package controller;

import model.\*;

import db.\*;

import java.sql.\*;

public class AdminController {

// Method for admin login validation

public boolean loginAdmin(String username, String password) {

String query = "SELECT \* FROM admins WHERE username = ? AND password = ?";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, username);

stmt.setString(2, password); // Password should be hashed in real apps

ResultSet rs = stmt.executeQuery();

return rs.next(); // If admin exists, return true

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

// Method to add a new question

public boolean addQuestion(Question question) {

QuestionDAO questionDAO = new QuestionDAO();

return questionDAO.addQuestion(question); // Ensure this method is properly calling the DAO

}

// Method to update an existing question

public boolean updateQuestion(Question question) {

QuestionDAO questionDAO = new QuestionDAO();

return questionDAO.updateQuestion(question);

}

// Method to delete a question

public boolean deleteQuestion(int questionId) {

QuestionDAO questionDAO = new QuestionDAO();

return questionDAO.deleteQuestion(questionId);

}

}

package controller;

import db.HighScoreDAO;

import model.Player;

import java.util.List;

public class HighScoreController {

// Method to fetch high scores by difficulty level

public List<Player> getHighScoresByLevel(String level) {

HighScoreDAO highScoreDAO = new HighScoreDAO();

return highScoreDAO.getHighScoresByLevel(level);

}

// Method to add a high score to the database

public boolean addHighScore(int playerId, int score, String level) {

HighScoreDAO highScoreDAO = new HighScoreDAO();

return highScoreDAO.addHighScore(playerId, score, level);

}

// Method to fetch all high scores

public List<Player> getAllHighScores() {

HighScoreDAO highScoreDAO = new HighScoreDAO();

return highScoreDAO.getAllHighScores();

}

}

package controller;

import model.\*;

import db.\*;

public class PlayerController {

// Method to register a new player

public boolean registerPlayer(Player player) {

PlayerDAO playerDAO = new PlayerDAO();

return playerDAO.registerPlayer(player);

}

// Method to log in an existing player

public int loginPlayer(String username, String password) {

PlayerDAO playerDAO = new PlayerDAO();

return playerDAO.loginPlayer(username, password); // Return playerId instead of boolean

}

}

package controller;

import model.\*;

import db.\*;

import java.util.\*;

public class QuizController {

// Method to fetch all questions for a specific difficulty level

public List<Question> getQuestionsByLevel(String difficulty) {

QuestionDAO questionDAO = new QuestionDAO();

return questionDAO.getQuestionsByDifficulty(difficulty);

}

// Method to calculate the score for the quiz

public int calculateScore(List<Question> questions, List<Integer> answers) {

int score = 0;

for (int i = 0; i < questions.size(); i++) {

if (questions.get(i).getCorrectAnswer() == answers.get(i)) {

score++;

}

}

return score;

}

}

package controller;

import model.\*;

import db.\*;

import java.util.List;

public class ReportController {

// Method to generate a report for the player after the quiz

public boolean generateReport(int playerId, int correctAnswers, int wrongAnswers, int score, String difficulty) {

ReportDAO reportDAO = new ReportDAO();

// Adjust the constructor to match the correct parameters

Report newReport = new Report(playerId, correctAnswers, wrongAnswers, score, difficulty);

return reportDAO.addReport(newReport); // Save the report to the database

}

// Method to fetch all reports for a player

public List<Report> getReportsForPlayer(int playerId) {

ReportDAO reportDAO = new ReportDAO();

return reportDAO.getReportsForPlayer(playerId); // Retrieve reports from the database

}

}

Database :

package db;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

public class DBConnection {

// Database URL, username, and password (change as necessary)

private static final String URL = "jdbc:mysql://localhost:3307/quiz\_app"; // Your database name

private static final String USER = "root"; // Database username

private static final String PASSWORD = ""; // Database password

// Method to get a connection to the database

public static Connection getConnection() {

try {

// Load the MySQL JDBC driver

Class.forName("com.mysql.cj.jdbc.Driver");

return DriverManager.getConnection(URL, USER, PASSWORD);

} catch (SQLException | ClassNotFoundException e) {

e.printStackTrace();

return null;

}

}

}

package db;

import model.Player;

import model.Report;

import java.sql.\*;

import java.util.\*;

public class HighScoreDAO {

// Method to add a new high score

public boolean addHighScore(int playerId, int score, String level) {

String query = "INSERT INTO high\_scores (player\_id, score, level) VALUES (?, ?, ?)";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, playerId);

stmt.setInt(2, score);

stmt.setString(3, level);

return stmt.executeUpdate() > 0;

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

// Method to get high scores by difficulty level

public List<Player> getHighScoresByLevel(String level) {

List<Player> highScores = new ArrayList<>();

String query = "SELECT p.name, hs.score FROM high\_scores hs JOIN players p ON hs.player\_id = p.player\_id WHERE hs.level = ? ORDER BY hs.score DESC LIMIT 10";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, level);

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

// Use the parameterized constructor

Player player = new Player(rs.getString("name"), "", "", ""); // Add default values for password and level if needed

player.setScore(rs.getInt("score"));

highScores.add(player);

}

} catch (SQLException e) {

e.printStackTrace();

}

return highScores;

}

// Method to get all high scores

public List<Player> getAllHighScores() {

List<Player> highScores = new ArrayList<>();

String query = "SELECT p.name, hs.score, hs.level FROM high\_scores hs JOIN players p ON hs.player\_id = p.player\_id ORDER BY hs.score DESC";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

// Use the parameterized constructor

Player player = new Player(rs.getString("name"), "", "", ""); // Add default values for password and level if needed

player.setScore(rs.getInt("score"));

player.setLevel(rs.getString("level"));

highScores.add(player);

}

} catch (SQLException e) {

e.printStackTrace();

}

return highScores;

}

public Report getHighestScore() {

String query = "SELECT p.name, hs.score FROM high\_scores hs JOIN players p ON hs.player\_id = p.player\_id ORDER BY hs.score DESC LIMIT 1";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

Report report = new Report();

report.setPlayerName(rs.getString("name"));

report.setScore(rs.getInt("score"));

return report;

}

} catch (SQLException e) {

e.printStackTrace();

}

return null;

}

}

package db;

import model.Player;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class PlayerDAO {

// Method to check if the username is already taken

public boolean isUsernameAvailable(String username) {

String query = "SELECT \* FROM players WHERE username = ?";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, username);

ResultSet rs = stmt.executeQuery();

// If a player with this username exists, return false (username is not available)

return !rs.next(); // If rs.next() is false, the username is available

} catch (SQLException e) {

e.printStackTrace();

return false; // Return false in case of an error

}

}

// Method to register a new player

public boolean registerPlayer(Player player) {

String query = "INSERT INTO players (username, password, name, level) VALUES (?, ?, ?, ?)";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

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

stmt.setString(2, player.getPassword()); // Ensure password is hashed if required

stmt.setString(3, player.getName());

stmt.setString(4, player.getLevel());

return stmt.executeUpdate() > 0;

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

// Method to login a player

public int loginPlayer(String username, String password) {

String query = "SELECT player\_id FROM players WHERE username = ? AND password = ?";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, username);

stmt.setString(2, password); // In real applications, use hashed passwords

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return rs.getInt("player\_id"); // Return the player's ID if login is successful

}

} catch (SQLException e) {

e.printStackTrace();

}

return -1; // Return -1 if login fails

}

public List<Player> getAllPlayers() {

List<Player> players = new ArrayList<>();

String query = "SELECT player\_id, name, username FROM players";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

Player player = new Player();

player.setPlayerId(rs.getInt("player\_id"));

player.setName(rs.getString("name"));

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

players.add(player);

}

} catch (SQLException e) {

e.printStackTrace();

}

return players;

}

public String getPlayerDifficulty(int playerId) {

String query = "SELECT level FROM players WHERE player\_id = ?";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, playerId);

ResultSet rs = stmt.executeQuery();

if (rs.next()) {

return rs.getString("level"); // Fetch the difficulty level dynamically

}

} catch (SQLException e) {

e.printStackTrace();

}

return "Beginner"; // Default to Beginner if no level is found

}

}

package db;

import model.Question;

import java.sql.\*;

import java.util.\*;

public class QuestionDAO {

// Method to add a question to the database

public boolean addQuestion(Question question) {

String query = "INSERT INTO questions (question, option1, option2, option3, option4, correct\_answer, difficulty) VALUES (?, ?, ?, ?, ?, ?, ?)";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, question.getQuestion());

stmt.setString(2, question.getOption1());

stmt.setString(3, question.getOption2());

stmt.setString(4, question.getOption3());

stmt.setString(5, question.getOption4());

stmt.setInt(6, question.getCorrectAnswer());

stmt.setString(7, question.getDifficulty());

return stmt.executeUpdate() > 0;

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

// Method to update an existing question

public boolean updateQuestion(Question question) {

String query = "UPDATE questions SET question = ?, option1 = ?, option2 = ?, option3 = ?, option4 = ?, correct\_answer = ?, difficulty = ? WHERE question\_id = ?";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, question.getQuestion());

stmt.setString(2, question.getOption1());

stmt.setString(3, question.getOption2());

stmt.setString(4, question.getOption3());

stmt.setString(5, question.getOption4());

stmt.setInt(6, question.getCorrectAnswer());

stmt.setString(7, question.getDifficulty());

stmt.setInt(8, question.getQuestionId());

return stmt.executeUpdate() > 0;

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

// Method to delete a question

public boolean deleteQuestion(int questionId) {

String query = "DELETE FROM questions WHERE question\_id = ?";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, questionId);

return stmt.executeUpdate() > 0;

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

// Method to get all questions for a specific difficulty level

public List<Question> getQuestionsByDifficulty(String difficulty) {

List<Question> questions = new ArrayList<>();

String query = "SELECT \* FROM questions WHERE difficulty = ?";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setString(1, difficulty);

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

Question question = new Question(

rs.getInt("question\_id"),

rs.getString("question"),

rs.getString("option1"),

rs.getString("option2"),

rs.getString("option3"),

rs.getString("option4"),

rs.getInt("correct\_answer"),

rs.getString("difficulty")

);

questions.add(question);

}

} catch (SQLException e) {

e.printStackTrace();

}

return questions;

}

}

package db;

import model.Report;

import java.sql.\*;

import java.util.\*;

public class ReportDAO {

// Method to add a new report

public boolean addReport(Report report) {

String query = "INSERT INTO reports (player\_id, correct\_answers, score, difficulty) VALUES (?, ?, ?, ?)";

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query, Statement.RETURN\_GENERATED\_KEYS)) {

// Set parameters

stmt.setInt(1, report.getPlayerId());

stmt.setInt(2, report.getCorrectAnswers());

stmt.setInt(3, report.getScore());

stmt.setString(4, report.getDifficulty());

// Execute update and retrieve auto-generated reportId

int affectedRows = stmt.executeUpdate();

if (affectedRows > 0) {

ResultSet generatedKeys = stmt.getGeneratedKeys();

if (generatedKeys.next()) {

// Get the auto-generated reportId and set it in the Report object

report.setReportId(generatedKeys.getInt(1));

}

return true;

} else {

return false;

}

} catch (SQLException e) {

e.printStackTrace();

return false;

}

}

// Method to get all reports for a specific player

public List<Report> getReportsForPlayer(int playerId) {

List<Report> reports = new ArrayList<>();

String query = "SELECT \*, (correct\_answers + (10 - correct\_answers)) AS total\_questions FROM reports WHERE player\_id = ?";

// Replace `(10 - correct\_answers)` with actual logic for calculating total questions.

try (Connection conn = DBConnection.getConnection();

PreparedStatement stmt = conn.prepareStatement(query)) {

stmt.setInt(1, playerId);

ResultSet rs = stmt.executeQuery();

while (rs.next()) {

Report report = new Report();

report.setReportId(rs.getInt("report\_id"));

report.setPlayerId(rs.getInt("player\_id"));

report.setCorrectAnswers(rs.getInt("correct\_answers"));

report.setScore(rs.getInt("score"));

report.setDifficulty(rs.getString("difficulty"));

report.setTotalQuestions(rs.getInt("total\_questions")); // Set total questions

reports.add(report);

}

} catch (SQLException e) {

e.printStackTrace();

}

return reports;

}

}

Model:

package model;

public class HighScore {

private int highScoreId;

private int playerId;

private int score;

private String level;

// Constructor

public HighScore(int playerId, int score, String level) {

this.playerId = playerId;

this.score = score;

this.level = level;

}

// Getters and Setters

public int getHighScoreId() {

return highScoreId;

}

public void setHighScoreId(int highScoreId) {

this.highScoreId = highScoreId;

}

public int getPlayerId() {

return playerId;

}

public void setPlayerId(int playerId) {

this.playerId = playerId;

}

public int getScore() {

return score;

}

public void setScore(int score) {

this.score = score;

}

public String getLevel() {

return level;

}

public void setLevel(String level) {

this.level = level;

}

}

package model;

public class Player {

private int playerId;

private String username;

private String password;

private String name;

private String level;

private int score;

// No-argument constructor

public Player() {

}

// Constructor with parameters

public Player(String username, String password, String name, String level) {

this.username = username;

this.password = password;

this.name = name;

this.level = level;

}

// Getters and Setters

public int getPlayerId() {

return playerId;

}

public void setPlayerId(int playerId) {

this.playerId = playerId;

}

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getLevel() {

return level;

}

public void setLevel(String level) {

this.level = level;

}

public int getScore() {

return score;

}

public void setScore(int score) {

this.score = score;

}

}

package model;

public class Question {

private int questionId;

private String question;

private String option1;

private String option2;

private String option3;

private String option4;

private int correctAnswer;

private String difficulty;

// Constructor

public Question(int questionId, String question, String option1, String option2, String option3,

String option4, int correctAnswer, String difficulty) {

this.questionId = questionId;

this.question = question;

this.option1 = option1;

this.option2 = option2;

this.option3 = option3;

this.option4 = option4;

this.correctAnswer = correctAnswer;

this.difficulty = difficulty;

}

// Getters and Setters

public int getQuestionId() {

return questionId;

}

public void setQuestionId(int questionId) {

this.questionId = questionId;

}

public String getQuestion() {

return question;

}

public void setQuestion(String question) {

this.question = question;

}

public String getOption1() {

return option1;

}

public void setOption1(String option1) {

this.option1 = option1;

}

public String getOption2() {

return option2;

}

public void setOption2(String option2) {

this.option2 = option2;

}

public String getOption3() {

return option3;

}

public void setOption3(String option3) {

this.option3 = option3;

}

public String getOption4() {

return option4;

}

public void setOption4(String option4) {

this.option4 = option4;

}

public int getCorrectAnswer() {

return correctAnswer;

}

public void setCorrectAnswer(int correctAnswer) {

this.correctAnswer = correctAnswer;

}

public String getDifficulty() {

return difficulty;

}

public void setDifficulty(String difficulty) {

this.difficulty = difficulty;

}

}

package model;

public class Report {

private int reportId;

private int playerId;

private int correctAnswers;

private int wrongAnswers;

private int score;

private String difficulty;

private int totalQuestions;

private String playerName;

// Default constructor

public Report() {}

// Constructor for saving a report

public Report(int playerId, int correctAnswers, int wrongAnswers, int score, String difficulty) {

this.playerId = playerId;

this.correctAnswers = correctAnswers;

this.wrongAnswers = wrongAnswers;

this.score = score;

this.difficulty = difficulty;

}

// Getters and setters

public int getReportId() {

return reportId;

}

public void setReportId(int reportId) {

this.reportId = reportId;

}

public int getPlayerId() {

return playerId;

}

public void setPlayerId(int playerId) {

this.playerId = playerId;

}

public int getCorrectAnswers() {

return correctAnswers;

}

public void setCorrectAnswers(int correctAnswers) {

this.correctAnswers = correctAnswers;

}

public int getWrongAnswers() {

return wrongAnswers;

}

public void setWrongAnswers(int wrongAnswers) {

this.wrongAnswers = wrongAnswers;

}

public int getScore() {

return score;

}

public void setScore(int score) {

this.score = score;

}

public String getDifficulty() {

return difficulty;

}

public void setDifficulty(String difficulty) {

this.difficulty = difficulty;

}

public int getTotalQuestions() {

return totalQuestions;

}

public void setTotalQuestions(int totalQuestions) {

this.totalQuestions = totalQuestions;

}

public String getPlayerName() {

return playerName;

}

public void setPlayerName(String playerName) {

this.playerName = playerName;

}

}

Utils:

package utils;

import java.security.MessageDigest;

import java.security.NoSuchAlgorithmException;

public class PasswordUtils {

// Hash the password using SHA-256

public static String hashPassword(String password) {

try {

// Create MessageDigest instance for SHA-256 hashing

MessageDigest md = MessageDigest.getInstance("SHA-256");

// Add password bytes to digest

byte[] hash = md.digest(password.getBytes());

// Convert the byte array into a hexadecimal format

StringBuilder hexString = new StringBuilder();

for (byte b : hash) {

hexString.append(String.format("%02x", b));

}

// Return the hashed password

return hexString.toString();

} catch (NoSuchAlgorithmException e) {

e.printStackTrace();

return null; // In case of error, return null

}

}

}

package utils;

import model.Question;

import java.util.Collections;

import java.util.List;

public class ShuffleQuestions {

// Method to shuffle a list of questions

public static void shuffle(List<Question> questions) {

Collections.shuffle(questions);

}

}

package utils;

import db.PlayerDAO;

public class ValidationUtils {

// Check if the username already exists in the database

public static boolean isUsernameAvailable(String username) {

PlayerDAO playerDAO = new PlayerDAO();

return playerDAO.isUsernameAvailable(username); // Assume this method checks if the username exists

}

// Validate the password length (e.g., minimum 6 characters)

public static boolean isValidPassword(String password) {

return password.length() >= 6;

}

// Validate the email format

public static boolean isValidEmail(String email) {

String emailRegex = "^[a-zA-Z0-9\_+&\*-]+(?:\\.[a-zA-Z0-9\_+&\*-]+)\*@(?:[a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,7}$";

return email.matches(emailRegex);

}

}

Tessting

package test;

import db.PlayerDAO;

import model.Player;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.\*;

public class PlayerDAOTest {

private PlayerDAO playerDAO;

private String uniqueEmail;

private String password;

@BeforeEach

void setUp() {

playerDAO = new PlayerDAO();

// Generate a unique email and password for the test

uniqueEmail = "Test" + System.currentTimeMillis() + "@gmail.com";

password = "#123";

// Register the new player

Player newPlayer = new Player(uniqueEmail, password, "Prashansa", "Intermediate");

playerDAO.registerPlayer(newPlayer);

}

@Test

public void testLoginPlayer\_ValidCredentials() {

// Use the generated email and password for login

int playerId = playerDAO.loginPlayer(uniqueEmail, password);

assertTrue(playerId > 0, "Player login should return a valid ID");

}

@Test

public void testLoginPlayer\_InvalidCredentials() {

int playerId = playerDAO.loginPlayer("wronguser", "wrongpassword");

assertEquals(-1, playerId, "Invalid login should return -1");

}

@Test

public void testGetPlayerDifficulty() {

String difficulty = playerDAO.getPlayerDifficulty(1); // Assuming player ID 1 exists

assertNotNull(difficulty, "Difficulty level should not be null");

assertTrue(difficulty.equals("Beginner") || difficulty.equals("Intermediate") || difficulty.equals("Advanced"),

"Difficulty should be Beginner, Intermediate, or Advanced");

}

@Test

public void testRegisterNewPlayer() {

// Generate a unique email for each test run to avoid duplication in the database

String uniqueEmail = "Test" + System.currentTimeMillis() + "@gmail.com";

// Create Player object with the unique email

Player newPlayer = new Player(uniqueEmail, "#123", "Prashansa", "Intermediate");

// Register the player

boolean isRegistered = playerDAO.registerPlayer(newPlayer);

// Assert that the player was successfully registered

assertTrue(isRegistered, "New player should be registered successfully");

}

}

package test;

import controller.QuizController;

import model.Question;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import java.util.List;

import static org.junit.jupiter.api.Assertions.\*;

public class QuizControllerTest {

private QuizController quizController;

@BeforeEach

public void setUp() {

quizController = new QuizController();

}

@Test

public void testGetQuestionsByLevel\_Beginner() {

List<Question> questions = quizController.getQuestionsByLevel("Beginner");

assertNotNull(questions, "Beginner questions should not be null");

assertFalse(questions.isEmpty(), "Beginner questions list should not be empty");

}

@Test

public void testGetQuestionsByLevel\_Intermediate() {

List<Question> questions = quizController.getQuestionsByLevel("Intermediate");

assertNotNull(questions, "Intermediate questions should not be null");

assertFalse(questions.isEmpty(), "Intermediate questions list should not be empty");

}

@Test

public void testGetQuestionsByLevel\_Advanced() {

List<Question> questions = quizController.getQuestionsByLevel("Advanced");

assertNotNull(questions, "Advanced questions should not be null");

assertFalse(questions.isEmpty(), "Advanced questions list should not be empty");

}

@Test

public void testGetQuestionsByInvalidLevel() {

List<Question> questions = quizController.getQuestionsByLevel("InvalidLevel");

assertNotNull(questions, "Questions should not be null");

assertTrue(questions.isEmpty(), "Invalid level should return empty list");

}

}

package test;

import view.QuizFrame;

import org.junit.jupiter.api.Test;

import javax.swing.\*;

import static org.junit.jupiter.api.Assertions.\*;

public class QuizFrameTest {

@Test

public void testQuizFrameCreation() {

SwingUtilities.invokeLater(() -> {

QuizFrame quizFrame = new QuizFrame(1); // Mock player ID

assertNotNull(quizFrame, "QuizFrame should be created");

quizFrame.dispose(); // Clean up after test

});

}

}

package test;

import db.ReportDAO;

import model.Report;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import java.util.List;

import static org.junit.jupiter.api.Assertions.\*;

public class ReportDAOTest {

private ReportDAO reportDAO;

@BeforeEach

void setUp() {

reportDAO = new ReportDAO();

}

@Test

public void testAddReport() {

Report report = new Report(1, 6, 4, 60, "Intermediate"); // Mock data

boolean result = reportDAO.addReport(report);

assertTrue(result, "Report should be added successfully");

}

@Test

public void testGetReportsForPlayer() {

List<Report> reports = reportDAO.getReportsForPlayer(1); // Assuming player ID 1 exists

assertNotNull(reports, "Report list should not be null");

assertFalse(reports.isEmpty(), "Reports list should not be empty");

}

@Test

public void testReportValues() {

List<Report> reports = reportDAO.getReportsForPlayer(1);

assertTrue(reports.size() > 0, "Player should have at least one report");

Report report = reports.get(0);

assertTrue(report.getCorrectAnswers() >= 0 && report.getCorrectAnswers() <= report.getTotalQuestions(),

"Correct answers should be between 0 and totalQuestions");

assertTrue(report.getWrongAnswers() >= 0 && report.getWrongAnswers() <= report.getTotalQuestions(),

"Wrong answers should be between 0 and totalQuestions");

assertTrue(report.getScore() >= 0, "Score should not be negative");

}

}

package test;

import org.junit.runner.JUnitCore;

import org.junit.runner.Result;

import org.junit.runner.notification.Failure;

public class TestRunner {

public static void main(String[] args) {

Result result = JUnitCore.runClasses(

QuizControllerTest.class,

ReportDAOTest.class,

PlayerDAOTest.class

);

for (Failure failure : result.getFailures()) {

System.out.println(failure.toString());

}

System.out.println("All tests passed: " + result.wasSuccessful());

}

}

UI

package view;

import javax.swing.\*;

import controller.AdminController;

import model.Question;

import java.awt.\*;

public class AddQuestionFrame extends JFrame {

private static final long serialVersionUID = 1L;

private JTextField questionField;

private JTextField option1Field;

private JTextField option2Field;

private JTextField option3Field;

private JTextField option4Field;

private JComboBox<String> difficultyComboBox;

private JComboBox<Integer> correctAnswerComboBox;

public AddQuestionFrame() {

setTitle("Add New Question");

setSize(600, 500);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

// Main panel with padding

JPanel mainPanel = new JPanel();

mainPanel.setLayout(new GridLayout(8, 2, 10, 10)); // 8 rows, 2 columns, 10px spacing

mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20)); // Padding

add(mainPanel);

// Create the UI components

JLabel questionLabel = new JLabel("Enter Question:");

questionLabel.setFont(new Font("Arial", Font.PLAIN, 16));

questionField = new JTextField();

questionField.setFont(new Font("Arial", Font.PLAIN, 14));

JLabel option1Label = new JLabel("Option 1:");

option1Label.setFont(new Font("Arial", Font.PLAIN, 16));

option1Field = new JTextField();

option1Field.setFont(new Font("Arial", Font.PLAIN, 14));

JLabel option2Label = new JLabel("Option 2:");

option2Label.setFont(new Font("Arial", Font.PLAIN, 16));

option2Field = new JTextField();

option2Field.setFont(new Font("Arial", Font.PLAIN, 14));

JLabel option3Label = new JLabel("Option 3:");

option3Label.setFont(new Font("Arial", Font.PLAIN, 16));

option3Field = new JTextField();

option3Field.setFont(new Font("Arial", Font.PLAIN, 14));

JLabel option4Label = new JLabel("Option 4:");

option4Label.setFont(new Font("Arial", Font.PLAIN, 16));

option4Field = new JTextField();

option4Field.setFont(new Font("Arial", Font.PLAIN, 14));

JLabel correctAnswerLabel = new JLabel("Correct Answer:");

correctAnswerLabel.setFont(new Font("Arial", Font.PLAIN, 16));

correctAnswerComboBox = new JComboBox<>(new Integer[] {1, 2, 3, 4});

correctAnswerComboBox.setFont(new Font("Arial", Font.PLAIN, 14));

JLabel difficultyLabel = new JLabel("Difficulty:");

difficultyLabel.setFont(new Font("Arial", Font.PLAIN, 16));

difficultyComboBox = new JComboBox<>(new String[] {"Beginner", "Intermediate", "Advanced"});

difficultyComboBox.setFont(new Font("Arial", Font.PLAIN, 14));

JButton addButton = new JButton("Add Question");

addButton.setFont(new Font("Arial", Font.BOLD, 16));

addButton.setBackground(new Color(102, 178, 255)); // Light blue

addButton.setForeground(Color.BLACK);

addButton.setFocusPainted(false);

JButton doneButton = new JButton("Done");

doneButton.setFont(new Font("Arial", Font.BOLD, 16));

doneButton.setBackground(new Color(102, 255, 178)); // Light green

doneButton.setForeground(Color.BLACK);

doneButton.setFocusPainted(false);

// Add components to the panel

mainPanel.add(questionLabel);

mainPanel.add(questionField);

mainPanel.add(option1Label);

mainPanel.add(option1Field);

mainPanel.add(option2Label);

mainPanel.add(option2Field);

mainPanel.add(option3Label);

mainPanel.add(option3Field);

mainPanel.add(option4Label);

mainPanel.add(option4Field);

mainPanel.add(correctAnswerLabel);

mainPanel.add(correctAnswerComboBox);

mainPanel.add(difficultyLabel);

mainPanel.add(difficultyComboBox);

// Add buttons to the panel

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10));

buttonPanel.add(addButton);

buttonPanel.add(doneButton);

mainPanel.add(new JLabel()); // Placeholder

mainPanel.add(buttonPanel);

// Action listener for "Add Question" button

addButton.addActionListener(\_ -> {

String questionText = questionField.getText();

String option1 = option1Field.getText();

String option2 = option2Field.getText();

String option3 = option3Field.getText();

String option4 = option4Field.getText();

int correctAnswer = (int) correctAnswerComboBox.getSelectedItem();

String difficulty = (String) difficultyComboBox.getSelectedItem();

Question newQuestion = new Question(0, questionText, option1, option2, option3, option4, correctAnswer, difficulty);

AdminController adminController = new AdminController();

boolean isAdded = adminController.addQuestion(newQuestion);

if (isAdded) {

new CustomDialog(this, "Success", "Question Added Successfully!");

clearFields(); // Clear input fields for the next question

} else {

new CustomDialog(this, "Error", "Error Adding Question!");

}

});

// Action listener for "Done" button

doneButton.addActionListener(\_ -> {

new AdminDashboardFrame().setVisible(true); // Go back to Admin Dashboard

dispose(); // Close the current frame

});

}

// Helper method to clear input fields

private void clearFields() {

questionField.setText("");

option1Field.setText("");

option2Field.setText("");

option3Field.setText("");

option4Field.setText("");

correctAnswerComboBox.setSelectedIndex(0);

difficultyComboBox.setSelectedIndex(0);

}

}

package view;

import javax.swing.\*;

import java.awt.\*;

public class AdminDashboardFrame extends JFrame {

private static final long serialVersionUID = 1L;

public AdminDashboardFrame() {

setTitle("Admin Dashboard");

setSize(500, 500); // Increased size to fit the additional button

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

// Main panel with padding

JPanel mainPanel = new JPanel();

mainPanel.setLayout(new GridLayout(5, 1, 10, 10)); // 5 rows, 10px vertical spacing

mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20)); // Add padding

add(mainPanel, BorderLayout.CENTER);

// Create a top panel for the logout button

JPanel topPanel = new JPanel(new BorderLayout());

topPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 0, 10)); // Padding at the top

add(topPanel, BorderLayout.NORTH);

// Create the logout button

JButton logoutButton = new JButton("Logout");

styleLogoutButton(logoutButton);

topPanel.add(logoutButton, BorderLayout.EAST); // Align the button to the right

// Create buttons for different actions

JButton addQuestionButton = new JButton("Add Question");

JButton updateQuestionButton = new JButton("Update Question");

JButton deleteQuestionButton = new JButton("Delete Question");

JButton viewReportsButton = new JButton("View Reports");

JButton viewStudentsButton = new JButton("View Students"); // New Button

// Style the buttons

styleButton(addQuestionButton, new Color(102, 178, 255)); // Light blue

styleButton(updateQuestionButton, new Color(102, 255, 178)); // Light green

styleButton(deleteQuestionButton, new Color(255, 153, 153)); // Light red

styleButton(viewReportsButton, new Color(255, 255, 153)); // Light yellow

styleButton(viewStudentsButton, new Color(204, 153, 255)); // Light purple

// Add buttons to the panel

mainPanel.add(addQuestionButton);

mainPanel.add(updateQuestionButton);

mainPanel.add(deleteQuestionButton);

mainPanel.add(viewReportsButton);

mainPanel.add(viewStudentsButton);

// Action listeners for buttons

addQuestionButton.addActionListener(\_ -> new AddQuestionFrame().setVisible(true));

updateQuestionButton.addActionListener(\_ -> new UpdateQuestionFrame().setVisible(true));

deleteQuestionButton.addActionListener(\_ -> new DeleteQuestionFrame().setVisible(true));

viewReportsButton.addActionListener(\_ -> new ViewReportsFrame().setVisible(true));

viewStudentsButton.addActionListener(\_ -> new ViewStudentsFrame().setVisible(true)); // New functionality

// Action listener for the logout button

logoutButton.addActionListener(\_ -> {

App.main(null); // Go back to the first page

dispose(); // Close the current frame

});

}

// Method to style buttons

private void styleButton(JButton button, Color backgroundColor) {

button.setFont(new Font("Arial", Font.BOLD, 16));

button.setBackground(backgroundColor);

button.setForeground(Color.BLACK);

button.setFocusPainted(false);

button.setBorder(BorderFactory.createLineBorder(Color.DARK\_GRAY, 1)); // Add a subtle border

}

// Method to style the logout button

private void styleLogoutButton(JButton button) {

button.setFont(new Font("Arial", Font.PLAIN, 12));

button.setBackground(new Color(255, 102, 102)); // Light red

button.setForeground(Color.WHITE);

button.setFocusPainted(false);

button.setBorder(BorderFactory.createLineBorder(Color.DARK\_GRAY, 1));

}

}

package view;

import javax.swing.\*;

import controller.AdminController;

import java.awt.\*;

public class AdminLoginFrame extends JFrame {

private static final long serialVersionUID = 1L;

private JTextField usernameField;

private JPasswordField passwordField;

public AdminLoginFrame() {

setTitle("Admin Login");

setSize(450, 300);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

// Main panel with padding

JPanel mainPanel = new JPanel();

mainPanel.setLayout(new BorderLayout());

mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

add(mainPanel);

// Title label

JLabel titleLabel = new JLabel("Admin Login", JLabel.CENTER);

titleLabel.setFont(new Font("Arial", Font.BOLD, 20));

titleLabel.setForeground(new Color(51, 102, 255)); // Light blue

mainPanel.add(titleLabel, BorderLayout.NORTH);

// Form panel for username and password

JPanel formPanel = new JPanel(new GridLayout(2, 2, 10, 10));

formPanel.setBorder(BorderFactory.createEmptyBorder(20, 0, 20, 0));

mainPanel.add(formPanel, BorderLayout.CENTER);

JLabel usernameLabel = new JLabel("Username:");

usernameLabel.setFont(new Font("Arial", Font.PLAIN, 16));

JLabel passwordLabel = new JLabel("Password:");

passwordLabel.setFont(new Font("Arial", Font.PLAIN, 16));

usernameField = new JTextField();

usernameField.setFont(new Font("Arial", Font.PLAIN, 14));

passwordField = new JPasswordField();

passwordField.setFont(new Font("Arial", Font.PLAIN, 14));

formPanel.add(usernameLabel);

formPanel.add(usernameField);

formPanel.add(passwordLabel);

formPanel.add(passwordField);

// Button panel

JPanel buttonPanel = new JPanel(new FlowLayout());

JButton loginButton = new JButton("Login");

// Style the button

loginButton.setFont(new Font("Arial", Font.BOLD, 14));

loginButton.setBackground(new Color(102, 178, 255)); // Light blue

loginButton.setForeground(Color.BLACK);

loginButton.setFocusPainted(false);

loginButton.setPreferredSize(new Dimension(100, 30));

buttonPanel.add(loginButton);

mainPanel.add(buttonPanel, BorderLayout.SOUTH);

// Action listener for login button

loginButton.addActionListener(\_ -> {

String username = usernameField.getText();

String password = new String(passwordField.getPassword());

AdminController adminController = new AdminController();

boolean isLoggedIn = adminController.loginAdmin(username, password);

if (isLoggedIn) {

new CustomDialog(this, "Success", "Login Successful");

new AdminDashboardFrame().setVisible(true);

dispose();

} else {

new CustomDialog(this, "Error", "Invalid username or password");

}

});

}

}

package view;

import javax.swing.\*;

import java.awt.\*;

public class App {

public static void main(String[] args) {

// Create a custom JFrame for the mode selection

JFrame modeSelectionFrame = new JFrame("Quiz App");

modeSelectionFrame.setSize(400, 200);

modeSelectionFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

modeSelectionFrame.setLocationRelativeTo(null);

modeSelectionFrame.setLayout(new BorderLayout());

// Add a title label with custom styling

JLabel titleLabel = new JLabel("Select Mode:", JLabel.CENTER);

titleLabel.setFont(new Font("Arial", Font.BOLD, 18));

titleLabel.setForeground(Color.DARK\_GRAY);

modeSelectionFrame.add(titleLabel, BorderLayout.NORTH);

// Create buttons for Admin and Player

JButton adminButton = new JButton("Admin");

JButton playerButton = new JButton("Player");

// Style the buttons

adminButton.setFont(new Font("Arial", Font.PLAIN, 16));

adminButton.setBackground(new Color(102, 178, 255)); // Light blue

adminButton.setForeground(Color.BLACK);

adminButton.setFocusPainted(false);

playerButton.setFont(new Font("Arial", Font.PLAIN, 16));

playerButton.setBackground(new Color(102, 255, 178)); // Light green

playerButton.setForeground(Color.BLACK);

playerButton.setFocusPainted(false);

// Add action listeners to buttons

adminButton.addActionListener(\_ -> {

new AdminLoginFrame().setVisible(true); // Open Admin Login

modeSelectionFrame.dispose(); // Close the selection frame

});

playerButton.addActionListener(\_ -> {

new PlayerLoginFrame().setVisible(true); // Open Player Login

modeSelectionFrame.dispose(); // Close the selection frame

});

// Add buttons to a panel with spacing

JPanel buttonPanel = new JPanel();

buttonPanel.setLayout(new GridLayout(1, 2, 20, 0)); // Horizontal layout with spacing

buttonPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20)); // Add padding

buttonPanel.add(adminButton);

buttonPanel.add(playerButton);

modeSelectionFrame.add(buttonPanel, BorderLayout.CENTER);

// Show the frame

modeSelectionFrame.setVisible(true);

}

}

package view;

import javax.swing.\*;

import java.awt.\*;

public class CustomDialog extends JDialog {

private static final long serialVersionUID = 1L;

public CustomDialog(JFrame parent, String title, String message) {

super(parent, title, true);

// Set dialog size and layout

setSize(300, 150);

setLayout(new BorderLayout());

setLocationRelativeTo(parent); // Center the dialog relative to the parent frame

// Title Label

JLabel messageLabel = new JLabel(message, JLabel.CENTER);

messageLabel.setFont(new Font("Arial", Font.BOLD, 16));

messageLabel.setForeground(new Color(0, 102, 0)); // Dark green for success messages

// OK Button

JButton okButton = new JButton("OK");

okButton.setFont(new Font("Arial", Font.BOLD, 14));

okButton.setBackground(new Color(102, 178, 255)); // Light blue

okButton.setForeground(Color.BLACK);

okButton.setFocusPainted(false);

okButton.addActionListener(\_ -> dispose()); // Close dialog on click

// Add components to the dialog

add(messageLabel, BorderLayout.CENTER);

JPanel buttonPanel = new JPanel();

buttonPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 10, 10));

buttonPanel.add(okButton);

add(buttonPanel, BorderLayout.SOUTH);

// Set dialog to visible

setVisible(true);

}

}

package view;

import javax.swing.\*;

import controller.AdminController;

import java.awt.\*;

public class DeleteQuestionFrame extends JFrame {

private static final long serialVersionUID = 1L;

private JTextField questionIdField;

public DeleteQuestionFrame() {

setTitle("Delete Question");

setSize(450, 250);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

// Main panel with padding

JPanel mainPanel = new JPanel();

mainPanel.setLayout(new GridLayout(3, 1, 10, 10)); // 3 rows, 1 column, 10px vertical spacing

mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20)); // Padding

add(mainPanel);

// Create the UI components

JLabel questionIdLabel = new JLabel("Enter Question ID to Delete:");

questionIdLabel.setFont(new Font("Arial", Font.PLAIN, 16));

questionIdField = new JTextField();

questionIdField.setFont(new Font("Arial", Font.PLAIN, 14));

JButton deleteButton = new JButton("Delete Question");

deleteButton.setFont(new Font("Arial", Font.BOLD, 16));

deleteButton.setBackground(new Color(255, 153, 153)); // Light red

deleteButton.setForeground(Color.BLACK);

deleteButton.setFocusPainted(false);

JButton doneButton = new JButton("Done");

doneButton.setFont(new Font("Arial", Font.BOLD, 16));

doneButton.setBackground(new Color(102, 255, 178)); // Light green

doneButton.setForeground(Color.BLACK);

doneButton.setFocusPainted(false);

// Add components to the panel

mainPanel.add(questionIdLabel);

mainPanel.add(questionIdField);

// Button panel

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10));

buttonPanel.add(deleteButton);

buttonPanel.add(doneButton);

mainPanel.add(buttonPanel);

// Action listener for the "Delete Question" button

deleteButton.addActionListener(\_ -> {

try {

int questionId = Integer.parseInt(questionIdField.getText());

// Call AdminController to delete the question

AdminController adminController = new AdminController();

boolean isDeleted = adminController.deleteQuestion(questionId);

if (isDeleted) {

new CustomDialog(this, "Success", "Question Deleted Successfully!");

questionIdField.setText(""); // Clear the input field

} else {

new CustomDialog(this, "Error", "Error Deleting Question!");

}

} catch (NumberFormatException ex) {

new CustomDialog(this, "Error", "Invalid Question ID!");

}

});

// Action listener for the "Done" button

doneButton.addActionListener(\_ -> {

new AdminDashboardFrame().setVisible(true); // Return to Admin Dashboard

dispose(); // Close the current frame

});

}

}

package view;

import javax.swing.\*;

import controller.HighScoreController;

import model.Player;

import java.awt.\*;

import java.util.List;

public class HighScoreFrame extends JFrame {

private static final long serialVersionUID = 1L;

public HighScoreFrame() {

setTitle("High Scores");

setSize(500, 400);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

HighScoreController highScoreController = new HighScoreController();

// Create the UI components

JLabel levelLabel = new JLabel("Select Difficulty Level:");

JComboBox<String> levelComboBox = new JComboBox<>(new String[] {"Beginner", "Intermediate", "Advanced"});

JButton viewScoresButton = new JButton("View High Scores");

// Layout

setLayout(new FlowLayout());

add(levelLabel);

add(levelComboBox);

add(viewScoresButton);

// Display the scores in a list

JList<String> highScoreList = new JList<>();

add(new JScrollPane(highScoreList));

viewScoresButton.addActionListener(\_ -> {

String selectedLevel = (String) levelComboBox.getSelectedItem();

List<Player> highScores = highScoreController.getHighScoresByLevel(selectedLevel);

// Display the high scores in the list

DefaultListModel<String> listModel = new DefaultListModel<>();

for (Player player : highScores) {

listModel.addElement(player.getName() + " - " + player.getScore());

}

highScoreList.setModel(listModel);

});

}

}

package view;

import javax.swing.\*;

import java.awt.\*;

public class PlayerDashboardFrame extends JFrame {

private int playerId;

public PlayerDashboardFrame(int playerId) {

this.playerId = playerId;

setTitle("Player Dashboard");

setSize(400, 300);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

// Main panel with layout and padding

JPanel mainPanel = new JPanel();

mainPanel.setLayout(new GridLayout(3, 1, 10, 10)); // 3 rows, spacing

mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20)); // Padding

add(mainPanel);

// Buttons for actions

JButton startQuizButton = new JButton("Start Quiz");

JButton viewReportButton = new JButton("View Report");

JButton logoutButton = new JButton("Logout");

styleButton(startQuizButton, new Color(102, 178, 255)); // Light blue

styleButton(viewReportButton, new Color(102, 255, 178)); // Light green

styleButton(logoutButton, new Color(255, 153, 153)); // Light red

// Add buttons to the panel

mainPanel.add(startQuizButton);

mainPanel.add(viewReportButton);

mainPanel.add(logoutButton);

// Action listeners for buttons

startQuizButton.addActionListener(\_ -> {

new QuizFrame(playerId).setVisible(true); // Pass playerId when starting the quiz

dispose();

});

viewReportButton.addActionListener(\_ -> {

new PlayerReportFrame(playerId).setVisible(true); // Navigate to Report

dispose();

});

logoutButton.addActionListener(\_ -> {

new PlayerLoginFrame().setVisible(true); // Navigate back to login

dispose();

});

}

// Method to style buttons

private void styleButton(JButton button, Color backgroundColor) {

button.setFont(new Font("Arial", Font.BOLD, 16));

button.setBackground(backgroundColor);

button.setForeground(Color.BLACK);

button.setFocusPainted(false);

button.setBorder(BorderFactory.createLineBorder(Color.DARK\_GRAY, 1));

}

}

package view;

import javax.swing.\*;

import controller.PlayerController;

import java.awt.\*;

public class PlayerLoginFrame extends JFrame {

private static final long serialVersionUID = 1L;

private JTextField usernameField;

private JPasswordField passwordField;

public PlayerLoginFrame() {

setTitle("Player Login");

setSize(450, 350);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

// Create a main panel with padding

JPanel mainPanel = new JPanel();

mainPanel.setLayout(new BorderLayout());

mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

add(mainPanel);

// Title label

JLabel titleLabel = new JLabel("Player Login", JLabel.CENTER);

titleLabel.setFont(new Font("Arial", Font.BOLD, 20));

titleLabel.setForeground(new Color(51, 102, 255)); // Light blue

mainPanel.add(titleLabel, BorderLayout.NORTH);

// Form panel for username and password

JPanel formPanel = new JPanel(new GridLayout(2, 2, 10, 10));

formPanel.setBorder(BorderFactory.createEmptyBorder(20, 0, 20, 0));

JLabel usernameLabel = new JLabel("Username:");

usernameLabel.setFont(new Font("Arial", Font.PLAIN, 16));

JLabel passwordLabel = new JLabel("Password:");

passwordLabel.setFont(new Font("Arial", Font.PLAIN, 16));

usernameField = new JTextField();

usernameField.setFont(new Font("Arial", Font.PLAIN, 14));

passwordField = new JPasswordField();

passwordField.setFont(new Font("Arial", Font.PLAIN, 14));

formPanel.add(usernameLabel);

formPanel.add(usernameField);

formPanel.add(passwordLabel);

formPanel.add(passwordField);

mainPanel.add(formPanel, BorderLayout.CENTER);

// Button panel for Login and Register buttons

JPanel buttonPanel = new JPanel(new GridLayout(1, 2, 10, 0));

JButton loginButton = new JButton("Login");

JButton registerButton = new JButton("Register");

// Style the buttons

loginButton.setFont(new Font("Arial", Font.BOLD, 14));

loginButton.setBackground(new Color(102, 178, 255)); // Light blue

loginButton.setForeground(Color.BLACK);

loginButton.setFocusPainted(false);

registerButton.setFont(new Font("Arial", Font.BOLD, 14));

registerButton.setBackground(new Color(102, 255, 178)); // Light green

registerButton.setForeground(Color.BLACK);

registerButton.setFocusPainted(false);

buttonPanel.add(loginButton);

buttonPanel.add(registerButton);

mainPanel.add(buttonPanel, BorderLayout.SOUTH);

// Action listener for login button

loginButton.addActionListener(\_ -> {

String username = usernameField.getText();

String password = new String(passwordField.getPassword());

PlayerController playerController = new PlayerController();

int playerId = playerController.loginPlayer(username, password);

if (playerId != -1) { // If login is successful

new CustomDialog(this, "Message", "Login Successful");

new PlayerDashboardFrame(playerId).setVisible(true); // Navigate to Player Dashboard instead of Quiz

dispose();

} else {

JOptionPane.showMessageDialog(this, "Invalid username or password");

}

});

// Action listener for register button

registerButton.addActionListener(\_ -> {

new RegisterPlayerFrame().setVisible(true);

dispose(); // Close the login frame

});

}

}

package view;

import javax.swing.\*;

import controller.ReportController;

import model.Report;

import db.HighScoreDAO;

import java.awt.\*;

import java.util.List;

public class PlayerReportFrame extends JFrame {

private int playerId;

public PlayerReportFrame(int playerId) {

this.playerId = playerId;

setTitle("Player Report");

setSize(500, 400);

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

setLocationRelativeTo(null);

// Main panel

JPanel mainPanel = new JPanel();

mainPanel.setLayout(new BorderLayout());

mainPanel.setBorder(BorderFactory.*createEmptyBorder*(20, 20, 20, 20)); // Padding

add(mainPanel);

// Text area to display report details

JTextArea reportArea = new JTextArea();

reportArea.setFont(new Font("Monospaced", Font.***PLAIN***, 14));

reportArea.setEditable(false);

JScrollPane scrollPane = new JScrollPane(reportArea);

scrollPane.setBorder(BorderFactory.*createTitledBorder*("Report Details"));

JButton backButton = new JButton("Back");

styleBackButton(backButton);

// Fetch player report and highest score

fetchPlayerReport(reportArea);

backButton.addActionListener(\_ -> {

new PlayerDashboardFrame(playerId).setVisible(true); // Back to Player Dashboard

dispose();

});

// Layout

mainPanel.add(scrollPane, BorderLayout.***CENTER***);

mainPanel.add(backButton, BorderLayout.***SOUTH***);

}

private void fetchPlayerReport(JTextArea reportArea) {

ReportController reportController = new ReportController();

List<Report> reports = reportController.getReportsForPlayer(playerId);

HighScoreDAO highScoreDAO = new HighScoreDAO();

Report highestReport = highScoreDAO.getHighestScore();

StringBuilder reportText = new StringBuilder();

if (!reports.isEmpty()) {

for (Report report : reports) {

reportText.append("Correct Answers: ").append(report.getCorrectAnswers())

.append("\nWrong Answers: ").append(report.getTotalQuestions() - report.getCorrectAnswers())

.append("\nScore: ").append(report.getScore())

.append("\n-------------------------\n");

}

} else {

reportText.append("No reports available for this player.\n");

}

reportText.append("\nHighest Scorer:\n");

if (highestReport != null) {

reportText.append("Name: ").append(highestReport.getPlayerName())

.append("\nScore: ").append(highestReport.getScore());

} else {

// Treat the current player's score as the high score if no other scores exist

reportText.append("Name: Current Player")

.append("\nScore: ").append(reports.isEmpty() ? 0 : reports.get(0).getScore());

}

reportArea.setText(reportText.toString());

}

// Method to style the back button

private void styleBackButton(JButton button) {

button.setFont(new Font("Arial", Font.***BOLD***, 14));

button.setBackground(new Color(255, 153, 153)); // Light red

button.setForeground(Color.***BLACK***);

button.setFocusPainted(false);

button.setBorder(BorderFactory.*createLineBorder*(Color.***DARK\_GRAY***, 1));

}

}

package view;

import javax.swing.\*;

import controller.QuizController;

import db.PlayerDAO;

import db.ReportDAO;

import model.Question;

import model.Report;

import java.awt.\*;

import java.util.List;

public class QuizFrame extends JFrame {

private static final long serialVersionUID = 1L;

private int playerId;

private String difficulty; // Store the player's difficulty level

private List<Question> questions;

private int currentQuestionIndex = 0;

private int score = 0;

private int correctAnswers = 0;

private ButtonGroup buttonGroup;

public QuizFrame(int playerId) {

this.playerId = playerId;

// Fetch player's difficulty level dynamically

PlayerDAO playerDAO = new PlayerDAO();

this.difficulty = playerDAO.getPlayerDifficulty(playerId);

setTitle("Quiz - " + difficulty + " Level");

setSize(600, 400);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

// Fetch questions based on the player's level

QuizController quizController = new QuizController();

questions = quizController.getQuestionsByLevel(difficulty);

if (questions == null || questions.isEmpty()) {

JOptionPane.showMessageDialog(this, "No questions available for this level.", "Error", JOptionPane.ERROR\_MESSAGE);

return;

}

JPanel mainPanel = new JPanel(new BorderLayout());

mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

add(mainPanel);

JLabel questionLabel = new JLabel("<html><h3>" + questions.get(currentQuestionIndex).getQuestion() + "</h3></html>");

questionLabel.setFont(new Font("Arial", Font.PLAIN, 16));

mainPanel.add(questionLabel, BorderLayout.NORTH);

JPanel optionsPanel = new JPanel(new GridLayout(4, 1, 10, 10));

JRadioButton option1 = new JRadioButton();

JRadioButton option2 = new JRadioButton();

JRadioButton option3 = new JRadioButton();

JRadioButton option4 = new JRadioButton();

buttonGroup = new ButtonGroup();

buttonGroup.add(option1);

buttonGroup.add(option2);

buttonGroup.add(option3);

buttonGroup.add(option4);

optionsPanel.add(option1);

optionsPanel.add(option2);

optionsPanel.add(option3);

optionsPanel.add(option4);

mainPanel.add(optionsPanel, BorderLayout.CENTER);

loadQuestionOptions(option1, option2, option3, option4);

JPanel buttonPanel = new JPanel();

JButton nextButton = new JButton("Next");

nextButton.setFont(new Font("Arial", Font.BOLD, 14));

nextButton.setBackground(new Color(102, 178, 255));

nextButton.setPreferredSize(new Dimension(100, 30));

buttonPanel.add(nextButton);

mainPanel.add(buttonPanel, BorderLayout.SOUTH);

nextButton.addActionListener(e -> {

Question currentQuestion = questions.get(currentQuestionIndex);

if (option1.isSelected() && currentQuestion.getCorrectAnswer() == 1) {

score += 10;

correctAnswers++;

} else if (option2.isSelected() && currentQuestion.getCorrectAnswer() == 2) {

score += 10;

correctAnswers++;

} else if (option3.isSelected() && currentQuestion.getCorrectAnswer() == 3) {

score += 10;

correctAnswers++;

} else if (option4.isSelected() && currentQuestion.getCorrectAnswer() == 4) {

score += 10;

correctAnswers++;

}

currentQuestionIndex++;

if (currentQuestionIndex < questions.size()) {

loadQuestionOptions(option1, option2, option3, option4);

questionLabel.setText("<html><h3>" + questions.get(currentQuestionIndex).getQuestion() + "</h3></html>");

buttonGroup.clearSelection();

} else {

saveReport();

JOptionPane.showMessageDialog(this, "Quiz Over! Your Score: " + score);

dispose();

new PlayerDashboardFrame(playerId).setVisible(true);

}

});

}

private void loadQuestionOptions(JRadioButton option1, JRadioButton option2, JRadioButton option3, JRadioButton option4) {

Question currentQuestion = questions.get(currentQuestionIndex);

option1.setText(currentQuestion.getOption1());

option2.setText(currentQuestion.getOption2());

option3.setText(currentQuestion.getOption3());

option4.setText(currentQuestion.getOption4());

}

private void saveReport() {

int totalQuestions = questions.size();

int wrongAnswers = totalQuestions - correctAnswers;

// Save report with correct difficulty level

Report report = new Report(playerId, correctAnswers, wrongAnswers, score, difficulty);

ReportDAO reportDAO = new ReportDAO();

boolean isSaved = reportDAO.addReport(report);

if (!isSaved) {

JOptionPane.showMessageDialog(this, "Error saving score!");

}

}

}

package view;

import javax.swing.\*;

import controller.PlayerController;

import model.Player;

import java.awt.\*;

public class RegisterPlayerFrame extends JFrame {

private static final long serialVersionUID = 1L;

private JTextField usernameField;

private JPasswordField passwordField;

private JTextField nameField;

private JComboBox<String> levelComboBox;

public RegisterPlayerFrame() {

setTitle("Register New Player");

setSize(500, 400);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

// Main panel with padding

JPanel mainPanel = new JPanel();

mainPanel.setLayout(new GridLayout(5, 2, 10, 10)); // 5 rows, 10px spacing

mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20)); // Padding

add(mainPanel, BorderLayout.CENTER);

// Create a top panel for the back button

JPanel topPanel = new JPanel(new BorderLayout());

topPanel.setBorder(BorderFactory.createEmptyBorder(10, 10, 0, 10)); // Padding at the top

add(topPanel, BorderLayout.NORTH);

// Create the back button

JButton backButton = new JButton("Back");

styleBackButton(backButton);

topPanel.add(backButton, BorderLayout.WEST); // Align the button to the left

// Create UI components

JLabel usernameLabel = new JLabel("Username:");

JLabel passwordLabel = new JLabel("Password:");

JLabel nameLabel = new JLabel("Name:");

JLabel levelLabel = new JLabel("Level:");

usernameField = new JTextField();

passwordField = new JPasswordField();

nameField = new JTextField();

levelComboBox = new JComboBox<>(new String[]{"Beginner", "Intermediate", "Advanced"});

JButton registerButton = new JButton("Register");

// Style the labels and fields

styleLabel(usernameLabel);

styleLabel(passwordLabel);

styleLabel(nameLabel);

styleLabel(levelLabel);

styleTextField(usernameField);

styleTextField(passwordField);

styleTextField(nameField);

styleComboBox(levelComboBox);

// Style the register button

styleRegisterButton(registerButton);

// Add components to the main panel

mainPanel.add(usernameLabel);

mainPanel.add(usernameField);

mainPanel.add(passwordLabel);

mainPanel.add(passwordField);

mainPanel.add(nameLabel);

mainPanel.add(nameField);

mainPanel.add(levelLabel);

mainPanel.add(levelComboBox);

mainPanel.add(new JLabel()); // Placeholder for alignment

mainPanel.add(registerButton);

// Action listener for the register button

registerButton.addActionListener(\_ -> {

String username = usernameField.getText();

String password = new String(passwordField.getPassword());

String name = nameField.getText();

String level = (String) levelComboBox.getSelectedItem();

Player newPlayer = new Player(username, password, name, level);

PlayerController playerController = new PlayerController();

boolean isRegistered = playerController.registerPlayer(newPlayer);

if (isRegistered) {

new CustomDialog(this, "Success", "Registration Successful!");

dispose(); // Close the registration frame

new PlayerLoginFrame().setVisible(true); // Navigate to Player Login

} else {

new CustomDialog(this, "Error", "Error Registering Player!");

}

});

// Action listener for the back button

backButton.addActionListener(\_ -> {

new PlayerLoginFrame().setVisible(true); // Return to Player Login

dispose(); // Close the current frame

});

}

// Method to style labels

private void styleLabel(JLabel label) {

label.setFont(new Font("Arial", Font.PLAIN, 16));

}

// Method to style text fields and combo box

private void styleTextField(JTextField textField) {

textField.setFont(new Font("Arial", Font.PLAIN, 14));

}

private void styleComboBox(JComboBox<String> comboBox) {

comboBox.setFont(new Font("Arial", Font.PLAIN, 14));

}

// Method to style the register button

private void styleRegisterButton(JButton button) {

button.setFont(new Font("Arial", Font.BOLD, 16));

button.setBackground(new Color(102, 178, 255)); // Light blue

button.setForeground(Color.BLACK);

button.setFocusPainted(false);

button.setBorder(BorderFactory.createLineBorder(Color.DARK\_GRAY, 1));

}

// Method to style the back button

private void styleBackButton(JButton button) {

button.setFont(new Font("Arial", Font.PLAIN, 12));

button.setBackground(new Color(255, 153, 153)); // Light red

button.setForeground(Color.WHITE);

button.setFocusPainted(false);

button.setBorder(BorderFactory.createLineBorder(Color.DARK\_GRAY, 1));

}

}

package view;

import javax.swing.\*;

import controller.AdminController;

import model.Question;

import java.awt.\*;

public class UpdateQuestionFrame extends JFrame {

private static final long serialVersionUID = 1L;

private JTextField questionField;

private JTextField option1Field;

private JTextField option2Field;

private JTextField option3Field;

private JTextField option4Field;

private JComboBox<String> difficultyComboBox;

private JComboBox<Integer> correctAnswerComboBox;

private JTextField questionIdField;

public UpdateQuestionFrame() {

setTitle("Update Question");

setSize(600, 500);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setLocationRelativeTo(null);

// Main panel with padding

JPanel mainPanel = new JPanel();

mainPanel.setLayout(new GridLayout(9, 2, 10, 10)); // 9 rows, 2 columns, 10px spacing

mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20)); // Padding

add(mainPanel);

// Create the UI components

JLabel questionIdLabel = new JLabel("Enter Question ID:");

questionIdLabel.setFont(new Font("Arial", Font.PLAIN, 16));

questionIdField = new JTextField();

questionIdField.setFont(new Font("Arial", Font.PLAIN, 14));

JLabel questionLabel = new JLabel("Enter Question:");

questionLabel.setFont(new Font("Arial", Font.PLAIN, 16));

questionField = new JTextField();

questionField.setFont(new Font("Arial", Font.PLAIN, 14));

JLabel option1Label = new JLabel("Option 1:");

option1Label.setFont(new Font("Arial", Font.PLAIN, 16));

option1Field = new JTextField();

option1Field.setFont(new Font("Arial", Font.PLAIN, 14));

JLabel option2Label = new JLabel("Option 2:");

option2Label.setFont(new Font("Arial", Font.PLAIN, 16));

option2Field = new JTextField();

option2Field.setFont(new Font("Arial", Font.PLAIN, 14));

JLabel option3Label = new JLabel("Option 3:");

option3Label.setFont(new Font("Arial", Font.PLAIN, 16));

option3Field = new JTextField();

option3Field.setFont(new Font("Arial", Font.PLAIN, 14));

JLabel option4Label = new JLabel("Option 4:");

option4Label.setFont(new Font("Arial", Font.PLAIN, 16));

option4Field = new JTextField();

option4Field.setFont(new Font("Arial", Font.PLAIN, 14));

JLabel correctAnswerLabel = new JLabel("Correct Answer:");

correctAnswerLabel.setFont(new Font("Arial", Font.PLAIN, 16));

correctAnswerComboBox = new JComboBox<>(new Integer[]{1, 2, 3, 4});

correctAnswerComboBox.setFont(new Font("Arial", Font.PLAIN, 14));

JLabel difficultyLabel = new JLabel("Difficulty:");

difficultyLabel.setFont(new Font("Arial", Font.PLAIN, 16));

difficultyComboBox = new JComboBox<>(new String[]{"Beginner", "Intermediate", "Advanced"});

difficultyComboBox.setFont(new Font("Arial", Font.PLAIN, 14));

JButton updateButton = new JButton("Update Question");

updateButton.setFont(new Font("Arial", Font.BOLD, 16));

updateButton.setBackground(new Color(102, 178, 255)); // Light blue

updateButton.setForeground(Color.BLACK);

updateButton.setFocusPainted(false);

JButton doneButton = new JButton("Done");

doneButton.setFont(new Font("Arial", Font.BOLD, 16));

doneButton.setBackground(new Color(102, 255, 178)); // Light green

doneButton.setForeground(Color.BLACK);

doneButton.setFocusPainted(false);

// Add components to the panel

mainPanel.add(questionIdLabel);

mainPanel.add(questionIdField);

mainPanel.add(questionLabel);

mainPanel.add(questionField);

mainPanel.add(option1Label);

mainPanel.add(option1Field);

mainPanel.add(option2Label);

mainPanel.add(option2Field);

mainPanel.add(option3Label);

mainPanel.add(option3Field);

mainPanel.add(option4Label);

mainPanel.add(option4Field);

mainPanel.add(correctAnswerLabel);

mainPanel.add(correctAnswerComboBox);

mainPanel.add(difficultyLabel);

mainPanel.add(difficultyComboBox);

// Button panel

JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 10, 10));

buttonPanel.add(updateButton);

buttonPanel.add(doneButton);

mainPanel.add(new JLabel()); // Placeholder for alignment

mainPanel.add(buttonPanel);

// Action listener for "Update Question" button

updateButton.addActionListener(\_ -> {

try {

int questionId = Integer.parseInt(questionIdField.getText());

String questionText = questionField.getText();

String option1 = option1Field.getText();

String option2 = option2Field.getText();

String option3 = option3Field.getText();

String option4 = option4Field.getText();

int correctAnswer = (int) correctAnswerComboBox.getSelectedItem();

String difficulty = (String) difficultyComboBox.getSelectedItem();

// Create updated question object

Question updatedQuestion = new Question(questionId, questionText, option1, option2, option3, option4, correctAnswer, difficulty);

AdminController adminController = new AdminController();

boolean isUpdated = adminController.addQuestion(updatedQuestion);

if (isUpdated) {

new CustomDialog(this, "Success", "Question Updated Successfully!");

clearFields(); // Clear the fields after update

} else {

new CustomDialog(this, "Error", "Error Updating Question!");

}

} catch (NumberFormatException ex) {

new CustomDialog(this, "Error", "Invalid Question ID!");

}

});

// Action listener for "Done" button

doneButton.addActionListener(\_ -> {

new AdminDashboardFrame().setVisible(true); // Return to Admin Dashboard

dispose(); // Close the current frame

});

}

// Helper method to clear input fields

private void clearFields() {

questionIdField.setText("");

questionField.setText("");

option1Field.setText("");

option2Field.setText("");

option3Field.setText("");

option4Field.setText("");

correctAnswerComboBox.setSelectedIndex(0);

difficultyComboBox.setSelectedIndex(0);

}

}

package view;

import javax.swing.\*;

import controller.ReportController;

import model.Report;

import java.awt.\*;

import java.util.List;

public class ViewReportsFrame extends JFrame {

private static final long ***serialVersionUID*** = 1L;

private JTextField playerIdField;

private JTextArea reportsArea;

public ViewReportsFrame() {

setTitle("View Reports");

setSize(600, 500);

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

setLocationRelativeTo(null);

// Main panel with padding

JPanel mainPanel = new JPanel();

mainPanel.setLayout(new BorderLayout());

mainPanel.setBorder(BorderFactory.*createEmptyBorder*(20, 20, 20, 20));

add(mainPanel);

// Input panel for player ID, view reports button, and back button

JPanel inputPanel = new JPanel(new FlowLayout(FlowLayout.***CENTER***, 10, 10));

JLabel playerIdLabel = new JLabel("Enter Player ID to View Reports:");

playerIdLabel.setFont(new Font("Arial", Font.***PLAIN***, 16));

playerIdField = new JTextField(15);

playerIdField.setFont(new Font("Arial", Font.***PLAIN***, 14));

JButton viewReportsButton = new JButton("View Reports");

viewReportsButton.setFont(new Font("Arial", Font.***BOLD***, 14));

viewReportsButton.setBackground(new Color(102, 178, 255)); // Light blue

viewReportsButton.setForeground(Color.***BLACK***);

viewReportsButton.setFocusPainted(false);

JButton backButton = new JButton("Back");

backButton.setFont(new Font("Arial", Font.***BOLD***, 14));

backButton.setBackground(new Color(255, 153, 153)); // Light red

backButton.setForeground(Color.***BLACK***);

backButton.setFocusPainted(false);

inputPanel.add(playerIdLabel);

inputPanel.add(playerIdField);

inputPanel.add(viewReportsButton);

inputPanel.add(backButton);

// Text area for displaying reports

reportsArea = new JTextArea();

reportsArea.setFont(new Font("Monospaced", Font.***PLAIN***, 14));

reportsArea.setEditable(false);

reportsArea.setBorder(BorderFactory.*createLineBorder*(Color.***LIGHT\_GRAY***));

JScrollPane scrollPane = new JScrollPane(reportsArea);

scrollPane.setBorder(BorderFactory.*createTitledBorder*("Reports"));

scrollPane.setPreferredSize(new Dimension(550, 350));

mainPanel.add(inputPanel, BorderLayout.***NORTH***);

mainPanel.add(scrollPane, BorderLayout.***CENTER***);

// Action listener for the "View Reports" button

viewReportsButton.addActionListener(\_ -> {

try {

int playerId = Integer.*parseInt*(playerIdField.getText());

// Fetch reports for the player

ReportController reportController = new ReportController();

List<Report> reports = reportController.getReportsForPlayer(playerId);

// Display the reports in the text area

StringBuilder reportText = new StringBuilder();

if (!reports.isEmpty()) {

for (Report report : reports) {

reportText.append("Report ID: ").append(report.getReportId())

.append("\nCorrect Answers: ").append(report.getCorrectAnswers())

.append("\nScore: ").append(report.getScore())

.append("\nDifficulty: ").append(report.getDifficulty())

.append("\n--------------------\n");

}

} else {

reportText.append("No reports found for Player ID: ").append(playerId);

}

reportsArea.setText(reportText.toString());

} catch (NumberFormatException ex) {

new CustomDialog(this, "Error", "Please enter a valid Player ID!");

}

});

// Action listener for the "Back" button

backButton.addActionListener(\_ -> {

new AdminDashboardFrame().setVisible(true); // Return to Admin Dashboard

dispose(); // Close the current frame

});

}

}

package view;

import javax.swing.\*;

import javax.swing.table.DefaultTableModel;

import db.PlayerDAO;

import model.Player;

import java.awt.\*;

import java.util.List;

public class ViewStudentsFrame extends JFrame {

private static final long serialVersionUID = 1L;

public ViewStudentsFrame() {

setTitle("View Students");

setSize(600, 400);

setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);

setLocationRelativeTo(null);

// Main panel

JPanel mainPanel = new JPanel(new BorderLayout());

mainPanel.setBorder(BorderFactory.createEmptyBorder(20, 20, 20, 20));

add(mainPanel);

// Table to display student details

String[] columnNames = {"Student ID", "Name", "Email"};

DefaultTableModel tableModel = new DefaultTableModel(columnNames, 0);

JTable studentTable = new JTable(tableModel);

// Fetch student data from the database

PlayerDAO playerDAO = new PlayerDAO();

List<Player> students = playerDAO.getAllPlayers(); // Assuming `getAllPlayers` is implemented

// Populate the table with student data

for (Player student : students) {

Object[] row = {student.getPlayerId(), student.getName(), student.getUsername()};

tableModel.addRow(row);

}

// Scroll pane for the table

JScrollPane scrollPane = new JScrollPane(studentTable);

mainPanel.add(scrollPane, BorderLayout.CENTER);

// Back button

JButton backButton = new JButton("Back");

backButton.setFont(new Font("Arial", Font.BOLD, 14));

backButton.setBackground(new Color(255, 153, 153));

backButton.setForeground(Color.BLACK);

backButton.setFocusPainted(false);

backButton.setBorder(BorderFactory.createLineBorder(Color.DARK\_GRAY, 1));

backButton.addActionListener(\_ -> dispose()); // Close the current frame

mainPanel.add(backButton, BorderLayout.SOUTH);

}

}

8. Known Bugs and Limitations

Security:

Plain text passwords stored in the database, which is an extreme security flaw.

Functionality:

Question randomization has not been implemented in the QuizController.

Statistical reporting (total players, maximum score for each level) is not completed.

Input validation is not completed, making the system susceptible to invalid input.

Error Handling: There is basic error handling, but the mechanisms need to be more robust.

Get all method not construcor

Java Quiz Application is a good start to a stable educational application. It has basic features, follows MVC architecture, and has unit tests for verification. Nevertheless, tackling the observed shortcomings, specifically on security, functionality, and exception handling, is essential in making the overall quality and dependability of the application better. Future improvements include the implementation of password hashing, completion of statistical reports, enhancement of error handling using custom exceptions and logging, and the addition of comprehensive input validation. The utilization of Javadoc comments would greatly enhance maintainability. The implementation of the default constructor in the Player class is crucial to the effective functionality of the getAllPlayers() methods. By making these improvements, the quiz application can be a more secure, feature-rich, and reliable testing and knowledge augmentation platform for students

The implementation of the default constructor in the Player class is crucial to the effective functionality of the getAllPlayers() methods