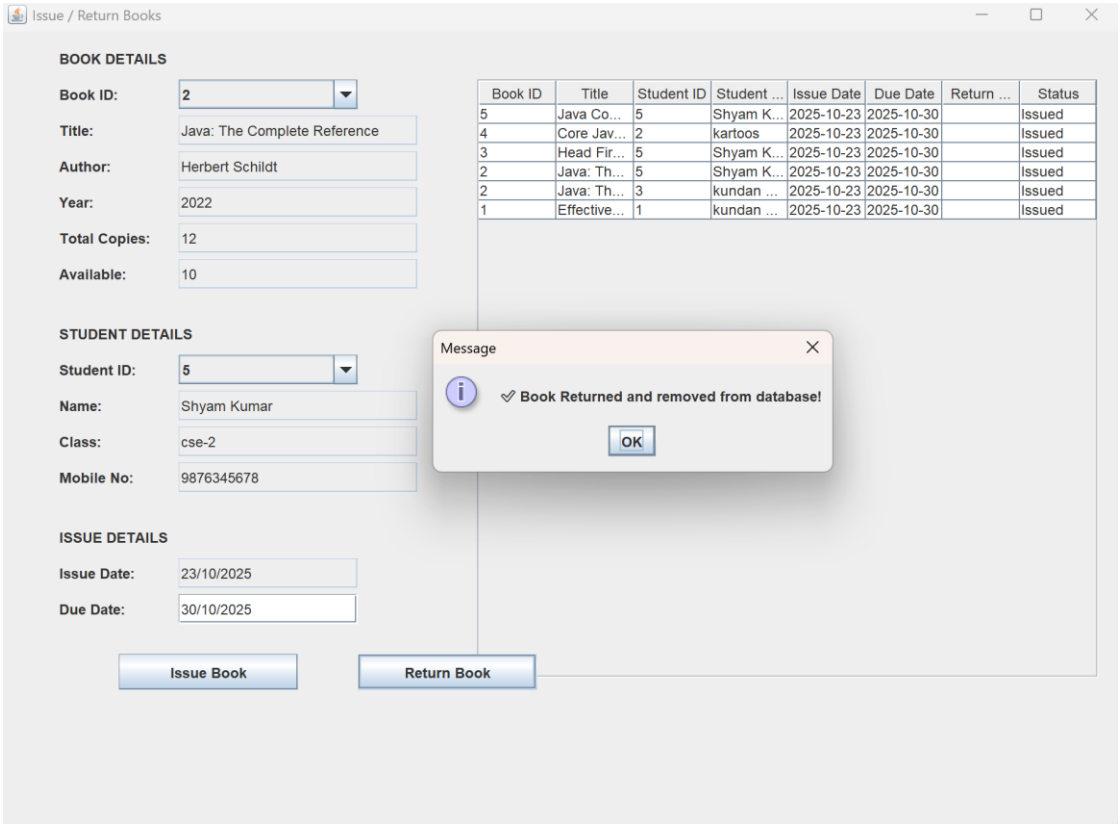

Fig no:-6 Book Return Page

References

Books

1. Schildt, H. (2022). Java: The Complete Reference (12th ed.). McGraw-Hill Education.
2. Sierra, K., & Bates, B. (2005). Head First Java (2nd ed.). O'Reilly Media.
3. Bloch, J. (2018). Effective Java (3rd ed.). Addison-Wesley.
4. Horstmann, C. S. (2021). Core Java Volume I – Fundamentals (12th ed.). Pearson.
5. Goetz, B. (2006). Java Concurrency in Practice. Addison-Wesley.

Websites

1. Oracle. (n.d.). Java™ Platform, Standard Edition Documentation. Retrieved from https://docs.oracle.com/javase/
2. W3Schools. (n.d.). JDBC Tutorial. Retrieved from https://www.w3schools.com/java/java_jdbc.asp
3. TutorialsPoint. (n.d.). Java Swing Tutorial. Retrieved from https://www.tutorialspoint.com/java_swing/index.htm
4. GeeksforGeeks. (n.d.). Library Management System using Java. Retrieved from https://www.geeksforgeeks.org/library-management-system-project-in-java/
5. MySQL Documentation. (n.d.). MySQL Reference Manual. Retrieved from https://dev.mysql.com/doc/