|  |  |
| --- | --- |
| **Name** | Hamza Sagheer(036)  Muhammad muzammil (077)  Shehroz Malik(113) |
| **Roll no.** | SP24-BSE-036  SP24-BSE-077  SP24-BSE-113 |
| **Submitted to** | Sir Shahid Bhatti |
| **Assignment no** | 4 |
| **Submission date** | 12-12-2024 |



# Admin login Part

# Class Overview: AdminLoginController

The AdminLoginController class is part of a JavaFX application responsible for managing the admin login functionality. The class handles the user interface interactions, verifies user credentials from a file, and provides feedback to the user. Here's a breakdown of the functionality and key components:

# Imports

The class imports several JavaFX components and other required libraries:

* + Timeline, KeyFrame from javafx.animation for animating label blinking.
  + TextField, PasswordField, Label from javafx.scene.control for creating input fields and labels in the UI.
  + FXML, FXMLLoader for loading the FXML files that define the GUI structure.
  + Node, Parent, Scene, Stage for handling the UI scene transitions and window management.
  + File, FileReader, BufferedReader for handling file I/O operations to read user credentials from a file.
  + Duration, ActionEvent for managing time-based animation and event handling.

# FXML Annotations

The @FXML annotations are used to bind the JavaFX UI components defined in the FXML file to the corresponding fields in the controller class. These are:

* + textField: A TextField used for entering the admin username.
  + passField: A PasswordField used for entering the admin password.
  + invalidLabel: A Label used for displaying error messages when login fails.

# File Handling

* + The class uses a File object named file to reference a text file (AdminLoginUserPass.txt), which contains the admin credentials (username and password) for login authentication.

# Methods Overview

## clickOnFields

* + **Purpose**: This method sets event listeners for both the textField and passField. When the user clicks on these fields, the method clears any previous error messages displayed in the invalidLabel.
  + **Functionality**: The event listeners are registered on the textField and passField using setOnMouseClicked. This ensures that whenever the user clicks into the fields, the error label is reset.

## signIn

* + **Purpose**: This method is triggered when the user clicks the "Sign In" button to attempt login.

### Functionality:

* + - First, it checks if the credentials file (AdminLoginUserPass.txt) exists.
    - If the file exists, it reads through the file line by line using BufferedReader.
    - Each line in the file is split into an array of strings, where the first element is the username and the second element is the password.
    - If the entered username and password match the credentials in the file, the user is considered authenticated.
    - If successful, the method loads a new scene (adminValidLogIn.fxml) using FXMLLoader and transitions to the admin's valid login view.
    - If the authentication fails, the username and password fields are cleared, and an error message ("Invalid User or Password") is displayed in red. Additionally, the error label is animated to blink using the blinkingLabel method.

## backBtn

* + **Purpose**: This method is triggered when the user clicks the "Back" button to return to the landing page.
  + **Functionality**: It loads the landingPage.fxml using FXMLLoader and switches to the landing page scene.

## blinkingLabel

* + **Purpose**: This method animates the invalidLabel to blink when invalid login credentials are entered.
  + **Functionality**: A Timeline is used to change the opacity of the label every 500 milliseconds, alternating between fully visible and invisible. The animation runs for 10 cycles (blinks).

# File Reading Logic

* + The file reading logic in the signIn method:
    - A FileReader and BufferedReader are used to read the contents of the file line by line.
    - The file is expected to contain lines where each line has a username and password separated by a space.
    - The input from the textField (username) and passField (password) is compared against each line of the file. If a match is found, the user is authenticated.
    - If no match is found after reading all the lines, an error message is shown.

# Error Handling

* + If the AdminLoginUserPass.txt file is not found, a message is printed to the console indicating that the file is missing.
  + If the user enters invalid credentials, the login fields are cleared, and an error message is shown with a blinking effect.

# Summary

The AdminLoginController class manages the admin login flow in a JavaFX application. It reads credentials from a file, authenticates the user, and provides feedback in the form of a blinking error message if the login fails. It also allows the user to navigate between scenes, including transitioning to the admin's dashboard or the landing page.

# Class Overview: AdminValidLoginController

The AdminValidLoginController class is responsible for handling the functionality after a successful login by the admin in the JavaFX application. It manages the main admin dashboard and provides options for managing students, including adding, deleting, and updating student information. Additionally, it controls the visibility of different UI sections and handles the admin logout process.

# Imports

The class imports various JavaFX components to handle GUI components (like buttons, text fields, labels, and event handling) and file operations to persist student data:

* + Alert, ButtonType: For showing confirmation dialogs (like logout).
  + ActionEvent, MouseEvent: For handling UI events like button clicks or mouse clicks.
  + Layout containers like HBox, VBox, AnchorPane, and BorderPane to organize the UI elements.
  + TextField, Label: For gathering and displaying user input and information.
  + File, ObjectOutputStream, ObjectInputStream: For serializing and deserializing student data.

# FXML Annotations

The @FXML annotations indicate that the fields are linked to UI components defined in the FXML file. These fields include:

* + **UI Components for Navigation and Display**: HBox and VBox for the navigation and layout of the UI elements, and Label to display the admin's username and student count.
  + **Text Fields for Input**: TextField fields for student data input (such as first name, username, password, semester, and fees).
  + **Paned UI Containers**: AnchorPane containers for different sections of the dashboard like student management, student adding, deleting, and updating.

# Instance Variables

* + **Student List**: A list of Student objects to store the student data (students). Initially, a few students are added manually.
  + **File Handling**: A file (studentsData.ser) is used for storing serialized student data. An ObjectOutputStream (os) is used to write data, and ObjectInputStream (is) is for reading data.
  + **Pane and Box Collections**: panes (list of AnchorPanes) and hboxes (list of HBoxs) manage the visibility and styling of the different sections of the admin dashboard.

# Methods Overview

## initialize

* + **Purpose**: This method initializes the panes and hboxes collections by adding corresponding UI elements (like different panes and navigation boxes). The byDefaultBlank method is called to initially hide all panes.

## clickOnDashBoard and clickOnStudents

* + **Purpose**: These methods handle the visibility and style changes when the admin clicks on the Dashboard or Students section of the navigation.
  + **Functionality**: Calls visiblitityAndStyleStatus to toggle the visibility of the corresponding pane and apply the active style to the navigation box.

## addStudent

* + **Purpose**: Adds a new student to the list and updates the studentsData.ser file.
  + **Functionality**: A new Student object is created using the data entered in the text fields (stFirstName, stUserName, etc.). The student count is updated, and the form fields are cleared after adding the student to the list.

## deleteStudent

* + **Purpose**: Deletes a student from the list by matching the registration number entered in delStRegField.
  + **Functionality**: An iterator is used to search through the student list. If the student is found, they are removed, and the updated list is saved back to the file. If the student is not found, an alert message is displayed to the user.

## stBackBtn

* + **Purpose**: Handles the "Back" button click to return to the student management section.
  + **Functionality**: Toggles visibility between the student management pane and the student navigation box.

## clickOnAddSt, clickOndelSt, clickOnUpdateSt

* + **Purpose**: These methods handle the click events for switching to the add, delete, or update student sections.
  + **Functionality**: Each method calls visiblitityAndStyleStatus to display the relevant pane and highlight the navigation box.

## dataByID

* + **Purpose**: Searches for a student by their registration ID and prints the details if found.
  + **Functionality**: An iterator searches through the students list to find a student with the matching ID and prints it to the console (likely for debugging purposes).

## clickOnTeachers, clickOnCourses, clickOnFees, clickOnSalary

* + **Purpose**: Placeholder methods for handling teacher, course, fee, and salary sections, but their functionality is not yet implemented.

## visiblitityAndStyleStatus

* + **Purpose**: Controls the visibility and styling of the active pane and navigation box.
  + **Functionality**: It makes the active pane visible while hiding all others. It also changes the background color and text color of the active navigation box to highlight the selected section.

## doLogout

* + **Purpose**: Handles the logout process.
  + **Functionality**: Displays a confirmation dialog asking the admin if they are sure about logging out. If confirmed, the scene is changed to the admin login screen.

## byDefaultBlank

* + **Purpose**: Hides all panes by default when the admin first logs in.
  + **Functionality**: Sets all panes to invisible, effectively hiding them from the UI.

## userName

* + **Purpose**: Sets the username of the logged-in admin.
  + **Functionality**: Updates the nameLabel with the admin's name.

# File Handling (Serialization)

* + **Purpose**: The class uses ObjectOutputStream and ObjectInputStream to save and load the student data. When a student is added or deleted, the list is serialized and saved to the file (studentsData.ser).
  + **Functionality**: The list of students is written to the file each time the data is updated.

# Alerts

* + **Purpose**: The class uses Alert to display informative messages to the user, such as when a student is successfully deleted or when the admin is asked to confirm the logout action.
  + **Functionality**: Information and confirmation dialogs are shown based on the action being performed (deleting a student or logging out).

# Summary

The AdminValidLoginController class is responsible for managing the admin dashboard after a successful login. It handles student management tasks such as adding, deleting, and updating students, as well as toggling between different sections of the dashboard. It also includes functionality for logging out and for managing the visibility and style of the dashboard's UI elements. The student data is persisted using file serialization.

# Class Overview: HelloApplication

The HelloApplication class is the entry point for a JavaFX application. It extends the Application class, which is a standard way of launching JavaFX applications. This class is responsible for setting up the initial user interface and displaying the main scene (which is loaded from an FXML file). Here's a detailed breakdown of the class:

# Imports

* + Application: The base class for all JavaFX applications. It provides lifecycle methods like start() that you override to set up the application's primary stage (window).
  + FXMLLoader: Used to load the FXML file that defines the UI structure.
  + Parent: The base class for all JavaFX nodes that can be used as a container (e.g., Pane, VBox, HBox).
  + Scene: Represents the content of a JavaFX stage. It contains all the visual elements displayed to the user.
  + Stage: Represents the main window of the JavaFX application.
  + IOException: Handles input-output exceptions that may arise while reading the FXML file.

# start Method

The start() method is the main entry point for the JavaFX application. It gets called automatically when the application is launched.

### FXMLLoader:

* + - The FXMLLoader is used to load the landingPage.fxml file. This FXML file is assumed to define the layout and UI components of the landing page of the application.
    - FXMLLoader.getClass().getResource("landingPage.fxml") locates the FXML file relative to the current class.

### Parent root:

* + - The FXMLLoader.load() method returns a Parent node, which is the root of the scene graph for the application.
    - This root node represents the entire structure of the UI, such as buttons, labels, and containers defined in the landingPage.fxml file.

### Scene:

* + - The root node is then placed inside a Scene object. A Scene in JavaFX defines the content to be displayed on the stage.

### Stage:

* + - The stage.setScene(scene) method sets the created scene to the main window (stage).
    - stage.show() makes the stage visible to the user, displaying the UI.

# main Method

* + The main() method calls launch(), which is a static method from the Application class.
  + This method is responsible for launching the JavaFX application, which, in turn, triggers the start() method to set up and display the application window.

# File Handling and UI Setup

* + The class does not directly handle any data or business logic. It simply sets up the UI by loading the landingPage.fxml file and presenting it in a window.
  + The landingPage.fxml file would contain the layout, which includes buttons, labels, text fields, etc., and is managed by the controllers linked to the FXML file.

# Summary

The HelloApplication class is the main entry point for a JavaFX application. It loads the landingPage.fxml file, creates a Scene with it, and sets it on a Stage to display the UI. The main method simply triggers the launch of the application by invoking launch(). This class is primarily concerned with initializing and showing the UI on startup.

# Class Overview: LandingPageController

The LandingPageController class manages the navigation for the landing page of the application. It is responsible for directing the user to different login pages, such as the admin login page, student login page, and teacher login page, when the respective buttons are clicked.

# Imports

* + ActionEvent: This is used to handle events triggered by UI elements, such as button clicks.
  + FXMLLoader: Used to load FXML files, which define the layout and structure of the user interface.
  + Node: A base class for all JavaFX UI components (like buttons, labels, etc.). It is used here to obtain the current stage (window).
  + Parent: Represents the root node of the scene graph. It is returned by the FXMLLoader.load() method to be set as the root of the scene.
  + Scene: Represents the content of the window, containing the layout and UI components.
  + Stage: Represents the window itself in the JavaFX application.
  + IOException: Handles input-output exceptions that might occur when loading the FXML files.

# UI Component

* + **Button**: A Button named adminBtn is declared, though it is not used explicitly in the code. This button likely corresponds to the action of navigating to the admin login page from the landing page.

# Methods Overview

## adminLoginPage

* + **Purpose**: This method handles the event when the admin login button is clicked.

### Functionality:

* + - It loads the adminLogin.fxml file using the FXMLLoader.
    - A Parent object, which represents the root of the loaded UI, is retrieved from the FXML loader.
    - The Stage representing the current window is obtained by casting the event.getSource() to a Node and getting the window with getScene().getWindow().
    - A new Scene is created using the loaded root.
    - The scene is set on the stage, and the stage is shown to display the admin login page.

## StudentLoginPage

* + **Purpose**: This method handles the event when the student login button is clicked.

### Functionality:

* + - It follows the same logic as adminLoginPage, but instead of loading the admin login page, it loads the studentLogin.fxml file, showing the student login page.

## teacherLoginPage

* + **Purpose**: This method handles the event when the teacher login button is clicked.

### Functionality:

* + - Similar to the other two methods, this one loads the teacherLogin.fxml file and sets the teacher login page as the new scene in the window.

# General Flow and Purpose

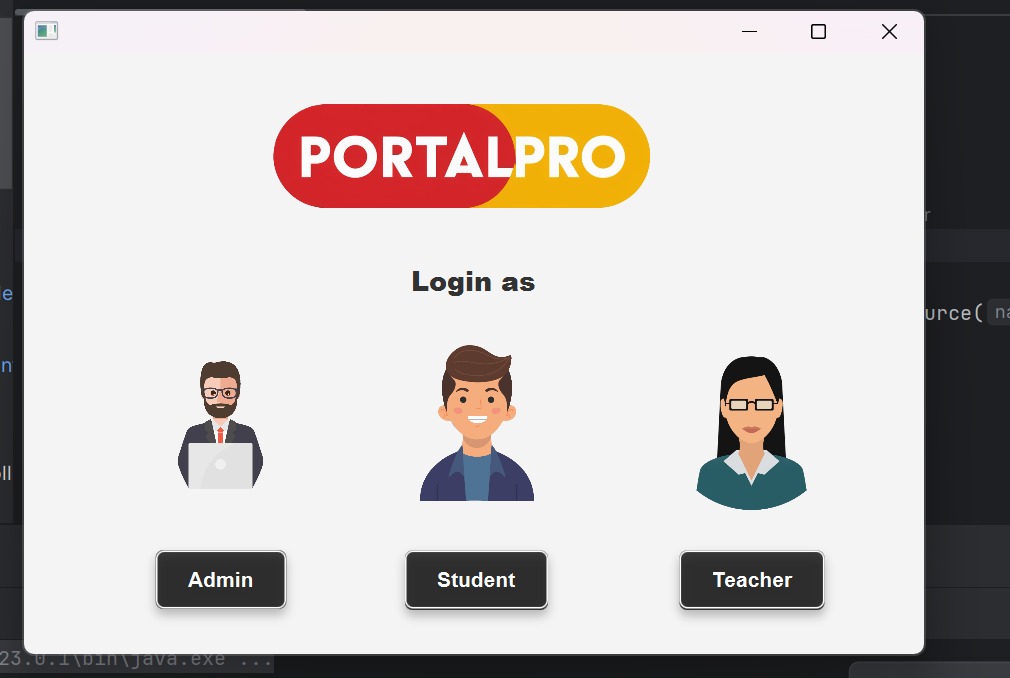
* + **Navigation Control**: The class acts as the controller for the landing page. It listens for events triggered by buttons, then loads the corresponding FXML file (for the admin, student, or teacher login) and changes the scene to the appropriate login page.
  + **Event Handling**: Each method takes an ActionEvent as a parameter, which is passed automatically when a button is clicked. The event provides information about the source of the action (in this case, the button that was clicked).
  + **Scene Management**: After loading the corresponding FXML file, the class creates a new Scene and sets it on the stage to switch the UI to the correct login page.

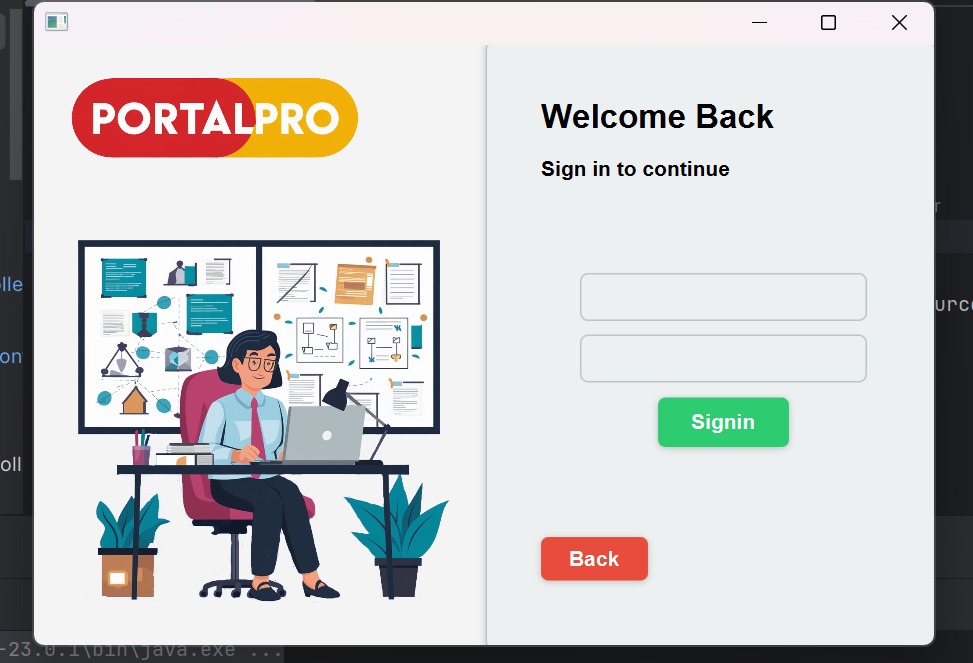
# Summary

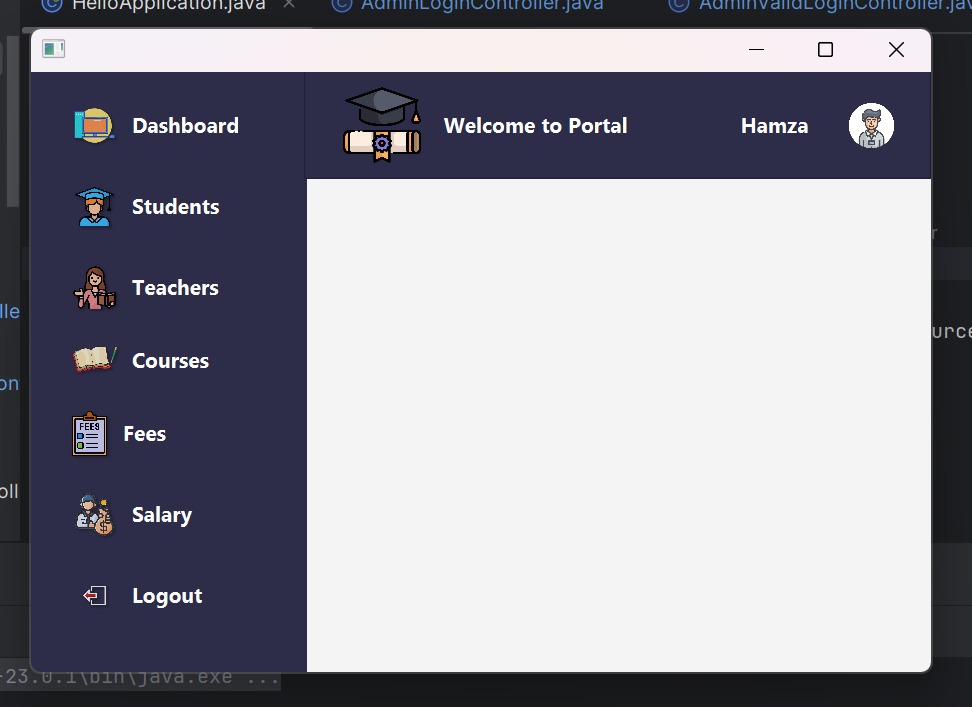
The LandingPageController class is designed to manage the navigation from the landing page to the login pages of different roles (admin, student, teacher). It listens for button clicks, loads the appropriate FXML file, and changes the scene to the corresponding login page. Each of the three methods (adminLoginPage, StudentLoginPage, and teacherLoginPage) follows a similar pattern of loading an FXML file and updating the scene.

***Final output***

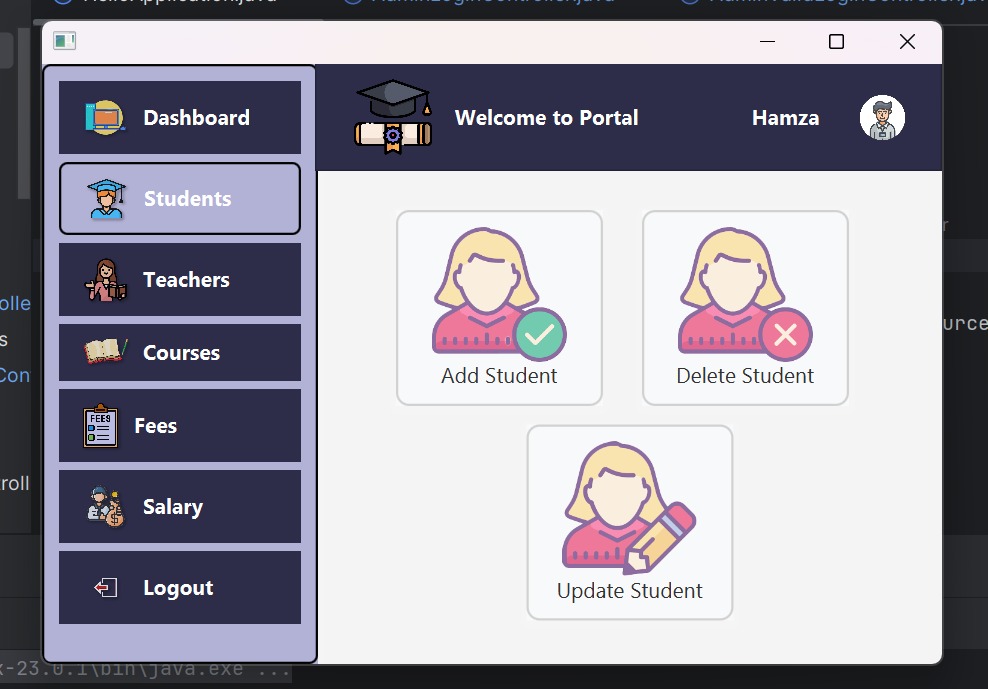
1. Landing page

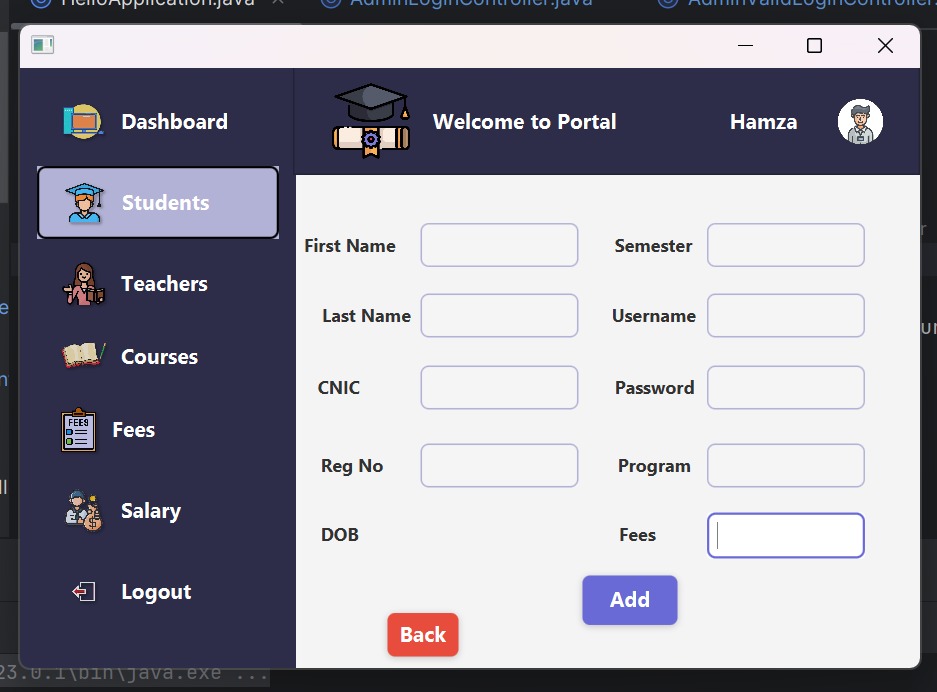


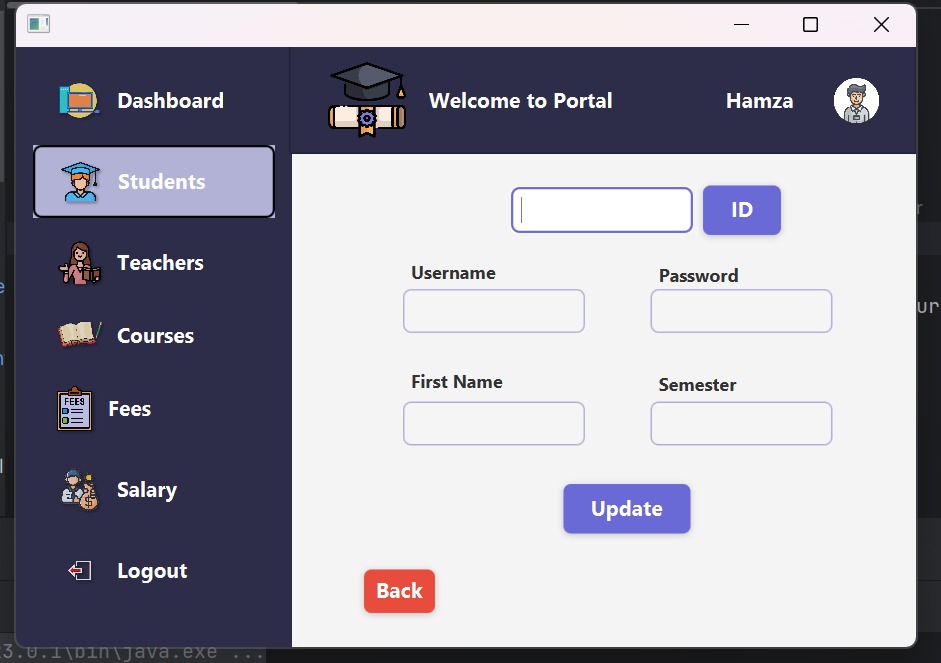
1. login controller
2. Entered in portal



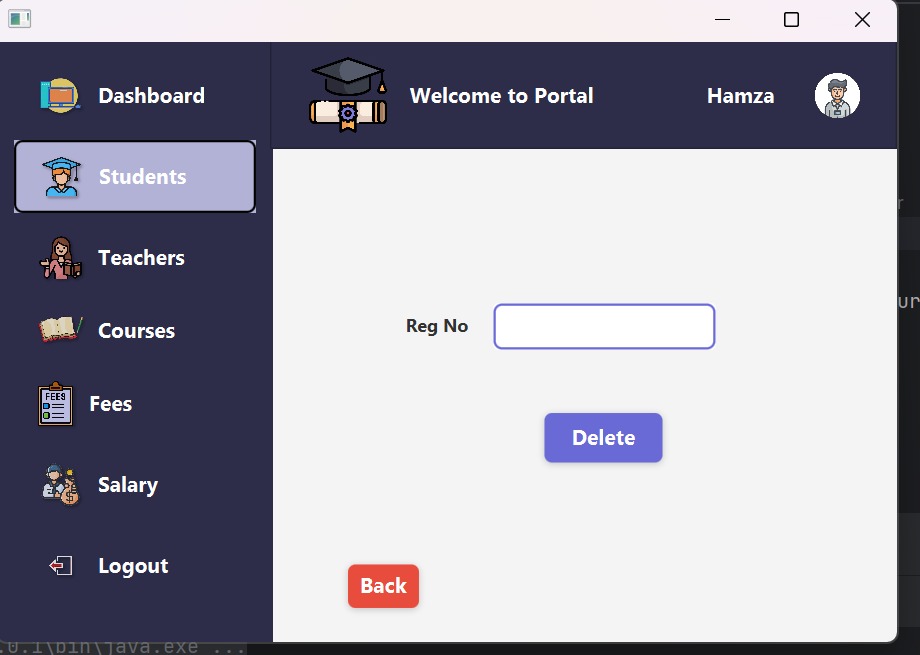
1. Students



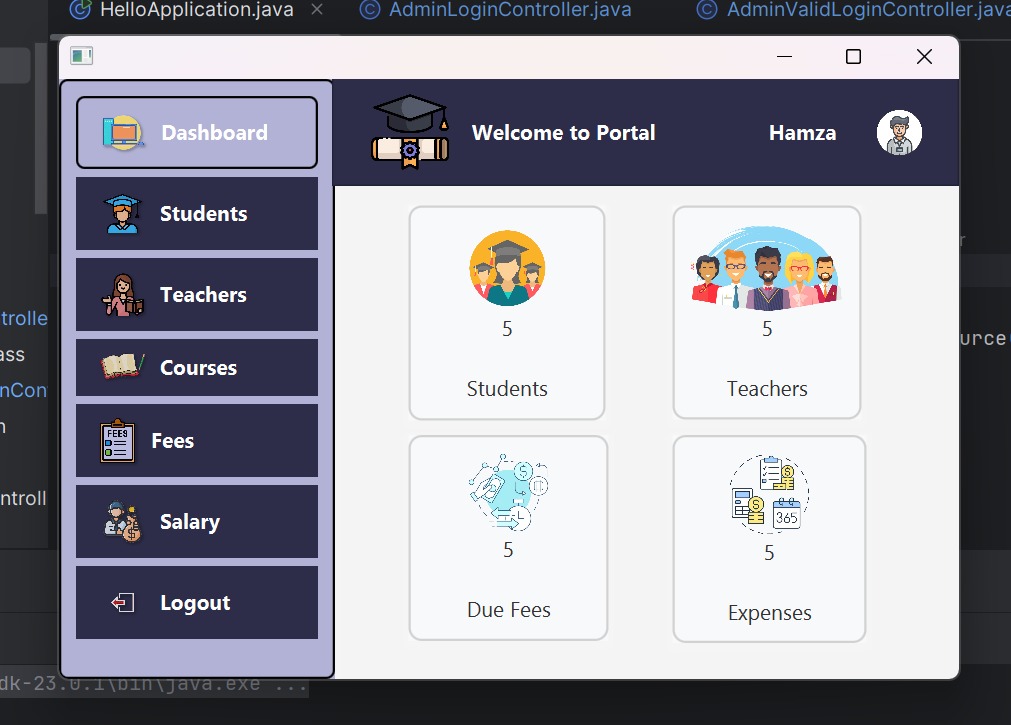
1. add Students
2. Update students



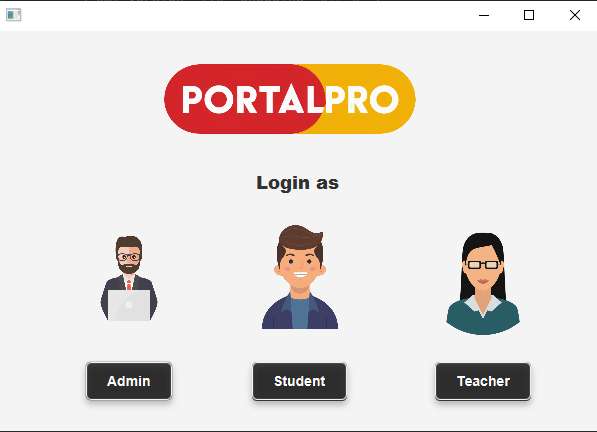
1. delete student



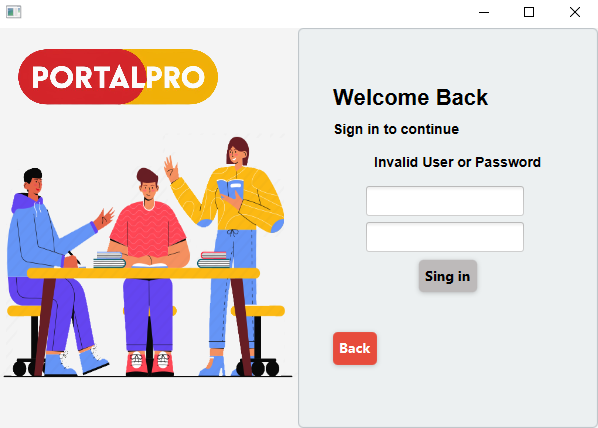
1. Dashboard



**Student login part**

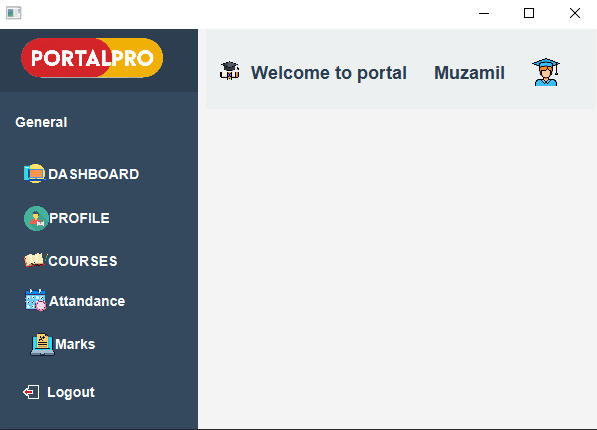


When execute a code the front page is appear

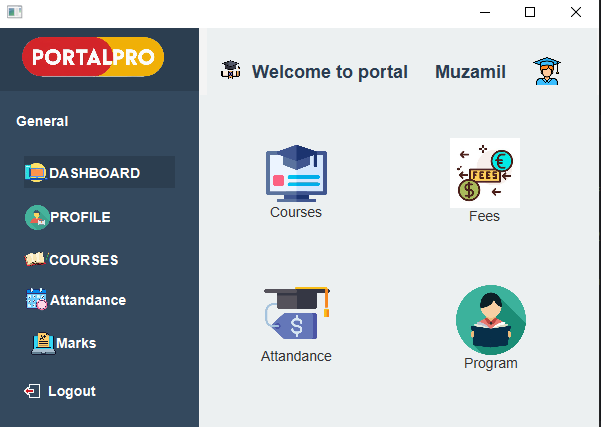


After clicking student button login page appear. When enter invalid password or user name then text show on the screen.

public void signIn(ActionEvent event) throws IOException {  
 if(!file.exists()) {  
 file.createNewFile();  
 System.*out*.println("File created");  
  
 }  
 FileWriter fileWriter=new FileWriter(file);  
 BufferedWriter bw=new BufferedWriter(fileWriter);  
 bw.write("Hamza 1000");  
 bw.newLine();  
 bw.write("Muzamil 1100");  
 bw.close();  
 if(file.exists()) {  
   
 FileReader fr = new FileReader(file);  
 BufferedReader br = new BufferedReader(fr);  
 String line;  
 boolean flag = false;  
  
 while ((line = br.readLine()) != null){  
 String [] userPassArray = line.split(" ");  
 String name=Textfield.getText();  
 if(userPassArray[0].equals(name) &&  
 userPassArray[1].equals(Passfield.getText())){  
 flag = true;  
 FXMLLoader loader = new FXMLLoader(getClass().getResource("Studentvalidlogin.fxml"));  
 Parent root =loader.load();  
 StudentvalidloginController controller = loader.getController();  
 controller.setlabel(name);  
 Stage stage = (Stage)((Node)event.getSource()).getScene().getWindow();  
 Scene scene = new Scene(root);  
 stage.setScene(scene);  
 stage.show();  
  
 }  
 }  
 if(!flag){  
 Textfield.clear();  
 Passfield.clear();  
 invalidLabel.setText("Invalid User or Password");  
 }  
 }  
  
}

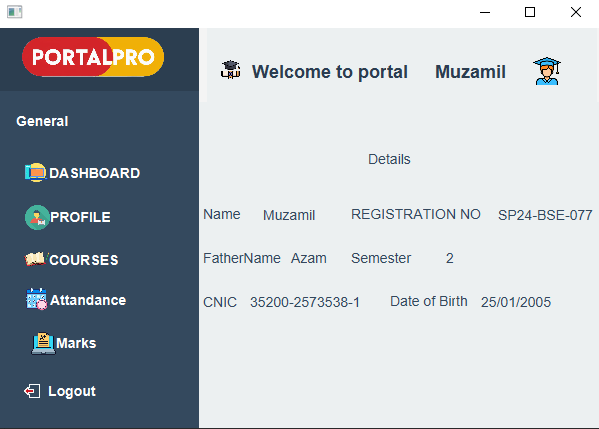


After enter valid username and password then portal successfully login



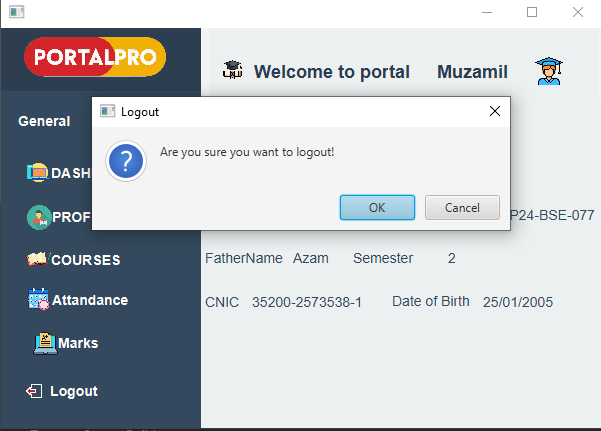
After clicking on dashboard these attributes are appear.

public void ClickDashboard(MouseEvent event) throws IOException  
{  
 setallFalse();  
 Dashboardpane.setVisible(true);  
}



After clicking on profile these attributes of students are show.

public void ClickOnProfile(MouseEvent event) throws IOException  
{  
 setallFalse();  
 profilePane.setVisible(true);



After click on logout then alert show for logout and after clicking OK then successfully logout.

public void ClickOnlogout(MouseEvent event) throws IOException {  
 Alert alert=new Alert(Alert.AlertType.*CONFIRMATION*);  
 alert.setTitle("Logout");  
 alert.setHeaderText(null);  
 alert.setContentText("Are you sure you want to logout!");  
 if(alert.showAndWait().get()==ButtonType.*OK*)  
 {  
 FXMLLoader fxmlLoader = new FXMLLoader(getClass().getResource("Studentlogin.fxml"));  
 Parent root = fxmlLoader.load();  
 Stage stage = (Stage)((Node)event.getSource()).getScene().getWindow();  
 Scene scene = new Scene(root);  
 stage.setScene(scene);  
 stage.show();  
 }  
 else {  
 event.consume();  
 }  
}

**Teacher login part**

**3.Documentation for teacherLoginController**

1. **Class Name: teacherLoginController**  
   This class handles the logic for the teacher login process in the "Portal Pro" application. It validates login credentials from a file and navigates the user to a success page upon correct login or returns to the landing page.
2. **Purpose**:  
   The teacherLoginController connects the teacher login interface to the application logic. It processes user input for authentication and manages navigation between scenes.
3. **Fields**:
   * **TextField textField**: Captures the teacher’s username input.
   * **PasswordField passField**: Captures the teacher’s password input.
   * **AnchorPane teacherLoginpane**: Represents the teacher login pane in the GUI.
   * **File file**: Refers to the file (TeacherLoginUserPass.txt) that stores teacher login credentials.
4. **Methods**:
   * **signIn(ActionEvent event)**:
     + Handles the sign-in process for teachers.
     + Steps:
       1. Checks if the file TeacherLoginUserPass.txt exists.
       2. Reads the file line by line.
       3. Splits each line into username and password.
       4. Compares the input username (textField) and password (passField) with the stored credentials.
       5. If a match is found:
          - Hides the current login pane.
          - Loads the teacherValidLogin.fxml scene.
          - Updates and displays the stage with the new scene.
       6. If no match is found or the file is missing, appropriate messages are logged.
     + **Throws**: IOException if file operations or scene loading fails.
   * **backBtn(ActionEvent event)**:
     + Handles navigation back to the landing page.
     + Steps:
       1. Loads the landingPage.fxml scene.
       2. Updates and displays the stage with the loaded scene.
     + **Throws**: IOException if the FXML file cannot be loaded.
5. **File Requirements**:
   * **TeacherLoginUserPass.txt**: This file must contain teacher login credentials in the format <username> <password>, separated by a space, with each credential pair on a new line.
   * **teacherValidLogin.fxml**: Defines the layout for the valid login page.
   * **landingPage.fxml**: Defines the layout for the landing page.
6. **Key Features**:
   * **Login Validation**:
     + Reads and parses credentials from an external file for flexibility and easy updates.
     + Validates user input against stored credentials.
   * **Navigation**:
     + Dynamically switches between the teacher login page, valid login page, and landing page.
7. **How It Works**:
   * The user enters their username and password in the provided fields and clicks the "Sign In" button.
   * The application reads credentials from TeacherLoginUserPass.txt and compares them to the input.
     + On success, the login pane is hidden, and the valid login page is displayed.
     + On failure, the pane remains visible, and no navigation occurs.
   * The "Back" button returns the user to the landing page.
8. **Notes**:
   * Ensure the file TeacherLoginUserPass.txt exists and follows the required format to avoid errors.
   * All FXML files (teacherValidLogin.fxml, landingPage.fxml) should be in the correct directories to load successfully.
   * Consider adding error messages or feedback for the user in the GUI to enhance usability.

**4. Documentation for teacherValidLoginController**

1. **Class Name: teacherValidLoginController**  
   This class manages the teacher's dashboard after a successful login. It enables navigation between various sections of the teacher interface, such as Dashboard, Profile, Attendance, and Courses, while also providing a logout functionality.
2. **Purpose**:  
   The teacherValidLoginController connects the graphical user interface (GUI) components to the application's backend logic. It facilitates seamless interaction with different sections of the teacher dashboard and updates the UI accordingly.
3. **FXML Mappings**:  
   The class includes several UI components mapped from the FXML file:
   * **AnchorPane**: Represents the main containers for each section (e.g., Dashboard, Profile, Attendance, Courses).
   * **HBox**: Represents horizontal layouts for buttons and grouped items.
   * **VBox**: Represents vertical layouts for specific sections within the dashboard or profile pages.
   * **Label**: Displays textual information in various sections.
4. **Fields**:
   * **AnchorPane**:
     + dashboardpane, defaultpane, profilepane, attendancepane, coursespane, borderPane: Represent the primary sections of the interface.
   * **HBox** (Buttons/Containers in default pane):
     + generalbox, dashboardbox, attendancebox, marksbox, coursesbox, profilebox.
   * **VBox** (Vertical Containers in Dashboard/Profile):
     + coursebox, salarybox, salarystatusbox, dashboardTbox, profileTbox.
   * **Label** (Dashboard/Profile Labels):
     + courseslabel, salarylabel, salarySlabel, noofcourseslabel, totalsalarylabel, salarystatuslabel, detaillabel, namelabel, nameSlabel, etc.
   * **Lists for Dynamic Management**:
     + panes: Stores all AnchorPane objects for easier visibility toggling.
     + hboxes: Stores HBox objects for button or layout management.
     + vboxes: Stores VBox objects for vertical layouts.
     + labels: Stores Label objects for text updates.
5. **Methods**:
   * **initialize()**:
     + Runs when the controller is initialized.
     + Adds all panes, HBox, VBox, and labels to respective lists for easier management.
     + Calls setAllFalse() to hide all sections and sets the default pane (defaultpane) to visible.
   * **setAllFalse()**:
     + Hides all AnchorPane sections by setting their visibility to false.
     + Ensures the defaultpane is visible by default.
   * **dashboard(MouseEvent event)**:
     + Activates the Dashboard section by:
       1. Calling setAllFalse() to hide all panes.
       2. Setting dashboardpane visibility to true.
   * **profile(MouseEvent event)**:
     + Activates the Profile section by:
       1. Calling setAllFalse().
       2. Setting profilepane visibility to true.
   * **attendance(MouseEvent event)**:
     + Activates the Attendance section by:
       1. Calling setAllFalse().
       2. Setting attendancepane visibility to true.
   * **courses(MouseEvent event)**:
     + Activates the Courses section by:
       1. Calling setAllFalse().
       2. Setting coursespane visibility to true.
   * **logout(MouseEvent event)**:
     + Logs the user out by:
       1. Loading the teacherLogin.fxml file using FXMLLoader.
       2. Updating the current stage with the teacher login scene.
6. **Key Features**:
   * **Section Switching**: Dynamically switches between different panes (e.g., Dashboard, Profile) without refreshing the entire application.
   * **Reusable Components**: Manages sections, layouts, and labels using lists for streamlined visibility toggling.
   * **Logout Functionality**: Seamlessly returns to the login page.
7. **How It Works**:
   * **Initialization**:
     + When the controller is loaded, the initialize() method sets up the default state by hiding all panes except the defaultpane.
   * **Section Navigation**:
     + Each navigation method (e.g., dashboard, profile) hides all panes and displays the relevant section by toggling its visibility.
   * **Logout**:
     + When the user clicks the logout button, the application transitions back to the teacher login page (teacherLogin.fxml).
8. **File Requirements**:
   * **FXML Files**:
     + teacherLogin.fxml: Login page layout.
     + teacherValidLogin.fxml: Dashboard layout.
   * Ensure the FXML files are correctly located in the resources directory.
9. **Notes**:
   * Enhance user experience by adding error messages or feedback if a pane cannot be displayed.
   * Optimize performance by dynamically updating only the visible panes instead of hiding all panes.
10. **Documentation for teacherLoginController**

The layout is managed by the **teacherLoginController** class, which handles the logic behind the UI, such as processing the input and managing user interactions. The file uses external stylesheets for visual customization.

The XML code you've provided represents a JavaFX layout for a teacher portal. It defines an interface where a teacher can interact with various components like dashboard, attendance, profile, and courses, among others.

Here’s a breakdown of what’s included in the layout:

### Main AnchorPane (borderPane):

* 1. This is the root container with a fixed width of 600px and height of 400px, containing several child AnchorPane components representing different sections of the portal.

### Default Pane (defaultpane):

* 1. This section appears to provide a navigation menu with options like "Dashboard," "Attendance," "Marks," "Courses," and "Profile," among others. These options are presented as labels within HBox containers.
  2. It has a teal background for the navigation box.

### Dashboard Pane (dashboardpane):

* 1. Displays three sections showing "Total Courses," "Salary Status," and "Salary," with corresponding images and labels.
  2. It uses VBox and ImageView to show icons along with text describing each category.

### Profile Pane (profilepane):

* 1. Displays detailed information about the teacher, including their name, employee ID, phone number, date of birth, joining date, and qualifications, laid out in a series of HBox containers.
  2. Each HBox contains a label describing the information (e.g., "Name," "Phone No.") and a Label element showing the actual data (e.g., "Ahmed Furqan").

### Attendance Pane (attendancepane):

* 1. Allows a teacher to mark attendance by entering the registration ID, course code, and selecting a date. It includes checkboxes for marking attendance (Present or Absent) and a button to confirm the action.

### Courses Pane (coursespane):

* 1. Displays a list of assigned courses with their details such as course name, course code, and credit hours. Each course is represented using a VBox.

### Top Navigation Bars (toprightbox and topleftbox):

* 1. These two HBox components create a top navigation bar, displaying the teacher’s

name and an image on the right side, and a logo on the left side.

1. **Key Features and Components:**

* **Labels (Label)**: Used throughout the UI to display text, such as "Dashboard," "Attendance," and personal information like the teacher's name or employee ID.
* **Buttons (Button)**: For actions such as "Mark Attendance" and navigation (back or logout).
* **Images (ImageView)**: Used to display icons related to different sections (dashboard, courses, salary, etc.).
* **VBox, HBox, and AnchorPane**: Layout containers for organizing the UI elements.

# Functionality:

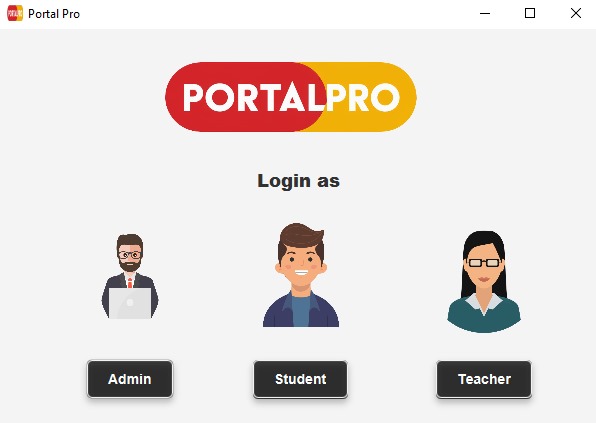
* The controller associated with the fx:controller="com.example.portal.teacherValidLoginController" will handle the logic and interaction between these UI elements.
* The action methods like #dashboard, #attendance, #logout are linked to UI elements (e.g., HBox click actions), which would trigger specific events to update the view or navigate to other sections.

This layout seems well-organized for a teacher’s portal and is likely to be interactive with backend

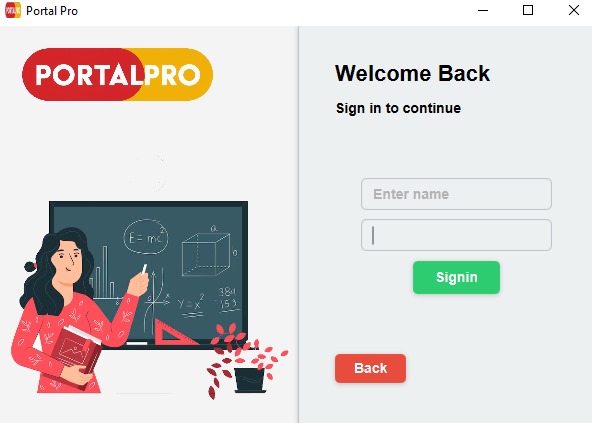
logic for functionalities like marking attendance, viewing courses, etc.

**Output/Display**

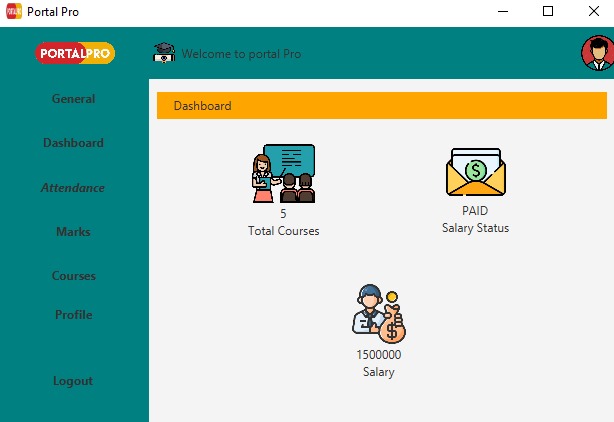
**Landing page**



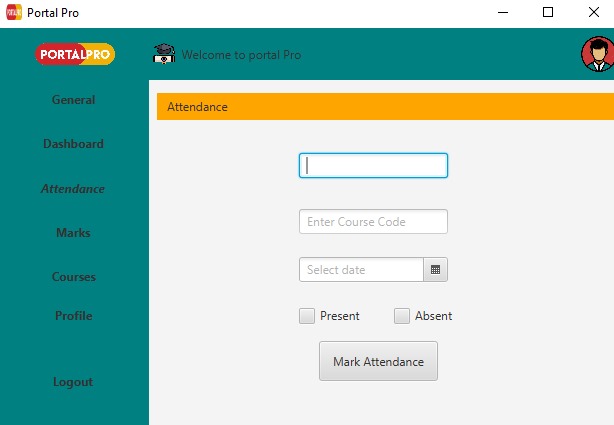
**Login page**



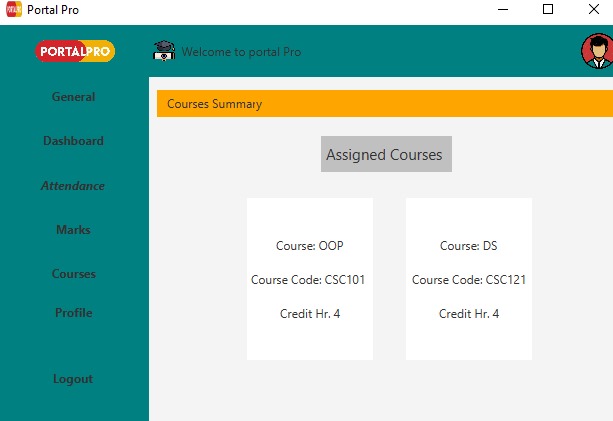
**Dashboard**

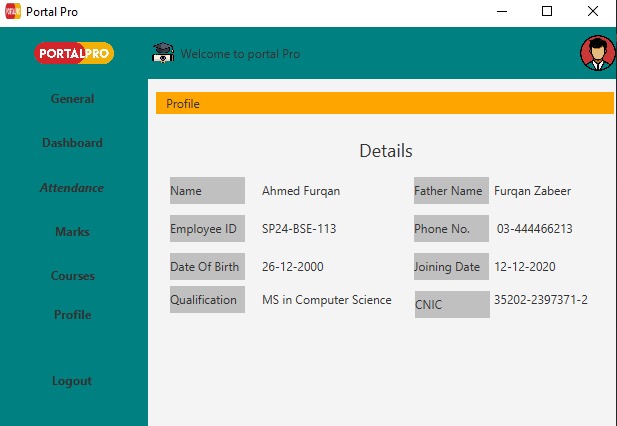


**Attendance**



**Course summary**

****



**Profile**