Project Title: Virtual Study Platform

Introduction:

The Virtual Study Platform is a Java-based software application developed using NetBeans that provides a structured and role-based digital learning environment. The system includes separate dashboards for the administrator, teachers, and students to ensure secure and organized access. The administrator is responsible for approving user registrations and maintaining overall system control. Teachers can create and manage classes, upload study materials and books, evaluate students, and track their marks through their individual dashboards. Students can participate in classes, access uploaded learning resources, and view their academic progress using their own dashboards. By combining all these features into a single platform, the Virtual Study Platform supports effective online learning, improves academic management, and enhances interaction between teachers and students in a simple and efficient manner.

Primary Objectives of the Project :

i) To design and develop a Java-based virtual learning system using NetBeans.

ii) To provide separate and secure dashboards for administrators, teachers, and students.

iii) To enable administrators to approve user registrations and manage system access.

iv) To allow teachers to create classes, upload study materials and books, and manage academic content.

v) To support student evaluation and systematic tracking of marks and performance.

vi) To enable students to access classes, learning materials, and view their academic progress.

vii) To improve the efficiency, transparency, and organization of online teaching and learning activities.

Major Functionalities of the Virtual Study Platform :

a) User registration and login with role-based access (Admin, Teacher, Student).

b) Admin dashboard to review and approve or reject user registrations.

c) Teacher dashboard for creating and managing virtual classes.

d) Uploading and managing study materials and digital books by teachers.

e) Student evaluation and marks entry by teachers.

f) Performance tracking and progress monitoring for students.

e) Student dashboard to join classes and access uploaded materials and books.

g) Viewing academic results and progress reports by students.

h) Secure data management to ensure authorized access and system reliability.

Major Categories of the Virtual Study Platform:

Administrator Module
This category handles overall system management. The administrator verifies and approves user registrations, manages user roles, and ensures secure and smooth operation of the platform.

Teacher Module
This category focuses on academic management. Teachers can create and manage classes, upload study materials and digital books, evaluate students, assign marks, and track student performance through their individual dashboards.

Student Module
This category supports learning activities. Students can join classes, access uploaded study materials and books, participate in academic activities, and view their marks and overall progress using their personal dashboards.

Authentication and Security Module
This category manages user login, role-based access control, and data security to ensure that only authorized users can access specific features of the system.

Benefits of Integrating the Virtual Study Platform:

1. Centralizes all academic activities into a single system for easy management.

2. Provides clear role-based access, ensuring secure and controlled use of features.

3. Improves communication and interaction between teachers and students.

4. Reduces manual work by automating class management, evaluation, and progress tracking.

5. Allows students to access study materials and performance reports anytime.

6. Helps teachers efficiently organize classes, materials, and student assessments.

7. Enables administrators to maintain system security and user authenticity.

8. Enhances overall efficiency, transparency, and reliability of the digital learning process.

The Virtual Study Platform integrates administrative control, teaching activities, and student learning into a single Java-based system developed using NetBeans. By providing separate dashboards for administrators, teachers, and students, the platform ensures secure access and clear role separation. Administrative approval of registrations increases system reliability, while teacher tools for class creation, material sharing, and evaluation support organized academic management. Students benefit from easy access to learning resources and continuous performance tracking. Overall, the integrated design improves efficiency, reduces manual effort, and creates a structured and effective virtual learning environment.

Software requirement collection through a questionnaire:

Software requirements data collection from a survey

Questionnaire:

1.Do you face difficulty in accessing study materials and books in one place?

2.Would you prefer a software that provides separate login panels for students and teachers?

3.Do you require secure login with unique ID and password?

4.Should teachers be able to upload study materials and mock exams easily?

5.Should students be able to attempt exams within the software or download/upload PDFs?

6.Do you want an option for teachers to view and evaluate student results?

7.How important is it to keep a digital library of books accessible anytime?

8.Should the system work offline (local software) or online (web-based)?

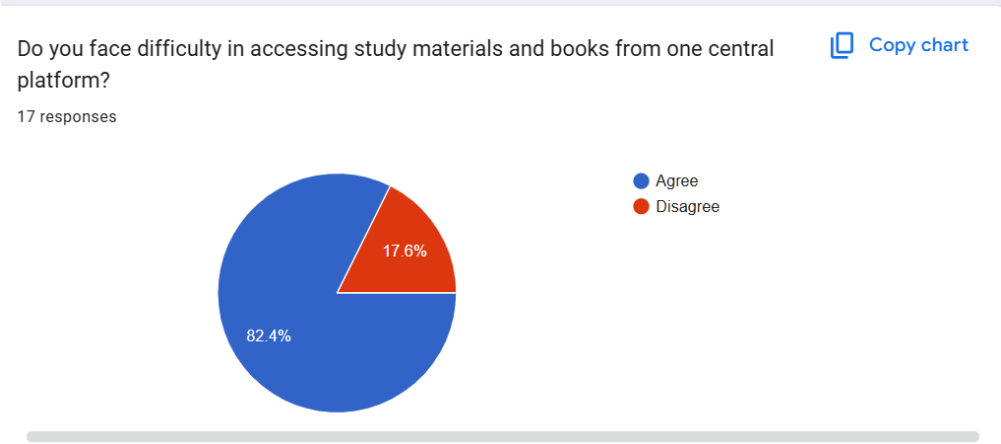
9.Do you wanStudents find it difficult to collect study materials from multiple sources.
t notifications/reminders for exams and new uploads?

10.Would you like the software to track individual performance over time?

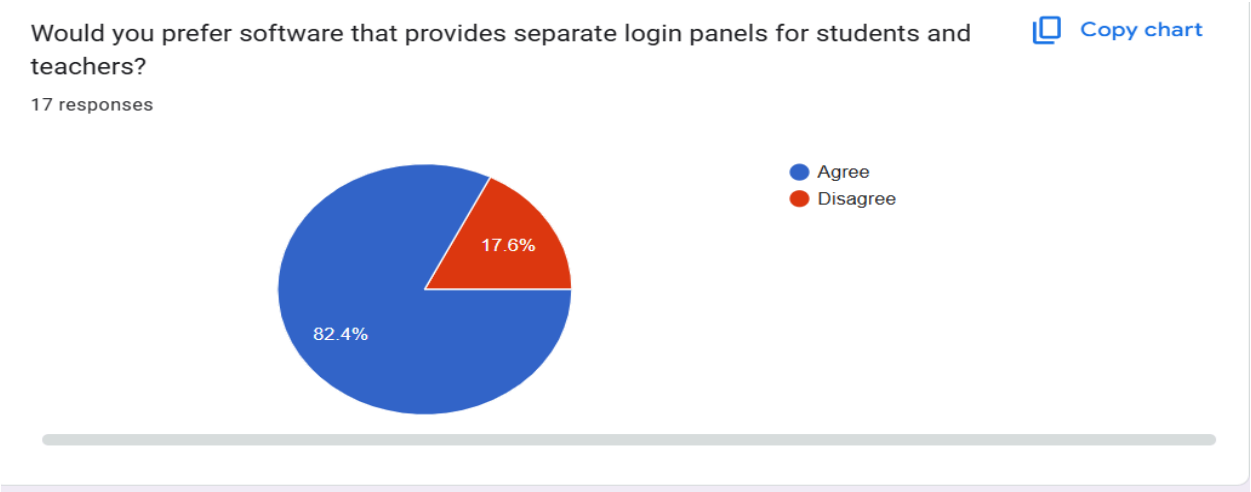
Response:

A total number of 17 response was gathered and their summary is given below

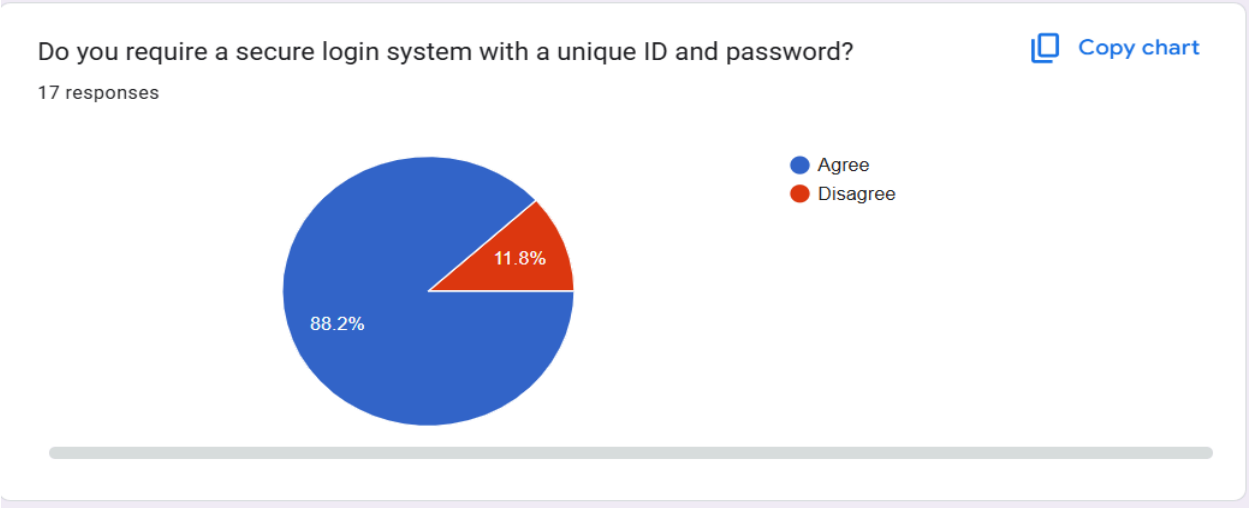
82.4% of people agreed that they face difficulty accessing study materials and books from one central platform.



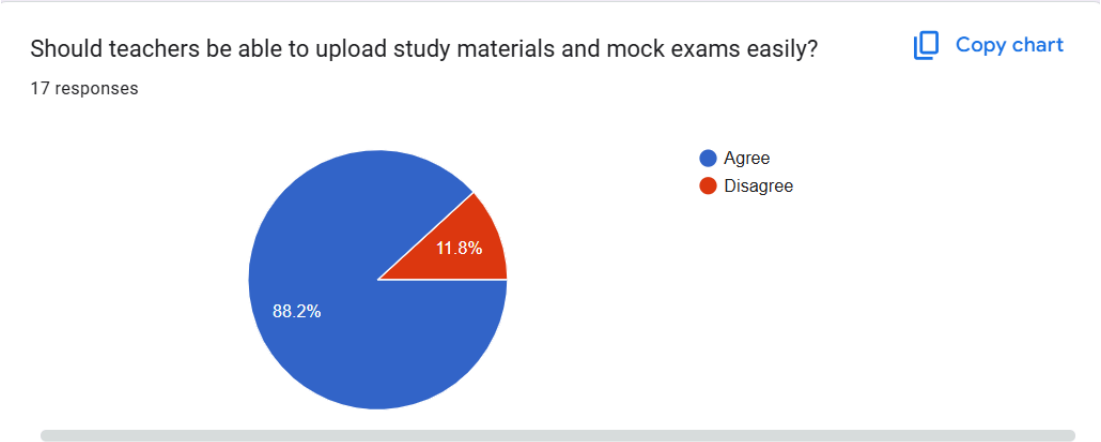
82.4% of people want separate login panels



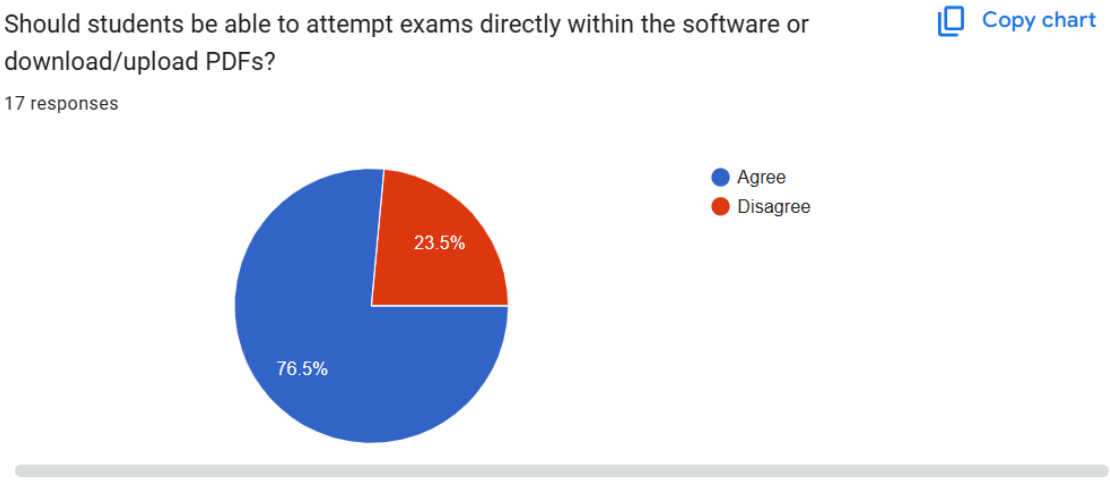
88.2% of people want secure login



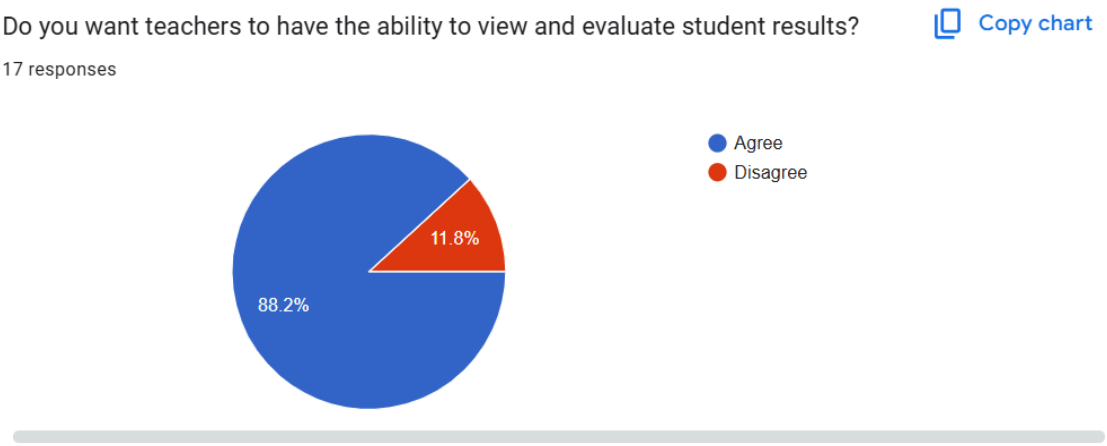
88.2% of people wants teachers to be able to upload study materials and mock exams easily.



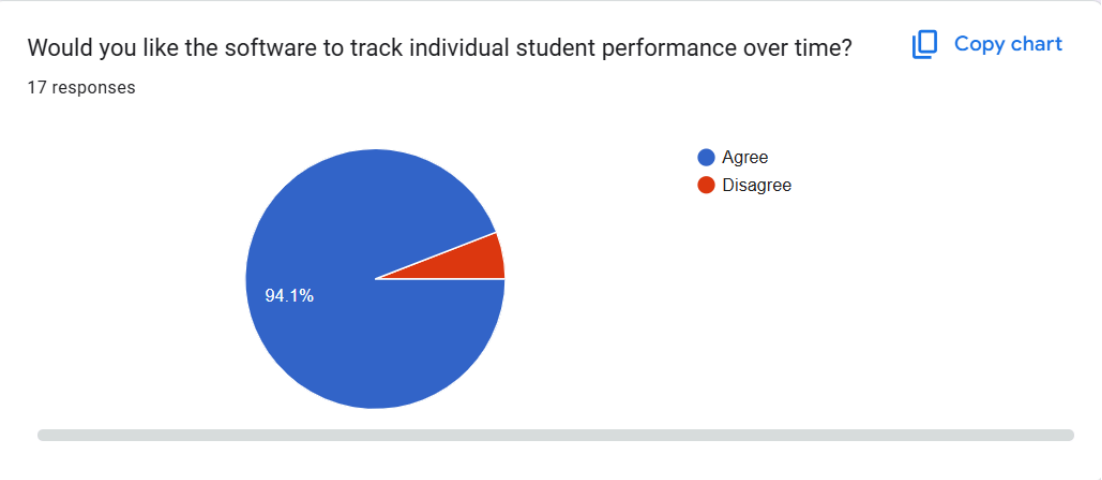
76.5 % of people wants to attempt exams directly



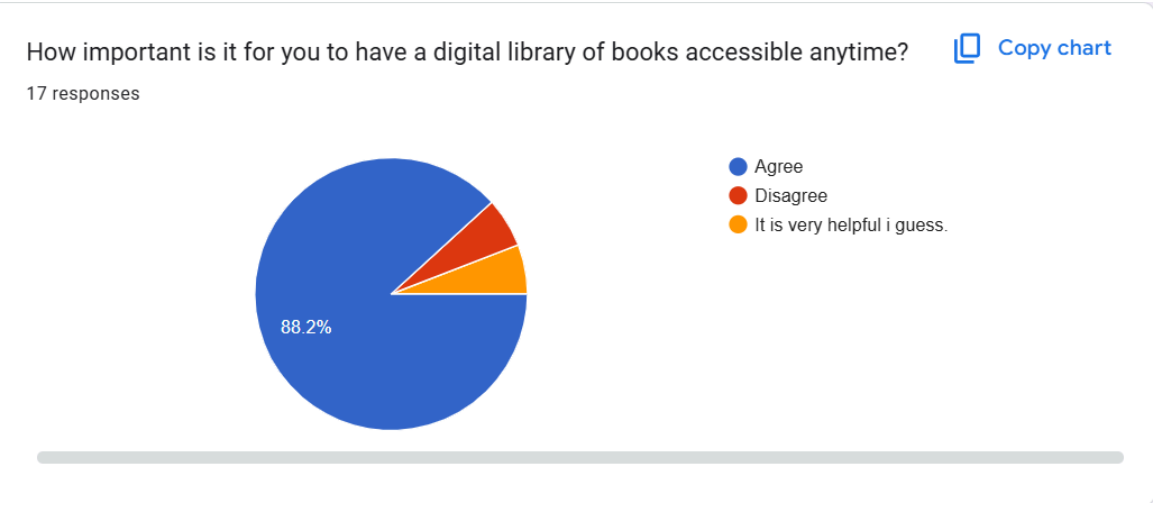
88.2% of people want teachers to have the ability to view and evaluate students



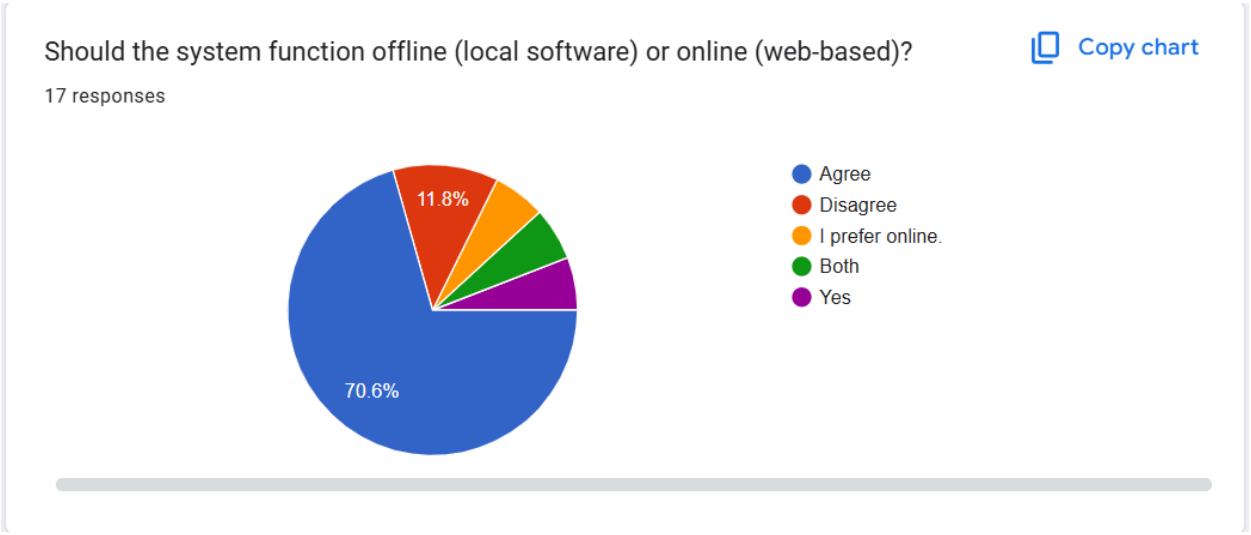
94.1% of people wants teachers to track individual performance over time



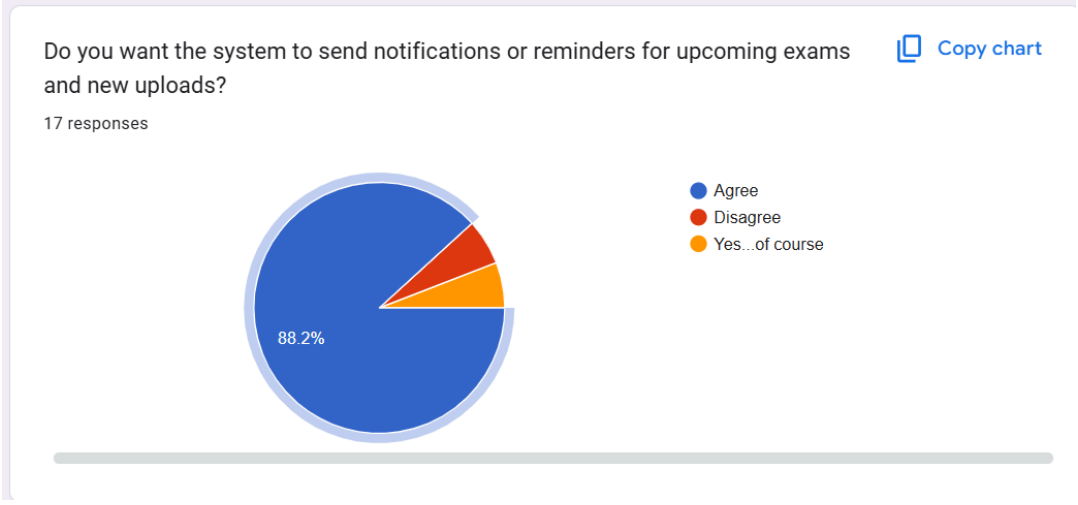
88.2% of people wants have a digital library



70.6% of people want the system function to be offline



88.2% people wants to get notified after receiving new uploads



Software requirements specification:

Virtual Study Platform is an offline Java-based application developed using NetBeans. The system is designed to manage academic activities by providing separate dashboards for administrators, teachers, and students without requiring an internet connection.

The Virtual Study Platform operates as an offline system installed on a local computer. It allows administrators to approve registrations, teachers to manage classes and academic content, and students to access learning materials and track their progress. All data is stored and managed locally using a database.

Dashboards:

Admin Dashboard

The admin dashboard serves as the central control panel for system management. Key functionalities include:

- **User Registration Approval**: Admin can review and approve or reject registration requests from teachers and students.
- **User Management**: Admin can view, edit, or remove user accounts.
- **System Monitoring**: Admin can monitor overall system usage and maintain data integrity.
- **Access Control**: Admin ensures that only authorized users can access the platform.

Teachers Dashboard

The teacher dashboard allows teachers to manage academic activities efficiently. Key functionalities include:

- **Class Creation and Management**: Teachers can create new classes, manage existing classes, and assign students.
- **Upload Study Materials and Books**: Teachers can upload study materials, digital books, and resources for students to access.
- **Student Evaluation**: Teachers can evaluate student assignments, quizzes, or exams and assign marks.
- **Progress Tracking**: Teachers can view and track the academic progress and performance of students.
- **Communication**: Teachers can interact with students regarding class updates and study material (offline notices/messages within the system).

Student Dashboard:

The student dashboard provides students with access to learning resources and performance tracking. Key functionalities include:

- **Class Participation**: Students can view and join their assigned classes.
- **Access to Study Materials**: Students can download or view uploaded books, notes, and other learning resources.
- **Progress Monitoring**: Students can view their marks, performance reports, and overall academic progress.
- **Self-Assessment**: Students can track their own learning and understand areas for improvement.

System Functionality:

Security

- The system shall require username and password authentication.
- The system shall restrict access based on user roles.

Performance

- The system shall respond quickly to user actions in offline mode.
- The system shall support multiple local users accessing the system at different times.

Usability

- The interface shall be simple and easy to use.
- Dashboards shall be clearly designed for each user role.

Reliability

- The system shall store data securely in a local database.
- The system shall prevent data loss during normal operation.

Constraints and Limitations :

- The system operates only in offline mode.
- Data sharing between multiple computers is not supported.
- Features such as live classes and real-time communication are not included.

Software Requirements:

- Java Development Kit (JDK)
- NetBeans IDE
- MySQL / SQLite Database Server
- Windows or Linux Operating System

The Virtual Study Platform is an offline Java-based application developed using NetBeans that provides a structured environment for managing educational activities. The system is designed around three main user roles—administrator, teacher, and student—each with a dedicated dashboard, ensuring secure, role-based access and clear separation of responsibilities. Administrators manage registrations and user accounts, teachers create and manage classes, upload study materials, evaluate students, and track their progress, while students participate in classes, access learning resources, and monitor their performance.

From a functional perspective, the system supports core academic operations such as offline user registration and approval, class creation and management, study material uploads, student evaluation, and progress tracking. Each dashboard provides features tailored to the user's role, enabling administrators to maintain control, teachers to manage teaching tasks efficiently, and students to engage with learning resources and track their academic growth.

In terms of non-functional aspects, the platform ensures secure authentication and role-based access, preventing unauthorized use. It is designed to be user-friendly and intuitive, providing simple navigation through dashboards. Performance is optimized for offline operation, with fast response times and reliable local data storage. Additionally, the system ensures data consistency, accuracy, and error handling, making it a dependable solution for managing educational activities in environments without internet connectivity.

Use Case Diagram:

Level 0:

In this assignment, the Level 0 use case diagram represents the main stakeholders and their interaction with the system. The stakeholders include Admin, Teacher, Student, and Guest. The Admin manages users, monitors activities, and maintains the overall system operations. The Teacher uploads study materials, creates and manages exams, and evaluates student performance. The Student accesses study resources, attempts mock tests, and checks results or feedback. The Guest can only view limited information such as available courses or general announcements without logging in. The Level-0 diagram shows these stakeholders outside the system boundary and connects them to the main functions they perform within the system, such as login, uploading materials, accessing resources, and viewing information.

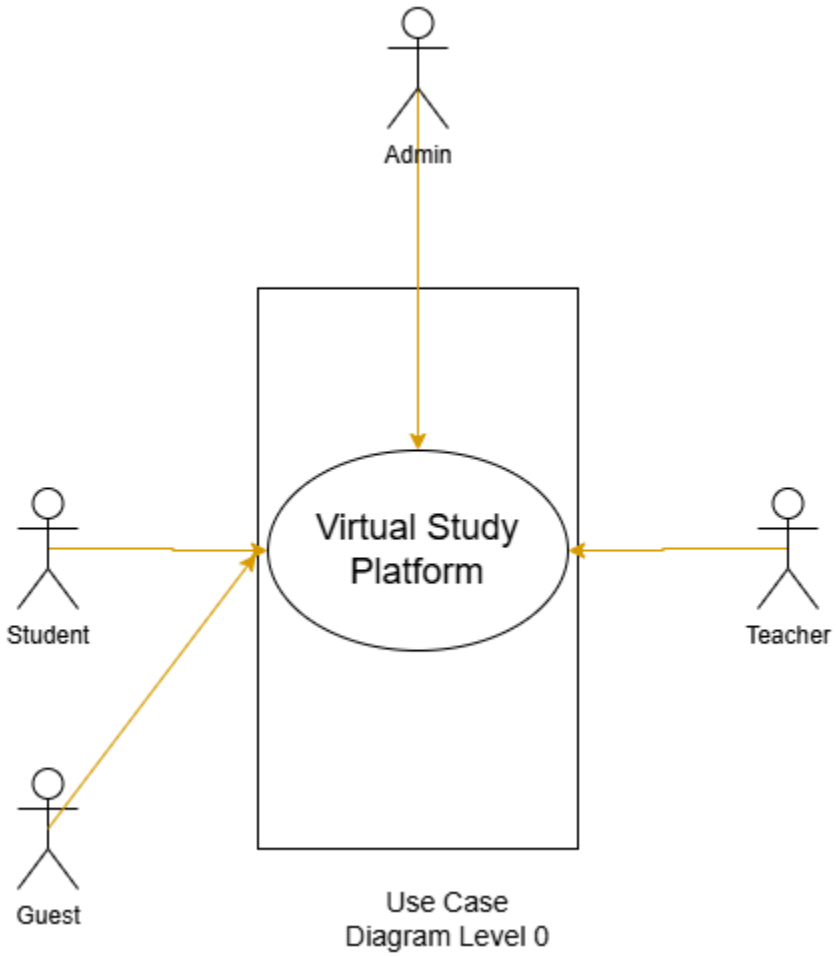


Fig 01: Level 0 use case diagram

Level 1:

At Level 1, the system is divided into several subsystems, each handling a specific part of the system's functionality and accessed by different stakeholders:

Authentication Module: Handles secure login, registration, and password recovery for all users, ensuring that only authorized users can access the system.

Digital Library Module: Allows teachers to upload and manage study materials, while students can browse, view, and download these resources easily.

Exam and Mock Test Management Module: Enables teachers to create, schedule, and publish exams or mock tests, and allows students to attempt and submit their tests through the system.

User Management Module: Provides the admin with the ability to add, update, or remove users, assign roles, and control access permissions.

Admin Panel Module: Offers the administrator complete control over system operations, including system settings, data management, and performance monitoring.

Role-Based Dashboard Module: Displays a customized interface for each type of user—students, teachers, and admins—showing only the features and information relevant to their role.
Result and Performance Tracking: Displays a customized interface for each student's records regarding their exam performance which the teachers can also access.

These subsystems together make up the overall structure of the system. The Level-1 use case diagram visually presents these modules within the system boundary, connecting them to the respective stakeholders who interact with each one. This structure ensures that all user roles and system functions are clearly organized and efficiently managed.

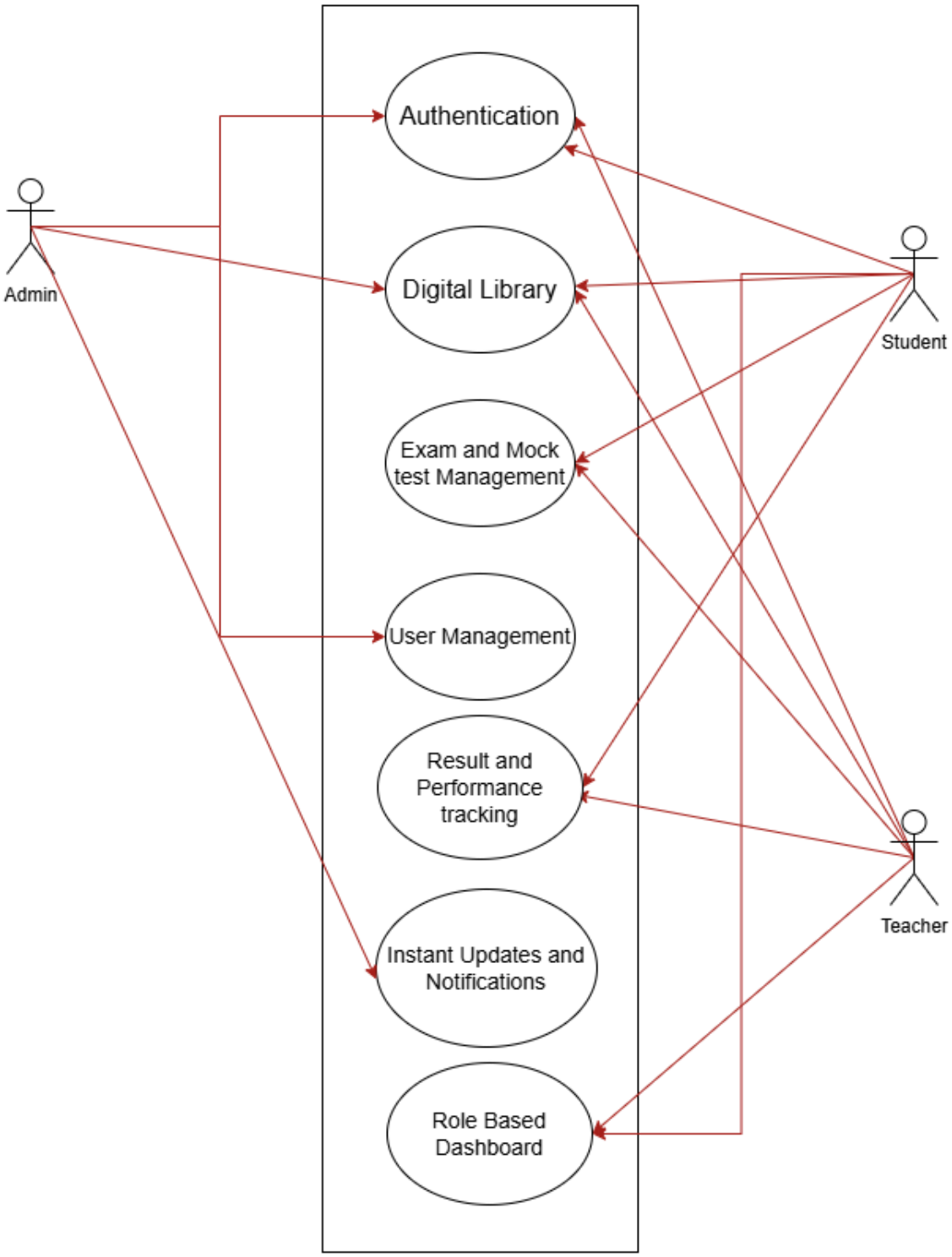


Fig 02: Level 1 use case diagram

Activity Diagram:

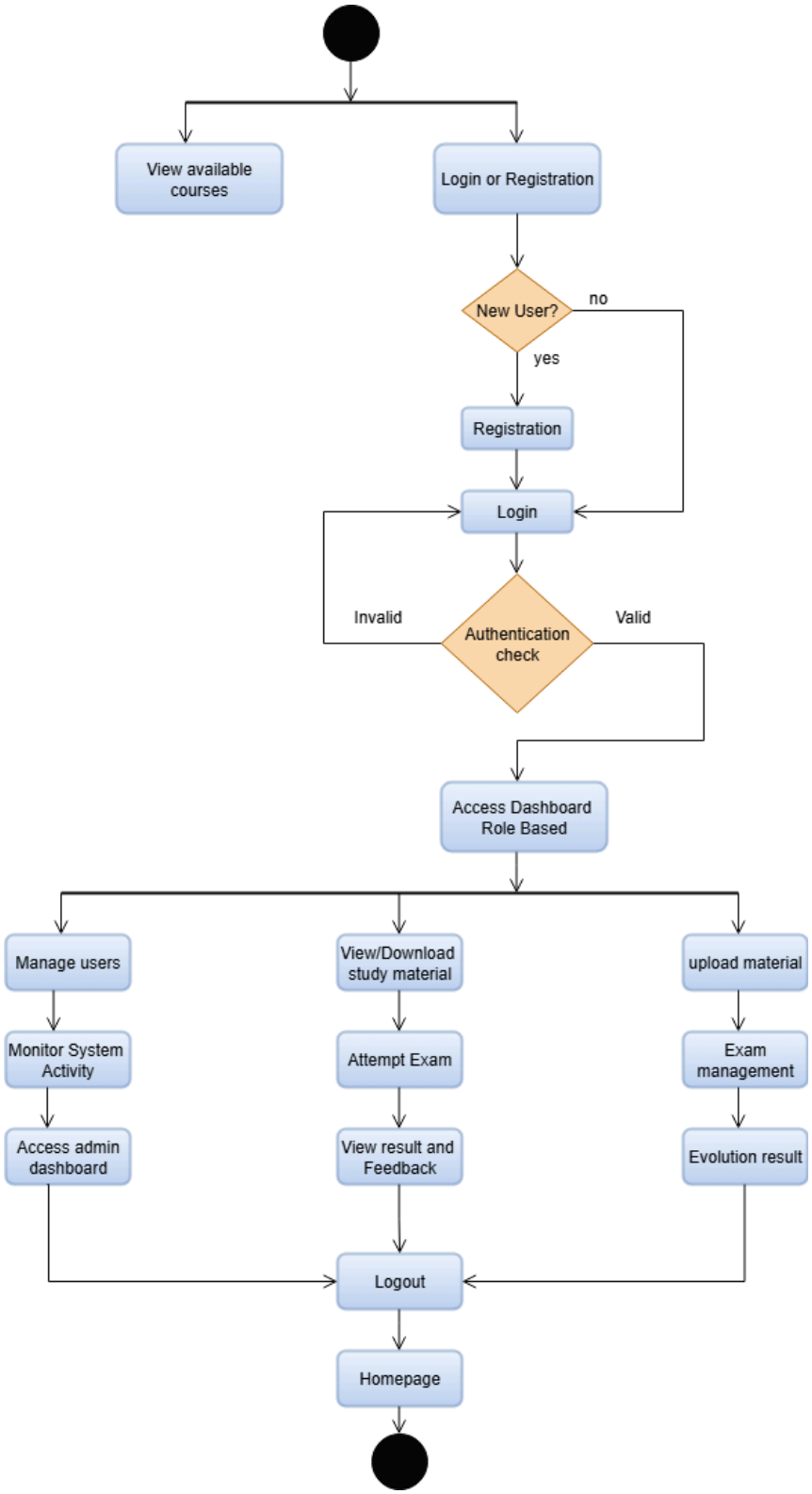


Fig 03: Activity diagram

Breakdown of the Activity Diagram:

1. Start of the System

- The black filled circle at the top represents the start node of the activity diagram.
- This indicates where the system flow begins when a user enters the platform.

2. Initial User Options

From the start, the flow splits into two parallel options:

- View available courses
- Login or Registration

3. New User Decision

The decision diamond “New User?” checks whether the user already has an account.

- Yes → The user goes to Registration
- No → The user directly proceeds to Login

4. Registration Process

New users complete the Registration process.

After successful registration, they are redirected to the Login page.

5. Login & Authentication

- User enters credentials in Login.
- System performs an Authentication check:
- Invalid → User is sent back to Login (loop)
- Valid → User is granted access

This ensures system security and controlled access.

6. Role-Based Dashboard Access

- After successful authentication, the user accesses a Role-Based Dashboard.

The available features depend on the user role (Admin, Student, Instructor).

7. Admin Activities

For Admin users, the flow includes:

- Manage users

- Monitor system activity
-
- Access admin dashboard

Admins oversee system operations and user management.

8. Student Activities

For Students, the activities include:

- View / Download study material
- Attempt Exam
- View result and feedback

These actions support learning and evaluation.

9. Instructor / Teacher Activities

- For Instructors, the flow includes:
- Upload material
- Exam management
- Evaluation result

They manage academic content and assessments.

10. Logout Process

After completing tasks, all user roles converge to Logout. This ensures a proper session termination.

11. End of the System

After logout, the user is redirected to the Homepage.

The black filled circle at the bottom represents the end node, marking the completion of the activity flow.

Overview:

The activity diagram shows how users interact with the Virtual Study Platform (VSP). It starts from the Homepage, where a user can either view available courses as a guest or go to the login or registration section. If the user is new, they complete the registration process; otherwise, they log in. The system then performs an authentication check. If the login is invalid, the user is asked to try again. If valid, the system allows the user to access the Role-Based Dashboard. Finally, all users can logout, returning to the Homepage, where the process ends.

Class Diagram:

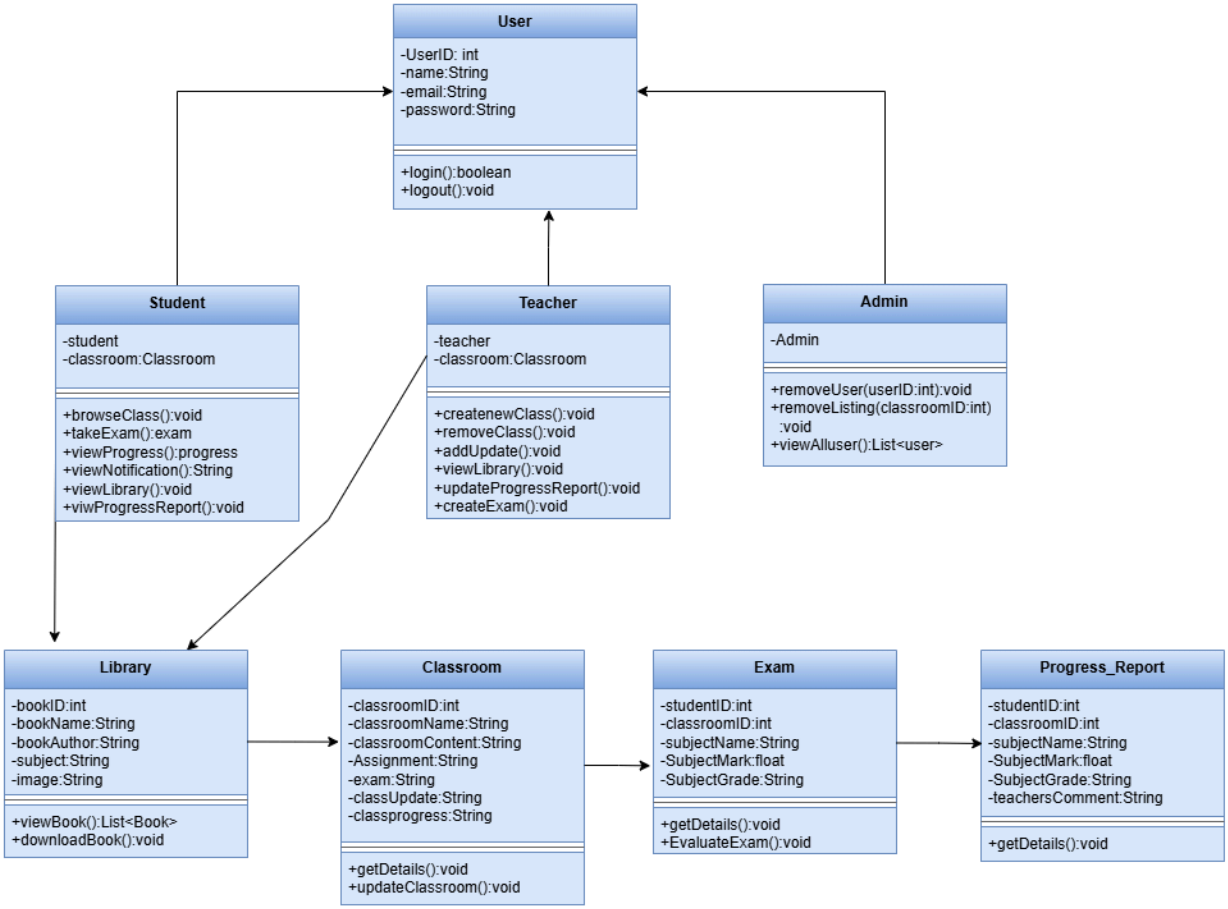


Fig 04: Class Diagram

Class Diagram: A class diagram is a UML diagram that shows the structure of a system by illustrating its classes, attributes, methods, and the relationships among them. It helps visualize how different parts of a system interact. For example, in a library system, classes like Book, Loan, and Member can be connected to show how books are borrowed and returned by members.

User Class: The User class is the parent or base class from which other user-related classes inherit. It contains essential user information such as userID, name, email, and password. These attributes provide identity and authentication for all system users. The class includes two main methods: login() and logout(), which manage user access and session control. This class serves as the foundation for specialization, allowing subclasses like Student, Teacher, and Admin to extend its features while maintaining common user properties.

Student Class: The Student class inherits from the User class and represents learners within the system. It includes attributes such as student and classroom, linking each student to a specific classroom. Its methods allow students to browse available classes, take exams, view progress, access notifications, and explore the library. The class supports learning and self-assessment functions, emphasizing user interaction and academic progress tracking.

Teacher Class: The Teacher class also extends the User class and represents instructors responsible for

managing academic activities. It has attributes for teacher identity and classroom association. The class methods allow teachers to create and remove classes, update content, view library materials, create exams, and update student progress reports. These operations highlight the teacher's role in organizing and monitoring educational processes.

Admin Class: The Admin class inherits from the User class but focuses on system-level management. It includes methods such as removeUser(), removeListing(), and viewAllUser(), enabling administrative control over user accounts and classroom listings. This class ensures that the system remains organized and that user data and course listings are properly maintained.

Library Class: The Library class handles educational resources available to users. It contains attributes like bookID, bookName, bookAuthor, subject, and image. Its methods, viewBook() and downloadBook(), allow users to explore and access study materials. This class supports both students and teachers in finding and using educational content efficiently.

Classroom Class: The Classroom class represents the learning environment. It includes attributes such as classroomID, classroomName, classroomContent, assignment, exam, and classProgress. The methods getDetails() and updateClassroom() manage and update classroom-related information. It serves as a central link between teachers and students for sharing content and assignments.

Exam Class: The Exam class manages assessment-related data. Its attributes include studentID, classroomID, subjectName, subjectMark, and subjectGrade. The methods getDetails() and EvaluateExam() allow the retrieval and evaluation of exam results. This class plays a crucial role in measuring student performance and linking results with both the classroom and progress report.

Progress_Report Class: The Progress_Report class records and evaluates student performance over time. It has attributes like studentID, classroomID, subjectName, subjectMark, subjectGrade, and teachersComment. The getDetails() method retrieves performance information for review. This class provides a detailed record of academic achievement and feedback, contributing to student evaluation and progress tracking

Conclusion:

The Virtual Study Platform is a comprehensive offline educational management system developed using Java in NetBeans. It successfully integrates administrative control, teaching functions, and student learning activities into a single, structured platform. By providing separate dashboards for administrators, teachers, and students, the system ensures role-based access, data security, and efficient management of academic tasks. Administrators can approve user registrations and manage accounts, maintaining overall system integrity. Teachers are empowered to create classes, upload study materials and books, evaluate students, and track their progress systematically. Students benefit from easy access to resources, participation in classes, and continuous monitoring of their academic performance.

The platform's offline functionality ensures that all operations can be performed without internet connectivity, making it reliable and accessible in environments with limited or no network access. Its user-friendly interface, efficient data handling, and structured workflows reduce manual effort, minimize errors, and enhance transparency in academic management. By consolidating various educational processes into one platform, the Virtual Study Platform promotes organized learning, improves communication between teachers and students, and provides a clear overview of student progress.

Overall, this system addresses the key challenges of traditional learning management by offering a secure, efficient, and scalable solution for educational institutions. It demonstrates the potential of technology to streamline academic operations, enhance student engagement, and support a structured approach to offline digital learning. The Virtual Study Platform, therefore, represents a practical and effective tool for improving the quality, accessibility, and management of education in a controlled offline environment.

Code in Appendix:

1. Login Page

```
package com.mycompany.vsp2.ui;

import com.mycompany.vsp2.AdminDashboard;
import com.mycompany.vsp2.StudentDashboard;
import com.mycompany.vsp2.TeacherDashboard;
import com.mycompany.vsp2.dao.UserDAO;
import com.mycompany.vsp2.model.User;

import javax.swing.*;
import java.awt.*;

public class LoginPage extends JFrame {

    public LoginPage() {
        initComponents();
    }

    // --------- CUSTOM GRADIENT BACKGROUND PANEL ----------
    class GradientPanel extends JPanel {
        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);

            Graphics2D g2 = (Graphics2D) g;
            int w = getWidth();
            int h = getHeight();

            Color top = new Color(70, 130, 180);      // Steel blue
            Color bottom = new Color(240, 248, 255);  // Light blue
            GradientPaint gp = new GradientPaint(0, 0, top, 0, h, bottom);

            g2.setPaint(gp);
            g2.fillRect(0, 0, w, h);
        }
    }

    @SuppressWarnings("unchecked")
    private void initComponents() {

        // gradient background
        GradientPanel background = new GradientPanel();
        setContentPane(background);
```

```
        // UI Components
        jLabel1 = new JLabel("Email:");
        jLabel2 = new JLabel("Password:");
        emailField = new JTextField();
        passwordField = new JPasswordField();
        loginButton = new JButton("Login");
        registerButton = new JButton("Register");

        // ----- TOP CARD PANEL -----
        JPanel infoCard = new JPanel();
        infoCard.setBackground(new Color(255, 255, 255, 210));
        infoCard.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

        JLabel appTitle = new JLabel("OZ'S Virtual Study Platform", SwingConstants.CENTER);
        appTitle.setFont(new Font("Segoe UI", Font.BOLD, 22));

        JLabel hook = new JLabel("Connect, Learn & Grow Together", SwingConstants.CENTER);
        hook.setFont(new Font("Segoe UI", Font.PLAIN, 14));

        JLabel desc = new JLabel(
                "<html><center>A modern learning platform enabling teachers and students "
                + "to collaborate, track progress, and enhance productivity.</center></html>",
                SwingConstants.CENTER);

        infoCard.setLayout(new BoxLayout(infoCard, BoxLayout.Y_AXIS));
        infoCard.add(appTitle);
        infoCard.add(Box.createVerticalStrut(10));
        infoCard.add(hook);
        infoCard.add(Box.createVerticalStrut(10));
        infoCard.add(desc);

        // Buttons
        loginButton.addActionListener(evt -> loginAction());
        registerButton.addActionListener(evt -> {
            new RegisterPage().setVisible(true);
            this.dispose();
        });

        // ----- LAYOUT -----
        GroupLayout layout = new GroupLayout(background);
        background.setLayout(layout);

        layout.setHorizontalGroup(
            layout.createParallelGroup(GroupLayout.Alignment.CENTER)
                .addComponent(infoCard, GroupLayout.PREFERRED_SIZE, 350,
GroupLayout.PREFERRED_SIZE)
```

```
            .addGroup(layout.createSequentialGroup()
                .addGroup(layout.createParallelGroup(GroupLayout.Alignment.CENTER)
                    .addComponent(jLabel1)
                    .addComponent(emailField, GroupLayout.PREFERRED_SIZE, 260,
GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel2)
                    .addComponent(passwordField, GroupLayout.PREFERRED_SIZE, 260,
GroupLayout.PREFERRED_SIZE)
                    .addComponent(loginButton, GroupLayout.PREFERRED_SIZE, 260,
GroupLayout.PREFERRED_SIZE)
                    .addComponent(registerButton, GroupLayout.PREFERRED_SIZE, 260,
GroupLayout.PREFERRED_SIZE)
                )
            )
        );

        layout.setVerticalGroup(
            layout.createSequentialGroup()
                .addGap(30)
                .addComponent(infoCard, GroupLayout.PREFERRED_SIZE,
GroupLayout.DEFAULT_SIZE, GroupLayout.PREFERRED_SIZE)
                .addGap(40)
                .addComponent(jLabel1)
                .addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(emailField, GroupLayout.PREFERRED_SIZE, 34,
GroupLayout.PREFERRED_SIZE)
                .addGap(18)
                .addComponent(jLabel2)
                .addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(passwordField, GroupLayout.PREFERRED_SIZE, 34,
GroupLayout.PREFERRED_SIZE)
                .addGap(30)
                .addComponent(loginButton, GroupLayout.PREFERRED_SIZE, 38,
GroupLayout.PREFERRED_SIZE)
                .addPreferredGap(LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(registerButton, GroupLayout.PREFERRED_SIZE, 38,
GroupLayout.PREFERRED_SIZE)
                .addGap(40)
        );


        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setTitle("Login");
        pack();
        setLocationRelativeTo(null);
    }
```

```
    private void loginAction() {
        String email = emailField.getText().trim();
        String password = String.valueOf(passwordField.getPassword()).trim();

        if (email.isEmpty() || password.isEmpty()) {
            JOptionPane.showMessageDialog(this, "Please enter both email and password.");
            return;
        }

        User user = UserDAO.login(email, password);

        if (user == null) {
            JOptionPane.showMessageDialog(this, "Invalid email or password!");
            return;
        }

        if (!user.isApproved()) {
            JOptionPane.showMessageDialog(this, "Your account is pending admin approval.");
            return;
        }

        switch (user.getRole().toLowerCase()) {
            case "student":
                new StudentDashboard(user).setVisible(true);
                break;
            case "teacher":
                new TeacherDashboard(user).setVisible(true);
                break;
            default:
                new AdminDashboard(user).setVisible(true);
                break;
        }

        this.dispose();
    }

    private JTextField emailField;
    private JPasswordField passwordField;
    private JButton loginButton;
    private JButton registerButton;
    private JLabel jLabel1;
    private JLabel jLabel2;
}
```

2. Registration Page

```
package com.mycompany.vsp2.ui;

import com.mycompany.vsp2.dao.UserDAO;
import com.mycompany.vsp2.model.User;
import javax.swing.*;

public class RegisterPage extends javax.swing.JFrame {

    public RegisterPage() {
        initComponents();
    }

    @SuppressWarnings("unchecked")
    private void initComponents() {
        nameField = new javax.swing.JTextField();
        emailField = new javax.swing.JTextField();
        passwordField = new javax.swing.JPasswordField();
        confirmPasswordField = new javax.swing.JPasswordField();

        roleCombo = new javax.swing.JComboBox<>();
        registerButton = new javax.swing.JButton();
        loginButton = new javax.swing.JButton();

        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jLabel3 = new javax.swing.JLabel();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Register");

        roleCombo.setModel(new javax.swing.DefaultComboBoxModel<>(new String[] { "student",
"teacher" }));

        registerButton.setText("Register");
        registerButton.addActionListener(evt -> registerAction());

        loginButton.setText("Go to Login");
        loginButton.addActionListener(e -> {
            new LoginPage().setVisible(true);
            this.dispose();
        });

        jLabel1.setText("Name:");
```

```
        jLabel2.setText("Email:");
        jLabel3.setText("Password:");
        jLabel4.setText("Confirm Password:");
        jLabel5.setText("Role:");

        javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(50, 50, 50)
                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
                    .addComponent(nameField)
                    .addComponent(emailField)
                    .addComponent(passwordField)
                    .addComponent(confirmPasswordField)
                    .addComponent(roleCombo, 0, 260, Short.MAX_VALUE)
                    .addComponent(registerButton, javax.swing.GroupLayout.DEFAULT_SIZE, 260,
Short.MAX_VALUE)
                    .addComponent(loginButton, javax.swing.GroupLayout.DEFAULT_SIZE, 260,
Short.MAX_VALUE)
                    .addComponent(jLabel1)
                    .addComponent(jLabel2)
                    .addComponent(jLabel3)
                    .addComponent(jLabel4)
                    .addComponent(jLabel5))
                .addContainerGap(60, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(40, 40, 40)
                .addComponent(jLabel1)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(nameField, javax.swing.GroupLayout.PREFERRED_SIZE, 32,
javax.swing.GroupLayout.PREFERRED_SIZE)

                .addGap(18, 18, 18)
                .addComponent(jLabel2)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(emailField, javax.swing.GroupLayout.PREFERRED_SIZE, 32,
javax.swing.GroupLayout.PREFERRED_SIZE)

                .addGap(18, 18, 18)
                .addComponent(jLabel3)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
```

```
            .addComponent(passwordField, javax.swing.GroupLayout.PREFERRED_SIZE, 32,
javax.swing.GroupLayout.PREFERRED_SIZE)

            .addGap(18, 18, 18)
            .addComponent(jLabel4)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(confirmPasswordField, javax.swing.GroupLayout.PREFERRED_SIZE,
32, javax.swing.GroupLayout.PREFERRED_SIZE)

            .addGap(18, 18, 18)
            .addComponent(jLabel5)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(roleCombo, javax.swing.GroupLayout.PREFERRED_SIZE, 32,
javax.swing.GroupLayout.PREFERRED_SIZE)

            .addGap(18, 18, 18)
            .addComponent(registerButton, javax.swing.GroupLayout.PREFERRED_SIZE, 37,
javax.swing.GroupLayout.PREFERRED_SIZE)

            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(loginButton, javax.swing.GroupLayout.PREFERRED_SIZE, 37,
javax.swing.GroupLayout.PREFERRED_SIZE)

            .addContainerGap(20, Short.MAX_VALUE))
        );

        pack();
        setLocationRelativeTo(null);
    }

    private void registerAction() {
        String name = nameField.getText();
        String email = emailField.getText();
        String pass = String.valueOf(passwordField.getPassword());
        String confirm = String.valueOf(confirmPasswordField.getPassword());
        String role = roleCombo.getSelectedItem().toString();

        if (!pass.equals(confirm)) {
            JOptionPane.showMessageDialog(this, "Passwords do not match!");
            return;
        }

        if (UserDAO.findByEmail(email) != null) {
            JOptionPane.showMessageDialog(this, "Email already exists!");
            return;
        }
```

```
        boolean success = UserDAO.register(new User(name, email, pass, role));

        if (success) {
            JOptionPane.showMessageDialog(this, "Registered Successfully!");
            new LoginPage().setVisible(true);
            this.dispose();
        } else {
            JOptionPane.showMessageDialog(this, "Failed to register!");
        }
    }

    private javax.swing.JTextField nameField;
    private javax.swing.JTextField emailField;
    private javax.swing.JPasswordField passwordField;
    private javax.swing.JPasswordField confirmPasswordField;
    private javax.swing.JComboBox<String> roleCombo;
    private javax.swing.JButton registerButton;
    private javax.swing.JButton loginButton;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
}
```

3. Admin Dashboard

```
package com.mycompany.vsp2;

import com.mycompany.vsp2.dao.UserDAO;
import com.mycompany.vsp2.model.User;
import com.mycompany.vsp2.ui.LoginPage;
import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.sql.SQLException;
import java.util.List;

public class AdminDashboard extends javax.swing.JFrame {

    private DefaultTableModel model;
    private User admin;
    private JButton logoutBtn;
    private JButton pendingBtn;
```

```
private JButton approveBtn;

public AdminDashboard() {
    initComponents();
    model = (DefaultTableModel) userTable.getModel();
    refreshBtn.addActionListener(e -> loadUsers());
    addBtn.addActionListener(e -> addUserDialog());
    deleteBtn.addActionListener(e -> deleteSelected());
    logoutBtn.addActionListener(e -> logout());
    loadUsers();
}

public AdminDashboard(User admin) {
    this.admin = admin;
    initComponents();
    model = (DefaultTableModel) userTable.getModel();
    refreshBtn.addActionListener(e -> loadUsers());
    addBtn.addActionListener(e -> addUserDialog());
    deleteBtn.addActionListener(e -> deleteSelected());
    logoutBtn.addActionListener(e -> logout());
    pendingBtn.addActionListener(e -> loadPendingUsers());
    approveBtn.addActionListener(e -> approveSelectedUser());

    loadUsers();
}

private void logout() {
    int opt = JOptionPane.showConfirmDialog(this,
            "Are you sure you want to logout?",
            "Logout", JOptionPane.YES_NO_OPTION);

    if (opt == JOptionPane.YES_OPTION) {
        this.dispose();
        new LoginPage().setVisible(true);
    }
}

private void loadUsers() {
    model.setRowCount(0);
    try {
        List<User> users = UserDAO.findAll();
        for (User u : users) {
            model.addRow(new Object[]{u.getId(), u.getName(), u.getEmail(), u.getRole()});
        }
    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this, "Failed to load users: " + ex.getMessage());
    }
```

```
    }

    private void addUserDialog() {
        JTextField name = new JTextField();
        JTextField email = new JTextField();
        JTextField pw = new JTextField();
        String[] roles = {"student", "teacher", "admin"};
        JComboBox<String> roleBox = new JComboBox<>(roles);

        Object[] fields = {"Name", name, "Email", email, "Password", pw, "Role", roleBox};

        int r = JOptionPane.showConfirmDialog(this, fields, "Add user",
JOptionPane.OK_CANCEL_OPTION);
        if (r != JOptionPane.OK_OPTION) return;

        User u = new User();
        u.setName(name.getText());
        u.setEmail(email.getText());
        u.setPassword(pw.getText());
        u.setRole((String) roleBox.getSelectedItem());

        try {
            UserDAO.insert(u);
            loadUsers();
            JOptionPane.showMessageDialog(this, "User added.");
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this, "Add failed: " + ex.getMessage());
        }
    }

    private void deleteSelected() {
        int sel = userTable.getSelectedRow();
        if (sel < 0) {
            JOptionPane.showMessageDialog(this, "Select a row.");
            return;
        }
        int id = (int) model.getValueAt(sel, 0);
        int confirm = JOptionPane.showConfirmDialog(this, "Delete user id " + id + "?", "Confirm",
JOptionPane.YES_NO_OPTION);
        if (confirm != JOptionPane.YES_OPTION) return;

        try {
            UserDAO.deleteById(id);
            loadUsers();
            JOptionPane.showMessageDialog(this, "Deleted.");
        } catch (SQLException ex) {
            JOptionPane.showMessageDialog(this, "Delete failed: " + ex.getMessage());
```

```
        }
    }
    private void loadPendingUsers() {
    model.setRowCount(0);

    try {
        List<User> pending = UserDAO.getPendingUsers();

        for (User u : pending) {
            model.addRow(new Object[]{
                    u.getId(),
                    u.getName(),
                    u.getEmail(),
                    u.getRole() + " (Pending)"
            });
        }

        if (pending.isEmpty()) {
            JOptionPane.showMessageDialog(this, "No pending users.");
        }

    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "Failed: " + e.getMessage());
    }
}

    private void approveSelectedUser() {
    int row = userTable.getSelectedRow();
    if (row < 0) {
        JOptionPane.showMessageDialog(this, "Select a user first.");
        return;
    }

    int id = (int) model.getValueAt(row, 0);

    int confirm = JOptionPane.showConfirmDialog(
            this,
            "Approve user ID " + id + "?",
            "Confirm",
            JOptionPane.YES_NO_OPTION
    );

    if (confirm != JOptionPane.YES_OPTION) return;

    try {
        UserDAO.approveUser(id);
        JOptionPane.showMessageDialog(this, "User approved!");
```

```
        loadPendingUsers(); // refresh
    } catch (Exception e) {
        JOptionPane.showMessageDialog(this, "Error: " + e.getMessage());
    }
}



    private javax.swing.JTable userTable;
    private javax.swing.JButton refreshBtn;
    private javax.swing.JButton addBtn;
    private javax.swing.JButton deleteBtn;

    private void initComponents() {

        setTitle("Admin Dashboard");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(700, 500);
        setLocationRelativeTo(null);
        setLayout(new BorderLayout());

        // --- TOP PANEL WITH LOGOUT BUTTON ---
        JPanel topBar = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        logoutBtn = new JButton("Logout");
        topBar.add(logoutBtn);
        add(topBar, BorderLayout.NORTH);

        // --- MAIN TABLE ---
        JScrollPane scrollPane = new JScrollPane();
        userTable = new javax.swing.JTable();
        JPanel bottom = new JPanel();
        refreshBtn = new javax.swing.JButton();
        addBtn = new javax.swing.JButton();
        deleteBtn = new javax.swing.JButton();

        userTable.setModel(new DefaultTableModel(
            new Object[][] {},
            new String[]{"ID","Name","Email","Role"}) {
            public boolean isCellEditable(int row, int col) { return false; }
        });

        scrollPane.setViewportView(userTable);

        // --- BOTTOM BUTTONS ---
        refreshBtn.setText("Home");
        addBtn.setText("Add");
        deleteBtn.setText("Delete");
```

```
        pendingBtn = new JButton("Pending Users");
        approveBtn = new JButton("Approve Selected");

        bottom.add(refreshBtn);
        bottom.add(addBtn);
        bottom.add(deleteBtn);
        bottom.add(pendingBtn);
        bottom.add(approveBtn);

        add(scrollPane, BorderLayout.CENTER);
        add(bottom, BorderLayout.SOUTH);

        pack();
    }
}
```

4. Student Dashboard

```
package com.mycompany.vsp2;

import com.mycompany.vsp2.dao.ClassDAO;
import com.mycompany.vsp2.dao.MarksDAO;
import com.mycompany.vsp2.dao.MaterialDAO;
import com.mycompany.vsp2.dao.DatabaseConnection;
import com.mycompany.vsp2.model.Material;
import com.mycompany.vsp2.model.User;
import com.mycompany.vsp2.ui.LoginPage;
import com.mycompany.vsp2.ui.PDFViewerDialog;
import com.mycompany.vsp2.ui.PDFViewerPanel;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.File;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.List;

public class StudentDashboard extends javax.swing.JFrame {

    private User student;
```

```
// Card names
private static final String CARD_HOME = "home";
private static final String CARD_CLASSES = "classes";
private static final String CARD_MATERIALS = "materials";
private static final String CARD_MARKS = "marks";
private static final String CARD_PROGRESS = "progress";

// UI components (sidebar)
private JPanel sidebar;
private JButton btnHome;
private JButton btnClasses;
private JButton btnMaterials;
private JButton btnLibrary;
private JButton btnMarks;
private JButton btnProgress;
private JButton btnJoinClass;

// main card panel
private JPanel cardPanel;
private CardLayout cardLayout;

// Home panel
private JLabel lblWelcome;

// Classes panel
private JPanel classesListPanel;
private JScrollPane classesScroll;

// Materials panel
private DefaultTableModel materialsModel;
private JTable materialsTable;

// Marks panel
private DefaultTableModel marksModel;
private JTable marksTable;

// Progress panel
private JProgressBar progressBar;
private JLabel progressLabel;
private JPanel progressContainer;

public StudentDashboard() {
    initComponents();
}

public StudentDashboard(User u) {
```

```
    this.student = u;
    initComponents();
    lblWelcome.setText("Welcome, " + (student != null ? student.getName() : "Student"));
    showHome();
}

private void initComponents() {
    setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    setTitle("Student Dashboard");
    setSize(900, 600);
    setLocationRelativeTo(null);
    setLayout(new BorderLayout());

    // Sidebar
    sidebar = new JPanel();
    sidebar.setLayout(new GridBagLayout());
    sidebar.setBackground(new Color(245,245,245));
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.gridx = 0;
    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.weightx = 1;
    gbc.insets = new Insets(8,8,8,8);


    btnHome = new JButton("Home");
    btnClasses = new JButton("Classes");
    btnMaterials = new JButton("Materials");
    btnLibrary = new JButton("Library");
    btnMarks = new JButton("Marks");
    btnProgress = new JButton("Progress");
    btnJoinClass = new JButton("Join Class");

    // LOGOUT BUTTON
    JButton btnLogout = new JButton("Logout");

    gbc.gridy = 0; sidebar.add(btnHome, gbc);
    gbc.gridy = 1; sidebar.add(btnClasses, gbc);
    gbc.gridy = 2; sidebar.add(btnMaterials, gbc);
    gbc.gridy = 3; sidebar.add(btnLibrary, gbc);
    gbc.gridy = 4; sidebar.add(btnMarks, gbc);
    gbc.gridy = 5; sidebar.add(btnProgress, gbc);

    gbc.gridy = 6; sidebar.add(btnJoinClass, gbc);
    gbc.gridy = 7; sidebar.add(btnLogout, gbc);

    add(sidebar, BorderLayout.WEST);
```

```
        // Wire logout
        btnLogout.addActionListener(e -> {
            new LoginPage().setVisible(true);
            this.dispose();
        });

        // Card panel (right)
        cardLayout = new CardLayout();
        cardPanel = new JPanel(cardLayout);

        // Home panel
        JPanel home = new JPanel(new BorderLayout());
        lblWelcome = new JLabel("Welcome, Student");
        lblWelcome.setFont(lblWelcome.getFont().deriveFont(18f));
        JPanel welcomeWrap = new JPanel(new FlowLayout(FlowLayout.LEFT));
        welcomeWrap.add(lblWelcome);
        home.add(welcomeWrap, BorderLayout.NORTH);
        JTextArea homeText = new JTextArea(
            "This is your student dashboard. Use the sidebar to navigate.\n\n" +
            "Classes: view and join classes.\nMaterials: view materials.\nMarks: see your
marks.\nProgress: average marks as percentage."
        );
        homeText.setEditable(false);
        homeText.setBackground(getBackground());
        home.add(homeText, BorderLayout.CENTER);

        // Classes panel
        JPanel classesPanel = new JPanel(new BorderLayout());
        classesListPanel = new JPanel();
        classesListPanel.setLayout(new BoxLayout(classesListPanel, BoxLayout.Y_AXIS));
        classesScroll = new JScrollPane(classesListPanel);
        classesPanel.add(classesScroll, BorderLayout.CENTER);
        JButton refreshClassesBtn = new JButton("Refresh Classes");
        classesPanel.add(refreshClassesBtn, BorderLayout.NORTH);

        // Materials panel
        JPanel materialsPanel = new JPanel(new BorderLayout());
        // 4th column (storedPath) will be hidden from user
        materialsModel = new DefaultTableModel(new String[]{"Class","Title","File","storedPath"},
0) {
            @Override
            public boolean isCellEditable(int row, int column) {
                return false;
            }
        };
        materialsTable = new JTable(materialsModel);
        materialsTable.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

```
materialsTable.getColumnModel().getColumn(3).setMinWidth(0);
materialsTable.getColumnModel().getColumn(3).setMaxWidth(0);
materialsTable.getColumnModel().getColumn(3).setPreferredWidth(0);

materialsPanel.add(new JScrollPane(materialsTable), BorderLayout.CENTER);
JButton refreshMaterialsBtn = new JButton("Refresh Materials");
materialsPanel.add(refreshMaterialsBtn, BorderLayout.NORTH);

// double-click -> open viewer dialog
materialsTable.addMouseListener(new MouseAdapter() {
   @Override
   public void mouseClicked(MouseEvent e) {
      if (e.getClickCount() == 2) {
         int row = materialsTable.getSelectedRow();
         if (row < 0) return;

         String className = (String) materialsModel.getValueAt(row, 0);
         String title = (String) materialsModel.getValueAt(row, 1);
         String storedPath = (String) materialsModel.getValueAt(row, 3);

         try {
            File f = new File(storedPath);
            if (!f.exists()) {
               JOptionPane.showMessageDialog(StudentDashboard.this,
                  "File missing on disk:\n" + storedPath,
                  "File missing", JOptionPane.WARNING_MESSAGE);
               return;
            }

            PDFViewerDialog dlg = new PDFViewerDialog(StudentDashboard.this, f);
            dlg.setVisible(true);

         } catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(StudentDashboard.this,
               "Error opening file: " + ex.getMessage(),
               "Error", JOptionPane.ERROR_MESSAGE);
         }
      }
   }
});

// Library panel
JPanel libraryPanel = new JPanel(new BorderLayout());

DefaultTableModel libraryModel = new DefaultTableModel(
      new String[]{"Title","Author","File","storedPath"}, 0
```

```
) {
   @Override public boolean isCellEditable(int r, int c) { return false; }
};

JTable libraryTable = new JTable(libraryModel);
libraryTable.getColumnModel().getColumn(3).setMinWidth(0);
libraryTable.getColumnModel().getColumn(3).setMaxWidth(0);

libraryPanel.add(new JScrollPane(libraryTable), BorderLayout.CENTER);
JButton refreshLibraryBtn = new JButton("Refresh Library");
libraryPanel.add(refreshLibraryBtn, BorderLayout.NORTH);

libraryTable.addMouseListener(new MouseAdapter() {
   @Override
   public void mouseClicked(MouseEvent e) {
      if (e.getClickCount() == 2) {
         int row = libraryTable.getSelectedRow();
         if (row < 0) return;

         String storedPath = (String) libraryModel.getValueAt(row, 3);

         File f = new File(storedPath);
         if (!f.exists()) {
            JOptionPane.showMessageDialog(StudentDashboard.this,
                  "Missing file:\n" + storedPath);
            return;
         }

         new PDFViewerDialog(StudentDashboard.this, f).setVisible(true);
      }
   }
});


// Marks panel
JPanel marksPanel = new JPanel(new BorderLayout());
marksModel = new DefaultTableModel(new String[]{"Class","Marks"}, 0);
marksTable = new JTable(marksModel);
marksPanel.add(new JScrollPane(marksTable), BorderLayout.CENTER);
JButton refreshMarksBtn = new JButton("Refresh Marks");
marksPanel.add(refreshMarksBtn, BorderLayout.NORTH);

// Progress panel (subject-wise)
JPanel progPanel = new JPanel(new BorderLayout());

JLabel progressTitle = new JLabel("Subject-wise Progress", SwingConstants.CENTER);
progressTitle.setFont(new java.awt.Font("Segoe UI", java.awt.Font.BOLD, 18));
```

```
        progPanel.add(progressTitle, BorderLayout.NORTH);

        // Container for progress bars
        progressContainer = new JPanel();
        progressContainer.setLayout(new BoxLayout(progressContainer, BoxLayout.Y_AXIS));
        JScrollPane progScroll = new JScrollPane(progressContainer);
        progPanel.add(progScroll, BorderLayout.CENTER);

        JButton refreshProgressBtn = new JButton("Refresh Progress");
        progPanel.add(refreshProgressBtn, BorderLayout.SOUTH);


        // Add cards
        cardPanel.add(home, CARD_HOME);
        cardPanel.add(classesPanel, CARD_CLASSES);
        cardPanel.add(materialsPanel, CARD_MATERIALS);
        cardPanel.add(libraryPanel, "library");
        cardPanel.add(marksPanel, CARD_MARKS);
        cardPanel.add(progPanel, CARD_PROGRESS);

        add(cardPanel, BorderLayout.CENTER);

        // Buttons
        btnHome.addActionListener(e -> showHome());
        btnClasses.addActionListener(e -> showClasses());
        btnMaterials.addActionListener(e -> showMaterialsCard());
        btnLibrary.addActionListener(e -> {
        cardLayout.show(cardPanel, "library");
        loadLibrary(libraryModel);
    });
        btnMarks.addActionListener(e -> showMarksCard());
        btnProgress.addActionListener(e -> showProgressCard());
        btnJoinClass.addActionListener(e -> joinClassDialog());

        refreshClassesBtn.addActionListener(e -> loadJoinedClasses());
        refreshMaterialsBtn.addActionListener(e -> loadMaterialsForJoined());
        refreshMarksBtn.addActionListener(e -> loadMarks());
        refreshProgressBtn.addActionListener(e -> loadIndividualProgress());

        cardLayout.show(cardPanel, CARD_HOME);
    }

    private void showHome() { cardLayout.show(cardPanel, CARD_HOME); }
    private void showClasses() { cardLayout.show(cardPanel, CARD_CLASSES);
loadJoinedClasses(); }
    private void showMaterialsCard() { cardLayout.show(cardPanel, CARD_MATERIALS);
loadMaterialsForJoined(); }
```

```
   private void showMarksCard() { cardLayout.show(cardPanel, CARD_MARKS); loadMarks(); }
   private void showProgressCard() { cardLayout.show(cardPanel, CARD_PROGRESS);
loadIndividualProgress(); }

   private void joinClassDialog() {
      try {
         List<ClassDAO.ClassSummary> classes = ClassDAO.listAllWithId();
         if (classes.isEmpty()) {
            JOptionPane.showMessageDialog(this, "No classes available.");
            return;
         }
         StringBuilder sb = new StringBuilder();
         for (ClassDAO.ClassSummary cs : classes)
            sb.append(cs.id).append(": ").append(cs.name).append("\n");

         String input = JOptionPane.showInputDialog(
            this,  "\nEnter class ID to join:"
         );

         if (input == null || input.trim().isEmpty()) return;

         int cid = Integer.parseInt(input.trim());

         try (Connection con = DatabaseConnection.getConnection();
              PreparedStatement ps = con.prepareStatement("INSERT INTO class_students (classId,
studentId) VALUES (?, ?)")) {
            ps.setInt(1, cid);
            ps.setInt(2, student.getId());
            ps.executeUpdate();
            JOptionPane.showMessageDialog(this, "Joined class successfully.");
            loadJoinedClasses();
            loadMaterialsForJoined();
            loadMarks();
            updateProgress();
         }
      } catch (Exception ex) {
         JOptionPane.showMessageDialog(this, "Failed to join class: " + ex.getMessage());
      }
   }

   private JPanel makeClassBlock(int id, String title, boolean joined) {
   JPanel p = new JPanel(new BorderLayout());
   p.setBorder(BorderFactory.createTitledBorder(title));

   JLabel lbl = new JLabel(title + (joined ? " (Joined)" : ""));
   p.add(lbl, BorderLayout.CENTER);
```

```
    JPanel btnPanel = new JPanel(new FlowLayout(FlowLayout.RIGHT));

    if (joined) {
        // === NEW: UPLOADS BUTTON ===
        JButton uploadsBtn = new JButton("Uploads");
        uploadsBtn.addActionListener(e -> openClassUploads(id, title));
        btnPanel.add(uploadsBtn);
    }

    if (!joined && id > 0) {
        JButton join = new JButton("Join +");
        join.addActionListener(e -> {
            try (Connection con = DatabaseConnection.getConnection();
                 PreparedStatement ps = con.prepareStatement(
                         "INSERT INTO class_students (classId, studentId) VALUES (?, ?)"
                 )) {
                ps.setInt(1, id);
                ps.setInt(2, student.getId());
                ps.executeUpdate();
                JOptionPane.showMessageDialog(this, "Joined " + title);
                loadJoinedClasses();
                loadMaterialsForJoined();
                loadMarks();
                updateProgress();
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(this, "Failed to join: " + ex.getMessage());
            }
        });
        btnPanel.add(join);
    }

    p.add(btnPanel, BorderLayout.EAST);

    p.setMaximumSize(new Dimension(Integer.MAX_VALUE, 70));
    return p;
}
    private void openClassUploads(int classId, String className) {
        JDialog dlg = new JDialog(this, "Uploads for " + className, true);
        dlg.setSize(600, 400);
        dlg.setLocationRelativeTo(this);

        JPanel panel = new JPanel(new BorderLayout());

        DefaultTableModel model = new DefaultTableModel(
                new String[]{"Title", "File Name", "Path"}, 0
        ) {
            @Override
```

```
        public boolean isCellEditable(int r, int c) { return false; }
    };

    JTable table = new JTable(model);
    table.getColumnModel().getColumn(2).setMinWidth(0);
    table.getColumnModel().getColumn(2).setMaxWidth(0);

    // Double click to open file
    table.addMouseListener(new MouseAdapter() {
        @Override
        public void mouseClicked(MouseEvent e) {
            if (e.getClickCount() == 2) {
                int row = table.getSelectedRow();
                if (row < 0) return;

                String path = (String) model.getValueAt(row, 2);
                try {
                    File f = new File(path);
                    if (!f.exists()) {
                        JOptionPane.showMessageDialog(dlg,
                                "File not found:\n" + path,
                                "Missing File", JOptionPane.WARNING_MESSAGE);
                        return;
                    }
                    new PDFViewerDialog(StudentDashboard.this, f).setVisible(true);
                } catch (Exception ex) {
                    JOptionPane.showMessageDialog(dlg,
                            "Error opening: " + ex.getMessage());
                }
            }
        }
    });

// Load materials for that class only
try {
    List<Material> mats = MaterialDAO.getMaterialsByClass(classId);
    for (Material m : mats) {
        String filePath = "uploaded_materials/" + m.getStoredName();
        model.addRow(new Object[]{
                m.getTitle(),
                m.getFileName(),
                filePath
        });
    }
} catch (Exception ex) {
    ex.printStackTrace();
    JOptionPane.showMessageDialog(this, "Error loading uploads.");
```

```
    }

    panel.add(new JScrollPane(table), BorderLayout.CENTER);

    dlg.add(panel);
    dlg.setVisible(true);
}


    private void loadMaterialsForJoined() {
    materialsModel.setRowCount(0);

    try {
        if (student == null) return;

        List<ClassDAO.ClassSummary> classes = ClassDAO.getClassesByStudent(student.getId());
        if (classes == null || classes.isEmpty()) return;

        for (ClassDAO.ClassSummary cs : classes) {

            int cid = cs.id;

            List<Material> mats = MaterialDAO.getMaterialsByClass(cid);

            for (Material m : mats) {

                // Build full local file path
                String fullPath = "uploaded_materials/" + m.getStoredName();

                materialsModel.addRow(new Object[]{
                        cs.name,          // class name
                        m.getTitle(),     // material title
                        m.getFileName(),  // original user-friendly file name
                        fullPath          // hidden path for file opening
                });
            }
        }

    } catch (Exception ex) {
        ex.printStackTrace();
    }
}

    private void loadLibrary(DefaultTableModel model) {
    model.setRowCount(0);
    try {
        List<Material> books = MaterialDAO.getBooks();
```

```
        for (Material b : books) {
            String path = "uploaded_books/" + b.getStoredName();
            model.addRow(new Object[]{
                    b.getTitle(),
                    b.getAuthor(),
                    b.getFileName(),
                    path
            });
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}



    private void loadMarks() {
        marksModel.setRowCount(0);
        try {
            if (student == null) return;
            List<String> marks = MarksDAO.getMarksByStudent(student.getId());
            for (String m : marks) {
                marksModel.addRow(new Object[]{
                        m.split("->")[0].trim(),
                        m.contains("->") ? m.split("->")[1].trim() : ""
                });
            }
        } catch (Exception ex) {
            marksModel.addRow(new Object[]{"Demo Class","78"});
        }
    }

    private void updateProgress() {
    loadIndividualProgress();  // Refresh all subject bars
    }

    private void loadIndividualProgress() {
    progressContainer.removeAll();

    try {
        List<ClassDAO.ClassSummary> classes =
                ClassDAO.getClassesByStudent(student.getId());

        if (classes == null || classes.isEmpty()) {
            progressContainer.add(new JLabel("No classes joined."));
            progressContainer.revalidate();
            progressContainer.repaint();
```

```
            return;
        }

        for (ClassDAO.ClassSummary cs : classes) {

            int marks = getMarksForClass(student.getId(), cs.id);

            // === ROW PANEL ===
            JPanel row = new JPanel(new BorderLayout());
            row.setBorder(BorderFactory.createEmptyBorder(8, 10, 8, 10));

            // row height
            row.setPreferredSize(new Dimension(400, 45));
            row.setMaximumSize(new Dimension(Integer.MAX_VALUE, 45));


            JLabel classLabel = new JLabel(cs.name);
            classLabel.setPreferredSize(new java.awt.Dimension(150, 25));
            classLabel.setFont(new java.awt.Font("Segoe UI", java.awt.Font.BOLD, 14));

            // === Progress Bar ===
            JProgressBar pb = new JProgressBar(0, 100);
            pb.setValue(marks);
            pb.setStringPainted(true);
            pb.setString(marks + "%");


            pb.setPreferredSize(new Dimension(400, 25));
            pb.setMaximumSize(new Dimension(Integer.MAX_VALUE, 25));

            if (marks < 40) {
                pb.setForeground(new java.awt.Color(220, 53, 69));  // red
            } else if (marks < 70) {
                pb.setForeground(new java.awt.Color(255, 193, 7));  // yellow
            } else {
                pb.setForeground(new java.awt.Color(40, 167, 69));  // green
            }


            row.add(classLabel, BorderLayout.WEST);
            row.add(pb, BorderLayout.CENTER);


            progressContainer.add(row);
        }

    } catch (Exception ex) {
```

```
            ex.printStackTrace();
            progressContainer.add(new JLabel("Error loading progress."));
        }

        progressContainer.revalidate();
        progressContainer.repaint();
    }



    private int getMarksForClass(int studentId, int classId) {
    try {
        return MarksDAO.getMarksForClass(studentId, classId);
    } catch (Exception e) {
        return 0;
    }
    }

    private void loadJoinedClasses() {
    try {
        if (student == null) return;

        List<ClassDAO.ClassSummary> joined = ClassDAO.getClassesByStudent(student.getId());

        classesListPanel.removeAll();

        if (joined.isEmpty()) {
            classesListPanel.add(new JLabel("You have not joined any classes yet."));
        } else {
            for (ClassDAO.ClassSummary cs : joined) {
                classesListPanel.add(makeClassBlock(cs.id, cs.name, true));
            }
        }

        classesListPanel.revalidate();
        classesListPanel.repaint();

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


    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new StudentDashboard().setVisible(true));
    }
}
```

5. Teachers Dashboard

```
package com.mycompany.vsp2;

import com.mycompany.vsp2.dao.ClassDAO;
import com.mycompany.vsp2.dao.MaterialDAO;
import com.mycompany.vsp2.dao.MarksDAO;
import com.mycompany.vsp2.dao.UserDAO;
import com.mycompany.vsp2.model.Material;
import com.mycompany.vsp2.model.User;
import com.mycompany.vsp2.ui.LoginPage;
import com.mycompany.vsp2.ui.PDFViewerDialog;

import javax.swing.*;
import javax.swing.table.DefaultTableModel;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.io.File;
import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.util.List;


public class TeacherDashboard extends javax.swing.JFrame {

    private User teacher;

    // Card names
    private static final String CARD_HOME = "home";
    private static final String CARD_CLASSES = "classes";
    private static final String CARD_UPLOAD = "upload";
    private static final String CARD_EVALUATE = "evaluate";
    private static final String CARD_PERFORMANCE = "performance";
    private static final String CARD_CREATE = "create";

    // Sidebar components
    private JPanel sidebar;
    private JButton btnHome;
    private JButton btnClasses;
    private JButton btnUpload;
    private JButton btnEvaluate;
    private JButton btnPerformance;
    private JButton btnCreateClass;
    private JButton btnLogout;
```

```
//library
private JButton btnLibrary;
private JButton btnUploadBook;
private DefaultTableModel libraryModel;
private JButton chooseBookBtn;
private JButton uploadBookBtn;
private JTextField bookTitle;
private JTextField bookAuthor;
private JTextField bookPath;


// Main card panel
private JPanel cardPanel;
private CardLayout cardLayout;

// --- Classes panel UI ---
private JPanel classesListPanel;
private JScrollPane classesScroll;
private JButton refreshClassesBtn;

// --- Upload panel UI ---
private JComboBox<ClassDAO.ClassSummary> uploadClassCombo;
private JTextField uploadTitleField;
private JTextField uploadPathField;
private JButton chooseFileBtn;
private JButton uploadBtn;

// --- Evaluate panel UI ---
private JComboBox<User> evalStudentCombo;
private JComboBox<ClassDAO.ClassSummary> evalClassCombo;
private JTextField evalMarksField;
private JButton evalSubmitBtn;

// --- Performance panel UI ---
private JComboBox<User> perfStudentCombo;
private DefaultTableModel perfTableModel;
private JTable perfTable;
private JButton perfRefreshBtn;

// --- Create class panel UI ---
private JTextField createClassNameField;
private JButton createClassBtn;

public TeacherDashboard() {
    this(null);
}
```

```
  public TeacherDashboard(User u) {
  this.teacher = u;
  initComponents();
  wireActions();

  // initial loads — now filtered by teacher ID
  loadClassesList();
  refreshUploadClasses();
  refreshEvalCombos();
  refreshPerfStudents();
}


  private void initComponents() {
     setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
     setTitle("Teacher Dashboard");
     setSize(900, 600);
     setLocationRelativeTo(null);
     setLayout(new BorderLayout());

     // --- Sidebar (left) similar to StudentDashboard style ---
     sidebar = new JPanel(new GridBagLayout());
     sidebar.setBackground(new Color(245, 245, 245));
     sidebar.setPreferredSize(new Dimension(220, getHeight()));

     GridBagConstraints gbc = new GridBagConstraints();
     gbc.gridx = 0;
     gbc.fill = GridBagConstraints.HORIZONTAL;
     gbc.weightx = 1;
     gbc.insets = new Insets(8, 8, 8, 8);

     JLabel title = new JLabel("Teacher Menu", SwingConstants.CENTER);
     title.setFont(new Font("Segoe UI", Font.BOLD, 16));

     btnHome = new JButton("Home");
     btnClasses = new JButton("Classes");
     btnUpload = new JButton("Upload Material");
     btnLibrary = new JButton("Library");
     btnUploadBook = new JButton("Upload Book");
     btnEvaluate = new JButton("Evaluate");
     btnPerformance = new JButton("Student Progress");
     btnCreateClass = new JButton("Create Class");
     btnLogout = new JButton("Logout");

     gbc.gridy = 0;
     sidebar.add(title, gbc);
     gbc.gridy = 1;
```

```
        sidebar.add(btnHome, gbc);
        gbc.gridy = 2;
        sidebar.add(btnClasses, gbc);
        gbc.gridy = 3;
        sidebar.add(btnUpload, gbc);
        gbc.gridy = 4;
        sidebar.add(btnEvaluate, gbc);
        gbc.gridy = 5;
        sidebar.add(btnPerformance, gbc);
        gbc.gridy = 6;
        sidebar.add(btnCreateClass, gbc);
        gbc.gridy = 7; sidebar.add(btnLibrary, gbc);
        gbc.gridy = 8; sidebar.add(btnUploadBook, gbc);
        gbc.gridy = 9; sidebar.add(btnLogout, gbc);

        add(sidebar, BorderLayout.WEST);

        // --- Card panel (right) ---
        cardLayout = new CardLayout();
        cardPanel = new JPanel(cardLayout);

        // --- Home card ---
        JPanel home = new JPanel(new BorderLayout());
        JLabel welcomeLabel = new JLabel("Welcome, " + (teacher != null ? teacher.getName() :
"Teacher"));
        welcomeLabel.setFont(welcomeLabel.getFont().deriveFont(20f));
        JPanel welcomeWrap = new JPanel(new FlowLayout(FlowLayout.LEFT));
        welcomeWrap.add(welcomeLabel);
        home.add(welcomeWrap, BorderLayout.NORTH);

        JTextArea homeText = new JTextArea(
                "This is your teacher dashboard. Use the sidebar to navigate.\n\n" +
                        "Classes: view classes and enrolled students.\n" +
                        "Upload Material: upload files for a class.\n" +
                        "Evaluate: set marks for students.\n" +
                        "Student Progress: view marks per class for a student.\n" +
                        "Create Class: create a new classroom."
        );
        homeText.setEditable(false);
        homeText.setBackground(getBackground());
        home.add(homeText, BorderLayout.CENTER);

        // --- Classes card ---
        JPanel classesPanel = new JPanel(new BorderLayout());
        classesListPanel = new JPanel();
        classesListPanel.setLayout(new BoxLayout(classesListPanel, BoxLayout.Y_AXIS));
        classesScroll = new JScrollPane(classesListPanel);
```

```
        classesPanel.add(classesScroll, BorderLayout.CENTER);
        refreshClassesBtn = new JButton("Refresh Classes");
        classesPanel.add(refreshClassesBtn, BorderLayout.NORTH);

        // --- Upload card ---
        JPanel uploadPanel = new JPanel(new GridBagLayout());
        GridBagConstraints ugb = new GridBagConstraints();
        ugb.insets = new Insets(6,6,6,6);
        ugb.gridx = 0; ugb.gridy = 0; ugb.anchor = GridBagConstraints.WEST;
        uploadPanel.add(new JLabel("Select Class:"), ugb);
        uploadClassCombo = new JComboBox<>();
        ugb.gridx = 1; ugb.fill = GridBagConstraints.HORIZONTAL; ugb.weightx = 1;
        uploadPanel.add(uploadClassCombo, ugb);

        ugb.gridx = 0; ugb.gridy = 1; ugb.fill = GridBagConstraints.NONE; ugb.weightx = 0;
        uploadPanel.add(new JLabel("Title:"), ugb);
        uploadTitleField = new JTextField();
        ugb.gridx = 1; ugb.fill = GridBagConstraints.HORIZONTAL;
        uploadPanel.add(uploadTitleField, ugb);

        ugb.gridx = 0; ugb.gridy = 2; ugb.fill = GridBagConstraints.NONE;
        uploadPanel.add(new JLabel("File:"), ugb);
        JPanel pathRow = new JPanel(new BorderLayout(6,0));
        uploadPathField = new JTextField();
        uploadPathField.setEditable(false);
        chooseFileBtn = new JButton("Choose...");
        pathRow.add(uploadPathField, BorderLayout.CENTER);
        pathRow.add(chooseFileBtn, BorderLayout.EAST);
        ugb.gridx = 1; ugb.fill = GridBagConstraints.HORIZONTAL;
        uploadPanel.add(pathRow, ugb);

        ugb.gridx = 1; ugb.gridy = 3; ugb.fill = GridBagConstraints.NONE; ugb.anchor =
GridBagConstraints.EAST;
        uploadBtn = new JButton("Upload");
        uploadPanel.add(uploadBtn, ugb);

        // --- Evaluate card ---
        JPanel evalPanel = new JPanel(new GridBagLayout());
        GridBagConstraints egb = new GridBagConstraints();
        egb.insets = new Insets(6,6,6,6);
        egb.gridx = 0; egb.gridy = 0; egb.anchor = GridBagConstraints.WEST;
        evalPanel.add(new JLabel("Select Student:"), egb);
        evalStudentCombo = new JComboBox<>();
        egb.gridx = 1; egb.fill = GridBagConstraints.HORIZONTAL; egb.weightx = 1;
        evalPanel.add(evalStudentCombo, egb);

        egb.gridx = 0; egb.gridy = 1; egb.fill = GridBagConstraints.NONE; egb.weightx = 0;
```

```
        evalPanel.add(new JLabel("Select Class:"), egb);
        evalClassCombo = new JComboBox<>();
        egb.gridx = 1; egb.fill = GridBagConstraints.HORIZONTAL;
        evalPanel.add(evalClassCombo, egb);

        egb.gridx = 0; egb.gridy = 2; egb.fill = GridBagConstraints.NONE;
        evalPanel.add(new JLabel("Marks:"), egb);
        evalMarksField = new JTextField();
        egb.gridx = 1; egb.fill = GridBagConstraints.HORIZONTAL;
        evalPanel.add(evalMarksField, egb);

        egb.gridx = 1; egb.gridy = 3; egb.fill = GridBagConstraints.NONE; egb.anchor =
GridBagConstraints.EAST;
        evalSubmitBtn = new JButton("Set Marks");
        evalPanel.add(evalSubmitBtn, egb);

        // --- Performance card ---
        JPanel perfPanel = new JPanel(new BorderLayout());
        JPanel perfTop = new JPanel(new FlowLayout(FlowLayout.LEFT));
        perfTop.add(new JLabel("Select Student:"));
        perfStudentCombo = new JComboBox<>();
        perfTop.add(perfStudentCombo);
        perfRefreshBtn = new JButton("Refresh");
        perfTop.add(perfRefreshBtn);
        perfPanel.add(perfTop, BorderLayout.NORTH);

        perfTableModel = new DefaultTableModel(new String[]{"Class","Marks"}, 0);
        perfTable = new JTable(perfTableModel);
        perfPanel.add(new JScrollPane(perfTable), BorderLayout.CENTER);

        // --- Create Class card ---
        JPanel createPanel = new JPanel(new GridBagLayout());
        GridBagConstraints cgb = new GridBagConstraints();
        cgb.insets = new Insets(6,6,6,6);
        cgb.gridx = 0; cgb.gridy = 0; cgb.anchor = GridBagConstraints.WEST;
        createPanel.add(new JLabel("Class Name:"), cgb);
        createClassNameField = new JTextField();
        createClassNameField.setColumns(20);
        cgb.gridx = 1; cgb.fill = GridBagConstraints.HORIZONTAL;
        createPanel.add(createClassNameField, cgb);
        cgb.gridx = 1; cgb.gridy = 1; cgb.fill = GridBagConstraints.NONE; cgb.anchor =
GridBagConstraints.EAST;
        createClassBtn = new JButton("Create");
        createPanel.add(createClassBtn, cgb);

        // --- Library panel ---
        JPanel libraryPanel = new JPanel(new BorderLayout());
```

```
// Use the class-level libraryModel field
libraryModel = new DefaultTableModel(
        new String[]{"Title", "Author", "File", "storedPath"}, 0
) {
    @Override
    public boolean isCellEditable(int row, int col) { return false; }
};

JTable libraryTable = new JTable(libraryModel);
libraryTable.getColumnModel().getColumn(3).setMinWidth(0);
libraryTable.getColumnModel().getColumn(3).setMaxWidth(0);

libraryPanel.add(new JScrollPane(libraryTable), BorderLayout.CENTER);
JButton refreshLibraryBtn = new JButton("Refresh Library");
libraryPanel.add(refreshLibraryBtn, BorderLayout.NORTH);

libraryTable.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseClicked(MouseEvent e) {
        if (e.getClickCount() == 2) {
            int row = libraryTable.getSelectedRow();
            if (row < 0) return;

            String path = (String) libraryModel.getValueAt(row, 3);
            new PDFViewerDialog(TeacherDashboard.this, new File(path)).setVisible(true);
        }
    }
});

// --- Upload Book panel ---
JPanel uploadBookPanel = new JPanel(new GridBagLayout());
GridBagConstraints bg = new GridBagConstraints();
bg.insets = new Insets(6, 6, 6, 6);

// Initialize class-level fields
bookTitle = new JTextField(20);
bookAuthor = new JTextField(20);
bookPath = new JTextField(20);
bookPath.setEditable(false);

chooseBookBtn = new JButton("Choose...");
uploadBookBtn = new JButton("Upload Book");

// Layout
bg.gridx = 0; bg.gridy = 0; uploadBookPanel.add(new JLabel("Book Title:"), bg);
bg.gridx = 1; uploadBookPanel.add(bookTitle, bg);
```

```
    bg.gridx = 0; bg.gridy = 1; uploadBookPanel.add(new JLabel("Author:"), bg);
    bg.gridx = 1; uploadBookPanel.add(bookAuthor, bg);

    bg.gridx = 0; bg.gridy = 2; uploadBookPanel.add(new JLabel("File:"), bg);
    JPanel row = new JPanel(new BorderLayout());
    row.add(bookPath, BorderLayout.CENTER);
    row.add(chooseBookBtn, BorderLayout.EAST);

    bg.gridx = 1; uploadBookPanel.add(row, bg);

    bg.gridx = 1; bg.gridy = 3; uploadBookPanel.add(uploadBookBtn, bg);


    // Add cards to cardPanel
    cardPanel.add(home, CARD_HOME);
    cardPanel.add(classesPanel, CARD_CLASSES);
    cardPanel.add(uploadPanel, CARD_UPLOAD);
    cardPanel.add(evalPanel, CARD_EVALUATE);
    cardPanel.add(perfPanel, CARD_PERFORMANCE);
    cardPanel.add(libraryPanel, "library");
    cardPanel.add(uploadBookPanel, "uploadBook");

    cardPanel.add(createPanel, CARD_CREATE);

    add(cardPanel, BorderLayout.CENTER);

    // initial view
    cardLayout.show(cardPanel, CARD_HOME);
}

private void wireActions() {
    // sidebar navigation
    btnHome.addActionListener(e -> cardLayout.show(cardPanel, CARD_HOME));
    btnClasses.addActionListener(e -> {
        cardLayout.show(cardPanel, CARD_CLASSES);
        loadClassesList();
    });
    btnUpload.addActionListener(e -> {
        cardLayout.show(cardPanel, CARD_UPLOAD);
        refreshUploadClasses();
    });
    btnEvaluate.addActionListener(e -> {
        cardLayout.show(cardPanel, CARD_EVALUATE);
        refreshEvalCombos();
    });
    btnPerformance.addActionListener(e -> {
```

```
        cardLayout.show(cardPanel, CARD_PERFORMANCE);
        refreshPerfStudents();
    });
    btnCreateClass.addActionListener(e -> {
        cardLayout.show(cardPanel, CARD_CREATE);
    });
    btnLibrary.addActionListener(e -> {
    cardLayout.show(cardPanel, "library");
    loadLibrary(libraryModel);
});

    btnUploadBook.addActionListener(e -> {
        cardLayout.show(cardPanel, "uploadBook");
});
    btnLogout.addActionListener(e -> {
        new LoginPage().setVisible(true);
        this.dispose();
    });

    // classes panel
    refreshClassesBtn.addActionListener(e -> loadClassesList());

    // upload panel
    chooseFileBtn.addActionListener(e -> {
        JFileChooser chooser = new JFileChooser();
        int res = chooser.showOpenDialog(this);
        if (res == JFileChooser.APPROVE_OPTION) {
            File f = chooser.getSelectedFile();
            uploadPathField.setText(f.getAbsolutePath());
        }
    });

    uploadBtn.addActionListener(e -> {
    ClassDAO.ClassSummary cs = (ClassDAO.ClassSummary)
uploadClassCombo.getSelectedItem();
    if (cs == null) {
        JOptionPane.showMessageDialog(this, "Select a class first.");
        return;
    }

    String title = uploadTitleField.getText().trim();
    String path = uploadPathField.getText().trim();

    if (title.isEmpty() || path.isEmpty()) {
        JOptionPane.showMessageDialog(this, "Provide title and choose a file.");
        return;
    }
```

```
        try {
            File selectedFile = new File(path);
            if (!selectedFile.exists()) {
                JOptionPane.showMessageDialog(this, "File not found.");
                return;
            }

            // upload with teacher ID
            MaterialDAO.uploadMaterial(
                    cs.id,
                    teacher.getId(),
                    title,
                    selectedFile
            );

            JOptionPane.showMessageDialog(this, "Material uploaded successfully!");

            uploadTitleField.setText("");
            uploadPathField.setText("");

        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Upload failed: " + ex.getMessage());
        }
    });


        // evaluate panel
        evalSubmitBtn.addActionListener(e -> {
            User s = (User) evalStudentCombo.getSelectedItem();
            ClassDAO.ClassSummary cs = (ClassDAO.ClassSummary)
evalClassCombo.getSelectedItem();
            if (s == null || cs == null) {
                JOptionPane.showMessageDialog(this, "Select student and class.");
                return;
            }
            String marksStr = evalMarksField.getText().trim();
            if (marksStr.isEmpty()) {
                JOptionPane.showMessageDialog(this, "Enter marks value.");
                return;
            }
            try {
                int marks = Integer.parseInt(marksStr);
                MarksDAO.setMarks(s.getId(), cs.id, marks);
                JOptionPane.showMessageDialog(this, "Marks set.");
                evalMarksField.setText("");
            } catch (NumberFormatException nfe) {
```

```
            JOptionPane.showMessageDialog(this, "Marks must be an integer.");
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Failed to set marks: " + ex.getMessage());
        }
    });

    // performance panel
    perfRefreshBtn.addActionListener(e -> refreshPerfTable());
    perfStudentCombo.addActionListener(e -> refreshPerfTable());

    chooseBookBtn.addActionListener(e -> {
  JFileChooser ch = new JFileChooser();
  int res = ch.showOpenDialog(this);
  if (res == JFileChooser.APPROVE_OPTION) {
      File selected = ch.getSelectedFile();
      if (selected.exists()) {
          bookPath.setText(selected.getAbsolutePath());
          System.out.println("Chosen file: " + selected.getAbsolutePath());
      } else {
          JOptionPane.showMessageDialog(this, "Selected file does not exist.");
      }
  }
});

uploadBookBtn.addActionListener(e -> {
  String path = bookPath.getText().trim();
  if (path.isEmpty()) {
      JOptionPane.showMessageDialog(this, "Choose a file first.");
      return;
  }

  File file = new File(path);
  if (!file.exists()) {
      JOptionPane.showMessageDialog(this, "File not found: " + path);
      return;
  }

  try {
      MaterialDAO.uploadBook(
              bookTitle.getText().trim(),
              bookAuthor.getText().trim(),
              file,
              teacher.getId()
      );
      JOptionPane.showMessageDialog(this, "Book uploaded successfully!");
      bookTitle.setText("");
      bookAuthor.setText("");
```

```
            bookPath.setText("");
        } catch (Exception ex) {
            ex.printStackTrace(); // prints any exception in console
            JOptionPane.showMessageDialog(this, "Upload failed: " + ex.getMessage());
        }
    });



        // create class panel
        createClassBtn.addActionListener(e -> {
            String name = createClassNameField.getText().trim();
            if (name.isEmpty()) {
                JOptionPane.showMessageDialog(this, "Enter a class name.");
                return;
            }
            try {
                int teacherId = teacher != null ? teacher.getId() : 0;
                ClassDAO.createClass(name, teacherId);
                JOptionPane.showMessageDialog(this, "Class created.");
                createClassNameField.setText("");
                loadClassesList();
                refreshUploadClasses();
                refreshEvalCombos();
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(this, "Failed to create class: " + ex.getMessage());
            }
        });
    }

    // --- helper methods to populate panels ---

    private void loadClassesList() {
    classesListPanel.removeAll();

    if (teacher == null) {
        classesListPanel.add(new JLabel("Teacher not loaded."));
        return;
    }

    try {
        // Filter classes by teacher ID
        List<ClassDAO.ClassSummary> classes = ClassDAO.listClassesByTeacher(teacher.getId());

        if (classes.isEmpty()) {
            classesListPanel.add(new JLabel("You have no classes."));
        } else {
```

```
            for (ClassDAO.ClassSummary cs : classes) {
                JPanel block = new JPanel(new BorderLayout());
                block.setBorder(BorderFactory.createTitledBorder(cs.name));
                JLabel lbl = new JLabel(cs.name);
                block.add(lbl, BorderLayout.CENTER);

                JButton viewStudents = new JButton("View Students");
                viewStudents.addActionListener(e -> showStudentsInClass(cs.id, cs.name));
                block.add(viewStudents, BorderLayout.EAST);

                block.setMaximumSize(new Dimension(Integer.MAX_VALUE, 60));
                classesListPanel.add(block);
            }
        }
    } catch (Exception ex) {
        classesListPanel.add(new JLabel("Failed to load classes: " + ex.getMessage()));
    }

    classesListPanel.revalidate();
    classesListPanel.repaint();
}


    private void showStudentsInClass(int classId, String className) {
        try {
            String sql = "SELECT u.id, u.name, u.email FROM class_students cs JOIN users u ON
cs.studentId = u.id WHERE cs.classId = ?";
            StringBuilder out = new StringBuilder();
            try (Connection con = com.mycompany.vsp2.dao.DatabaseConnection.getConnection();
                 PreparedStatement ps = con.prepareStatement(sql)) {
                ps.setInt(1, classId);
                try (ResultSet rs = ps.executeQuery()) {
                    while (rs.next()) {
                        out.append(rs.getInt("id")).append(": ").append(rs.getString("name"))
                                .append(" (").append(rs.getString("email")).append(")\n");
                    }
                }
            }
            JTextArea area = new JTextArea(out.length() == 0 ? "No students in this class." :
out.toString());
            area.setEditable(false);
            JOptionPane.showMessageDialog(this, new JScrollPane(area), "Students in " + className,
JOptionPane.INFORMATION_MESSAGE);
        } catch (Exception ex) {
            JOptionPane.showMessageDialog(this, "Failed to list students: " + ex.getMessage());
        }
    }
```

```
  private void refreshUploadClasses() {
     uploadClassCombo.removeAllItems();
     try {
        List<ClassDAO.ClassSummary> classes = ClassDAO.listAllWithId();
        for (ClassDAO.ClassSummary cs : classes) uploadClassCombo.addItem(cs);
     } catch (Exception ex) {
        // ignore
     }
  }

  private void refreshEvalCombos() {
  evalStudentCombo.removeAllItems();
  evalClassCombo.removeAllItems();

  if (teacher == null) return;

  try {
     // Load classes owned by this teacher
     List<ClassDAO.ClassSummary> classes = ClassDAO.listClassesByTeacher(teacher.getId());
     for (ClassDAO.ClassSummary cs : classes)
        evalClassCombo.addItem(cs);

     // Load students from those classes
     List<User> studs = UserDAO.findStudentsOfTeacher(teacher.getId());
     for (User s : studs)
        evalStudentCombo.addItem(s);

  } catch (Exception ex) {
     ex.printStackTrace();
  }
}


  private void refreshPerfStudents() {
     perfStudentCombo.removeAllItems();
     try {
        List<User> studs = UserDAO.findStudents();
        for (User s : studs) perfStudentCombo.addItem(s);
     } catch (Exception ex) {
        // ignore
     }
  }

  private void refreshPerfTable() {
  perfTableModel.setRowCount(0);
  User sel = (User) perfStudentCombo.getSelectedItem();
```

```
    if (sel == null) return;

    try {
        String sql = "SELECT c.className, m.marks " +
                     "FROM marks m " +
                     "JOIN classroom c ON m.classId = c.classId " +
                     "WHERE m.studentId = ? AND c.teacherId = ?";

        try (Connection con = com.mycompany.vsp2.dao.DatabaseConnection.getConnection();
             PreparedStatement ps = con.prepareStatement(sql)) {

            ps.setInt(1, sel.getId());         // student
            ps.setInt(2, teacher.getId());     // teacher  ← missing line FIXED

            try (ResultSet rs = ps.executeQuery()) {
                while (rs.next()) {
                    perfTableModel.addRow(new Object[]{
                            rs.getString("className"),
                            rs.getInt("marks")
                    });
                }
            }
        }

    } catch (Exception ex) {
        JOptionPane.showMessageDialog(this,
                "Failed to load performance: " + ex.getMessage());
    }
}

        private void loadLibrary(DefaultTableModel model) {
    model.setRowCount(0);
    try {
        List<Material> books = MaterialDAO.getBooks();
        for (Material b : books) {
            String path = "uploaded_books/" + b.getStoredName();
            model.addRow(new Object[]{
                    b.getTitle(),
                    b.getAuthor(),
                    b.getFileName(),
                    path
            });
        }
    } catch (Exception ex) {
        ex.printStackTrace();
    }
}
```

```
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new TeacherDashboard(new User("Demo Teacher",
"demo@x", "p", "teacher")).setVisible(true));
    }
}
```

6. User (Model)

```
package com.mycompany.vsp2.model;

public class User {

    public static final String ROLE_STUDENT = "student";
    public static final String ROLE_TEACHER = "teacher";
    public static final String ROLE_ADMIN = "admin";

    private int id;
    private String name;
    private String email;
    private String password;
    private String role;
    private boolean isApproved;

    public User() {}

    public User(int id, String name, String email, String password, String role) {
        this.id = id;
        this.name = name;
        this.email = email;
        this.password = password;
        this.role = role;
    }

    public User(String name, String email, String password, String role) {
        this(0, name, email, password, role);
    }

    public int getId() { return id; }
    public void setId(int id) { this.id = id; }

    public String getName() { return name; }
    public void setName(String name) { this.name = name; }
```

```
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }

    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }

    public String getRole() { return role; }
    public void setRole(String role) { this.role = role; }

    public boolean isApproved() { return isApproved; }
    public void setApproved(boolean approved) { this.isApproved = approved; }

    @Override
    public String toString() {
        return name;
    }
}
```

7. User (DAO)

```
package com.mycompany.vsp2.dao;

import com.mycompany.vsp2.model.User;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class UserDAO {

    // Create new user (core method)
    public static boolean createUser(User user) {
        String sql = "INSERT INTO users (name, email, password, role, isApproved) VALUES (?, ?, ?,
?, 0)";


        try (Connection c = DatabaseConnection.getConnection();
             PreparedStatement ps = c.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {

            ps.setString(1, user.getName());
            ps.setString(2, user.getEmail());
            ps.setString(3, user.getPassword());
            ps.setString(4, user.getRole());

            int affected = ps.executeUpdate();
```

```
            if (affected == 0) return false;

            try (ResultSet keys = ps.getGeneratedKeys()) {
                if (keys.next()) {
                    user.setId(keys.getInt(1)); // Save generated ID
                }
            }

            return true;

        } catch (SQLException ex) {
            ex.printStackTrace();
            return false;
        }
    }

    public static List<User> getPendingUsers() throws SQLException {
    List<User> users = new ArrayList<>();

    String sql = "SELECT * FROM users WHERE isApproved = 0";
    try (Connection con = DatabaseConnection.getConnection();
         PreparedStatement ps = con.prepareStatement(sql);
         ResultSet rs = ps.executeQuery()) {

        while (rs.next()) {
            User u = new User(
                rs.getInt("id"),
                rs.getString("name"),
                rs.getString("email"),
                rs.getString("password"),
                rs.getString("role")
            );
            u.setApproved(false);
            users.add(u);
        }
    }
    return users;
}

    public static void approveUser(int userId) throws SQLException {
    String sql = "UPDATE users SET isApproved = 1 WHERE id = ?";
    try (Connection con = DatabaseConnection.getConnection();
         PreparedStatement ps = con.prepareStatement(sql)) {

        ps.setInt(1, userId);
        ps.executeUpdate();
    }
```

```
}


    // Wrapper used by UI that expects SQLException
    public static boolean insert(User user) throws SQLException {
        if (!createUser(user)) {
            throw new SQLException("Insert failed");
        }
        return true;
    }

    // Wrapper used by RegisterPage
    public static boolean register(User user) {
        return createUser(user);
    }

    // Login lookup
   public static User findByEmail(String email) {
    String sql = "SELECT id, name, email, password, role, isApproved FROM users WHERE email
= ?";

    try (Connection c = DatabaseConnection.getConnection();
         PreparedStatement ps = c.prepareStatement(sql)) {

        ps.setString(1, email);

        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                User u = new User(
                        rs.getInt("id"),
                        rs.getString("name"),
                        rs.getString("email"),
                        rs.getString("password"),
                        rs.getString("role")
                );
                u.setApproved(rs.getBoolean("isApproved")); // ⭐ CRITICAL
                return u;
            }
        }

    } catch (SQLException ex) {
        ex.printStackTrace();
    }

    return null;
 }
   public static User login(String email, String password) {
```

```
   String sql = "SELECT id, name, email, password, role, isApproved FROM users WHERE email
= ? AND password = ?";

   try (Connection c = DatabaseConnection.getConnection();
        PreparedStatement ps = c.prepareStatement(sql)) {

      ps.setString(1, email);
      ps.setString(2, password);

      try (ResultSet rs = ps.executeQuery()) {
         if (rs.next()) {
            User u = new User(
                  rs.getInt("id"),
                  rs.getString("name"),
                  rs.getString("email"),
                  rs.getString("password"),
                  rs.getString("role")
            );
            u.setApproved(rs.getBoolean("isApproved"));
            return u;
         }
      }

   } catch (SQLException ex) {
      ex.printStackTrace();
   }

   return null;
 }



   // List all users
   public static List<User> listAll() {
      List<User> out = new ArrayList<>();
      String sql = "SELECT id, name, email, password, role FROM users";

      try (Connection c = DatabaseConnection.getConnection();
           PreparedStatement ps = c.prepareStatement(sql);
           ResultSet rs = ps.executeQuery()) {

         while (rs.next()) {
            out.add(new User(
                  rs.getInt("id"),
                  rs.getString("name"),
                  rs.getString("email"),
                  rs.getString("password"),
```

```
                rs.getString("role")
            ));
        }

    } catch (SQLException ex) {
        ex.printStackTrace();
    }

    return out;
}

// Alias expected by UI
public static List<User> findAll() {
    return listAll();
}

// Delete a user
public static boolean deleteById(int id) throws SQLException {
    String sql = "DELETE FROM users WHERE id = ?";

    try (Connection c = DatabaseConnection.getConnection();
         PreparedStatement ps = c.prepareStatement(sql)) {

        ps.setInt(1, id);
        return ps.executeUpdate() > 0;

    } catch (SQLException ex) {
        ex.printStackTrace();
        throw ex;
    }
}

public static List<User> findStudentsOfTeacher(int teacherId) throws Exception {
List<User> list = new ArrayList<>();

String sql = """
    SELECT DISTINCT u.*
    FROM class_students cs
    JOIN classroom c ON cs.classId = c.classId
    JOIN users u ON cs.studentId = u.id
    WHERE c.teacherId = ?
""";

try (Connection con = DatabaseConnection.getConnection();
     PreparedStatement ps = con.prepareStatement(sql)) {

    ps.setInt(1, teacherId);
```

```
        ResultSet rs = ps.executeQuery();

        while (rs.next()) {
            list.add(new User(
                    rs.getInt("id"),
                    rs.getString("name"),
                    rs.getString("email"),
                    rs.getString("password"),
                    rs.getString("role")
            ));
        }
    }
    return list;
}


    // Fetch users with role=student
    public static List<User> findStudents() {
        List<User> out = new ArrayList<>();
        String sql = "SELECT id, name, email, password, role FROM users WHERE role = ?";

        try (Connection c = DatabaseConnection.getConnection();
             PreparedStatement ps = c.prepareStatement(sql)) {

            ps.setString(1, User.ROLE_STUDENT);

            try (ResultSet rs = ps.executeQuery()) {
                while (rs.next()) {
                    out.add(new User(
                            rs.getInt("id"),
                            rs.getString("name"),
                            rs.getString("email"),
                            rs.getString("password"),
                            rs.getString("role")
                    ));
                }
            }

        } catch (SQLException ex) {
            ex.printStackTrace();
        }

        return out;
    }

}
```

7. Class (DAO)

```
package com.mycompany.vsp2.dao;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class ClassDAO {

   // ----------------------------------------
   // CHECK IF CLASS NAME EXISTS
   // ----------------------------------------
   public static boolean classExists(String className) {
      String sql = "SELECT COUNT(*) FROM classroom WHERE className = ?";
      try (Connection c = DatabaseConnection.getConnection();
           PreparedStatement ps = c.prepareStatement(sql)) {

         ps.setString(1, className);
         try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) return rs.getInt(1) > 0;
         }
      } catch (SQLException ex) {
         ex.printStackTrace();
      }
      return false;
   }

   // ----------------------------------------
   // ✔ CREATE CLASS (with duplicate name check)
   // ----------------------------------------
   public static void createClass(String className, int teacherId) throws SQLException {

      if (classExists(className)) {
         throw new SQLException("A class with this name already exists!");
      }

      String sql = "INSERT INTO classroom (className, teacherId) VALUES (?, ?)";
      try (Connection c = DatabaseConnection.getConnection();
           PreparedStatement ps = c.prepareStatement(sql)) {
         ps.setString(1, className);
         ps.setInt(2, teacherId);
         ps.executeUpdate();
      }
   }

   // ----------------------------------------
```

```
// List all class names only
// ----------------------------------------
public static List<String> listAll() {
    List<String> out = new ArrayList<>();
    String sql = "SELECT className FROM classroom";
    try (Connection c = DatabaseConnection.getConnection();
         PreparedStatement ps = c.prepareStatement(sql);
         ResultSet rs = ps.executeQuery()) {

        while (rs.next()) {
            out.add(rs.getString("className"));
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return out;
}

// ----------------------------------------
// List classes a student has joined
// ----------------------------------------
public static List<ClassSummary> getClassesByStudent(int studentId) {
    List<ClassSummary> out = new ArrayList<>();
    String sql =
        "SELECT c.classId, c.className " +
        "FROM class_students cs " +
        "JOIN classroom c ON cs.classId = c.classId " +
        "WHERE cs.studentId = ?";

    try (Connection c = DatabaseConnection.getConnection();
         PreparedStatement ps = c.prepareStatement(sql)) {

        ps.setInt(1, studentId);

        try (ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                out.add(new ClassSummary(rs.getInt("classId"), rs.getString("className")));
            }
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return out;
}

// ----------------------------------------
// List all classes with ID
```

```
// ----------------------------------------
public static List<ClassSummary> listAllWithId() {
    List<ClassSummary> out = new ArrayList<>();
    String sql = "SELECT classId, className FROM classroom";

    try (Connection c = DatabaseConnection.getConnection();
         PreparedStatement ps = c.prepareStatement(sql);
         ResultSet rs = ps.executeQuery()) {

        while (rs.next()) {
            out.add(new ClassSummary(
                rs.getInt("classId"),
                rs.getString("className")
            ));
        }

    } catch (SQLException ex) {
        ex.printStackTrace();
    }
    return out;
}

// ----------------------------------------
// Get class ID by name
// ----------------------------------------
public static int getClassIdByName(String className) {
    String sql = "SELECT classId FROM classroom WHERE className = ?";

    try (Connection con = DatabaseConnection.getConnection();
         PreparedStatement ps = con.prepareStatement(sql)) {

        ps.setString(1, className);

        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) return rs.getInt("classId");
        }
    } catch (SQLException ex) {
        ex.printStackTrace();
    }

    return -1;
}

// ----------------------------------------
// CHECK IF STUDENT ALREADY ENROLLED
// ----------------------------------------
public static boolean isStudentInClass(int studentId, int classId) {
```

```
        String sql = "SELECT COUNT(*) FROM class_students WHERE studentId = ? AND classId
= ?";
        try (Connection con = DatabaseConnection.getConnection();
             PreparedStatement ps = con.prepareStatement(sql)) {

            ps.setInt(1, studentId);
            ps.setInt(2, classId);

            ResultSet rs = ps.executeQuery();
            if (rs.next()) return rs.getInt(1) > 0;

        } catch (SQLException ex) {
            ex.printStackTrace();
        }
        return false;
    }

    // ------------------------------------------
    //avoids duplicates
    // ------------------------------------------
    public static void enrollStudent(int studentId, int classId) throws SQLException {

        if (isStudentInClass(studentId, classId)) {
            throw new SQLException("Student already enrolled in this class!");
        }

        String sql = "INSERT INTO class_students (studentId, classId) VALUES (?, ?)";
        try (Connection con = DatabaseConnection.getConnection();
             PreparedStatement ps = con.prepareStatement(sql)) {

            ps.setInt(1, studentId);
            ps.setInt(2, classId);
            ps.executeUpdate();
        }
    }

    // ------------------------------------------
    // List classes by teacher
    // ------------------------------------------
    public static List<ClassSummary> listClassesByTeacher(int teacherId) throws Exception {
        List<ClassSummary> list = new ArrayList<>();

        String sql = "SELECT classId, className FROM classroom WHERE teacherId = ?";
        try (Connection con = DatabaseConnection.getConnection();
             PreparedStatement ps = con.prepareStatement(sql)) {

            ps.setInt(1, teacherId);
```

```
            ResultSet rs = ps.executeQuery();

            while (rs.next()) {
                list.add(new ClassSummary(
                        rs.getInt("classId"),
                        rs.getString("className")
                ));
            }
        }
        return list;
    }

    public static class ClassSummary {
        public final int id;
        public final String name;

        public ClassSummary(int id, String name) {
            this.id = id;
            this.name = name;
        }

        @Override
        public String toString() {
            return name;
        }
    }
}
```

8. Marks (DAO)

```
package com.mycompany.vsp2.dao;

import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class MarksDAO {

    public static void setMarks(int studentId, int classId, int marks) throws SQLException {
        String sql = "REPLACE INTO marks (studentId, classId, marks) VALUES (?, ?, ?)";

        try (Connection c = DatabaseConnection.getConnection();
             PreparedStatement ps = c.prepareStatement(sql)) {

            ps.setInt(1, studentId);
            ps.setInt(2, classId);
```

```
        ps.setInt(3, marks);
        ps.executeUpdate();
    }
}

public static List<String> getMarksByStudent(int studentId) {
    List<String> out = new ArrayList<>();

    String sql = """
        SELECT m.classId, c.className, m.marks
        FROM marks m
        JOIN classroom c ON m.classId = c.classId
        WHERE m.studentId = ?
    """;

    try (Connection c = DatabaseConnection.getConnection();
         PreparedStatement ps = c.prepareStatement(sql)) {

        ps.setInt(1, studentId);

        try (ResultSet rs = ps.executeQuery()) {
            while (rs.next()) {
                String className = rs.getString("className");
                int marks = rs.getInt("marks");
                out.add("Class: " + className + " -> " + marks);
            }
        }

    } catch (SQLException ex) {
        ex.printStackTrace();
    }

    return out;
}

public static int getProgress(int studentId) {
    String sql = "SELECT AVG(marks) AS avg_mark FROM marks WHERE studentId = ?";

    try (Connection c = DatabaseConnection.getConnection();
         PreparedStatement ps = c.prepareStatement(sql)) {

        ps.setInt(1, studentId);

        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                double avg = rs.getDouble("avg_mark");
                if (rs.wasNull()) return 0;
```

```
                return (int) Math.round(avg);
            }
        }

    } catch (SQLException ex) {
        ex.printStackTrace();
    }

    return 0;
}
public static int getMarksForClass(int studentId, int classId) {
String sql = "SELECT marks FROM marks WHERE studentId = ? AND classId = ?";

try (Connection c = DatabaseConnection.getConnection();
     PreparedStatement ps = c.prepareStatement(sql)) {

    ps.setInt(1, studentId);
    ps.setInt(2, classId);

    try (ResultSet rs = ps.executeQuery()) {
        if (rs.next()) {
            return rs.getInt("marks");
        }
    }

} catch (SQLException ex) {
    ex.printStackTrace();
}

return 0;
}

}
```

9. Material (Model)

```
package com.mycompany.vsp2.model;

import java.sql.Timestamp;

public class Material {
    private int id;
    private int classId;
    private int uploadedBy;
    private String title;
    private String fileName;
```

```
    private String storedName;
    private Timestamp uploadDate;  // NEW: matches DB
    private String author;


    public Material() {}

    public int getId() { return id; }
    public int getClassId() { return classId; }
    public int getUploadedBy() { return uploadedBy; }
    public String getTitle() { return title; }
    public String getFileName() { return fileName; }
    public String getStoredName() { return storedName; }
    public Timestamp getUploadDate() { return uploadDate; }
    public String getAuthor() { return author; }

    public void setId(int id) { this.id = id; }
    public void setClassId(int classId) { this.classId = classId; }
    public void setUploadedBy(int uploadedBy) { this.uploadedBy = uploadedBy; }
    public void setTitle(String title) { this.title = title; }
    public void setFileName(String fileName) { this.fileName = fileName; }
    public void setStoredName(String storedName) { this.storedName = storedName; }
    public void setUploadDate(Timestamp uploadDate) { this.uploadDate = uploadDate; }
    public void setAuthor(String a) { this.author = a; }
}
```

10. MAterial (DAO)

```
package com.mycompany.vsp2.dao;

import com.mycompany.vsp2.model.Material;

import java.io.File;
import java.nio.file.Files;
import java.nio.file.StandardCopyOption;
import java.sql.*;
import java.util.ArrayList;
import java.util.List;

public class MaterialDAO {

    private static final String STORAGE_DIR = "uploaded_materials";

    static {
        try {
            Files.createDirectories(new File(STORAGE_DIR).toPath());
```

```
        } catch (Exception ignored) {}
    }

    // Upload material
    public static void uploadMaterial(int classId, int teacherId, String title, File sourceFile) throws
Exception {

        String originalName = sourceFile.getName();
        String storedName = System.currentTimeMillis() + "_" + originalName;

        File dest = new File(STORAGE_DIR, storedName);

        // Copy file to storage directory
        Files.copy(sourceFile.toPath(), dest.toPath(), StandardCopyOption.REPLACE_EXISTING);

        // FIXED: correct column names and order
        String sql =
            "INSERT INTO materials (class_id, uploaded_by, title, file_name, stored_name) " +
            "VALUES (?, ?, ?, ?, ?)";

        try (Connection c = DatabaseConnection.getConnection();
             PreparedStatement ps = c.prepareStatement(sql)) {

            ps.setInt(1, classId);
            ps.setInt(2, teacherId);
            ps.setString(3, title);
            ps.setString(4, originalName);
            ps.setString(5, storedName);

            ps.executeUpdate();
        }
    }

    // Retrieve materials by class
    public static List<Material> getMaterialsByClass(int classId) throws Exception {
        List<Material> list = new ArrayList<>();

        String sql =
            "SELECT id, class_id, uploaded_by, title, file_name, stored_name, upload_date " +
            "FROM materials WHERE class_id = ? ORDER BY upload_date DESC";

        try (Connection c = DatabaseConnection.getConnection();
             PreparedStatement ps = c.prepareStatement(sql)) {

            ps.setInt(1, classId);

            try (ResultSet rs = ps.executeQuery()) {
```

```
            while (rs.next()) {
                Material m = new Material();
                m.setId(rs.getInt("id"));
                m.setClassId(rs.getInt("class_id"));
                m.setUploadedBy(rs.getInt("uploaded_by"));
                m.setTitle(rs.getString("title"));
                m.setFileName(rs.getString("file_name"));
                m.setStoredName(rs.getString("stored_name"));
                m.setUploadDate(rs.getTimestamp("upload_date"));

                list.add(m);
            }
        }
    }

    return list;
}

// Retrieve single material
public static Material getMaterial(int classId, String title) throws Exception {
    String sql = "SELECT * FROM materials WHERE class_id = ? AND title = ?";

    try (Connection c = DatabaseConnection.getConnection();
         PreparedStatement ps = c.prepareStatement(sql)) {

        ps.setInt(1, classId);
        ps.setString(2, title);

        try (ResultSet rs = ps.executeQuery()) {
            if (rs.next()) {
                Material m = new Material();
                m.setId(rs.getInt("id"));
                m.setClassId(rs.getInt("class_id"));
                m.setUploadedBy(rs.getInt("uploaded_by"));
                m.setTitle(rs.getString("title"));
                m.setFileName(rs.getString("file_name"));
                m.setStoredName(rs.getString("stored_name"));
                m.setUploadDate(rs.getTimestamp("upload_date"));

                return m;
            }
        }
    }

    return null;
}
```

```
   public static void uploadBook(String title, String author, File file, int teacherId) throws Exception
{
   if (file == null || !file.exists()) {
      throw new Exception("File does not exist.");
   }

   // Create folder if missing
   File folder = new File("uploaded_books");
   if (!folder.exists()) folder.mkdirs();

   // Create stored filename for server use
   String storedName = System.currentTimeMillis() + "_" + file.getName();
   File dest = new File(folder, storedName);

   // Copy file to server folder
   try (java.io.FileInputStream in = new java.io.FileInputStream(file);
        java.io.FileOutputStream out = new java.io.FileOutputStream(dest)) {

      byte[] buffer = new byte[1024];
      int len;
      while ((len = in.read(buffer)) > 0)
         out.write(buffer, 0, len);
   }

   // Insert DB record
   String sql = "INSERT INTO books (title, author, fileName, storedName, teacherId) VALUES (?,
?, ?, ?, ?)";

   try (Connection con = DatabaseConnection.getConnection();
        PreparedStatement ps = con.prepareStatement(sql)) {

      ps.setString(1, title);
      ps.setString(2, author);
      ps.setString(3, file.getName());   // original file name
      ps.setString(4, storedName);       // stored file name
      ps.setInt(5, teacherId);

      ps.executeUpdate();
   }
}



   public static List<Material> getBooks() throws Exception {
   List<Material> list = new java.util.ArrayList<>();

   String sql = "SELECT * FROM books ORDER BY uploadedAt DESC";
```

```
    try (Connection con = DatabaseConnection.getConnection();
         PreparedStatement ps = con.prepareStatement(sql);
         ResultSet rs = ps.executeQuery()) {

        while (rs.next()) {
            Material m = new Material();
            m.setId(rs.getInt("id"));
            m.setTitle(rs.getString("title"));
            m.setAuthor(rs.getString("author"));
            m.setFileName(rs.getString("fileName"));
            m.setStoredName(rs.getString("storedName"));
            list.add(m);
        }
    }

    return list;
}

}
```

11. PDF viewer dialog

```
package com.mycompany.vsp2.ui;

import javax.swing.*;
import java.awt.*;
import java.io.File;

public class PDFViewerDialog extends JDialog {

    public PDFViewerDialog(Window owner, File pdfFile) {
        super(owner, "PDF Viewer - " + pdfFile.getName(), ModalityType.APPLICATION_MODAL);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setLayout(new BorderLayout());

        PDFViewerPanel panel = new PDFViewerPanel(pdfFile);
        add(panel, BorderLayout.CENTER);

        JPanel bottom = new JPanel(new FlowLayout(FlowLayout.RIGHT));
        JButton close = new JButton("Close");
        close.addActionListener(e -> dispose());
        bottom.add(close);
        add(bottom, BorderLayout.SOUTH);

        setSize(900, 700);
        setLocationRelativeTo(owner);
```

```
    }
}
```

12. PDF viewer panel

```
package com.mycompany.vsp2.ui;

import org.apache.pdfbox.pdmodel.PDDocument;
import org.apache.pdfbox.rendering.PDFRenderer;

import javax.swing.*;
import java.awt.*;
import java.awt.image.BufferedImage;
import java.io.File;

public class PDFViewerPanel extends JPanel {

    public PDFViewerPanel(File pdfFile) {
        setLayout(new BorderLayout());

        try {
            PDDocument document = PDDocument.load(pdfFile);
            PDFRenderer renderer = new PDFRenderer(document);

            BufferedImage pageImage = renderer.renderImageWithDPI(0, 100);

            JLabel imageLabel = new JLabel(new ImageIcon(pageImage));
            add(new JScrollPane(imageLabel), BorderLayout.CENTER);

            document.close();

        } catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(this, "Failed to load PDF: " + ex.getMessage());
        }
    }
}
```

13. Database Connection

```
package com.mycompany.vsp2.dao;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

```
public class DatabaseConnection {

   private static Connection connection;
   private static final String URL =
"jdbc:mysql://localhost:3306/vsp_db?useSSL=false&serverTimezone=UTC";
   private static final String USER = "root";
   private static final String PASSWORD = "Kksolive1234";

   private DatabaseConnection() { }

   public static synchronized Connection getConnection() throws SQLException {
      if (connection == null || connection.isClosed()) {
         try {
            Class.forName("com.mysql.cj.jdbc.Driver");
         } catch (ClassNotFoundException ex) {
            System.err.println("MySQL Driver not found. Add connector/J to classpath.");
            ex.printStackTrace();
            throw new SQLException("JDBC Driver missing", ex);
         }
         connection = DriverManager.getConnection(URL, USER, PASSWORD);
      }
      return connection;
   }
}
```

14. Main Function

```
package com.mycompany.vsp2;

import com.mycompany.vsp2.ui.LoginPage;

import javax.swing.SwingUtilities;

public class VSP2 {

   public static void main(String[] args) {
      SwingUtilities.invokeLater(() -> {
         new LoginPage().setVisible(true);
      });
   }
}
```

15. Database for the Virtual Study Platform Software

```
CREATE DATABASE IF NOT EXISTS vsp_db CHARACTER SET utf8mb4 COLLATE
utf8mb4_general_ci;

USE vsp_db;

-- USERS TABLE
CREATE TABLE IF NOT EXISTS users (
  id INT PRIMARY KEY AUTO_INCREMENT,
  name VARCHAR(255),
  email VARCHAR(255) UNIQUE,
  password VARCHAR(255),
  role ENUM('teacher','student','admin') DEFAULT 'student'
);

INSERT INTO users (name, email, password, role) VALUES
('Admin User', 'admin@example.com', 'admin123', 'admin'),
('Alice Teacher', 'alice.teacher@example.com', 'teacher123', 'teacher'),
('Bob Teacher', 'bob.teacher@example.com', 'teacher123', 'teacher'),
('Charlie Student', 'charlie.student@example.com', 'student123', 'student'),
('Diana Student', 'diana.student@example.com', 'student123', 'student'),
('Edward Student', 'edward.student@example.com', 'student123', 'student');


-- CLASSROOM TABLE
CREATE TABLE IF NOT EXISTS classroom (
  classId INT PRIMARY KEY AUTO_INCREMENT,
  className VARCHAR(255),
  teacherId INT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (teacherId) REFERENCES users(id) ON DELETE SET NULL
);

-- CLASS STUDENTS
CREATE TABLE IF NOT EXISTS class_students (
  id INT PRIMARY KEY AUTO_INCREMENT,
  classId INT,
  studentId INT,
  created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  FOREIGN KEY (classId) REFERENCES classroom(classId) ON DELETE CASCADE,
  FOREIGN KEY (studentId) REFERENCES users(id) ON DELETE CASCADE
);


CREATE TABLE IF NOT EXISTS marks (
  studentId INT,
  classId INT,
  marks INT,
```

```
  PRIMARY KEY (studentId, classId),
  FOREIGN KEY (studentId) REFERENCES users(id) ON DELETE CASCADE,
  FOREIGN KEY (classId) REFERENCES classroom(classId) ON DELETE CASCADE
);


CREATE TABLE materials (
    id INT AUTO_INCREMENT PRIMARY KEY,
    class_id INT NOT NULL,
    title VARCHAR(255) NOT NULL,
    description TEXT,
    file_name VARCHAR(255) NOT NULL,
    stored_name VARCHAR(255) NOT NULL,
    uploaded_by INT NOT NULL,
    upload_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,

    FOREIGN KEY (class_id) REFERENCES classroom(classId),
    FOREIGN KEY (uploaded_by) REFERENCES users(id)
);


CREATE TABLE IF NOT EXISTS books (
    id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(255) NOT NULL,
    author VARCHAR(255),
    fileName VARCHAR(255) NOT NULL,
    storedName VARCHAR(255) NOT NULL,
    teacherId INT,
    uploadedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (teacherId) REFERENCES users(id) ON DELETE SET NULL
);


SET FOREIGN_KEY_CHECKS = 0;
TRUNCATE TABLE class_students;
TRUNCATE TABLE materials;
TRUNCATE TABLE marks;
TRUNCATE TABLE classroom;
TRUNCATE TABLE users;

SET FOREIGN_KEY_CHECKS = 1;
```