

**Title: Online Course Registration System**

CS313 Advanced Programming Language

Section: 5C9

|  |  |  |
| --- | --- | --- |
| **No.** | **Student name** | **Student Number** |
| 1 | Leen Alsanad |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |

**Year: 1447**

|  |  |
| --- | --- |
| **Content** | **Obtained mark** |
| **Project Description** |  |
| **Class Diagram** |  |
| **GUI Design** |  |
| **Database Tables (Schema, Creation, explanation)** |  |
| **Total Score** |  |

Table of Contents

[**1.**](#_heading=h.mjcndnlhzd2f) **Project Description:** 3

[**2.**](#_heading=h.8cafqw5j7xws) **System hierarchy:** 4

[**3.**](#_heading=h.gbun8tmtwn9t) **Class Diagram:** 4

[**4. Database:** 5](#_heading=h.4i4y5o2e8z7)

[**5.**](#_heading=h.3y1ku2o126es) **GUI Design:** 5

[**6. Code:** 19](#_heading=h.jibdr19weqyl)

## **Project Description:**

The aim of this project is to develop an Online Course Registration System using JavaFX technology. The system provides secure, role-based login access for three types of users: Students, Instructors, and Administrators, each with specific privileges.

· Students can view available courses, register for courses, and drop courses from their schedules.

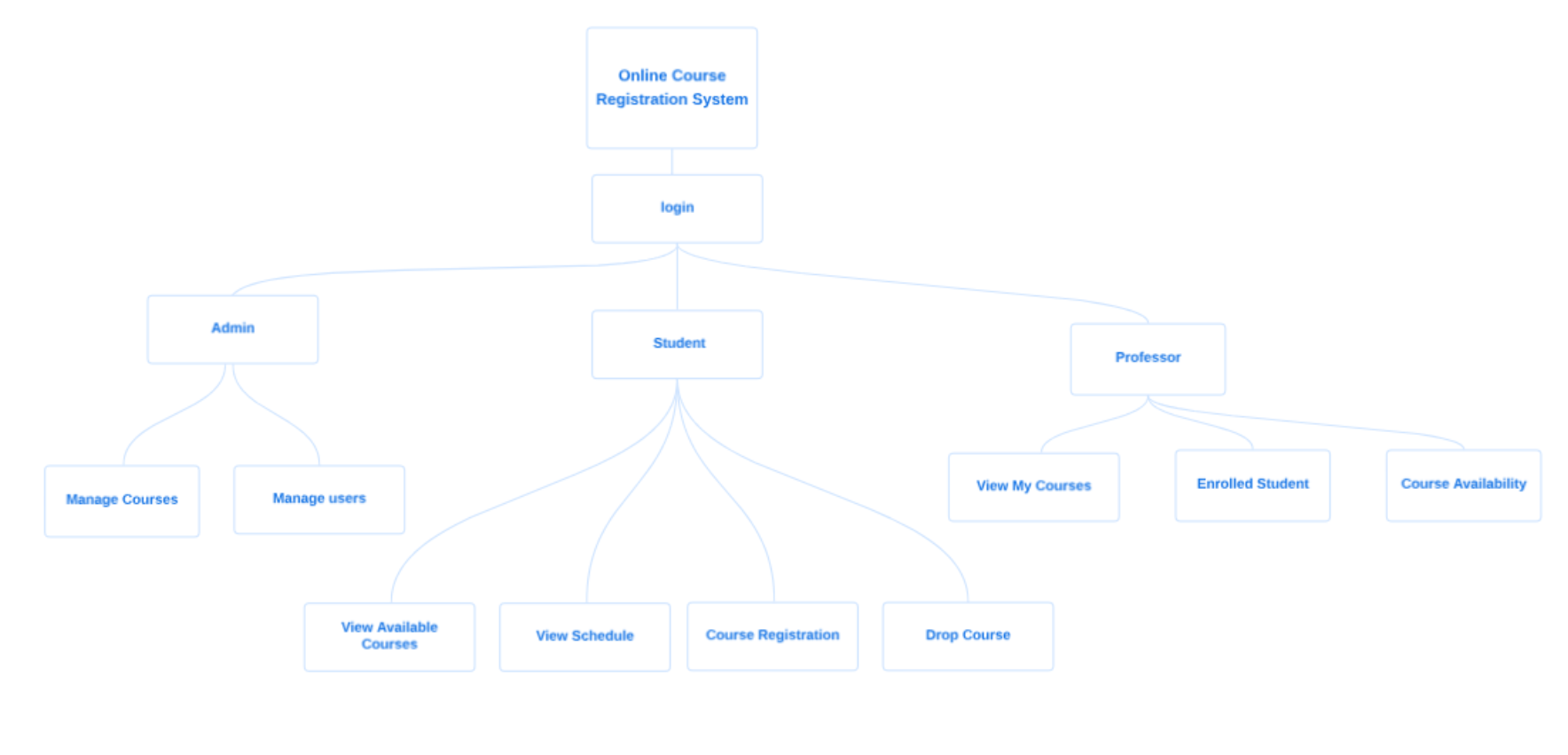
· Instructors can manage the courses they teach, view lists of enrolled students, and update course availability.

· Administrators have full oversight of the system, with the ability to manage user accounts (create, update, delete), manage the course catalog, and oversee all registration records.

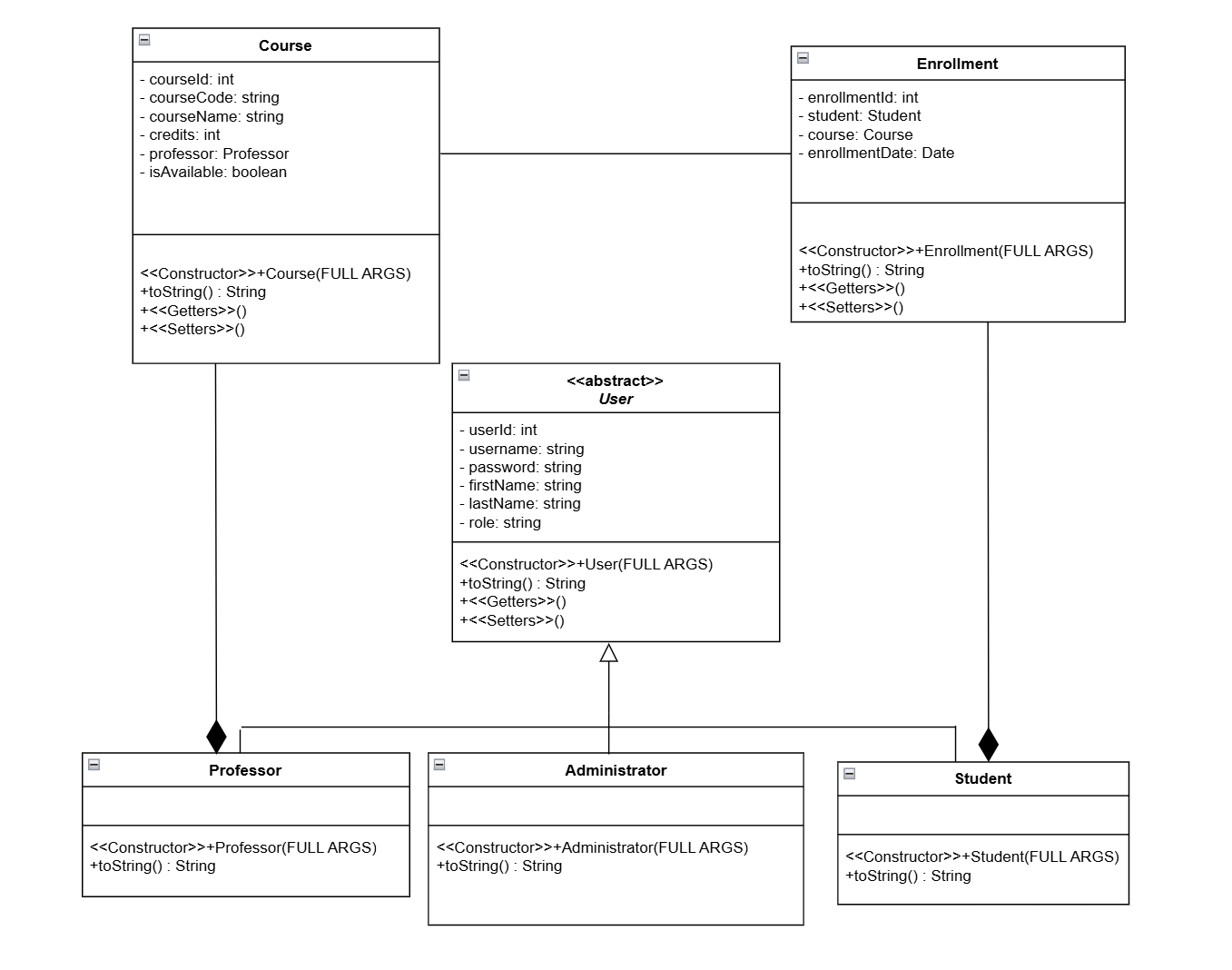
To support these functions, the system uses a MySQL database as its backend to store and manage all data related to users, courses, and enrollments. The frontend is built using JavaFX and Scene Builder, which together create an intuitive and responsive Graphical User Interface (GUI). The GUI includes dedicated screens for login, course listing, and registration operations. It connects directly to the database, enabling real-time data interactions and ensuring that all users see up-to-date information.

This project is structured into multiple Java classes that handle the core system logic, including data models, database access (DAO classes), and application controllers. This modular architecture, supported by several well-designed GUI forms, ensures an efficient, robust, and user-friendly system that provides an optimal course registration experience.

## **System hierarchy:**

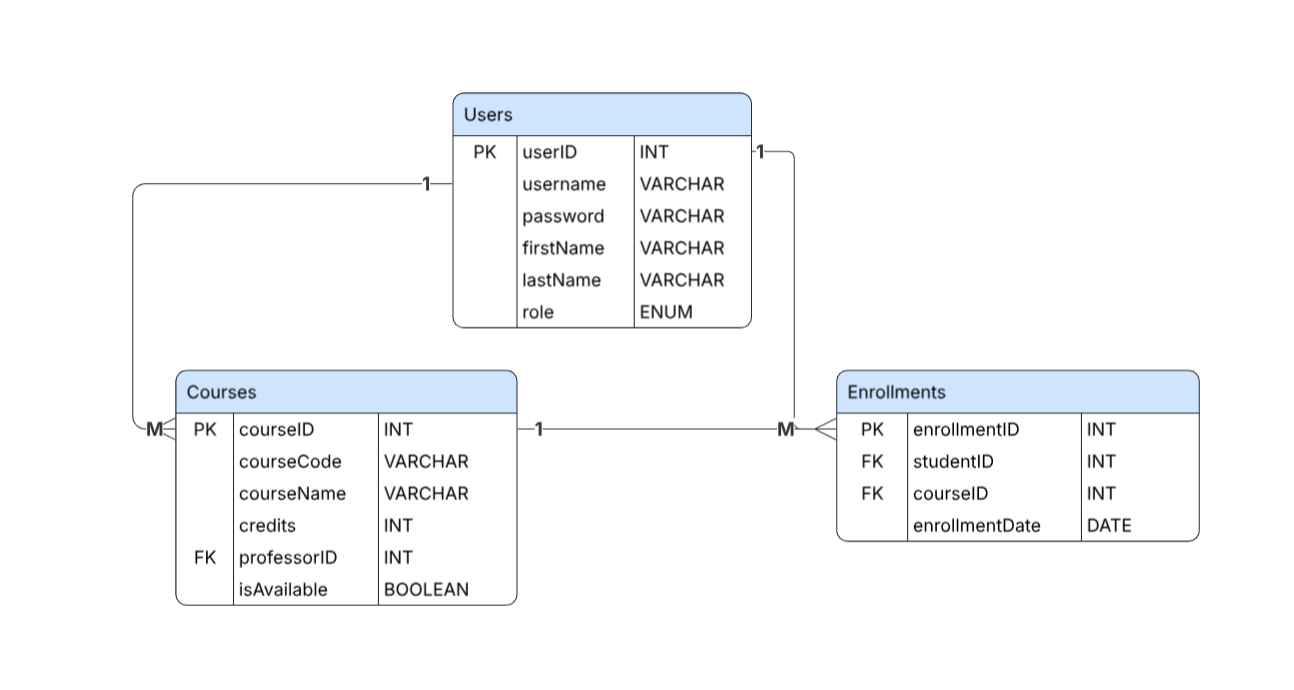


## **Class Diagram:**

****

## 

## **4. Database:**

****

## **GUI Design:**

**صورة تحتوي على نص, برمجيات, برامج الوسائط المتعددة, لقطة شاشة

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.صورة تحتوي على نص, لقطة شاشة, برامج الوسائط المتعددة, برمجيات

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.**

**صورة تحتوي على نص, لقطة شاشة, زهري, أرجواني

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.**

**صورة تحتوي على نص, لقطة شاشة, برمجيات, صفحة ويب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.**

**صورة تحتوي على نص, برمجيات, صفحة ويب, أيقونة الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.**

**صورة تحتوي على نص, لقطة شاشة, برمجيات, صفحة ويب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.**

**صورة تحتوي على نص, برمجيات, لقطة شاشة

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.**

**صورة تحتوي على نص, برمجيات, أيقونة الحاسوب, صفحة ويب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.**

صورة تحتوي على نص, برمجيات, لقطة شاشة

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, برمجيات, لقطة شاشة

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, برمجيات, أيقونة الحاسوب, صفحة ويب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, لقطة شاشة, برمجيات, برامج الوسائط المتعددة

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, لقطة شاشة, التصميم

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, برمجيات, صفحة ويب, أيقونة الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, برمجيات, أيقونة الحاسوب, صفحة ويب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, برمجيات, أيقونة الحاسوب, صفحة ويب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, برمجيات, أيقونة الحاسوب, صفحة ويب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, برمجيات, أيقونة الحاسوب, نظام التشغيل

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, لقطة شاشة, برمجيات, صفحة ويب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, لقطة شاشة, برمجيات, برامج الوسائط المتعددة

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, لقطة شاشة, الخط, التصميم

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على لقطة شاشة, نص, برمجيات

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, لقطة شاشة, برمجيات, صفحة ويب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, لقطة شاشة, برمجيات, أيقونة الحاسوب

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على لقطة شاشة, نص

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على نص, لقطة شاشة, التصميم

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على لقطة شاشة, نص, برمجيات

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

صورة تحتوي على لقطة شاشة, نص

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

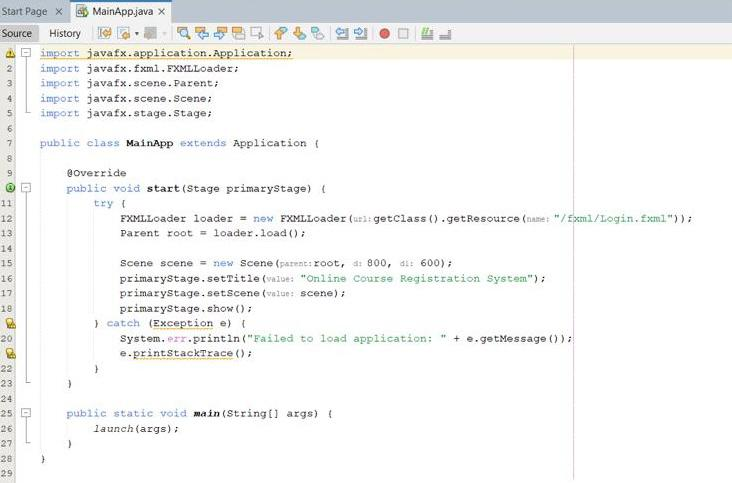
صورة تحتوي على لقطة شاشة, نص

قد يكون المحتوى الذي تم إنشاؤه بواسطة الذكاء الاصطناعي غير صحيح.

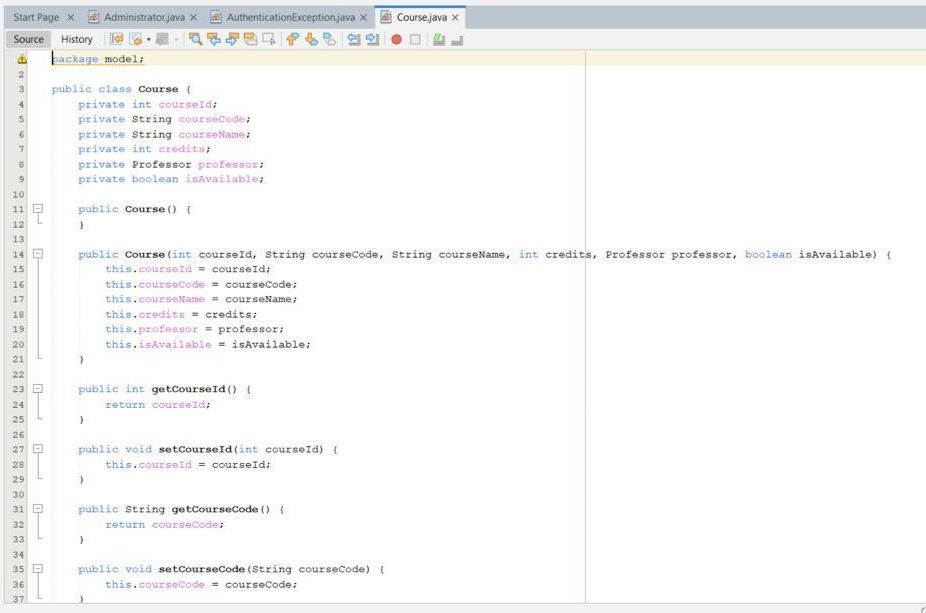
## **6. Code:**

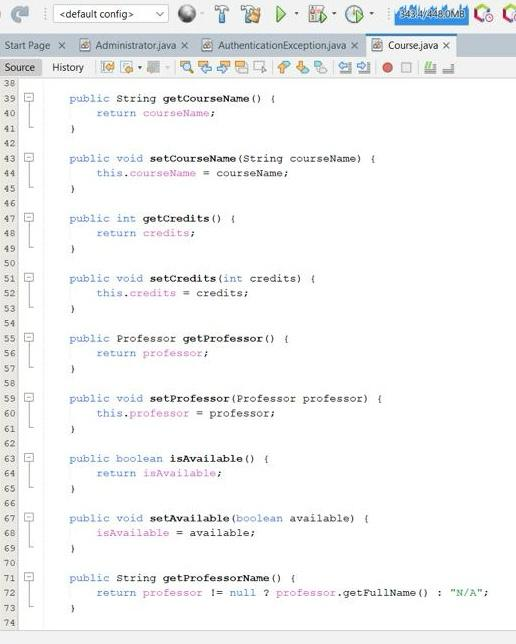
**-Classes**

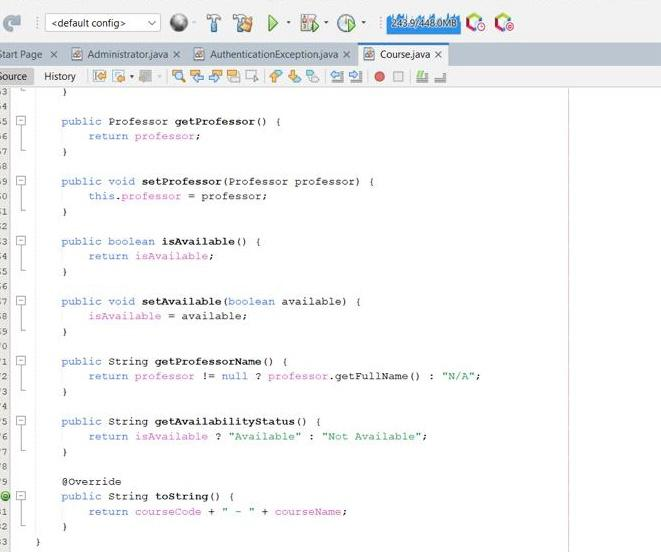
Main:



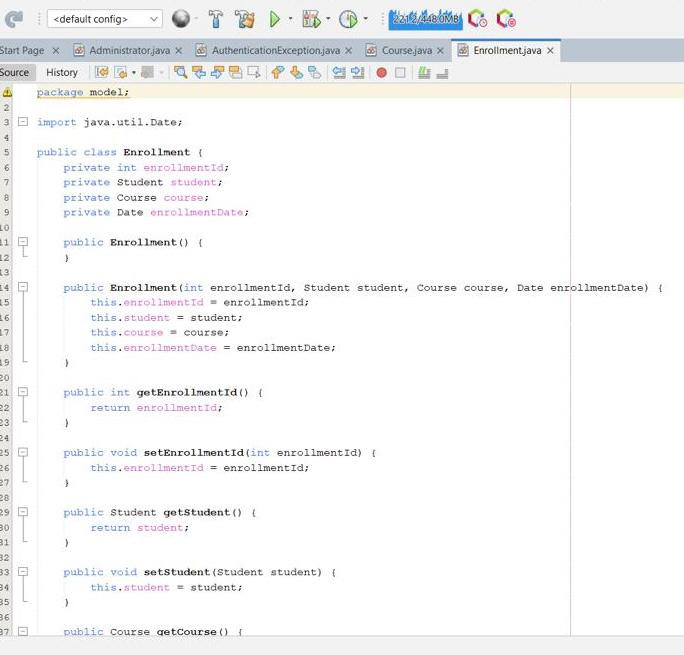
Course:

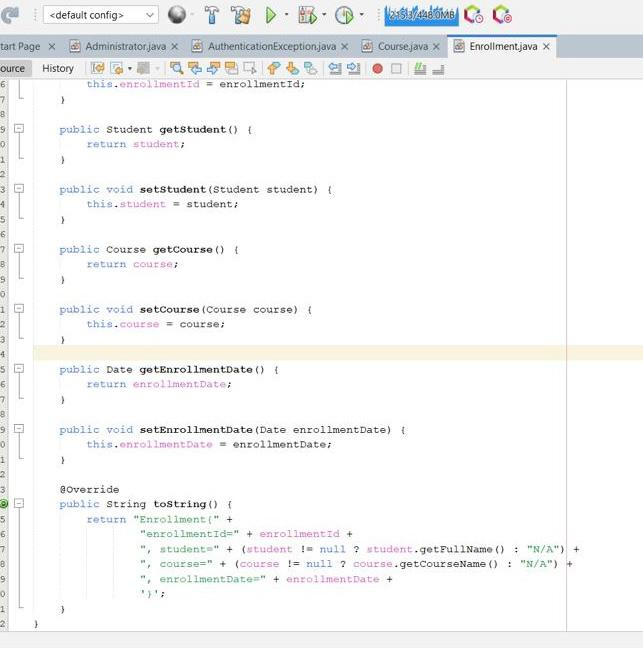


****

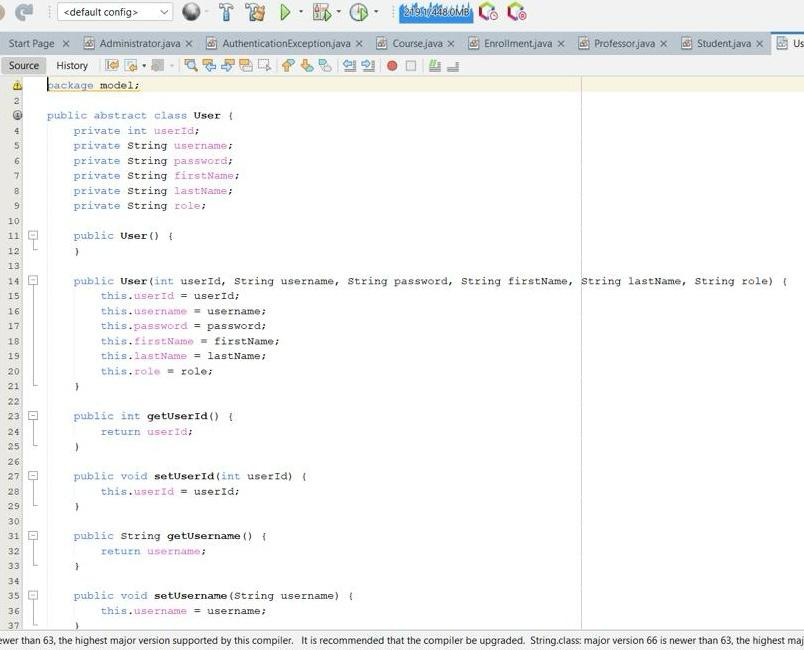


Enrollment:



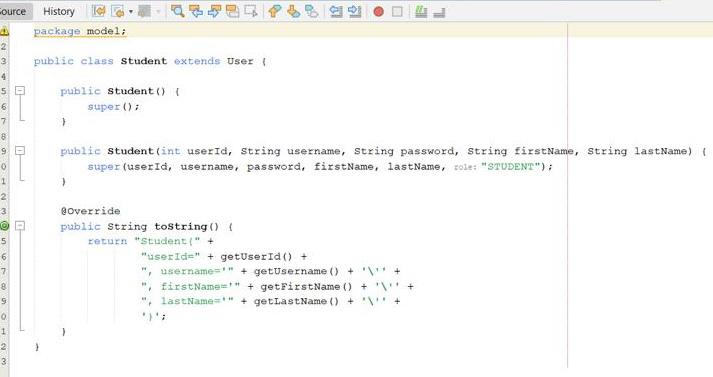


User:

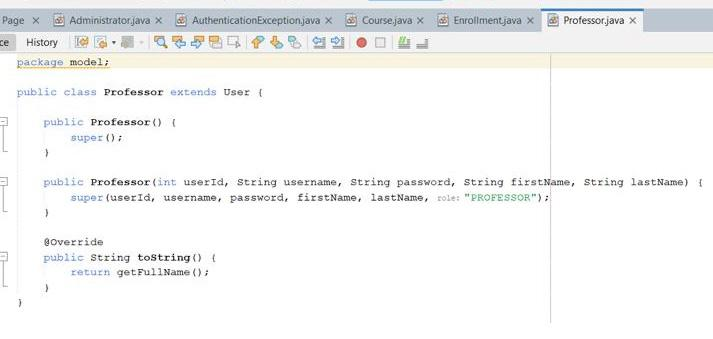




Student:



Professor:



Authentication Exception:



Administrator:



**-Appendix**

**DataBase:**

**Manage Database:**

public interface Manageable<T> {

void add(T entity) throws DatabaseException;

void update(T entity) throws DatabaseException;

void delete(int id) throws DatabaseException;

}

**Database exception:**

public class DatabaseException extends Exception {

public DatabaseException(String message) {

super(message);

}

public DatabaseException(String message, Throwable cause) {

super(message, cause);}}

**Database connection:**

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DatabaseConnection {

private static final String DB\_URL = "jdbc:oracle:thin:@LAPTOP-PC6KPOAP:1522/XE";

private static final String DB\_USERNAME = "system";

private static final String DB\_PASSWORD = "2005";

public static Connection getConnection() throws DatabaseException {

try {

Class.forName("oracle.jdbc.driver.OracleDriver");

return DriverManager.getConnection(DB\_URL, DB\_USERNAME, DB\_PASSWORD);

} catch (ClassNotFoundException e) {

throw new DatabaseException("Oracle JDBC Driver not found", e);

} catch (SQLException e) {

throw new DatabaseException("Failed to connect to database", e);

}

}

public static void closeConnection(Connection conn) {

try {

if (conn != null && !conn.isClosed()) {

conn.close();

}

} catch (SQLException e) {

System.err.println("Error closing connection: " + e.getMessage());

}}}

**Course Database:**

import model.Course;

import model.Professor;

import java.sql.\*;

import java.util.ArrayList;

public class CourseDAO implements Manageable<Course> {

private UserDAO userDAO = new UserDAO();

public ArrayList<Course> getAllCourses() throws DatabaseException {

ArrayList<Course> courses = new ArrayList<>();

String sql = "SELECT \* FROM Courses ORDER BY course\_code";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql);

ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

courses.add(createCourseFromResultSet(rs));

}

return courses;

} catch (SQLException e) {

System.err.println("SQL Error in getAllCourses: " + e.getMessage());

e.printStackTrace();

throw new DatabaseException("Failed to retrieve courses: " + e.getMessage(), e);

} catch (Exception e) {

System.err.println("Error in getAllCourses: " + e.getMessage());

e.printStackTrace();

throw new DatabaseException("Failed to retrieve courses: " + e.getMessage());

}

}

public ArrayList<Course> getAvailableCourses() throws DatabaseException {

ArrayList<Course> courses = new ArrayList<>();

String sql = "SELECT \* FROM Courses WHERE is\_available = 1 ORDER BY course\_code";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql);

ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

courses.add(createCourseFromResultSet(rs));

}

return courses;

} catch (SQLException e) {

throw new DatabaseException("Failed to retrieve available courses", e);

}

}

public ArrayList<Course> getCoursesByProfessor(int professorId) throws DatabaseException {

ArrayList<Course> courses = new ArrayList<>();

String sql = "SELECT \* FROM Courses WHERE professor\_id = ? ORDER BY course\_code";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, professorId);

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

courses.add(createCourseFromResultSet(rs));

}

return courses;

} catch (SQLException e) {

throw new DatabaseException("Failed to retrieve professor courses", e);

}

}

public Course getCourseById(int courseId) throws DatabaseException {

String sql = "SELECT \* FROM Courses WHERE course\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, courseId);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return createCourseFromResultSet(rs);

}

return null;

} catch (SQLException e) {

throw new DatabaseException("Failed to retrieve course", e);

}

}

@Override

public void add(Course course) throws DatabaseException {

String sql = "INSERT INTO Courses (course\_id, course\_code, course\_name, credits, professor\_id, is\_available) " +

"VALUES (course\_seq.NEXTVAL, ?, ?, ?, ?, ?)";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setString(1, course.getCourseCode());

stmt.setString(2, course.getCourseName());

stmt.setInt(3, course.getCredits());

// Set professor\_id as NULL if no professor assigned

if (course.getProfessor() != null) {

stmt.setInt(4, course.getProfessor().getUserId());

} else {

stmt.setNull(4, java.sql.Types.INTEGER);

}

stmt.setInt(5, course.isAvailable() ? 1 : 0);

stmt.executeUpdate();

} catch (SQLException e) {

throw new DatabaseException("Failed to add course: " + e.getMessage(), e);

}

}

@Override

public void update(Course course) throws DatabaseException {

String sql = "UPDATE Courses SET course\_code = ?, course\_name = ?, credits = ?, professor\_id = ?, is\_available = ? " +

"WHERE course\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setString(1, course.getCourseCode());

stmt.setString(2, course.getCourseName());

stmt.setInt(3, course.getCredits());

// Set professor\_id as NULL if no professor assigned

if (course.getProfessor() != null) {

stmt.setInt(4, course.getProfessor().getUserId());

} else {

stmt.setNull(4, java.sql.Types.INTEGER);

}

stmt.setInt(5, course.isAvailable() ? 1 : 0);

stmt.setInt(6, course.getCourseId());

stmt.executeUpdate();

} catch (SQLException e) {

throw new DatabaseException("Failed to update course: " + e.getMessage(), e);

}

}

public void updateAvailability(int courseId, boolean isAvailable) throws DatabaseException {

String sql = "UPDATE Courses SET is\_available = ? WHERE course\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, isAvailable ? 1 : 0);

stmt.setInt(2, courseId);

stmt.executeUpdate();

} catch (SQLException e) {

throw new DatabaseException("Failed to update course availability", e);

}

}

@Override

public void delete(int courseId) throws DatabaseException {

String sql = "DELETE FROM Courses WHERE course\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, courseId);

stmt.executeUpdate();

} catch (SQLException e) {

throw new DatabaseException("Failed to delete course", e);

}

}

private Course createCourseFromResultSet(ResultSet rs) throws SQLException, DatabaseException {

Course course = new Course();

course.setCourseId(rs.getInt("course\_id"));

course.setCourseCode(rs.getString("course\_code"));

course.setCourseName(rs.getString("course\_name"));

course.setCredits(rs.getInt("credits"));

course.setAvailable(rs.getInt("is\_available") == 1);

// Handle professor assignment - check for NULL and valid professor

int professorId = rs.getInt("professor\_id");

if (!rs.wasNull() && professorId > 0) {

try {

Professor prof = (Professor) userDAO.getUserById(professorId);

if (prof != null) {

course.setProfessor(prof);

}

} catch (Exception e) {

// Professor not found or error - leave professor as null

System.err.println("Warning: Could not load professor " + professorId + " for course " + course.getCourseCode());

}

}

return course; }}

**User Database:**

import model.AuthenticationException;

import model.\*;

import java.sql.\*;

import java.util.ArrayList;

public class UserDAO implements Manageable<User> {

public User login(String username, String password, String role) throws AuthenticationException, DatabaseException {

String sql = "SELECT \* FROM Users WHERE username = ? AND password = ? AND role = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setString(1, username);

stmt.setString(2, password);

stmt.setString(3, role);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return createUserFromResultSet(rs);

} else {

throw new AuthenticationException("Invalid username, password, or role");

}

} catch (SQLException e) {

throw new DatabaseException("Login failed", e);

}}

public ArrayList<User> getAllUsers() throws DatabaseException {

ArrayList<User> users = new ArrayList<>();

String sql = "SELECT \* FROM Users ORDER BY user\_id";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql);

ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

users.add(createUserFromResultSet(rs));

}

return users;

} catch (SQLException e) {

throw new DatabaseException("Failed to retrieve users", e);

}}

public User getUserById(int userId) throws DatabaseException {

String sql = "SELECT \* FROM Users WHERE user\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, userId);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return createUserFromResultSet(rs);

}

return null;

} catch (SQLException e) {

throw new DatabaseException("Failed to retrieve user", e);

}}

public ArrayList<Professor> getAllProfessors() throws DatabaseException {

ArrayList<Professor> professors = new ArrayList<>();

String sql = "SELECT \* FROM Users WHERE role = 'PROFESSOR' ORDER BY last\_name";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql);

ResultSet rs = stmt.executeQuery()) {

while (rs.next()) {

Professor prof = new Professor();

prof.setUserId(rs.getInt("user\_id"));

prof.setUsername(rs.getString("username"));

prof.setPassword(rs.getString("password"));

prof.setFirstName(rs.getString("first\_name"));

prof.setLastName(rs.getString("last\_name"));

prof.setRole(rs.getString("role"));

professors.add(prof);

}

return professors;

} catch (SQLException e) {

throw new DatabaseException("Failed to retrieve professors", e);}}

@Override

public void add(User user) throws DatabaseException {

String sql = "INSERT INTO Users (user\_id, username, password, first\_name, last\_name, role) " +

"VALUES (user\_seq.NEXTVAL, ?, ?, ?, ?, ?)";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setString(1, user.getUsername());

stmt.setString(2, user.getPassword());

stmt.setString(3, user.getFirstName());

stmt.setString(4, user.getLastName());

stmt.setString(5, user.getRole());

stmt.executeUpdate();

} catch (SQLException e) {

throw new DatabaseException("Failed to add user", e);}}

@Override

public void update(User user) throws DatabaseException {

String sql = "UPDATE Users SET username = ?, password = ?, first\_name = ?, last\_name = ?, role = ? " +

"WHERE user\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setString(1, user.getUsername());

stmt.setString(2, user.getPassword());

stmt.setString(3, user.getFirstName());

stmt.setString(4, user.getLastName());

stmt.setString(5, user.getRole());

stmt.setInt(6, user.getUserId());

stmt.executeUpdate();

} catch (SQLException e) {

throw new DatabaseException("Failed to update user", e);}}

@Override

public void delete(int userId) throws DatabaseException {

String sql = "DELETE FROM Users WHERE user\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, userId);

stmt.executeUpdate();

} catch (SQLException e) {

throw new DatabaseException("Failed to delete user", e);}}

private User createUserFromResultSet(ResultSet rs) throws SQLException {

String role = rs.getString("role");

User user;

switch (role) {

case "STUDENT":

user = new Student();

break;

case "PROFESSOR":

user = new Professor();

break;

case "ADMIN":

user = new Administrator();

break;

default:

throw new SQLException("Unknown user role: " + role);}

user.setUserId(rs.getInt("user\_id"));

user.setUsername(rs.getString("username"));

user.setPassword(rs.getString("password"));

user.setFirstName(rs.getString("first\_name"));

user.setLastName(rs.getString("last\_name"));

user.setRole(role);

return user;}}

**Enrollment database:**

import model.\*;

import java.sql.\*;

import java.util.ArrayList;

public class EnrollmentDAO implements Manageable<Enrollment> {

private UserDAO userDAO = new UserDAO();

private CourseDAO courseDAO = new CourseDAO();

public void enrollStudent(int studentId, int courseId) throws DatabaseException {

String sql = "INSERT INTO Enrollments (enrollment\_id, student\_id, course\_id, enrollment\_date) " +

"VALUES (enrollment\_seq.NEXTVAL, ?, ?, SYSDATE)";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, studentId);

stmt.setInt(2, courseId);

stmt.executeUpdate();

} catch (SQLException e) {

if (e.getErrorCode() == 1) {

throw new DatabaseException("Student is already enrolled in this course");

}

throw new DatabaseException("Failed to enroll student", e);}}

public void dropCourse(int studentId, int courseId) throws DatabaseException {

String sql = "DELETE FROM Enrollments WHERE student\_id = ? AND course\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, studentId);

stmt.setInt(2, courseId);

int rowsAffected = stmt.executeUpdate();

if (rowsAffected == 0) {

throw new DatabaseException("Enrollment not found");

}

} catch (SQLException e) {

throw new DatabaseException("Failed to drop course", e);

}}

public ArrayList<Course> getEnrollmentsByStudent(int studentId) throws DatabaseException {

ArrayList<Course> courses = new ArrayList<>();

String sql = "SELECT c.\* FROM Courses c " +

"JOIN Enrollments e ON c.course\_id = e.course\_id " +

"WHERE e.student\_id = ? " +

"ORDER BY c.course\_code";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, studentId);

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

Course course = new Course();

course.setCourseId(rs.getInt("course\_id"));

course.setCourseCode(rs.getString("course\_code"));

course.setCourseName(rs.getString("course\_name"));

course.setCredits(rs.getInt("credits"));

course.setAvailable(rs.getInt("is\_available") == 1);

int professorId = rs.getInt("professor\_id");

if (professorId > 0) {

Professor prof = (Professor) userDAO.getUserById(professorId);

course.setProfessor(prof);}

courses.add(course);}

return courses;

} catch (SQLException e) {

throw new DatabaseException("Failed to retrieve student enrollments", e);}}

public ArrayList<Student> getEnrollmentsByCourse(int courseId) throws DatabaseException {

ArrayList<Student> students = new ArrayList<>();

String sql = "SELECT u.\* FROM Users u " +

"JOIN Enrollments e ON u.user\_id = e.student\_id " +

"WHERE e.course\_id = ? AND u.role = 'STUDENT' " +

"ORDER BY u.last\_name, u.first\_name";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, courseId);

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

Student student = new Student();

student.setUserId(rs.getInt("user\_id"));

student.setUsername(rs.getString("username"));

student.setPassword(rs.getString("password"));

student.setFirstName(rs.getString("first\_name"));

student.setLastName(rs.getString("last\_name"));

student.setRole(rs.getString("role"));

students.add(student);}

return students;

} catch (SQLException e) {

throw new DatabaseException("Failed to retrieve course enrollments", e);}}

@Override

public void add(Enrollment enrollment) throws DatabaseException {

enrollStudent(enrollment.getStudent().getUserId(), enrollment.getCourse().getCourseId());}

@Override

public void update(Enrollment enrollment) throws DatabaseException {

throw new UnsupportedOperationException("Updating enrollments is not supported");}

@Override

public void delete(int enrollmentId) throws DatabaseException {

String sql = "DELETE FROM Enrollments WHERE enrollment\_id = ?";

try (Connection conn = DatabaseConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(sql)) {

stmt.setInt(1, enrollmentId);

stmt.executeUpdate();

} catch (SQLException e) {

throw new DatabaseException("Failed to delete enrollment", e);}}}

**Controller:**

**Admin Dashboard:**

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Label;

import javafx.stage.Stage;

import model.User;

import java.io.IOException;

import java.net.URL;

import java.util.ResourceBundle;

public class AdminDashboardController implements Initializable {

@FXML

private Label welcomeLabel;

private User currentUser;

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

currentUser = LoginController.getCurrentUser();

if (currentUser != null) {

welcomeLabel.setText("Welcome, " + currentUser.getFullName() + "!");}} @FXML

private void handleManageUsers() {

loadScene("/fxml/UserManagement.fxml", "Manage Users");}

@FXML

private void handleManageCourses() {

loadScene("/fxml/CourseManagement.fxml", "Manage Courses");}

@FXML

private void handleLogout() {

loadScene("/fxml/Login.fxml", "Login");}

private void loadScene(String fxmlFile, String title) {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource(fxmlFile));

Parent root = loader.load();

Stage stage = (Stage) welcomeLabel.getScene().getWindow();

stage.setScene(new Scene(root, 800, 600));

stage.setTitle(title);

} catch (IOException e) {

System.err.println("Failed to load scene: " + e.getMessage());

}}}

**Course Management:**

import dao.CourseDAO;

import dao.UserDAO;

import dao.DatabaseException;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.stage.Stage;

import model.Course;

import model.Professor;

import java.io.IOException;

import java.net.URL;

import java.util.ArrayList;

import java.util.ResourceBundle;

public class CourseManagementController implements Initializable {

@FXML

private TextField courseCodeField;

@FXML

private TextField courseNameField;

@FXML

private TextField creditsField;

@FXML

private ComboBox<Professor> professorComboBox;

@FXML

private CheckBox availableCheckBox;

@FXML

private TableView<Course> coursesTable;

@FXML

private TableColumn<Course, Integer> courseIdColumn;

@FXML

private TableColumn<Course, String> courseCodeColumn;

@FXML

private TableColumn<Course, String> courseNameColumn;

@FXML

private TableColumn<Course, Integer> creditsColumn;

@FXML

private TableColumn<Course, String> professorColumn;

@FXML

private TableColumn<Course, String> availabilityColumn;

private CourseDAO courseDAO = new CourseDAO();

private UserDAO userDAO = new UserDAO();

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

courseIdColumn.setCellValueFactory(new PropertyValueFactory<>("courseId"));

courseCodeColumn.setCellValueFactory(new PropertyValueFactory<>("courseCode"));

courseNameColumn.setCellValueFactory(new PropertyValueFactory<>("courseName"));

creditsColumn.setCellValueFactory(new PropertyValueFactory<>("credits"));

professorColumn.setCellValueFactory(cellData ->

new javafx.beans.property.SimpleStringProperty(cellData.getValue().getProfessorName()));

availabilityColumn.setCellValueFactory(cellData ->

new javafx.beans.property.SimpleStringProperty(

cellData.getValue().isAvailable() ? "Available" : "Unavailable"));

coursesTable.getSelectionModel().selectedItemProperty().addListener((obs, oldSelection, newSelection) -> {

if (newSelection != null) {

populateFields(newSelection); }});

loadProfessors();

loadCourses();}

private void loadProfessors() {

try {

ArrayList<Professor> professors = userDAO.getAllProfessors();

ObservableList<Professor> professorList = FXCollections.observableArrayList(professors);

professorComboBox.setItems(professorList);

} catch (DatabaseException e) {

showError("Failed to load professors: " + e.getMessage());}}

private void loadCourses() {

try {

ArrayList<Course> courses = courseDAO.getAllCourses();

ObservableList<Course> courseList = FXCollections.observableArrayList(courses);

coursesTable.setItems(courseList);

} catch (DatabaseException e) {

showError("Failed to load courses: " + e.getMessage());}]

private void populateFields(Course course) {

courseCodeField.setText(course.getCourseCode());

courseNameField.setText(course.getCourseName());

creditsField.setText(String.valueOf(course.getCredits()));

availableCheckBox.setSelected(course.isAvailable());

if (course.getProfessor() != null) {

for (Professor prof : professorComboBox.getItems()) {

if (prof.getUserId() == course.getProfessor().getUserId()) {

professorComboBox.setValue(prof);

break;}}}}

@FXML

private void handleAdd() {

if (!validateInput()) {

return;}

try {

Course course = createCourseFromFields();

courseDAO.add(course);

showSuccess("Course added successfully");

loadCourses();

handleClear();

} catch (DatabaseException e) {

showError("Failed to add course: " + e.getMessage());}}

@FXML

private void handleUpdate() {

Course selectedCourse = coursesTable.getSelectionModel().getSelectedItem();

if (selectedCourse == null) {

showWarning("Please select a course to update");

return;}

if (!validateInput()) {

return;}

try {

Course course = createCourseFromFields();

course.setCourseId(selectedCourse.getCourseId());

courseDAO.update(course);

showSuccess("Course updated successfully");

loadCourses();

handleClear();

} catch (DatabaseException e) {

showError("Failed to update course: " + e.getMessage());}}

@FXML

private void handleDelete() {

Course selectedCourse = coursesTable.getSelectionModel().getSelectedItem();

if (selectedCourse == null) {

showWarning("Please select a course to delete");

return;}

try {

courseDAO.delete(selectedCourse.getCourseId());

showSuccess("Course deleted successfully");

loadCourses();

handleClear();

} catch (DatabaseException e) {

showError("Failed to delete course: " + e.getMessage());}}

@FXML

private void handleClear() {

courseCodeField.clear();

courseNameField.clear();

creditsField.clear();

professorComboBox.setValue(null);

availableCheckBox.setSelected(false);

coursesTable.getSelectionModel().clearSelection();}

@FXML

private void handleBack() {

loadScene("/fxml/AdminDashboard.fxml", "Admin Dashboard");}

private boolean validateInput() {

if (courseCodeField.getText().trim().isEmpty() ||

courseNameField.getText().trim().isEmpty() ||

creditsField.getText().trim().isEmpty()) {

showWarning("Please fill in all required fields");

return false;}

try {

Integer.parseInt(creditsField.getText().trim());

} catch (NumberFormatException e) {

showWarning("Credits must be a valid number");

return false;}

return true;}

private Course createCourseFromFields() {

Course course = new Course();

course.setCourseCode(courseCodeField.getText().trim());

course.setCourseName(courseNameField.getText().trim());

course.setCredits(Integer.parseInt(creditsField.getText().trim()));

course.setProfessor(professorComboBox.getValue());

course.setAvailable(availableCheckBox.isSelected());

return course;}

private void loadScene(String fxmlFile, String title) {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource(fxmlFile));

Parent root = loader.load();

Stage stage = (Stage) coursesTable.getScene().getWindow();

stage.setScene(new Scene(root, 800, 600));

stage.setTitle(title);

} catch (IOException e) {

System.err.println("Failed to load scene: " + e.getMessage());}}

private void showSuccess(String message) {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Success");

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();}

private void showWarning(String message) {

Alert alert = new Alert(Alert.AlertType.WARNING);

alert.setTitle("Warning");

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();}

private void showError(String message) {

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Error");

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();}}

**Login Controller:**

import dao.UserDAO;

import dao.DatabaseException;

import model.AuthenticationException;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.stage.Stage;

import model.User;

import java.io.IOException;

import java.net.URL;

import java.util.ResourceBundle;

public class LoginController implements Initializable {

@FXML

private TextField usernameField;

@FXML

private PasswordField passwordField;

@FXML

private ComboBox<String> roleComboBox;

@FXML

private Label errorLabel;

private UserDAO userDAO = new UserDAO();

private static User currentUser;

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

roleComboBox.getItems().addAll("STUDENT", "PROFESSOR", "ADMIN");

roleComboBox.setValue("STUDENT");}

@FXML

private void handleLogin() {

String username = usernameField.getText().trim();

String password = passwordField.getText();

String role = roleComboBox.getValue();

if (username.isEmpty() || password.isEmpty()) {

errorLabel.setText("Please enter username and password");

return;}

try {

User user = userDAO.login(username, password, role);

currentUser = user;

String fxmlFile = "";

switch (role) {

case "STUDENT":

fxmlFile = "/fxml/StudentDashboard.fxml";

break;

case "PROFESSOR":

fxmlFile = "/fxml/ProfessorDashboard.fxml";

break;

case "ADMIN":

fxmlFile = "/fxml/AdminDashboard.fxml";

break;}

loadScene(fxmlFile, user.getRole() + " Dashboard");

} catch (AuthenticationException e) {

errorLabel.setText(e.getMessage());

} catch (DatabaseException e) {

errorLabel.setText("Database error: " + e.getMessage());}}

private void loadScene(String fxmlFile, String title) {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource(fxmlFile));

Parent root = loader.load();

Stage stage = (Stage) usernameField.getScene().getWindow();

stage.setScene(new Scene(root, 800, 600));

stage.setTitle(title);

} catch (IOException e) {

errorLabel.setText("Failed to load dashboard: " + e.getMessage());}}

public static User getCurrentUser() {

return currentUser;}}

**Professor Controller:**

import dao.CourseDAO;

import dao.EnrollmentDAO;

import dao.DatabaseException;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.stage.Stage;

import model.Course;

import model.Student;

import model.User;

import java.io.IOException;

import java.net.URL;

import java.util.ArrayList;

import java.util.ResourceBundle;

/\*\*

\* Consolidated controller for professor course management

\* Handles viewing courses, enrolled students, and updating course availability

\*/

public class ProfessorCourseManagementController implements Initializable {

@FXML

private TabPane professorTabPane;

// My Courses Tab Components

@FXML

private TableView<Course> myCoursesTable;

@FXML

private TableColumn<Course, String> myCourseCodeColumn;

@FXML

private TableColumn<Course, String> myCourseNameColumn;

@FXML

private TableColumn<Course, Integer> myCreditsColumn;

@FXML

private TableColumn<Course, String> myAvailabilityColumn;

// Enrolled Students Tab Components

@FXML

private ComboBox<Course> courseComboBox;

@FXML

private TableView<Student> studentsTable;

@FXML

private TableColumn<Student, Integer> studentIdColumn;

@FXML

private TableColumn<Student, String> firstNameColumn;

@FXML

private TableColumn<Student, String> lastNameColumn;

@FXML

private TableColumn<Student, String> usernameColumn;

// Update Availability Tab Components

@FXML

private TableView<Course> availabilityTable;

@FXML

private TableColumn<Course, String> availCourseCodeColumn;

@FXML

private TableColumn<Course, String> availCourseNameColumn;

@FXML

private TableColumn<Course, Integer> availCreditsColumn;

@FXML

private TableColumn<Course, String> availStatusColumn;

@FXML

private Button makeAvailableButton;

@FXML

private Button makeUnavailableButton;

private CourseDAO courseDAO = new CourseDAO();

private EnrollmentDAO enrollmentDAO = new EnrollmentDAO();

private User currentUser;

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

currentUser = LoginController.getCurrentUser();

// Initialize My Courses Tab

myCourseCodeColumn.setCellValueFactory(new PropertyValueFactory<>("courseCode"));

myCourseNameColumn.setCellValueFactory(new PropertyValueFactory<>("courseName"));

myCreditsColumn.setCellValueFactory(new PropertyValueFactory<>("credits"));

myAvailabilityColumn.setCellValueFactory(cellData ->

new javafx.beans.property.SimpleStringProperty(

cellData.getValue().isAvailable() ? "Available" : "Unavailable"));

// Initialize Enrolled Students Tab

studentIdColumn.setCellValueFactory(new PropertyValueFactory<>("userId"));

firstNameColumn.setCellValueFactory(new PropertyValueFactory<>("firstName"));

lastNameColumn.setCellValueFactory(new PropertyValueFactory<>("lastName"));

usernameColumn.setCellValueFactory(new PropertyValueFactory<>("username"));

// Initialize Update Availability Tab

availCourseCodeColumn.setCellValueFactory(new PropertyValueFactory<>("courseCode"));

availCourseNameColumn.setCellValueFactory(new PropertyValueFactory<>("courseName"));

availCreditsColumn.setCellValueFactory(new PropertyValueFactory<>("credits"));

availStatusColumn.setCellValueFactory(cellData ->

new javafx.beans.property.SimpleStringProperty(

cellData.getValue().isAvailable() ? "Available" : "Unavailable"));

// Load data

loadProfessorCourses();

loadCourseComboBox();

// Add tab change listener to refresh data

professorTabPane.getSelectionModel().selectedItemProperty().addListener((obs, oldTab, newTab) -> {

if (newTab != null) {

loadProfessorCourses();

loadCourseComboBox();}}); }

private void loadProfessorCourses() {

try {

ArrayList<Course> courses = courseDAO.getCoursesByProfessor(currentUser.getUserId());

ObservableList<Course> courseList = FXCollections.observableArrayList(courses);

myCoursesTable.setItems(courseList);

availabilityTable.setItems(courseList);

} catch (DatabaseException e) {

showError("Failed to load courses: " + e.getMessage());}}

private void loadCourseComboBox() {

try {

ArrayList<Course> courses = courseDAO.getCoursesByProfessor(currentUser.getUserId());

ObservableList<Course> courseList = FXCollections.observableArrayList(courses);

courseComboBox.setItems(courseList);

} catch (DatabaseException e) {

showError("Failed to load courses: " + e.getMessage());}}

@FXML

private void handleCourseSelection() {

Course selectedCourse = courseComboBox.getSelectionModel().getSelectedItem();

if (selectedCourse != null) {

loadEnrolledStudents(selectedCourse.getCourseId());}}

private void loadEnrolledStudents(int courseId) {

try {

ArrayList<Student> students = enrollmentDAO.getEnrollmentsByCourse(courseId);

ObservableList<Student> studentList = FXCollections.observableArrayList(students);

studentsTable.setItems(studentList);

} catch (DatabaseException e) {

showError("Failed to load enrolled students: " + e.getMessage());}}

@FXML

private void handleMakeAvailable() {

Course selectedCourse = availabilityTable.getSelectionModel().getSelectedItem();

if (selectedCourse == null) {

showWarning("Please select a course");

return;}

try {

courseDAO.updateAvailability(selectedCourse.getCourseId(), true);

showSuccess("Course " + selectedCourse.getCourseCode() + " is now available");

loadProfessorCourses();

} catch (DatabaseException e) {

showError("Failed to update availability: " + e.getMessage());}}

@FXML

private void handleMakeUnavailable() {

Course selectedCourse = availabilityTable.getSelectionModel().getSelectedItem();

if (selectedCourse == null) {

showWarning("Please select a course");

return;}

try {

courseDAO.updateAvailability(selectedCourse.getCourseId(), false);

showSuccess("Course " + selectedCourse.getCourseCode() + " is now unavailable");

loadProfessorCourses();

} catch (DatabaseException e) {

showError("Failed to update availability: " + e.getMessage());}}

@FXML

private void handleBack() {

loadScene("/fxml/ProfessorDashboard.fxml", "Professor Dashboard");}

private void loadScene(String fxmlFile, String title) {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource(fxmlFile));

Parent root = loader.load();

Stage stage = (Stage) professorTabPane.getScene().getWindow();

stage.setScene(new Scene(root, 800, 600));

stage.setTitle(title);

} catch (IOException e) {

System.err.println("Failed to load scene: " + e.getMessage());}}

private void showSuccess(String message) {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Success");

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();}

private void showWarning(String message) {

Alert alert = new Alert(Alert.AlertType.WARNING);

alert.setTitle("Warning");

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();}

private void showError(String message) {

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Error");

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();}}

**Professor Dashboard:**

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Label;

import javafx.stage.Stage;

import model.User;

import java.io.IOException;

import java.net.URL;

import java.util.ResourceBundle;

public class ProfessorDashboardController implements Initializable {

@FXML

private Label welcomeLabel;

private User currentUser;

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

currentUser = LoginController.getCurrentUser();

if (currentUser != null) {

welcomeLabel.setText("Welcome, " + currentUser.getFullName() + "!");}}

@FXML

private void handleViewMyCourses() {

loadScene("/fxml/ProfessorCourseManagement.fxml", "Course Management");}

@FXML

private void handleViewEnrolledStudents() {

loadScene("/fxml/ProfessorCourseManagement.fxml", "Course Management");}

@FXML

private void handleUpdateCourseAvailability() {

loadScene("/fxml/ProfessorCourseManagement.fxml", "Course Management");}

@FXML

private void handleLogout() {

loadScene("/fxml/Login.fxml", "Login");}

private void loadScene(String fxmlFile, String title) {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource(fxmlFile));

Parent root = loader.load();

Stage stage = (Stage) welcomeLabel.getScene().getWindow();

stage.setScene(new Scene(root, 800, 600));

stage.setTitle(title);

} catch (IOException e) {

System.err.println("Failed to load scene: " + e.getMessage());}}

**Student Course View Controller:**

import dao.CourseDAO;

import dao.EnrollmentDAO;

import dao.DatabaseException;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.stage.Stage;

import model.Course;

import model.User;

import java.io.IOException;

import java.net.URL;

import java.util.ArrayList;

import java.util.ResourceBundle;

/\*\*

\* Consolidated controller for student course views

\* Handles both viewing available courses and student schedule

\*/

public class StudentCourseViewController implements Initializable {

@FXML

private TabPane courseViewTabPane;

// Available Courses Tab Components

@FXML

private TableView<Course> availableCoursesTable;

@FXML

private TableColumn<Course, String> availCourseCodeColumn;

@FXML

private TableColumn<Course, String> availCourseNameColumn;

@FXML

private TableColumn<Course, Integer> availCreditsColumn;

@FXML

private TableColumn<Course, String> availProfessorColumn;

@FXML

private TableColumn<Course, String> availabilityColumn;

// My Schedule Tab Components

@FXML

private TableView<Course> scheduleTable;

@FXML

private TableColumn<Course, String> schedCourseCodeColumn;

@FXML

private TableColumn<Course, String> schedCourseNameColumn;

@FXML

private TableColumn<Course, Integer> schedCreditsColumn;

@FXML

private TableColumn<Course, String> schedProfessorColumn;

private CourseDAO courseDAO = new CourseDAO();

private EnrollmentDAO enrollmentDAO = new EnrollmentDAO();

private User currentUser;

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

currentUser = LoginController.getCurrentUser();

// Initialize Available Courses Tab

availCourseCodeColumn.setCellValueFactory(new PropertyValueFactory<>("courseCode"));

availCourseNameColumn.setCellValueFactory(new PropertyValueFactory<>("courseName"));

availCreditsColumn.setCellValueFactory(new PropertyValueFactory<>("credits"));

availProfessorColumn.setCellValueFactory(new PropertyValueFactory<>("professorName"));

availabilityColumn.setCellValueFactory(new PropertyValueFactory<>("availabilityStatus"));

// Initialize My Schedule Tab

schedCourseCodeColumn.setCellValueFactory(new PropertyValueFactory<>("courseCode"));

schedCourseNameColumn.setCellValueFactory(new PropertyValueFactory<>("courseName"));

schedCreditsColumn.setCellValueFactory(new PropertyValueFactory<>("credits"));

schedProfessorColumn.setCellValueFactory(new PropertyValueFactory<>("professorName"));

// Load data

loadAvailableCourses();

loadSchedule();

// Add tab change listener to refresh data

courseViewTabPane.getSelectionModel().selectedItemProperty().addListener((obs, oldTab, newTab) -> {

if (newTab != null) {

loadAvailableCourses();

loadSchedule(); }}); }

private void loadAvailableCourses() {

try {

ArrayList<Course> courses = courseDAO.getAvailableCourses();

ObservableList<Course> courseList = FXCollections.observableArrayList(courses);

availableCoursesTable.setItems(courseList);

} catch (DatabaseException e) {

showError("Failed to load courses: " + e.getMessage());}}

private void loadSchedule() {

try {

ArrayList<Course> courses = enrollmentDAO.getEnrollmentsByStudent(currentUser.getUserId());

ObservableList<Course> courseList = FXCollections.observableArrayList(courses);

scheduleTable.setItems(courseList);

} catch (DatabaseException e) {

showError("Failed to load schedule: " + e.getMessage());}}

@FXML

private void handleBack() {

loadScene("/fxml/StudentDashboard.fxml", "Student Dashboard");}

private void loadScene(String fxmlFile, String title) {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource(fxmlFile));

Parent root = loader.load();

Stage stage = (Stage) courseViewTabPane.getScene().getWindow();

stage.setScene(new Scene(root, 800, 600));

stage.setTitle(title);

} catch (IOException e) {

showError("Failed to load scene: " + e.getMessage());}}

private void showError(String message) {

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Error");

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();}}

**Student DashBoard Controller:**

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.Label;

import javafx.stage.Stage;

import model.User;

import java.io.IOException;

import java.net.URL;

import java.util.ResourceBundle;

public class StudentDashboardController implements Initializable {

@FXML

private Label welcomeLabel;

private User currentUser;

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

currentUser = LoginController.getCurrentUser();

if (currentUser != null) {

welcomeLabel.setText("Welcome, " + currentUser.getFullName() + "!");}}

@FXML

private void handleViewAvailableCourses() {

loadScene("/fxml/StudentCourseView.fxml", "Course Information");}

@FXML

private void handleViewMySchedule() {

loadScene("/fxml/StudentCourseView.fxml", "Course Information");}

@FXML

private void handleRegisterCourse() {

loadScene("/fxml/StudentEnrollment.fxml", "Course Enrollment");}

@FXML

private void handleDropCourse() {

loadScene("/fxml/StudentEnrollment.fxml", "Course Enrollment");}

@FXML

private void handleLogout() {

loadScene("/fxml/Login.fxml", "Login");}

private void loadScene(String fxmlFile, String title) {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource(fxmlFile));

Parent root = loader.load();

Stage stage = (Stage) welcomeLabel.getScene().getWindow();

stage.setScene(new Scene(root, 800, 600));

stage.setTitle(title);

} catch (IOException e) {

System.err.println("Failed to load scene: " + e.getMessage());}}

**Student Enrollment Controller:**

import dao.CourseDAO;

import dao.EnrollmentDAO;

import dao.DatabaseException;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.stage.Stage;

import model.Course;

import model.User;

import java.io.IOException;

import java.net.URL;

import java.util.ArrayList;

import java.util.ResourceBundle;

/\*\*

\* Consolidated controller for student enrollment operations

\* Handles both course registration and dropping courses

\*/

public class StudentEnrollmentController implements Initializable {

@FXML

private TabPane enrollmentTabPane;

// Registration Tab Components

@FXML

private TableView<Course> availableCoursesTable;

@FXML

private TableColumn<Course, String> regCourseCodeColumn;

@FXML

private TableColumn<Course, String> regCourseNameColumn;

@FXML

private TableColumn<Course, Integer> regCreditsColumn;

@FXML

private TableColumn<Course, String> regProfessorColumn;

@FXML

private Button registerButton;

// Drop Course Tab Components

@FXML

private TableView<Course> enrolledCoursesTable;

@FXML

private TableColumn<Course, String> dropCourseCodeColumn;

@FXML

private TableColumn<Course, String> dropCourseNameColumn;

@FXML

private TableColumn<Course, Integer> dropCreditsColumn;

@FXML

private TableColumn<Course, String> dropProfessorColumn;

@FXML

private Button dropButton;

private CourseDAO courseDAO = new CourseDAO();

private EnrollmentDAO enrollmentDAO = new EnrollmentDAO();

private User currentUser;

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

currentUser = LoginController.getCurrentUser();

// Initialize Registration Tab

regCourseCodeColumn.setCellValueFactory(new PropertyValueFactory<>("courseCode"));

regCourseNameColumn.setCellValueFactory(new PropertyValueFactory<>("courseName"));

regCreditsColumn.setCellValueFactory(new PropertyValueFactory<>("credits"));

regProfessorColumn.setCellValueFactory(cellData ->

new javafx.beans.property.SimpleStringProperty(cellData.getValue().getProfessorName()));

// Initialize Drop Tab

dropCourseCodeColumn.setCellValueFactory(new PropertyValueFactory<>("courseCode"));

dropCourseNameColumn.setCellValueFactory(new PropertyValueFactory<>("courseName"));

dropCreditsColumn.setCellValueFactory(new PropertyValueFactory<>("credits"));

dropProfessorColumn.setCellValueFactory(cellData ->

new javafx.beans.property.SimpleStringProperty(cellData.getValue().getProfessorName()));

// Load data

loadAvailableCourses();

loadEnrolledCourses();

// Add tab change listener to refresh data

enrollmentTabPane.getSelectionModel().selectedItemProperty().addListener((obs, oldTab, newTab) -> {

if (newTab != null) {

loadAvailableCourses();

loadEnrolledCourses();

}});}

private void loadAvailableCourses() {

try {

ArrayList<Course> courses = courseDAO.getAvailableCourses();

ObservableList<Course> courseList = FXCollections.observableArrayList(courses);

availableCoursesTable.setItems(courseList);

} catch (DatabaseException e) {

showError("Failed to load available courses: " + e.getMessage());}}

private void loadEnrolledCourses() {

try {

ArrayList<Course> courses = enrollmentDAO.getEnrollmentsByStudent(currentUser.getUserId());

ObservableList<Course> courseList = FXCollections.observableArrayList(courses);

enrolledCoursesTable.setItems(courseList);

} catch (DatabaseException e) {

showError("Failed to load enrolled courses: " + e.getMessage());}}

@FXML

private void handleRegister() {

Course selectedCourse = availableCoursesTable.getSelectionModel().getSelectedItem();

if (selectedCourse == null) {

showWarning("Please select a course to register");

return;}

try {

enrollmentDAO.enrollStudent(currentUser.getUserId(), selectedCourse.getCourseId());

showSuccess("Successfully registered for " + selectedCourse.getCourseCode());

loadAvailableCourses();

loadEnrolledCourses();

} catch (DatabaseException e) {

showError("Registration failed: " + e.getMessage());}}

@FXML

private void handleDrop() {

Course selectedCourse = enrolledCoursesTable.getSelectionModel().getSelectedItem();

if (selectedCourse == null) {

showWarning("Please select a course to drop");

return;}

try {

enrollmentDAO.dropCourse(currentUser.getUserId(), selectedCourse.getCourseId());

showSuccess("Successfully dropped " + selectedCourse.getCourseCode());

loadAvailableCourses();

loadEnrolledCourses();

} catch (DatabaseException e) {

showError("Failed to drop course: " + e.getMessage());}}

@FXML

private void handleBack() {

loadScene("/fxml/StudentDashboard.fxml", "Student Dashboard");}

private void loadScene(String fxmlFile, String title) {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource(fxmlFile));

Parent root = loader.load();

Stage stage = (Stage) enrollmentTabPane.getScene().getWindow();

stage.setScene(new Scene(root, 800, 600));

stage.setTitle(title);

} catch (IOException e) {

System.err.println("Failed to load scene: " + e.getMessage());}}

private void showSuccess(String message) {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Success");

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();}

private void showWarning(String message) {

Alert alert = new Alert(Alert.AlertType.WARNING);

alert.setTitle("Warning");

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();}

private void showError(String message) {

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Error");

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();}}

**User Management Controller:**

import dao.UserDAO;

import dao.DatabaseException;

import javafx.collections.FXCollections;

import javafx.collections.ObservableList;

import javafx.fxml.FXML;

import javafx.fxml.FXMLLoader;

import javafx.fxml.Initializable;

import javafx.scene.Parent;

import javafx.scene.Scene;

import javafx.scene.control.\*;

import javafx.scene.control.cell.PropertyValueFactory;

import javafx.stage.Stage;

import model.\*;

import java.io.IOException;

import java.net.URL;

import java.util.ArrayList;

import java.util.ResourceBundle;

public class UserManagementController implements Initializable {

@FXML

private TextField usernameField;

@FXML

private TextField passwordField;

@FXML

private TextField firstNameField;

@FXML

private TextField lastNameField;

@FXML

private ComboBox<String> roleComboBox;

@FXML

private TableView<User> usersTable;

@FXML

private TableColumn<User, Integer> userIdColumn;

@FXML

private TableColumn<User, String> usernameColumn;

@FXML

private TableColumn<User, String> firstNameColumn;

@FXML

private TableColumn<User, String> lastNameColumn;

@FXML

private TableColumn<User, String> roleColumn;

private UserDAO userDAO = new UserDAO();

@Override

public void initialize(URL url, ResourceBundle resourceBundle) {

roleComboBox.setItems(FXCollections.observableArrayList("STUDENT", "PROFESSOR", "ADMIN"));

userIdColumn.setCellValueFactory(new PropertyValueFactory<>("userId"));

usernameColumn.setCellValueFactory(new PropertyValueFactory<>("username"));

firstNameColumn.setCellValueFactory(new PropertyValueFactory<>("firstName"));

lastNameColumn.setCellValueFactory(new PropertyValueFactory<>("lastName"));

roleColumn.setCellValueFactory(new PropertyValueFactory<>("role"));

usersTable.getSelectionModel().selectedItemProperty().addListener((obs, oldSelection, newSelection) -> {

if (newSelection != null) {

populateFields(newSelection); }});

loadUsers(); }

private void loadUsers() {

try {

ArrayList<User> users = userDAO.getAllUsers();

ObservableList<User> userList = FXCollections.observableArrayList(users);

usersTable.setItems(userList);

} catch (DatabaseException e) {

showError("Failed to load users: " + e.getMessage());}}

private void populateFields(User user) {

usernameField.setText(user.getUsername());

passwordField.setText(user.getPassword());

firstNameField.setText(user.getFirstName());

lastNameField.setText(user.getLastName());

roleComboBox.setValue(user.getRole());}

@FXML

private void handleAdd() {

if (!validateInput()) {

return;}

try {

User user = createUserFromFields();

userDAO.add(user);

showSuccess("User added successfully");

loadUsers();

handleClear();

} catch (DatabaseException e) {

showError("Failed to add user: " + e.getMessage());}}

@FXML

private void handleUpdate() {

User selectedUser = usersTable.getSelectionModel().getSelectedItem();

if (selectedUser == null) {

showWarning("Please select a user to update");

return;}

if (!validateInput()) {

return;}

try {

User user = createUserFromFields();

user.setUserId(selectedUser.getUserId());

userDAO.update(user);

showSuccess("User updated successfully");

loadUsers();

handleClear();

} catch (DatabaseException e) {

showError("Failed to update user: " + e.getMessage());}}

@FXML

private void handleDelete() {

User selectedUser = usersTable.getSelectionModel().getSelectedItem();

if (selectedUser == null) {

showWarning("Please select a user to delete");

return

try {

userDAO.delete(selectedUser.getUserId());

showSuccess("User deleted successfully");

loadUsers();

handleClear();

} catch (DatabaseException e) {

showError("Failed to delete user: " + e.getMessage());}}

@FXML

private void handleClear() {

usernameField.clear();

passwordField.clear();

firstNameField.clear();

lastNameField.clear();

roleComboBox.setValue(null);

usersTable.getSelectionModel().clearSelection();}

@FXML

private void handleBack() {

loadScene("/fxml/AdminDashboard.fxml", "Admin Dashboard");}

private boolean validateInput() {

if (usernameField.getText().trim().isEmpty() ||

passwordField.getText().trim().isEmpty() ||

firstNameField.getText().trim().isEmpty() ||

lastNameField.getText().trim().isEmpty() ||

roleComboBox.getValue() == null) {

showWarning("Please fill in all fields");

return false;}

return true;}

private User createUserFromFields() {

String role = roleComboBox.getValue();

User user;

switch (role) {

case "STUDENT":

user = new Student();

break;

case "PROFESSOR":

user = new Professor();

break;

case "ADMIN":

user = new Administrator();

break;

default:

user = new Student();}

user.setUsername(usernameField.getText().trim());

user.setPassword(passwordField.getText().trim());

user.setFirstName(firstNameField.getText().trim());

user.setLastName(lastNameField.getText().trim());

user.setRole(role);

return user;}

private void loadScene(String fxmlFile, String title) {

try {

FXMLLoader loader = new FXMLLoader(getClass().getResource(fxmlFile));

Parent root = loader.load();

Stage stage = (Stage) usersTable.getScene().getWindow();

stage.setScene(new Scene(root, 800, 600));

stage.setTitle(title);

} catch (IOException e) {

System.err.println("Failed to load scene: " + e.getMessage());}}

private void showSuccess(String message) {

Alert alert = new Alert(Alert.AlertType.INFORMATION);

alert.setTitle("Success");

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();}

private void showWarning(String message) {

Alert alert = new Alert(Alert.AlertType.WARNING);

alert.setTitle("Warning");

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();}

private void showError(String message) {

Alert alert = new Alert(Alert.AlertType.ERROR);

alert.setTitle("Error");

alert.setHeaderText(null);

alert.setContentText(message);

alert.showAndWait();}}

**Util:**

**Check Professor:**

import dao.DatabaseConnection;

import dao.DatabaseException;

import java.sql.\*;

public class CheckProfessors {

public static void main(String[] args) {

System.out.println("=== Checking Professors ===");

try {

Connection conn = DatabaseConnection.getConnection();

String sql = "SELECT \* FROM Users WHERE role = 'PROFESSOR'";

PreparedStatement stmt = conn.prepareStatement(sql);

ResultSet rs = stmt.executeQuery();

int count = 0;

while (rs.next()) {

count++;

System.out.println("\nProfessor " + count + ":");

System.out.println(" user\_id: " + rs.getInt("user\_id"));

System.out.println(" username: " + rs.getString("username"));

System.out.println(" first\_name: " + rs.getString("first\_name"));

System.out.println(" last\_name: " + rs.getString("last\_name"));}

System.out.println("\nTotal professors: " + count);

// Now check which professors are referenced in courses

System.out.println("\n=== Professors referenced in Courses ===");

sql = "SELECT DISTINCT professor\_id FROM Courses WHERE professor\_id IS NOT NULL ORDER BY professor\_id";

stmt = conn.prepareStatement(sql);

rs = stmt.executeQuery();

while (rs.next()) {

int profId = rs.getInt("professor\_id");

System.out.println(" Professor ID: " + profId);}

rs.close();

stmt.close();

} catch (DatabaseException e) {

System.err.println("Database connection failed: " + e.getMessage());

e.printStackTrace();

} catch (SQLException e) {

System.err.println("SQL Error: " + e.getMessage());

e.printStackTrace();}}}

**DataBase Check:**

import dao.DatabaseConnection;

import dao.DatabaseException;

import java.sql.\*;

public class DatabaseCheck {

public static void main(String[] args) {

System.out.println("=== Database Connection Check ===");

try {

Connection conn = DatabaseConnection.getConnection();

System.out.println("✓ Database connection successful!");

// Check if Courses table exists

System.out.println("\n=== Checking Courses Table ===");

String sql = "SELECT \* FROM Courses";

PreparedStatement stmt = conn.prepareStatement(sql);

ResultSet rs = stmt.executeQuery();

// Get column info

ResultSetMetaData metaData = rs.getMetaData();

int columnCount = metaData.getColumnCount();

System.out.println("Columns in Courses table:");

for (int i = 1; i <= columnCount; i++) {

System.out.println(" - " + metaData.getColumnName(i) + " (" + metaData.getColumnTypeName(i) + ")");}

// Count rows

int count = 0;

while (rs.next()) {

count++;

System.out.println("\nRow " + count + ":");

System.out.println(" course\_id: " + rs.getInt("course\_id"));

System.out.println(" course\_code: " + rs.getString("course\_code"));

System.out.println(" course\_name: " + rs.getString("course\_name"));

System.out.println(" credits: " + rs.getInt("credits"));

System.out.println(" professor\_id: " + rs.getInt("professor\_id") + " (wasNull: " + rs.wasNull() + ")");

System.out.println(" is\_available: " + rs.getInt("is\_available"));}

System.out.println("\nTotal courses: " + count);

rs.close();

stmt.close();

} catch (DatabaseException e) {

System.err.println("✗ Database connection failed: " + e.getMessage());

e.printStackTrace();

} catch (SQLException e) {

System.err.println("✗ SQL Error: " + e.getMessage());

e.printStackTrace();}}}