Advanced Java Programming

|  |  |
| --- | --- |
| Student Admission System Project | |
| **Name** | **Humber N number** |
| Sunny Chauhan | N 01658618 |
| Aleem Wadhwaniya | N 01639084 |

# Student Admission Management System Overview

## Purpose:

The Student Admission Management System is designed to streamline the process of managing student admissions for educational institutions, specifically focusing on:

1. **Efficiency:** Automating and simplifying the admission process from application submission to enrollment.
2. **User Roles:** Providing different access levels for administrators, registrars, and applicants, ensuring security and data privacy.

## Key Features:

1. **Student Registration:**

* **Application Form:** Enables students to fill out and submit admission applications online.
* **Document Upload:** Allows applicants to upload required documents (e.g., transcripts, ID proofs).
* **Login and Signup Pages:** Provides secure authentication and registration for applicants.

1. **Admin Dashboard:**

* **Dashboard Interface:** Provides administrators with a centralized dashboard for managing admissions.
* **Registrar Management:** Allows adding, updating, and deleting registrar information (e.g., name, contact details).

1. **Registrar Dashboard:**

* **Registrar Interface:** Allows registrars to view, update, and manage student applications.
* **Document Verification:** Enables registrars to verify uploaded documents and update application statuses.
* **Report Generation:** Generates reports on student admissions, including applicant details and enrollment statistics.

## Technologies Used:

* **Programming Languages:** Java, JavaFX for the frontend UI.
* **Database: MySQL** (In Phase 2)
* **Frameworks/Libraries:** JavaFX for building rich desktop applications, JDBC for database connectivity.

## Benefits:

* **Efficiency:** Reduces manual effort in processing applications and managing student records.
* **Accuracy:** Minimizes errors associated with manual data entry and processing.
* **Accessibility:** Provides 24/7 access to application status updates for applicants and real-time data for administrators and registrars.

## Database Schema

* **Admin**

## A screenshot of a computer Description automatically generated

* **Registrar**

A screenshot of a computer

Description automatically generated

* **Student**

A screenshot of a computer

Description automatically generated

A group of squares with black text

Description automatically generated

## Project Flow (MVC Structure)

A screenshot of a computer

Description automatically generated

1. **Model (M):**

* **Student.java:** Represents the student entity with fields like name, email, phone number, etc.
* **Registrar.java:** Represents the registrar entity with fields like ID, name, email, phone number, department, and address.

1. **View (V):**

* **Login.java:** Handles the login interface and logic.
* **Signup. java:** Manages the signup interface and registration logic.
* **StudentDashboard.java:** Displays the student dashboard with profile information, update profile functionality, document upload, show result, status, and notifications.
* **AdminDashboard.java:** Shows the admin dashboard with registrar management (add, update, delete), report generation, and logout functionality.
* **RegistrarDashboard.java:** Displays the registrar dashboard with student application management, document verification, and application status updates.

1. **Controller (C):**

* **DBConnect.java:** Provides database connectivity and management. Ensures efficient interaction with the MySQL database, handling connections and queries.
* **CSVGenerator.java:** Facilitates exporting data into CSV format, supporting functionalities like report generation and data backup.
* **RegisterDAO.java:** Manages operations related to registrar records, including adding, updating, deleting, and retrieving registrar data from the database.
* **StudentDAO.java:** Handles operations related to student records, including adding, updating, deleting, retrieving, and searching student information.

## Code And Screenshots

1. **Resources:**

|  |
| --- |
|  |
| Style.css |
| /\* style.css \*/ .root {  -fx-font-family: 'Arial';  -fx-font-size: 14px;  -fx-background-color: #f0f4f8; }  .label {  -fx-font-weight: bold; }  .text-field, .password-field {  -fx-pref-width: 200px; }  .login-button {  -fx-background-color: #4CAF50;  -fx-text-fill: white;  -fx-font-size: 14px;  -fx-padding: 8 16; }  .login-button:hover {  -fx-background-color: #45a049; }  .signup-button {  -fx-background-color: #2196F3;  -fx-text-fill: white;  -fx-font-size: 14px;  -fx-padding: 8 16; }  .signup-button:hover {  -fx-background-color: #1976D2; }  .grid-pane {  -fx-background-color: white;  -fx-border-color: #d3d3d3;  -fx-border-width: 1;  -fx-border-radius: 5;  -fx-padding: 20;  -fx-background-radius: 5; }  .hbox {  -fx-spacing: 10px; }  .dashboard-button {  -fx-font-size: 16px;  -fx-padding: 10px 20px;  -fx-background-color: #3498db;  -fx-text-fill: white;  -fx-border-radius: 5;  -fx-background-radius: 5; }  .dashboard-button:hover {  -fx-background-color: #2980b9; }  .logout-button {  -fx-font-size: 14px;  -fx-padding: 5px 10px;  -fx-background-color: #e74c3c;  -fx-text-fill: white;  -fx-border-radius: 5;  -fx-background-radius: 5; }  .logout-button:hover {  -fx-background-color: #c0392b; }  .welcome-label {  -fx-font-size: 24px;  -fx-font-weight: bold; }  .header {  -fx-background-color: #f0f8ff;  -fx-padding: 10px;  -fx-alignment: center; } |

1. **Login Page:**

|  |
| --- |
| LoginApplication.java |
| package com.example.studentadmissionsystem.View;  import com.example.studentadmissionsystem.Controller.DBConnect; import com.example.studentadmissionsystem.Controller.StudentDAO; import javafx.application.Application; import javafx.geometry.Insets; import javafx.geometry.Pos; import javafx.scene.Scene; import javafx.scene.control.\*; import javafx.scene.image.Image; import javafx.scene.image.ImageView; import javafx.scene.layout.GridPane; import javafx.scene.layout.HBox; import javafx.scene.layout.VBox; import javafx.stage.Stage;  import java.sql.Connection; import java.sql.SQLException;  public class LoginApplication extends Application {   public static void main(String[] args) {  *launch*(args);  }   @Override  public void start(Stage primaryStage) {  // Create UI components  Label usernameLabel = new Label("Username:");  TextField usernameField = new TextField();  Label passwordLabel = new Label("Password:");  PasswordField passwordField = new PasswordField();  Button loginButton = new Button("Login");  Button signupButton = new Button("Sign Up");  Label errorMessageLabel = new Label();  errorMessageLabel.setStyle("-fx-text-fill: red;");  errorMessageLabel.setWrapText(true);   // Apply style classes to buttons  loginButton.getStyleClass().add("login-button");  signupButton.getStyleClass().add("signup-button");   // Create layout  GridPane gridPane = new GridPane();  gridPane.setAlignment(Pos.*CENTER*);  gridPane.setHgap(10);  gridPane.setVgap(10);  gridPane.setPadding(new Insets(25, 25, 25, 25));   gridPane.add(usernameLabel, 0, 0);  gridPane.add(usernameField, 1, 0);  gridPane.add(passwordLabel, 0, 1);  gridPane.add(passwordField, 1, 1);  gridPane.add(errorMessageLabel, 1, 2, 2, 1);   HBox hBox = new HBox(10);  hBox.setAlignment(Pos.*BOTTOM\_RIGHT*);  hBox.getChildren().addAll(loginButton, signupButton);  gridPane.add(hBox, 1, 3);   // Handle login button click  loginButton.setOnAction(event -> handleLogin(usernameField, passwordField, errorMessageLabel, primaryStage));   // Handle signup button click  signupButton.setOnAction(event -> handleSignup(primaryStage));   // Load the logo image  ImageView logoImageView = new ImageView();  try {  Image logoImage = new Image(getClass().getResource("/college\_logo.png").toExternalForm());  logoImageView.setImage(logoImage);  logoImageView.setFitHeight(100);  logoImageView.setPreserveRatio(true);  } catch (NullPointerException e) {  System.*out*.println("Logo image not found");  }   // Create the root layout  VBox root = new VBox(10);  root.setAlignment(Pos.*TOP\_CENTER*);  root.getChildren().addAll(logoImageView, gridPane);  root.setPadding(new Insets(10, 10, 10, 10));   Scene scene = new Scene(root, 380, 350); // Increased height to accommodate error message  try {  scene.getStylesheets().add(getClass().getResource("/Style.css").toExternalForm());  } catch (NullPointerException e) {  System.*out*.println("CSS file not found");  }   primaryStage.setTitle("Login Page");  primaryStage.setScene(scene);  primaryStage.show();  }   private void handleLogin(TextField usernameField, PasswordField passwordField, Label errorMessageLabel, Stage primaryStage) {  String username = usernameField.getText();  String password = passwordField.getText();   if (username.isEmpty() || password.isEmpty()) {  errorMessageLabel.setText("Username and password cannot be empty.");  } else {  errorMessageLabel.setText("");  DBConnect dbConnect = new DBConnect();  try (Connection connection = dbConnect.*getConnection*()) {  StudentDAO studentDAO = new StudentDAO(connection);  boolean isAuthenticated = studentDAO.authenticateUser(username, password);   System.*out*.println("Authentication result: " + isAuthenticated);   if (isAuthenticated) {  if (username.equals("admin@humber.ca") && password.equals("admin1234")) {  clearFields(usernameField, passwordField);  primaryStage.hide();  AdminDashboard adminDashboard = new AdminDashboard();  Stage adminStage = new Stage();  adminDashboard.start(adminStage);  } else if (username.contains("admissions") || username.contains("finance") ||  username.contains("it") || username.contains("marketing") ||  username.contains("humanresources")) {   clearFields(usernameField, passwordField);  primaryStage.hide();   // Pass the username and password to the RegistrarDashboard  RegistrarDashboard registrarDashboard = new RegistrarDashboard(username, password);  Stage registrarStage = new Stage();  registrarDashboard.start(registrarStage);  } else {  clearFields(usernameField, passwordField);  primaryStage.hide();  StudentDashboard studentDashboard = new StudentDashboard(username, password);  Stage dashboardStage = new Stage();  studentDashboard.start(dashboardStage);  }  } else {  errorMessageLabel.setText("Invalid username or password.");  }  } catch (SQLException e) {  e.printStackTrace();  errorMessageLabel.setText("An error occurred during login.");  System.*err*.println("SQL Exception: " + e.getMessage());  }  }  }   private void handleSignup(Stage primaryStage) {  System.*out*.println("Redirecting to Sign Up page...");  primaryStage.hide();  SignupApplication signupApp = new SignupApplication();  Stage signupStage = new Stage();  signupApp.start(signupStage);  }   // Helper method to clear text fields  private void clearFields(TextField usernameField, PasswordField passwordField) {  usernameField.clear();  passwordField.clear();  }  } |
| Output & Validation : |

1. **Signup Page:**

|  |
| --- |
| SignupApplication.java |
| package com.example.studentadmissionsystem.View;  import com.example.studentadmissionsystem.Controller.DBConnect;  import com.example.studentadmissionsystem.Controller.StudentDAO;  import com.example.studentadmissionsystem.Model.Student;  import javafx.application.Application;  import javafx.collections.FXCollections;  import javafx.collections.ObservableList;  import javafx.geometry.Insets;  import javafx.geometry.Pos;  import javafx.scene.Scene;  import javafx.scene.control.\*;  import javafx.scene.image.Image;  import javafx.scene.image.ImageView;  import javafx.scene.layout.GridPane;  import javafx.scene.layout.HBox;  import javafx.scene.layout.VBox;  import javafx.stage.Stage;  import java.sql.Connection;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import java.sql.SQLException;  public class SignupApplication extends Application {  public static void main(String[] args) {  launch(args);  }  @Override  public void start(Stage primaryStage) {  // Create UI components  Label fullNameLabel = new Label("Full Name:");  TextField fullNameField = new TextField();  Label emailLabel = new Label("Email:");  TextField emailField = new TextField();  Label passwordLabel = new Label("Password:");  PasswordField passwordField = new PasswordField();  Label confirmPasswordLabel = new Label("Confirm Password:");  PasswordField confirmPasswordField = new PasswordField();  Label dobLabel = new Label("Date of Birth:");  DatePicker dobPicker = new DatePicker();  Label genderLabel = new Label("Gender:");  ComboBox<String> genderComboBox = new ComboBox<>();  ObservableList<String> gender = FXCollections.observableArrayList(  "Male", "Female", "Other"  );  genderComboBox.setItems(gender);  genderComboBox.setPromptText("Select Gender");  Label phoneNumberLabel = new Label("Phone Number:");  TextField phoneNumberField = new TextField();  Label citizenshipLabel = new Label("Citizenship:");  // Citizenship ComboBox  ComboBox<String> citizenshipComboBox = new ComboBox<>();  ObservableList<String> citizenship = FXCollections.observableArrayList(  "United States", "Canada", "United Kingdom", "Australia", "Germany", "France", "Japan", "India",  "China", "South Korea", "Brazil", "Mexico", "Russia", "South Africa", "Other"  );  citizenshipComboBox.setItems(citizenship);  citizenshipComboBox.setPromptText("Select Citizenship");  citizenshipComboBox.getStyleClass().add("combo-box");  CheckBox termsCheckBox = new CheckBox("I agree to the Terms and Conditions");  Button signupButton = new Button("Sign Up");  // Apply style classes to buttons and fields  signupButton.getStyleClass().add("signup-button");  // Create "Already Have An Account?" link  Hyperlink loginLink = new Hyperlink("Already Have An Account?");  loginLink.setOnAction(event -> {  // Close the current SignupApplication window  primaryStage.hide();  // Launch LoginApplication in a new window  LoginApplication loginApp = new LoginApplication();  Stage loginStage = new Stage();  loginApp.start(loginStage);  });  // Create layout  GridPane gridPane = new GridPane();  gridPane.setAlignment(Pos.CENTER);  gridPane.setHgap(10);  gridPane.setVgap(10);  gridPane.setPadding(new Insets(25, 25, 25, 25));  // Adding UI components in the specified order  gridPane.add(fullNameLabel, 0, 0);  gridPane.add(fullNameField, 1, 0);  gridPane.add(emailLabel, 0, 1);  gridPane.add(emailField, 1, 1);  gridPane.add(passwordLabel, 0, 2);  gridPane.add(passwordField, 1, 2);  gridPane.add(confirmPasswordLabel, 0, 3);  gridPane.add(confirmPasswordField, 1, 3);  gridPane.add(dobLabel, 0, 4);  gridPane.add(dobPicker, 1, 4);  gridPane.add(genderLabel, 0, 5);  gridPane.add(genderComboBox, 1, 5);  gridPane.add(phoneNumberLabel, 0, 7);  gridPane.add(phoneNumberField, 1, 7);  gridPane.add(citizenshipLabel, 0, 8);  gridPane.add(citizenshipComboBox, 1, 8);  gridPane.add(termsCheckBox, 1, 9);  HBox hBox = new HBox(10);  hBox.setAlignment(Pos.BOTTOM\_RIGHT);  hBox.getChildren().add(signupButton);  gridPane.add(hBox, 1, 10);  // Load the logo image  ImageView logoImageView = new ImageView();  try {  Image logoImage = new Image(getClass().getResource("/college\_logo.png").toExternalForm());  logoImageView.setImage(logoImage);  logoImageView.setFitHeight(100);  logoImageView.setPreserveRatio(true);  } catch (NullPointerException e) {  System.out.println("Logo image not found");  }  // Create the root layout  VBox root = new VBox(10);  root.setAlignment(Pos.TOP\_CENTER);  root.getChildren().addAll(logoImageView, gridPane, loginLink); // Add loginLink to root  root.setPadding(new Insets(10, 10, 10, 10));  Scene scene = new Scene(root, 450, 750); // Increased height to accommodate all fields  try {  scene.getStylesheets().add(getClass().getResource("/Style.css").toExternalForm());  } catch (NullPointerException e) {  System.out.println("CSS file not found");  }  // Validation logic for signup button click  signupButton.setOnAction(event -> {  boolean isValid = true;  StringBuilder errorMessage = new StringBuilder();  // Validate email  String email = emailField.getText();  if (email.isEmpty() || !isValidEmail(email)) {  isValid = false;  errorMessage.append("Valid email is required.\n");  }  // Validate password  String password = passwordField.getText();  if (password.isEmpty()) {  isValid = false;  errorMessage.append("Password is required.\n");  }  // Validate confirm password  String confirmPassword = confirmPasswordField.getText();  if (!password.equals(confirmPassword)) {  isValid = false;  errorMessage.append("Passwords do not match.\n");  }  // Validate full name  if (fullNameField.getText().isEmpty()) {  isValid = false;  errorMessage.append("Full name is required.\n");  }  // Validate date of birth  if (dobPicker.getValue() == null) {  isValid = false;  errorMessage.append("Date of Birth is required.\n");  }  // Validate phone number  if (phoneNumberField.getText().isEmpty()) {  isValid = false;  errorMessage.append("Phone number is required.\n");  }  // Validate citizenship  if (citizenshipComboBox.getSelectionModel().isEmpty()) {  isValid = false;  errorMessage.append("Citizenship is required.\n");  }  // Validate terms and conditions  if (!termsCheckBox.isSelected()) {  isValid = false;  errorMessage.append("You must agree to the Terms and Conditions.\n");  }  // Display error messages if any validation fails  if (isValid) {  // Save student data to the database  DBConnect dbConnect = new DBConnect();  try (Connection connection = dbConnect.getConnection()) {  StudentDAO studentDAO = new StudentDAO(connection);  Student student = new Student(  0, // ID will be auto-generated  fullNameField.getText(),  emailField.getText(),  passwordField.getText(),  phoneNumberField.getText(),  dobPicker.getValue() != null ? dobPicker.getValue().toString() : null,  genderComboBox.getValue(),  citizenshipComboBox.getValue(),  "Pending", // Assuming default status is "Pending"  null, // DocumentName is initially null  false // UploadedDocuments initially false  );  studentDAO.insertStudent(student);  Alert alert = new Alert(Alert.AlertType.INFORMATION, "Signup successful!");  alert.showAndWait();  // Close the current SignupApplication window  primaryStage.hide();  // Launch LoginApplication in a new window  LoginApplication loginApp = new LoginApplication();  Stage loginStage = new Stage();  loginApp.start(loginStage);  } catch (SQLException e) {  e.printStackTrace();  Alert alert = new Alert(Alert.AlertType.ERROR, "An error occurred during signup.");  alert.showAndWait();  }  } else {  Alert alert = new Alert(Alert.AlertType.ERROR, errorMessage.toString());  alert.showAndWait();  }  });  primaryStage.setTitle("Signup Page");  primaryStage.setScene(scene);  primaryStage.show();  }  // Helper method to clear text fields  private void clearFields(TextField fullNameField, TextField emailField,  PasswordField passwordField, PasswordField confirmPasswordField, DatePicker dobPicker,  ComboBox<String> genderComboBox, TextField phoneNumberField) {  fullNameField.clear();  emailField.clear();  passwordField.clear();  confirmPasswordField.clear();  dobPicker.setValue(null);  genderComboBox.getSelectionModel().clearSelection();  phoneNumberField.clear();  }  // Helper method to validate email format  private boolean isValidEmail(String email) {  // Simple email validation, can be improved based on requirements  return email.matches("[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}");  }  // Helper method to show alert dialog  private void showAlert(Alert.AlertType alertType, String title, String headerText, String contentText) {  Alert alert = new Alert(alertType);  alert.setTitle(title);  alert.setHeaderText(headerText);  alert.setContentText(contentText);  alert.showAndWait();  }  // // Method to authenticate user  // public boolean authenticateUser(String username, String password) throws SQLException {  // String query = "SELECT COUNT(\*) FROM students WHERE email = ? AND password = ?";  // try (PreparedStatement stmt = connection.prepareStatement(query)) {  // stmt.setString(1, username);  // stmt.setString(2, password);  // ResultSet rs = stmt.executeQuery();  // if (rs.next()) {  // return rs.getInt(1) > 0; // Returns true if there is at least one matching record  // }  // }  // return false;  // } |
| Output & Validation : |
|  |

1. **Student Dashboard:**

|  |
| --- |
| SudentDashboard.java |
| package com.example.studentadmissionsystem.View;  import com.example.studentadmissionsystem.Controller.DBConnect;  import com.example.studentadmissionsystem.Controller.StudentDAO;  import com.example.studentadmissionsystem.Model.Student;  import javafx.application.Application;  import javafx.geometry.Insets;  import javafx.geometry.Pos;  import javafx.scene.Scene;  import javafx.scene.control.\*;  import javafx.scene.image.Image;  import javafx.scene.image.ImageView;  import javafx.scene.layout.BorderPane;  import javafx.scene.layout.GridPane;  import javafx.scene.layout.HBox;  import javafx.scene.layout.VBox;  import javafx.stage.FileChooser;  import javafx.stage.Stage;  import java.io.File;  import java.sql.Connection;  import java.sql.SQLException;  import java.util.regex.Pattern;  public class StudentDashboard extends Application {  private Label statusLabel;  private TextField nameField, emailField, phoneNumberField;  public ComboBox<String> departmentComboBox;  private ListView<String> notificationsList;  private StudentDAO studentDAO;  private Button uploadDocButton, resultButton;  private String username;  private String password;  private String statusLabelPrefix = "Application Status: ";  private String status = "Pending";  public StudentDashboard(String username, String password) {  this.username = username;  this.password = password;  }  public static void main(String[] args) {  launch(args);  }  @Override  public void start(Stage primaryStage) {  primaryStage.setTitle("Student Dashboard");  // Initialize DB and DAO  DBConnect dbConnect = new DBConnect();  Connection connection = dbConnect.getConnection();  studentDAO = new StudentDAO(connection);  // Profile information fields  nameField = new TextField();  emailField = new TextField();  phoneNumberField = new TextField();  // Department ComboBox  departmentComboBox = new ComboBox<>();  departmentComboBox.getItems().addAll("Admissions", "Finance", "IT", "Marketing", "Human Resources");  departmentComboBox.setPromptText("Select Department");  // Notifications ListView (initialize here)  Label notificationsLabel = new Label("Notifications:");  notificationsList = new ListView<>();  statusLabel = new Label(statusLabelPrefix + status);  // Fetch and fill data  populateProfileData();  // Profile information labels  Label nameLabel = new Label("Name:");  Label emailLabel = new Label("Email:");  Label phoneNumberLabel = new Label("Phone Number:");  Label departmentLabel = new Label("Department:");  // Profile information layout  GridPane profileGrid = new GridPane();  profileGrid.setPadding(new Insets(10));  profileGrid.setHgap(10);  profileGrid.setVgap(10);  profileGrid.setAlignment(Pos.CENTER);  profileGrid.add(nameLabel, 0, 0);  profileGrid.add(nameField, 1, 0);  profileGrid.add(emailLabel, 0, 1);  profileGrid.add(emailField, 1, 1);  profileGrid.add(phoneNumberLabel, 0, 2);  profileGrid.add(phoneNumberField, 1, 2);  profileGrid.add(departmentLabel, 0, 3);  profileGrid.add(departmentComboBox, 1, 3);  Button updateProfileButton = new Button("Update Profile");  updateProfileButton.getStyleClass().add("dashboard-button");  updateProfileButton.setOnAction(e -> updateProfile());  uploadDocButton = new Button("Upload Documents");  uploadDocButton.getStyleClass().add("dashboard-button");  uploadDocButton.setOnAction(e -> uploadDocuments(primaryStage));  // Result Button  resultButton = new Button("Show Result");  resultButton.getStyleClass().add("dashboard-button");  resultButton.setOnAction(e -> showResult());  // Main layout  VBox mainLayout = new VBox(20);  mainLayout.setPadding(new Insets(10));  mainLayout.setAlignment(Pos.CENTER);  mainLayout.getChildren().addAll(profileGrid, updateProfileButton, statusLabel, uploadDocButton, resultButton, notificationsLabel, notificationsList);  // College logo  Image logo = new Image(getClass().getResourceAsStream("/college\_logo.png"));  ImageView logoView = new ImageView(logo);  logoView.setFitHeight(100);  logoView.setPreserveRatio(true);  Label welcomeLabel = new Label("Student Dashboard");  welcomeLabel.getStyleClass().add("welcome-label");  Button logoutButton = new Button("Logout");  logoutButton.getStyleClass().add("logout-button");  logoutButton.setOnAction(event -> {  primaryStage.hide();  LoginApplication loginApp = new LoginApplication();  Stage loginStage = new Stage();  loginApp.start(loginStage);  });  HBox header = new HBox(10, logoView, welcomeLabel, logoutButton);  header.setAlignment(Pos.CENTER);  header.setPadding(new Insets(10));  header.getStyleClass().add("header");  BorderPane root = new BorderPane();  root.setTop(header);  root.setCenter(mainLayout);  root.getStyleClass().add("root");  Scene scene = new Scene(root, 1000, 600);  scene.getStylesheets().add(getClass().getResource("/Style.css").toExternalForm());  primaryStage.setScene(scene);  primaryStage.show();  }  private void populateProfileData() {  try {  Student student = studentDAO.getStudentByUsernameAndPassword(username, password);  if (student != null) {  nameField.setText(student.getName());  emailField.setText(student.getEmail());  phoneNumberField.setText(student.getPhone());  if (student.getDepartment() != null && !student.getDepartment().isEmpty()) {  departmentComboBox.setValue(student.getDepartment());  }  status = student.getStatus();  statusLabel.setText(statusLabelPrefix + status);  // Fetch and display the document in the notifications  String documentPath = studentDAO.getDocumentPathByEmail(student.getEmail());  if (documentPath != null && !documentPath.isEmpty()) {  addNotification("Document uploaded: " + new File(documentPath).getName());  } else {  addNotification("No documents uploaded.");  }  } else {  showAlert("No student found with the provided credentials.");  }  } catch (SQLException e) {  showAlert("An error occurred while fetching the student data.");  e.printStackTrace();  }  }  private void updateProfile() {  String name = nameField.getText();  String email = emailField.getText();  String phoneNumber = phoneNumberField.getText();  String department = departmentComboBox.getValue();  if (name.isEmpty() || email.isEmpty() || phoneNumber.isEmpty()) {  showAlert("Please fill in all required fields.");  return;  }  if (!isValidEmail(email)) {  showAlert("Please enter a valid email address.");  return;  }  if (!isValidPhoneNumber(phoneNumber)) {  showAlert("Phone number must be 10 digits long.");  return;  }  // If department is not selected, fetch existing department from the database  if (department == null) {  try {  Student student = studentDAO.getStudentByUsernameAndPassword(username, password);  if (student != null) {  department = student.getDepartment(); // Get existing department  } else {  showAlert("Failed to retrieve existing department.");  return;  }  } catch (SQLException e) {  showAlert("An error occurred while fetching the department data.");  e.printStackTrace();  return;  }  }  try {  boolean isUpdated = studentDAO.updateStudentProfile(name, email, phoneNumber, department);  if (isUpdated) {  showConfirmation("Profile updated successfully!");  } else {  showAlert("Failed to update profile.");  }  } catch (SQLException e) {  showAlert("An error occurred while updating the profile.");  e.printStackTrace();  }  }  private void showResult() {  String resultMessage = getResultMessage(status);  Alert resultAlert = new Alert(Alert.AlertType.INFORMATION);  resultAlert.setTitle("Application Result");  resultAlert.setHeaderText(null);  resultAlert.setContentText(resultMessage);  resultAlert.showAndWait();  }  private String getResultMessage(String status) {  switch (status) {  case "Pending":  return "The application has been submitted but has not yet been reviewed by the admissions committee.\n\n" +  "Required documents are missing or incomplete, awaiting submission by the applicant.\n\n" +  "The application fee has not been paid or processed.";  case "Under Process":  return "The application is currently being reviewed by the admissions committee.\n\n" +  "The applicant's academic records are being verified.\n\n" +  "The interview process or additional assessments are underway.\n\n" +  "Background checks or references are being contacted for further evaluation.";  case "Accepted":  return "The applicant meets all the admission criteria and has been offered a place in the program.\n\n" +  "The academic records and supporting documents have been verified and approved.\n\n" +  "The applicant has successfully passed the interview and any required assessments.";  case "Rejected":  return "The applicant does not meet the minimum admission requirements.\n\n" +  "Academic records or supporting documents did not meet the institution's standards.\n\n" +  "The applicant was unsuccessful in the interview or other assessments.\n\n" +  "The program has reached its capacity, and the applicant was not selected for admission.";  default:  return "Application status unknown.";  }  }  private void addNotification(String notification) {  notificationsList.getItems().add(notification);  }  private void uploadDocuments(Stage stage) {  if (nameField.getText().isEmpty() || emailField.getText().isEmpty() || phoneNumberField.getText().isEmpty() || departmentComboBox.getValue() == null) {  showAlert("Please fill in all profile fields before uploading a document.");  return;  }  if (!isValidEmail(emailField.getText())) {  showAlert("Please enter a valid email address.");  return;  }  if (!isValidPhoneNumber(phoneNumberField.getText())) {  showAlert("Phone number must be 10 digits long.");  return;  }  FileChooser fileChooser = new FileChooser();  fileChooser.setTitle("Upload Document");  File file = fileChooser.showOpenDialog(stage);  if (file != null) {  try {  // Create the folder structure based on the student's email  String email = emailField.getText();  File folder = new File("StudentDocuments" + File.separator + email); // Main folder named "StudentDocuments" followed by student's email  if (!folder.exists()) {  folder.mkdirs(); // Create the folder if it doesn't exist  }  // Save the file in the created folder  File destFile = new File(folder, file.getName());  file.renameTo(destFile);  boolean isSaved = studentDAO.saveDocument(destFile.getAbsolutePath(), emailField.getText());  if (isSaved) {  addNotification("Document uploaded: " + file.getName());  statusLabel.setText("Application Status: Under Review");  showConfirmation("Thank you for applying! Your application is now under review.");  } else {  showAlert("Failed to upload document.");  }  } catch (Exception e) {  showAlert("An error occurred while uploading the document.");  e.printStackTrace();  }  }  }  private void showAlert(String message) {  Alert alert = new Alert(Alert.AlertType.WARNING);  alert.setTitle("Warning");  alert.setHeaderText(null);  alert.setContentText(message);  alert.showAndWait();  }  private void showConfirmation(String message) {  Alert alert = new Alert(Alert.AlertType.INFORMATION);  alert.setTitle("Success");  alert.setHeaderText(null);  alert.setContentText(message);  alert.showAndWait();  }  private boolean isValidEmail(String email) {  String emailPattern = "^[a-zA-Z0-9.\_%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,6}$";  return Pattern.matches(emailPattern, email);  }  private boolean isValidPhoneNumber(String phoneNumber) {  String phonePattern = "\\d{10}";  return Pattern.matches(phonePattern, phoneNumber);  }  } |
| Output : |
|  |

1. **Admin Dashboard:**

|  |
| --- |
| AdminDashboard.java |
| package com.example.studentadmissionsystem.View;  import com.example.studentadmissionsystem.Controller.CSVGenerator;  import com.example.studentadmissionsystem.Controller.DBConnect;  import com.example.studentadmissionsystem.Model.Registrar;  import javafx.application.Application;  import javafx.beans.property.SimpleObjectProperty;  import javafx.collections.FXCollections;  import javafx.collections.ObservableList;  import javafx.geometry.Insets;  import javafx.geometry.Pos;  import javafx.scene.Scene;  import javafx.scene.control.\*;  import javafx.scene.image.Image;  import javafx.scene.image.ImageView;  import javafx.scene.layout.BorderPane;  import javafx.scene.layout.GridPane;  import javafx.scene.layout.HBox;  import javafx.scene.layout.VBox;  import javafx.stage.Stage;  import java.io.BufferedReader;  import java.io.FileReader;  import java.io.IOException;  import java.sql.Connection;  import java.sql.PreparedStatement;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.util.ArrayList;  import java.util.List;  import com.example.studentadmissionsystem.Controller.CSVGenerator; // Update this path based on your package structure  public class AdminDashboard extends Application {  private TableView<Registrar> registrarTable;  private List<Registrar> registrars;  public static void main(String[] args) {  launch(args);  }  @Override  public void start(Stage primaryStage) {  primaryStage.setTitle("Administrator Dashboard");  // Initialize the list of registrars from the database  registrars = loadRegistrarsFromDatabase();  // Create the table view for registrars  registrarTable = new TableView<>();  registrarTable.setPrefWidth(800);  registrarTable.setItems(FXCollections.observableArrayList(registrars));  // Define columns  TableColumn<Registrar, Integer> idColumn = new TableColumn<>("ID");  idColumn.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue().getId()));  TableColumn<Registrar, String> nameColumn = new TableColumn<>("Name");  nameColumn.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue().getName()));  TableColumn<Registrar, String> emailColumn = new TableColumn<>("Email");  emailColumn.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue().getEmail()));  TableColumn<Registrar, String> phoneNumberColumn = new TableColumn<>("Phone Number");  phoneNumberColumn.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue().getPhoneNumber()));  TableColumn<Registrar, String> departmentColumn = new TableColumn<>("Department");  departmentColumn.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue().getDepartment()));  TableColumn<Registrar, String> addressColumn = new TableColumn<>("Address");  addressColumn.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue().getAddress()));  TableColumn<Registrar, String> passwordColumn = new TableColumn<>("Password");  passwordColumn.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue().getPassword()));  // Add columns to the table  registrarTable.getColumns().addAll(idColumn, nameColumn, emailColumn, phoneNumberColumn, departmentColumn, addressColumn, passwordColumn);  // Create buttons  Button addButton = new Button("Add Registrar");  addButton.getStyleClass().add("dashboard-button");  Button updateButton = new Button("Update Registrar");  updateButton.getStyleClass().add("dashboard-button");  Button deleteButton = new Button("Delete Registrar");  deleteButton.getStyleClass().add("dashboard-button");  Button generateReportButton = new Button("Generate Report");  generateReportButton.getStyleClass().add("dashboard-button");  Button logoutButton = new Button("Logout");  logoutButton.getStyleClass().add("logout-button");  // Set button actions  addButton.setOnAction(e -> addRegistrar());  updateButton.setOnAction(e -> updateRegistrar());  deleteButton.setOnAction(e -> deleteRegistrar());  generateReportButton.setOnAction(e -> generateReport());  logoutButton.setOnAction(event -> {  System.out.println("Redirecting to Login page...");  primaryStage.hide();  LoginApplication loginApp = new LoginApplication();  Stage loginStage = new Stage();  loginApp.start(loginStage);  });  // Layout setup  VBox buttonBox = new VBox(10, addButton, updateButton, deleteButton, generateReportButton, logoutButton);  buttonBox.setAlignment(Pos.CENTER);  buttonBox.setPadding(new Insets(20));  HBox mainContent = new HBox(20, buttonBox, registrarTable);  mainContent.setPadding(new Insets(20));  mainContent.setAlignment(Pos.CENTER);  // College logo  Image logo = new Image(getClass().getResourceAsStream("/college\_logo.png"));  ImageView logoView = new ImageView(logo);  logoView.setFitHeight(100);  logoView.setPreserveRatio(true);  Label welcomeLabel = new Label("Administrator Dashboard");  welcomeLabel.getStyleClass().add("welcome-label");  HBox header = new HBox(10, logoView, welcomeLabel);  header.setAlignment(Pos.CENTER);  header.setPadding(new Insets(10));  header.getStyleClass().add("header");  BorderPane root = new BorderPane();  root.setTop(header);  root.setCenter(mainContent);  root.getStyleClass().add("root");  Scene scene = new Scene(root, 1000, 600);  scene.getStylesheets().add(getClass().getResource("/Style.css").toExternalForm());  primaryStage.setScene(scene);  primaryStage.show();  }  private List<Registrar> loadRegistrarsFromDatabase() {  List<Registrar> registrarList = new ArrayList<>();  String query = "SELECT \* FROM registrar";  try (Connection connection = DBConnect.getConnection();  PreparedStatement pstmt = connection.prepareStatement(query);  ResultSet rs = pstmt.executeQuery()) {  while (rs.next()) {  int id = rs.getInt("id");  String name = rs.getString("name");  String email = rs.getString("email");  String phoneNumber = rs.getString("phone");  String department = rs.getString("department");  String address = rs.getString("address");  String password = rs.getString("password");  Registrar registrar = new Registrar(id, name, email, phoneNumber, department, address, password);  registrarList.add(registrar);  }  } catch (SQLException e) {  e.printStackTrace();  }  return registrarList;  }  private void addRegistrar() {  Stage stage = new Stage();  stage.setTitle("Add Registrar");  GridPane gridPane = new GridPane();  gridPane.setPadding(new Insets(10));  gridPane.setHgap(10);  gridPane.setVgap(10);  gridPane.setAlignment(Pos.CENTER);  gridPane.getStyleClass().add("grid-pane");  TextField idField = new TextField();  idField.setPromptText("ID");  idField.getStyleClass().add("text-field");  TextField nameField = new TextField();  nameField.setPromptText("Name");  nameField.getStyleClass().add("text-field");  TextField emailField = new TextField();  emailField.setPromptText("Email");  emailField.getStyleClass().add("text-field");  TextField phoneNumberField = new TextField();  phoneNumberField.setPromptText("Phone Number");  phoneNumberField.getStyleClass().add("text-field");  ComboBox<String> departmentComboBox = new ComboBox<>();  ObservableList<String> departments = FXCollections.observableArrayList(  "Admissions", "Finance", "IT", "Marketing", "Human Resources"  );  departmentComboBox.setItems(departments);  departmentComboBox.setPromptText("Select Department");  departmentComboBox.getStyleClass().add("combo-box");  TextField addressField = new TextField();  addressField.setPromptText("Address");  addressField.getStyleClass().add("text-field");  TextField passwordField = new TextField("Registrar@101");  passwordField.setEditable(false);  passwordField.setVisible(false);  Button submitButton = new Button("Submit");  submitButton.getStyleClass().add("dashboard-button");  submitButton.setOnAction(e -> {  String id = idField.getText();  String name = nameField.getText();  String email = emailField.getText();  String phoneNumber = phoneNumberField.getText();  String department = departmentComboBox.getValue();  String address = addressField.getText();  if (!id.isEmpty() && !name.isEmpty() && !email.isEmpty() && !phoneNumber.isEmpty() && department != null && !address.isEmpty()) {  if (isDepartmentAvailable(department)) {  // Add registrar to the list and database  addRegistrarToDatabase(Integer.parseInt(id), name, email, phoneNumber, department, address, "Registrar@101");  registrars.add(new Registrar(Integer.parseInt(id), name, email, phoneNumber, department, address, "Registrar@101"));  registrarTable.setItems(FXCollections.observableArrayList(registrars));  stage.close();  } else {  showAlert("Department already has a registrar.");  }  } else {  showAlert("Please fill in all fields.");  }  });  gridPane.add(new Label("ID:"), 0, 0);  gridPane.add(idField, 1, 0);  gridPane.add(new Label("Name:"), 0, 1);  gridPane.add(nameField, 1, 1);  gridPane.add(new Label("Email:"), 0, 2);  gridPane.add(emailField, 1, 2);  gridPane.add(new Label("Phone Number:"), 0, 3);  gridPane.add(phoneNumberField, 1, 3);  gridPane.add(new Label("Department:"), 0, 4);  gridPane.add(departmentComboBox, 1, 4);  gridPane.add(new Label("Address:"), 0, 5);  gridPane.add(addressField, 1, 5);  gridPane.add(submitButton, 1, 6);  Scene scene = new Scene(gridPane, 400, 300);  scene.getStylesheets().add(getClass().getResource("/Style.css").toExternalForm());  stage.setScene(scene);  stage.show();  }  private boolean isDepartmentAvailable(String department) {  for (Registrar registrar : registrars) {  if (registrar.getDepartment().equals(department)) {  return false;  }  }  return true;  }  private void addRegistrarToDatabase(int id, String name, String email, String phoneNumber, String department, String address, String password) {  String query = "INSERT INTO registrar (id, name, email, phone, department, address, password) VALUES (?, ?, ?, ?, ?, ?, ?)";  try (Connection connection = DBConnect.getConnection();  PreparedStatement pstmt = connection.prepareStatement(query)) {  pstmt.setInt(1, id);  pstmt.setString(2, name);  pstmt.setString(3, email);  pstmt.setString(4, phoneNumber);  pstmt.setString(5, department);  pstmt.setString(6, address);  pstmt.setString(7, password);  int rowsAffected = pstmt.executeUpdate();  System.out.println("Rows affected: " + rowsAffected); // Debugging output  if (rowsAffected > 0) {  System.out.println("Registrar added successfully!");  } else {  System.out.println("Failed to add registrar.");  }  } catch (SQLException e) {  e.printStackTrace();  System.out.println("SQL Exception: " + e.getMessage());  }  }  private void updateRegistrar() {  Registrar selectedRegistrar = registrarTable.getSelectionModel().getSelectedItem();  if (selectedRegistrar != null) {  Stage stage = new Stage();  stage.setTitle("Update Registrar");  GridPane gridPane = new GridPane();  gridPane.setPadding(new Insets(10));  gridPane.setHgap(10);  gridPane.setVgap(10);  gridPane.setAlignment(Pos.CENTER);  gridPane.getStyleClass().add("grid-pane");  TextField idField = new TextField(String.valueOf(selectedRegistrar.getId()));  idField.setEditable(false);  idField.getStyleClass().add("text-field");  TextField nameField = new TextField(selectedRegistrar.getName());  nameField.getStyleClass().add("text-field");  TextField emailField = new TextField(selectedRegistrar.getEmail());  emailField.getStyleClass().add("text-field");  TextField phoneNumberField = new TextField(selectedRegistrar.getPhoneNumber());  phoneNumberField.getStyleClass().add("text-field");  ComboBox<String> departmentComboBox = new ComboBox<>();  ObservableList<String> departments = FXCollections.observableArrayList(  "Admissions", "Finance", "IT", "Marketing", "Human Resources"  );  departmentComboBox.setItems(departments);  departmentComboBox.setValue(selectedRegistrar.getDepartment());  departmentComboBox.getStyleClass().add("combo-box");  TextField addressField = new TextField(selectedRegistrar.getAddress());  addressField.getStyleClass().add("text-field");  TextField passwordField = new TextField(selectedRegistrar.getPassword());  passwordField.setEditable(false);  passwordField.setVisible(false);  Button submitButton = new Button("Submit");  submitButton.getStyleClass().add("dashboard-button");  submitButton.setOnAction(e -> {  String name = nameField.getText();  String email = emailField.getText();  String phoneNumber = phoneNumberField.getText();  String department = departmentComboBox.getValue();  String address = addressField.getText();  if (!name.isEmpty() && !email.isEmpty() && !phoneNumber.isEmpty() && department != null && !address.isEmpty()) {  if (isDepartmentAvailableOrSame(department, selectedRegistrar.getId())) {  // Update the selected registrar in the list and database  updateRegistrarInDatabase(selectedRegistrar.getId(), name, email, phoneNumber, department, address, passwordField.getText());  selectedRegistrar.setName(name);  selectedRegistrar.setEmail(email);  selectedRegistrar.setPhoneNumber(phoneNumber);  selectedRegistrar.setDepartment(department);  selectedRegistrar.setAddress(address);  registrarTable.refresh();  stage.close();  } else {  showAlert("Department already has a registrar.");  }  } else {  showAlert("Please fill in all fields.");  }  });  gridPane.add(new Label("ID:"), 0, 0);  gridPane.add(idField, 1, 0);  gridPane.add(new Label("Name:"), 0, 1);  gridPane.add(nameField, 1, 1);  gridPane.add(new Label("Email:"), 0, 2);  gridPane.add(emailField, 1, 2);  gridPane.add(new Label("Phone Number:"), 0, 3);  gridPane.add(phoneNumberField, 1, 3);  gridPane.add(new Label("Department:"), 0, 4);  gridPane.add(departmentComboBox, 1, 4);  gridPane.add(new Label("Address:"), 0, 5);  gridPane.add(addressField, 1, 5);  gridPane.add(submitButton, 1, 6);  Scene scene = new Scene(gridPane, 400, 300);  scene.getStylesheets().add(getClass().getResource("/Style.css").toExternalForm());  stage.setScene(scene);  stage.show();  } else {  showAlert("Please select a registrar to update.");  }  }  private boolean isDepartmentAvailableOrSame(String department, int currentRegistrarId) {  for (Registrar registrar : registrars) {  if (registrar.getDepartment().equals(department) && registrar.getId() != currentRegistrarId) {  return false;  }  }  return true;  }  private void updateRegistrarInDatabase(int id, String name, String email, String phoneNumber, String department, String address, String password) {  String query = "UPDATE registrar SET name = ?, email = ?, phone = ?, department = ?, address = ?, password = ? WHERE id = ?";  try (Connection connection = DBConnect.getConnection();  PreparedStatement pstmt = connection.prepareStatement(query)) {  pstmt.setString(1, name);  pstmt.setString(2, email);  pstmt.setString(3, phoneNumber);  pstmt.setString(4, department);  pstmt.setString(5, address);  pstmt.setString(6, password);  pstmt.setInt(7, id);  pstmt.executeUpdate();  } catch (SQLException e) {  e.printStackTrace();  }  }  private void deleteRegistrar() {  Registrar selectedRegistrar = registrarTable.getSelectionModel().getSelectedItem();  if (selectedRegistrar != null) {  registrars.remove(selectedRegistrar);  deleteRegistrarFromDatabase(selectedRegistrar.getId());  registrarTable.setItems(FXCollections.observableArrayList(registrars));  } else {  showAlert("Please select a registrar to delete.");  }  }  private void deleteRegistrarFromDatabase(int id) {  String query = "DELETE FROM registrar WHERE id = ?";  try (Connection connection = DBConnect.getConnection();  PreparedStatement pstmt = connection.prepareStatement(query)) {  pstmt.setInt(1, id);  pstmt.executeUpdate();  } catch (SQLException e) {  e.printStackTrace();  }  }  private void generateReport() {  String filePath = "registrars\_report.csv";  // Generate CSV file  CSVGenerator.generateCSV(registrars, filePath);  // Show CSV content  showCSVReport(filePath);  }  private void showCSVReport(String filePath) {  Stage stage = new Stage();  stage.setTitle("CSV Report");  TextArea textArea = new TextArea();  textArea.setEditable(false);  textArea.setWrapText(true);  try (BufferedReader reader = new BufferedReader(new FileReader(filePath))) {  StringBuilder content = new StringBuilder();  String line;  while ((line = reader.readLine()) != null) {  content.append(line).append("\n");  }  textArea.setText(content.toString());  } catch (IOException e) {  e.printStackTrace();  textArea.setText("Error reading CSV file.");  }  Scene scene = new Scene(textArea, 600, 400);  stage.setScene(scene);  stage.show();  }  private void showAlert(String message) {  Alert alert = new Alert(Alert.AlertType.INFORMATION);  alert.setTitle("Information");  alert.setHeaderText(null);  alert.setContentText(message);  alert.showAndWait();  }  } |
| Output: |
|  |

1. **Registrar Dashboard:**

|  |
| --- |
| RegistrarDashboard.java |
| package com.example.studentadmissionsystem.View;  import com.example.studentadmissionsystem.Controller.DBConnect;  import com.example.studentadmissionsystem.Controller.StudentDAO;  import com.example.studentadmissionsystem.Model.Student;  import javafx.application.Application;  import javafx.beans.property.SimpleObjectProperty;  import javafx.collections.FXCollections;  import javafx.collections.ObservableList;  import javafx.geometry.Insets;  import javafx.geometry.Pos;  import javafx.scene.Scene;  import javafx.scene.control.\*;  import javafx.scene.image.Image;  import javafx.scene.image.ImageView;  import javafx.scene.layout.BorderPane;  import javafx.scene.layout.GridPane;  import javafx.scene.layout.HBox;  import javafx.scene.layout.VBox;  import javafx.stage.FileChooser;  import javafx.stage.Stage;  import java.io.IOException;  import java.io.InputStream;  import java.io.\*;  import java.nio.file.\*;  import javafx.stage.\*;  import java.io.File;  import java.io.FileWriter;  import java.io.IOException;  import java.sql.Connection;  import java.sql.SQLException;  import java.util.List;  public class RegistrarDashboard extends Application {  private TableView<Student> studentTable;  private List<Student> students;  private Student selectedStudent;  private String username;  private String password;  private File destFile;  private StudentDAO studentDAO;  public RegistrarDashboard(String username, String password) {  this.username = username;  this.password = password;  }  public static void main(String[] args) {  launch(args);  }  @Override  public void start(Stage primaryStage) throws SQLException {  primaryStage.setTitle("Registrar Dashboard");  // Initialize DB and DAO  DBConnect dbConnect = new DBConnect();  Connection connection = dbConnect.getConnection();  studentDAO = new StudentDAO(connection);  // Initialize the list of students  students = FXCollections.observableArrayList();  //students.add(new Student(1, "John Doe", "john.doe@example.com", "123-456-7890", "Computer Science", "Accepted", "123 Main St", "Springfield", "IL", "62701", "USA", true));  //students.add(new Student(2, "Jane Smith", "jane.smith@example.com", "987-654-3210", "Electrical Engineering", "Pending", "456 Elm St", "Springfield", "IL", "62701", "USA", false));  // Create the table view for students  studentTable = new TableView<>();  studentTable.setPrefWidth(900);  studentTable.setItems(FXCollections.observableArrayList(students));  // Define columns  TableColumn<Student, Object> idColumn = new TableColumn<>("ID");  idColumn.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue().getId()));  TableColumn<Student, Object> nameColumn = new TableColumn<>("Name");  nameColumn.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue().getName()));  TableColumn<Student, Object> emailColumn = new TableColumn<>("Email");  emailColumn.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue().getEmail()));  TableColumn<Student, Object> phoneNumberColumn = new TableColumn<>("Phone Number");  phoneNumberColumn.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue().getPhone()));  TableColumn<Student, Object> departmentColumn = new TableColumn<>("Department");  departmentColumn.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue().getDepartment()));  TableColumn<Student, Object> statusColumn = new TableColumn<>("Status");  statusColumn.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue().getStatus()));  TableColumn<Student, Object> documentNameColumn = new TableColumn<>("Document Name");  documentNameColumn.setCellValueFactory(data -> new SimpleObjectProperty<>(data.getValue().getDocumentName()));  TableColumn<Student, Void> downloadColumn = new TableColumn<>("Download");  downloadColumn.setCellFactory(param -> new TableCell<>() {  private final Button downloadButton = new Button("Download");  {  downloadButton.setOnAction(event -> {  Student student = getTableView().getItems().get(getIndex());  String documentName = student.getDocumentName();  // Implement download logic  downloadFile(documentName);  });  }  private void downloadFile(String documentName) {  try {  // Use a File object instead of URL  File file = new File(documentName);  if (file.exists()) {  // Open file chooser for saving the file  FileChooser fileChooser = new FileChooser();  fileChooser.setTitle("Save File");  fileChooser.setInitialFileName(file.getName());  File destinationFile = fileChooser.showSaveDialog(new Stage());  if (destinationFile != null) {  // Copy the file to the chosen path  try (InputStream inputStream = new FileInputStream(file);  OutputStream outputStream = new FileOutputStream(destinationFile)) {  byte[] buffer = new byte[1024];  int bytesRead;  while ((bytesRead = inputStream.read(buffer)) != -1) {  outputStream.write(buffer, 0, bytesRead);  }  System.out.println("File downloaded successfully: " + destinationFile.getAbsolutePath());  }  }  } else {  System.err.println("File not found: " + documentName);  }  } catch (IOException e) {  e.printStackTrace();  System.err.println("Error downloading file: " + e.getMessage());  }  }  @Override  protected void updateItem(Void item, boolean empty) {  super.updateItem(item, empty);  setGraphic(empty ? null : downloadButton);  }  });  studentTable.getColumns().addAll(idColumn, nameColumn, emailColumn, phoneNumberColumn, departmentColumn, statusColumn, documentNameColumn, downloadColumn);  // Fetch students and set data  // Create an instance of StudentDAO  ObservableList<Student> students = FXCollections.observableArrayList(studentDAO.fetchStudentsFromDatabase());  studentTable.setItems(students);  // Create buttons  Button addButton = new Button("Add Student");  addButton.getStyleClass().add("dashboard-button");  Button updateButton = new Button("Update Student");  updateButton.getStyleClass().add("dashboard-button");  Button deleteButton = new Button("Delete Student");  deleteButton.getStyleClass().add("dashboard-button");  Button generateReportButton = new Button("Generate Report");  generateReportButton.getStyleClass().add("dashboard-button");  Button logoutButton = new Button("Logout");  logoutButton.getStyleClass().add("logout-button");  // Button changePasswordButton = new Button("Change Password");  // changePasswordButton.getStyleClass().add("dashboard-button");  // Set button actions  addButton.setOnAction(e -> addStudent());  updateButton.setOnAction(e -> updateStudent());  deleteButton.setOnAction(e -> deleteStudent());  generateReportButton.setOnAction(e -> generateReport());  logoutButton.setOnAction(event -> {  System.out.println("Redirecting to Login page...");  primaryStage.hide();  LoginApplication loginApp = new LoginApplication();  Stage loginStage = new Stage();  loginApp.start(loginStage);  });  // changePasswordButton.setOnAction(e -> changePassword());  // Layout setup  VBox buttonBox = new VBox(10, addButton, updateButton, deleteButton, generateReportButton, logoutButton);  buttonBox.setAlignment(Pos.CENTER);  buttonBox.setPadding(new Insets(20));  HBox mainContent = new HBox(20, buttonBox, studentTable);  mainContent.setPadding(new Insets(20));  mainContent.setAlignment(Pos.CENTER);  // College logo  Image logo = new Image(getClass().getResourceAsStream("/college\_logo.png"));  ImageView logoView = new ImageView(logo);  logoView.setFitHeight(100);  logoView.setPreserveRatio(true);  Label welcomeLabel = new Label("Registrar Dashboard");  welcomeLabel.getStyleClass().add("welcome-label");  HBox header = new HBox(10, logoView, welcomeLabel);  header.setAlignment(Pos.CENTER);  header.setPadding(new Insets(10));  header.getStyleClass().add("header");  BorderPane root = new BorderPane();  root.setTop(header);  root.setCenter(mainContent);  root.getStyleClass().add("root");  Scene scene = new Scene(root, 1000, 600);  scene.getStylesheets().add(getClass().getResource("/Style.css").toExternalForm());  primaryStage.setScene(scene);  primaryStage.show();  }  private File selectedFile = null; // Class-level variable to store the selected file  private void addStudent() {  Stage stage = new Stage();  stage.setTitle("Add Student");  GridPane gridPane = new GridPane();  gridPane.setPadding(new Insets(10));  gridPane.setHgap(10);  gridPane.setVgap(10);  gridPane.setAlignment(Pos.CENTER);  gridPane.getStyleClass().add("grid-pane");  TextField idField = new TextField();  idField.setPromptText("ID");  TextField nameField = new TextField();  nameField.setPromptText("Name");  TextField emailField = new TextField();  emailField.setPromptText("Email");  TextField phoneNumberField = new TextField();  phoneNumberField.setPromptText("Phone Number");  // Department ComboBox  ComboBox<String> departmentComboBox = new ComboBox<>();  ObservableList<String> departments = FXCollections.observableArrayList(  "Admissions", "Finance", "IT", "Marketing", "Human Resources"  );  departmentComboBox.setItems(departments);  departmentComboBox.setPromptText("Select Department");  departmentComboBox.getStyleClass().add("combo-box");  // Status ComboBox  ComboBox<String> statusComboBox = new ComboBox<>();  ObservableList<String> status1 = FXCollections.observableArrayList(  "Pending", "Under Process", "Accepted", "Rejected", "Conditionally Accepted"  );  statusComboBox.setItems(status1);  statusComboBox.setPromptText("Select Status");  statusComboBox.getStyleClass().add("combo-box");  CheckBox uploadedDocumentsCheckBox = new CheckBox("Documents Uploaded");  // Add document selection  TextField documentNameField = new TextField();  documentNameField.setPromptText("Document Name");  Button selectDocumentButton = new Button("Select Document");  selectDocumentButton.setOnAction(e -> {  FileChooser fileChooser = new FileChooser();  fileChooser.setTitle("Select Document");  selectedFile = fileChooser.showOpenDialog(stage); // Store the selected file  if (selectedFile != null) {  documentNameField.setText(selectedFile.getName());  }  });  Button submitButton = new Button("Submit");  submitButton.setOnAction(e -> {  String id = idField.getText();  String name = nameField.getText();  String email = emailField.getText();  String phoneNumber = phoneNumberField.getText();  String department = departmentComboBox.getValue();  String status = statusComboBox.getValue();  boolean uploadedDocuments = uploadedDocumentsCheckBox.isSelected();  String documentName = documentNameField.getText();  // Validation  if (!id.isEmpty() && !name.isEmpty() && !email.isEmpty() && !phoneNumber.isEmpty() && !department.isEmpty() && status != null && !documentName.isEmpty()) {  if (!email.matches("^[\\w-\_.]+@([\\w-]+\\.)+[\\w-]{2,4}$")) {  showAlert(Alert.AlertType.ERROR, "Invalid Email", "Please enter a valid email address.");  return;  }  if (!phoneNumber.matches("\\d{10}")) {  showAlert(Alert.AlertType.ERROR, "Invalid Phone Number", "Phone number must be exactly 10 digits.");  return;  }  try {  Student newStudent = new Student(  Integer.parseInt(id),  name,  email,  phoneNumber,  department,  status,  uploadedDocuments,  documentName  );  studentDAO.insertStudentByRegistrar(newStudent);  // Handle document upload after student data is inserted  if (selectedFile != null) {  // Create the folder structure based on the student's email  File folder = new File("StudentDocuments" + File.separator + email);  if (!folder.exists()) {  folder.mkdirs();  }  // Define the destination file  File destFile = new File(folder, selectedFile.getName());  // Copy the file to the destination folder  try {  Files.copy(selectedFile.toPath(), destFile.toPath(), StandardCopyOption.REPLACE\_EXISTING);  boolean isSaved = studentDAO.saveDocument(destFile.getAbsolutePath(), email);  if (isSaved) {  showConfirmation("Document uploaded successfully.");  } else {  showAlert(Alert.AlertType.ERROR, "Failed to upload document.", "Document upload failed.");  }  } catch (IOException ex) {  showAlert(Alert.AlertType.ERROR, "File Error", "An error occurred while copying the file.");  ex.printStackTrace();  }  }  students = FXCollections.observableArrayList(studentDAO.fetchStudentsFromDatabase());  studentTable.setItems(FXCollections.observableArrayList(students));  stage.close();  } catch (SQLException ex) {  showAlert(Alert.AlertType.ERROR, "Database Error", "Failed to add student to the database.");  ex.printStackTrace();  } catch (NumberFormatException ex) {  showAlert(Alert.AlertType.ERROR, "Invalid ID", "ID must be a number.");  ex.printStackTrace();  }  } else {  showAlert(Alert.AlertType.ERROR, "Validation Error", "Please fill in all required fields.");  }  });  gridPane.add(new Label("ID:"), 0, 0);  gridPane.add(idField, 1, 0);  gridPane.add(new Label("Name:"), 0, 1);  gridPane.add(nameField, 1, 1);  gridPane.add(new Label("Email:"), 0, 2);  gridPane.add(emailField, 1, 2);  gridPane.add(new Label("Phone Number:"), 0, 3);  gridPane.add(phoneNumberField, 1, 3);  gridPane.add(new Label("Department:"), 0, 4);  gridPane.add(departmentComboBox, 1, 4);  gridPane.add(new Label("Status:"), 0, 5);  gridPane.add(statusComboBox, 1, 5);  gridPane.add(uploadedDocumentsCheckBox, 0, 6, 2, 1);  gridPane.add(new Label("Document Name:"), 0, 7);  gridPane.add(documentNameField, 1, 7);  gridPane.add(selectDocumentButton, 1, 8);  gridPane.add(submitButton, 1, 9);  Scene scene = new Scene(gridPane, 400, 500);  stage.setScene(scene);  stage.show();  }  private void showConfirmation(String message) {  Alert alert = new Alert(Alert.AlertType.INFORMATION);  alert.setTitle("Information");  alert.setHeaderText(null);  alert.setContentText(message);  alert.showAndWait();  }  private void updateStudent() {  Student student = studentTable.getSelectionModel().getSelectedItem();  if (student == null) {  showAlert(Alert.AlertType.ERROR, "No Student Selected", "Please select a student to update.");  return;  }  Stage stage = new Stage();  stage.setTitle("Update Student");  GridPane gridPane = new GridPane();  gridPane.setPadding(new Insets(10));  gridPane.setHgap(10);  gridPane.setVgap(10);  gridPane.setAlignment(Pos.CENTER);  gridPane.getStyleClass().add("grid-pane");  TextField idField = new TextField();  idField.setPromptText("ID");  idField.setText(String.valueOf(student.getId()));  idField.setDisable(true);  TextField nameField = new TextField();  nameField.setPromptText("Name");  nameField.setText(student.getName());  TextField emailField = new TextField();  emailField.setPromptText("Email");  emailField.setText(student.getEmail());  TextField phoneNumberField = new TextField();  phoneNumberField.setPromptText("Phone Number");  phoneNumberField.setText(student.getPhone());  // Department ComboBox  ComboBox<String> departmentComboBox = new ComboBox<>();  ObservableList<String> departments = FXCollections.observableArrayList(  "Admissions", "Finance", "IT", "Marketing", "Human Resources"  );  departmentComboBox.setItems(departments);  departmentComboBox.setPromptText("Select Department");  departmentComboBox.getStyleClass().add("combo-box");  departmentComboBox.setValue(student.getDepartment());  // Status ComboBox  ComboBox<String> statusComboBox = new ComboBox<>();  ObservableList<String> statusList = FXCollections.observableArrayList(  "Pending", "Under Process", "Accepted", "Rejected", "Conditionally Accepted"  );  statusComboBox.setItems(statusList);  statusComboBox.setPromptText("Select Status");  statusComboBox.getStyleClass().add("combo-box");  statusComboBox.setValue(student.getStatus());  CheckBox uploadedDocumentsCheckBox = new CheckBox("Documents Uploaded");  uploadedDocumentsCheckBox.setSelected(student.isUploadedDocuments());  TextField documentNameField = new TextField();  documentNameField.setPromptText("Document Name");  documentNameField.setText(student.getDocumentName());  Button selectDocumentButton = new Button("Select Document");  selectDocumentButton.setOnAction(e -> {  FileChooser fileChooser = new FileChooser();  fileChooser.setTitle("Select Document");  selectedFile = fileChooser.showOpenDialog(stage); // Store the selected file  if (selectedFile != null) {  documentNameField.setText(selectedFile.getName());  }  });  Button submitButton = new Button("Submit");  submitButton.setOnAction(e -> {  String name = nameField.getText();  String email = emailField.getText();  String phoneNumber = phoneNumberField.getText();  String department = departmentComboBox.getValue();  String status = statusComboBox.getValue();  boolean uploadedDocuments = uploadedDocumentsCheckBox.isSelected();  String documentName = documentNameField.getText();  File documentFile = null; // Get the selected document file  if (!name.isEmpty() && !email.isEmpty() && !phoneNumber.isEmpty() && !department.isEmpty() && status != null) {  if (!email.matches("^[\\w-\_.]+@([\\w-]+\\.)+[\\w-]{2,4}$")) {  showAlert(Alert.AlertType.ERROR, "Invalid Email", "Please enter a valid email address.");  return;  }  if (!phoneNumber.matches("\\d{10}")) {  showAlert(Alert.AlertType.ERROR, "Invalid Phone Number", "Phone number must be exactly 10 digits.");  return;  }  // Update student data  student.setName(name);  student.setEmail(email);  student.setPhone(phoneNumber);  student.setDepartment(department);  student.setStatus(status);  student.setUploadedDocuments(uploadedDocuments);  student.setDocumentName(documentName);  // Perform the update operation using StudentDAO  try {  studentDAO.updateStudent(student);  // Handle document upload after student data is inserted  if (selectedFile != null) {  // Create the folder structure based on the student's email  File folder = new File("StudentDocuments" + File.separator + email);  if (!folder.exists()) {  folder.mkdirs();  }  // Define the destination file  File destFile = new File(folder, selectedFile.getName());  // Copy the file to the destination folder  try {  Files.copy(selectedFile.toPath(), destFile.toPath(), StandardCopyOption.REPLACE\_EXISTING);  boolean isSaved = studentDAO.saveDocument(destFile.getAbsolutePath(), email);  if (isSaved) {  showConfirmation("Document uploaded successfully.");  } else {  showAlert(Alert.AlertType.ERROR, "Failed to upload document.", "Document upload failed.");  }  } catch (IOException ex) {  showAlert(Alert.AlertType.ERROR, "File Error", "An error occurred while copying the file.");  ex.printStackTrace();  }  }  students = FXCollections.observableArrayList(studentDAO.fetchStudentsFromDatabase());  studentTable.setItems(FXCollections.observableArrayList(students));  stage.close();  studentTable.refresh(); // Refresh the table view  stage.close(); // Close the update window  } catch (SQLException ex) {  showAlert(Alert.AlertType.ERROR, "Database Error", "Failed to update student in the database.");  ex.printStackTrace();  }  } else {  showAlert(Alert.AlertType.ERROR, "Please fill in all fields.", "Please fill in all required fields.");  }  });  gridPane.add(new Label("ID:"), 0, 0);  gridPane.add(idField, 1, 0);  gridPane.add(new Label("Name:"), 0, 1);  gridPane.add(nameField, 1, 1);  gridPane.add(new Label("Email:"), 0, 2);  gridPane.add(emailField, 1, 2);  gridPane.add(new Label("Phone Number:"), 0, 3);  gridPane.add(phoneNumberField, 1, 3);  gridPane.add(new Label("Department:"), 0, 4);  gridPane.add(departmentComboBox, 1, 4);  gridPane.add(new Label("Status:"), 0, 5);  gridPane.add(statusComboBox, 1, 5);  gridPane.add(uploadedDocumentsCheckBox, 0, 6, 2, 1);  gridPane.add(new Label("Document Name:"), 0, 7);  gridPane.add(documentNameField, 1, 7);  gridPane.add(selectDocumentButton, 1, 8);  gridPane.add(submitButton, 1, 9);  Scene scene = new Scene(gridPane, 400, 500);  stage.setScene(scene);  stage.show();  }  private void deleteStudent() {  Student student = studentTable.getSelectionModel().getSelectedItem();  if (student == null) {  showAlert(Alert.AlertType.ERROR, "No Student Selected", "Please select a student to delete.");  return;  }  try {  studentDAO.deleteStudent(student.getId());  // Directly remove the student from the original observable list  students.remove(student);  students = FXCollections.observableArrayList(studentDAO.fetchStudentsFromDatabase());  studentTable.setItems(FXCollections.observableArrayList(students));  showAlert(Alert.AlertType.INFORMATION, "Student Deleted", "The student record has been deleted successfully.");  } catch (SQLException e) {  showAlert(Alert.AlertType.ERROR, "Database Error", "Failed to delete student from the database.");  e.printStackTrace();  }  }  private void generateReport() {  FileChooser fileChooser = new FileChooser();  fileChooser.setTitle("Save Report");  fileChooser.getExtensionFilters().add(new FileChooser.ExtensionFilter("CSV Files", "\*.csv"));  File file = fileChooser.showSaveDialog(null);  if (file != null) {  try (FileWriter writer = new FileWriter(file)) {  writer.append("ID,Name,Email,Phone Number,Department,Status,Document Name\n");  for (Student student : students) {  writer.append(String.format("%d,%s,%s,%s,%s,%s,%s\n",  student.getId(), student.getName(), student.getEmail(), student.getPhone(),  student.getDepartment(), student.getStatus(), student.getDocumentName()));  }  writer.flush();  showAlert(Alert.AlertType.INFORMATION, "Report Generated", "The report has been generated successfully.");  } catch (IOException e) {  showAlert(Alert.AlertType.ERROR, "Error", "An error occurred while generating the report.");  }  }  }  private void changePassword() {  Stage stage = new Stage();  stage.setTitle("Change Password");  GridPane gridPane = new GridPane();  gridPane.setPadding(new Insets(10));  gridPane.setHgap(10);  gridPane.setVgap(10);  gridPane.setAlignment(Pos.CENTER);  gridPane.getStyleClass().add("grid-pane");  TextField emailField = new TextField();  emailField.setPromptText("Email Address");  PasswordField currentPasswordField = new PasswordField();  currentPasswordField.setPromptText("Current Password");  PasswordField newPasswordField = new PasswordField();  newPasswordField.setPromptText("New Password");  PasswordField confirmPasswordField = new PasswordField();  confirmPasswordField.setPromptText("Confirm New Password");  Button submitButton = new Button("Submit");  submitButton.setOnAction(e -> {  String email = emailField.getText();  String currentPassword = currentPasswordField.getText();  String newPassword = newPasswordField.getText();  String confirmPassword = confirmPasswordField.getText();  if (email.isEmpty() || currentPassword.isEmpty() || newPassword.isEmpty() || confirmPassword.isEmpty()) {  showAlert(Alert.AlertType.ERROR, "Validation Error", "Please fill in all fields.");  return;  }  if (!email.matches("^[\\w-\_.]+@([\\w-]+\\.)+[\\w-]{2,4}$")) {  showAlert(Alert.AlertType.ERROR, "Invalid Email", "Please enter a valid email address.");  return;  }  if (!newPassword.equals(confirmPassword)) {  showAlert(Alert.AlertType.ERROR, "Password Mismatch", "New passwords do not match.");  return;  }  // Mockup for password change  // In a real application, you'd need to verify the current password and update it in the database.  showAlert(Alert.AlertType.INFORMATION, "Password Changed", "Password has been successfully changed.");  stage.close();  });  gridPane.add(new Label("Email Address:"), 0, 0);  gridPane.add(emailField, 1, 0);  gridPane.add(new Label("Current Password:"), 0, 1);  gridPane.add(currentPasswordField, 1, 1);  gridPane.add(new Label("New Password:"), 0, 2);  gridPane.add(newPasswordField, 1, 2);  gridPane.add(new Label("Confirm New Password:"), 0, 3);  gridPane.add(confirmPasswordField, 1, 3);  gridPane.add(submitButton, 1, 4);  Scene scene = new Scene(gridPane, 400, 300);  stage.setScene(scene);  stage.show();  }  private void showAlert(Alert.AlertType alertType, String title, String message) {  Alert alert = new Alert(alertType);  alert.setTitle(title);  alert.setHeaderText(null);  alert.setContentText(message);  alert.showAndWait();  }  private void showAlert1(String message) {  Alert alert = new Alert(Alert.AlertType.WARNING);  alert.setTitle("Warning");  alert.setHeaderText(null);  alert.setContentText(message);  alert.showAndWait();  }  } |
| Output: |
|  |

## Conclusion:

The Student Admission Management System aims to enhance operational efficiency, improve data management, and provide a seamless experience for students applying for admission. By leveraging technology, educational institutions can streamline their admission processes and focus more on delivering quality education to their students.

![A black and white text

Description automatically generated]()