GROUP ASSIGNMENT REPORT.

TECHNOLOGY PARK MALAYSIA.

CT044-3-2-IOOP.

INTRODUCTION TO OBJECT ORIENTED PROGRAMMING.

APD1F2209CS(CYB).

HAND OUT DATE: WEEK 3.

HAND IN DATE: WEEK 12.

WEIGHTAGE: 50%

LECTURER: Sathiapriya Ramiah.

GROUP # 1 STUDENT NAMES & TP NUMBERS:

1) ABDULRAHMAN GAMIL MOHAMMED AHMED (TP071012).

2) IBRAHEEM MOHAMMED IMADELDIN AWAD (TP070765).

3) AMANULLAH GHAURI (TP071215)

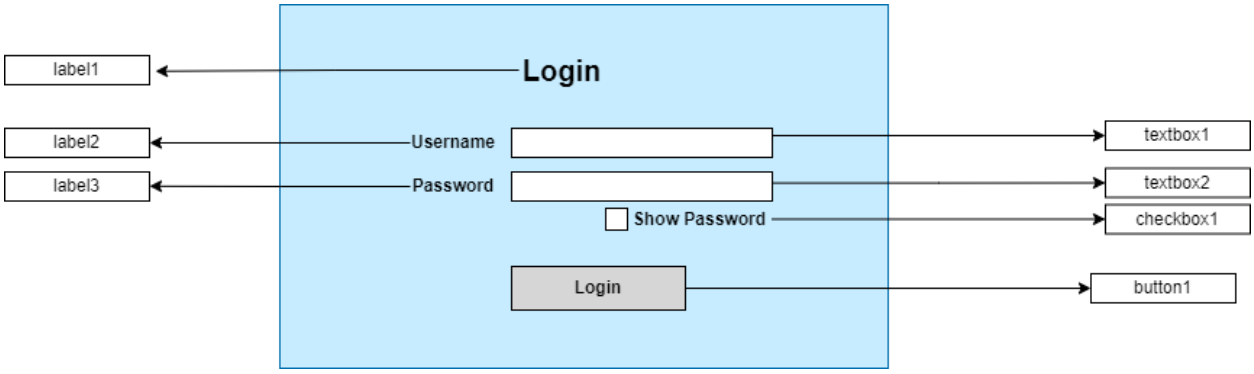
4) ABDULELAH HUSSEIN ABDULRAHMAN AL-KAF (TP069319)

Table of Contents

1.0 Storyboard

1.1 Administrator (Ibraheem Mohammed Imadeldin Awad TP070765)

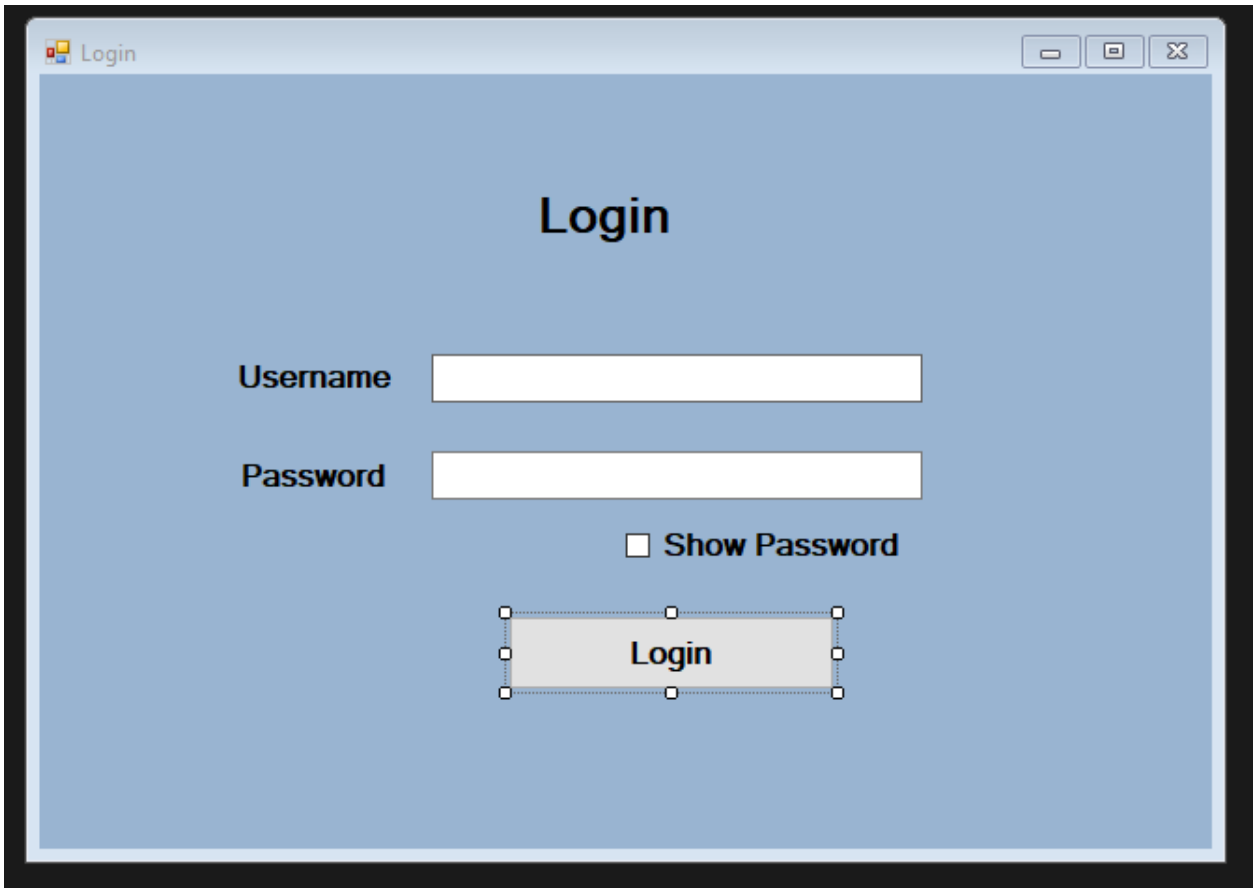
Storyboard for Login



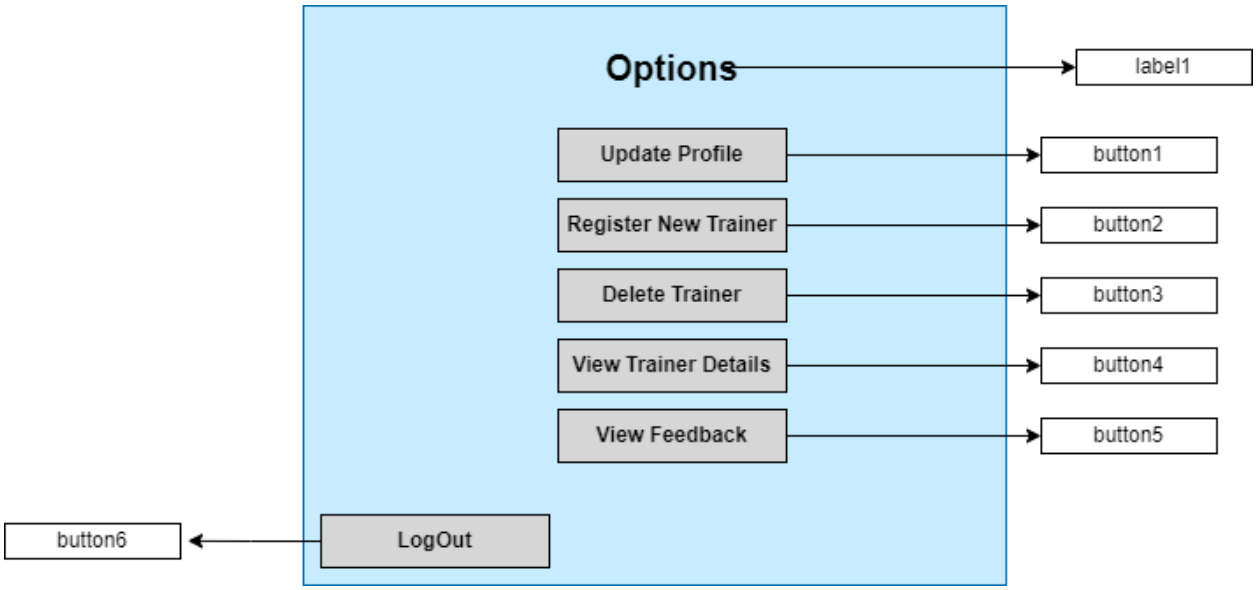
Storyboard Description

Control	Control Name	Description
label1	lbl_Login	To label the form header
label2	lbl_Username	To label the related controls
label3	lbl_Password	
textbox1	txt_Username	To allow user to enter username
textbox2	txt_Password	To allow user to enter password
checkbox1	chckbx_ShowPassword	To show the password
button1	btn_Login	To enable user to login into system

User Interface for Login



Storyboard for Options



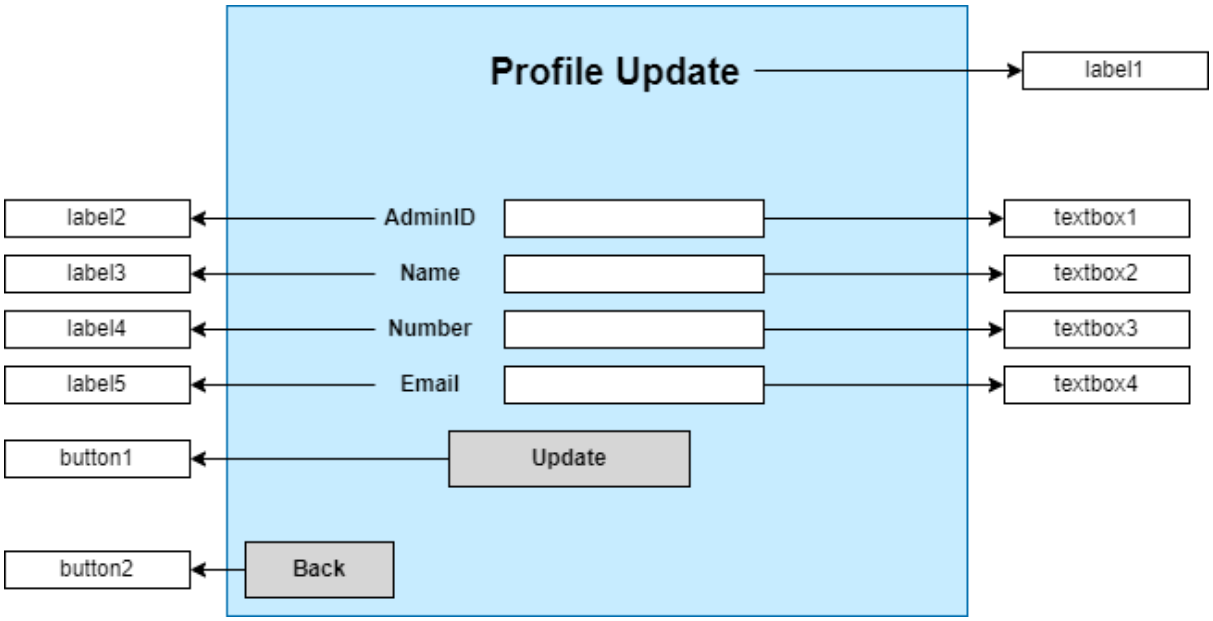
Storyboard Description

Control	Control Name	Description
label1	lbl_Options	To label the form header
button1	btn_UpdateProfile	To take the user to the update profile form
button2	btn_RegisterNewTrainer	To take the user to the Trainer Registration form
button3	btn_DeleteTrainer	To take the user to the Deleting Trainer form
button4	btn_TrainerDetails	To take the user to the View Trainer Details form

button5	btn_Feedback	To take the user to the View Feedback form
button6	btn_LogOut	To take the user back to the login form

User Interface for Options

Storyboard for Profile Update

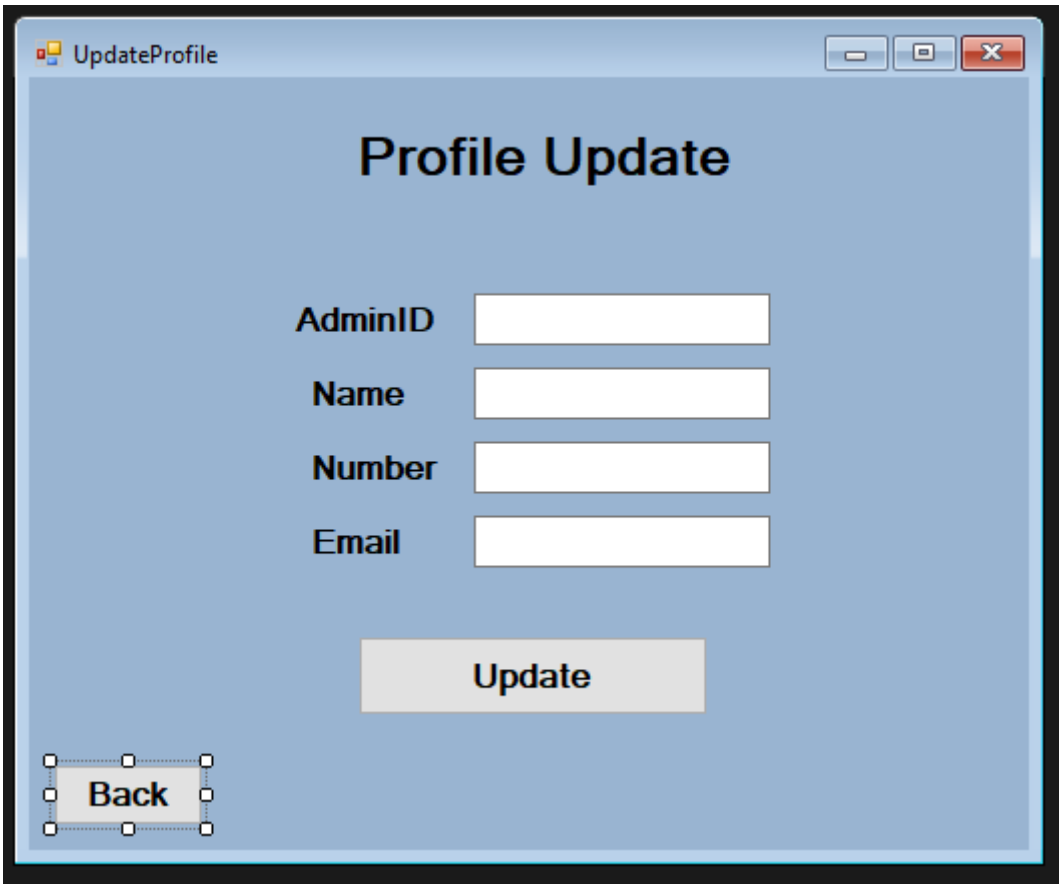


Storyboard Description

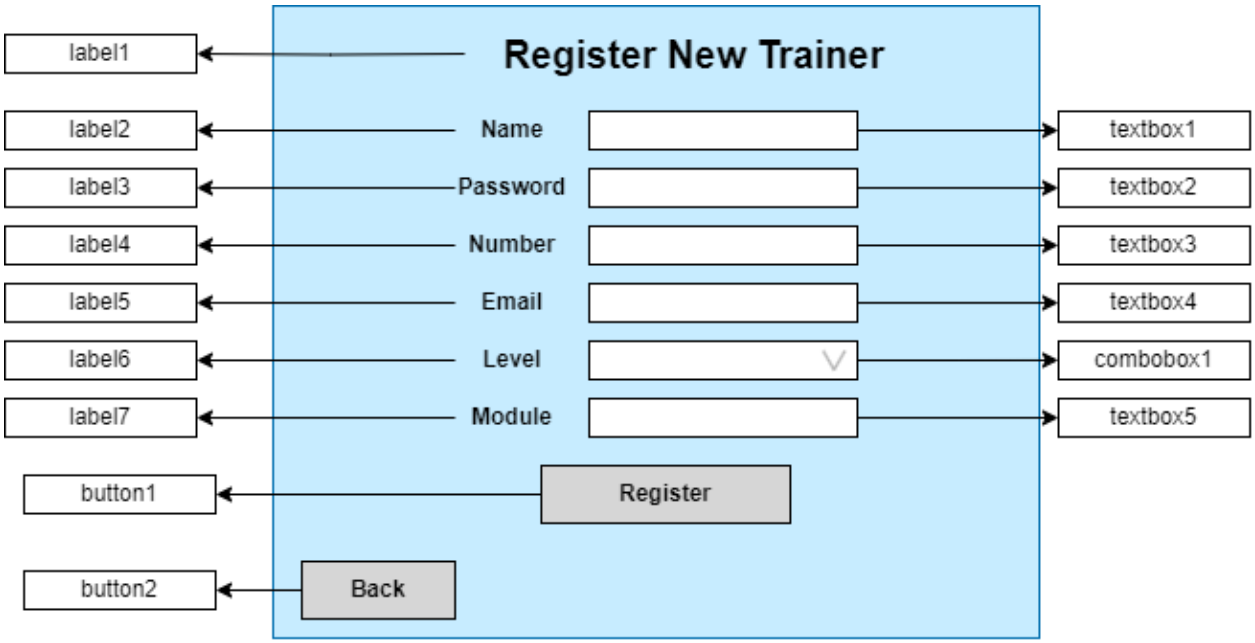
Control	Control Name	Description
label1	lbl_ProfileUpdate	To label form header
label2	lbl_AdminID	To label the related controls
label3	lbl_Name	
label4	lbl_Number	
label5	lbl_Email	
textbox1	txt_AdminID	To Display the Admin ID based on the Login Details

textbox2	txt_Name	To allow user to enter new name
textbox3	txt_Number	To allow user to enter new Number
textbox4	txt_Email	To allow user to enter new Email
button1	btn_Update	To allow user to update their details in the database
button2	btn_Back	To take the user back to the Options form

User Interface for Profile Update



Storyboard for Trainer Registration

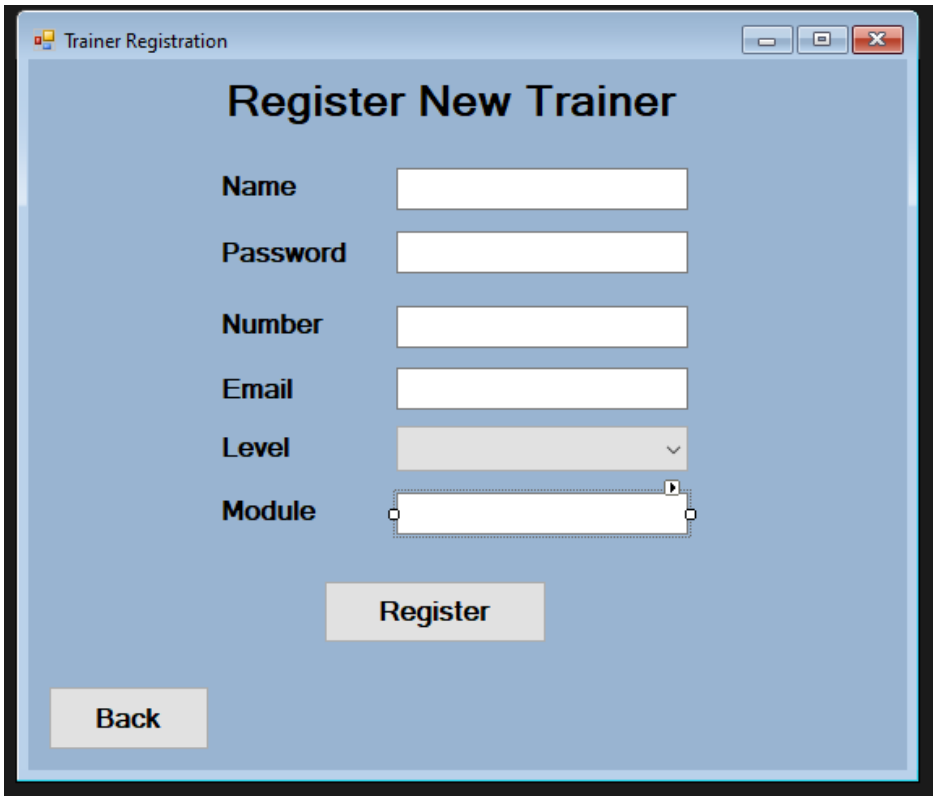


Storyboard Description

Control	Control Name	Description
label1	lbl_NewTrainer	To label form header
label2	lbl_AdminID	To label the related controls
label3	lbl_Name	
label4	lbl_Number	
label5	lbl_Email	
label6	lbl_Level	
label7	lbl_Module	
textbox1	txt_Name	To allow admin to enter the trainer's Name

textbox2	txt_Password	To allow admin to enter the trainer's Password
textbox3	txt_Number	To allow admin to enter new trainer's Number
textbox4	txt_Email	To allow admin to enter the trainer's Email
combobox1	cmb_Level	To allow admin to select the trainer's Level from three options
textbox5	txt_Module	To allow admin to enter the trainer's Module
button1	btn_Register	To allow admin to register the details of the trainer in the database
button2	btn_Back	To take the user back to the Options form

User Interface for Trainer Registration

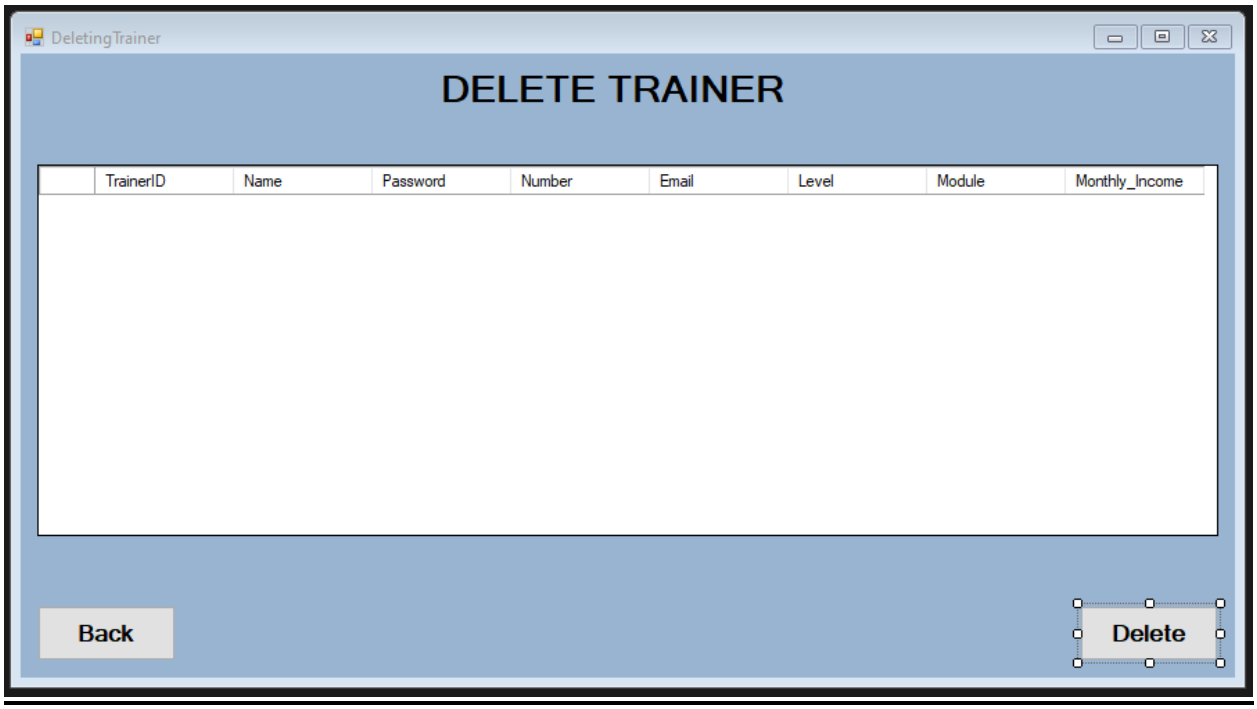


Storyboard for Deleting Trainer

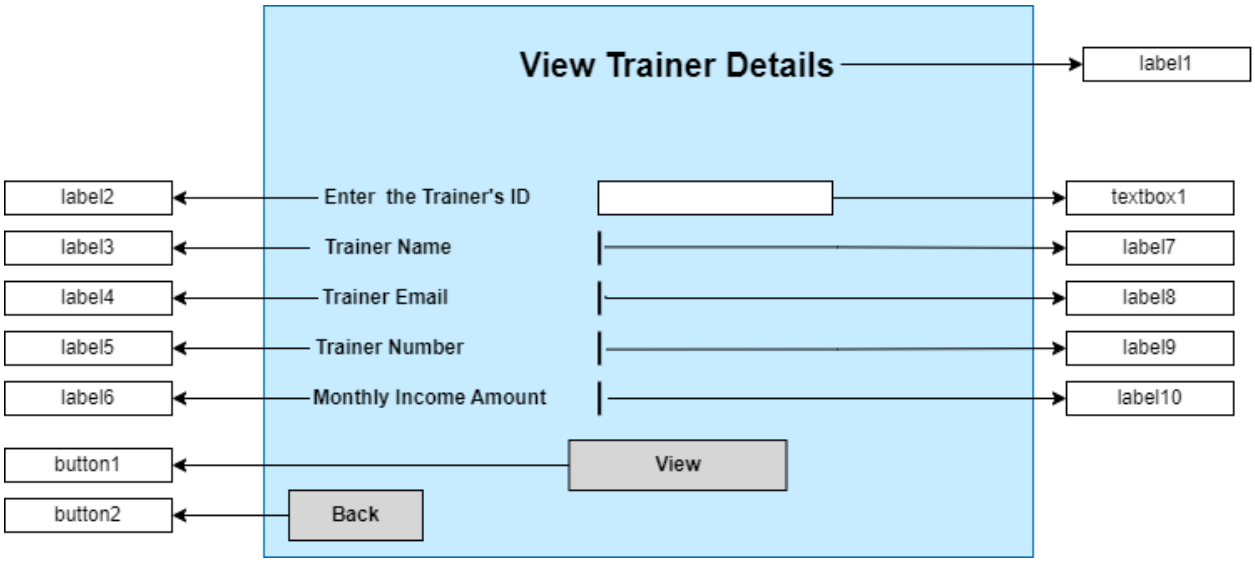
Storyboard Description

Control	Control Name	Description
label1	lbl_DeleteTrainer	To label form header
dataGridView1	grd_Trainers	To display the trainers and their information and allow the admin to select the.
button1	btn_Back	To allow the admin to go back to options form.
button2	btn_Delete	To allow the admin to delete the selected trainer from the database.

User Interface for Deleting Trainer



Storyboard for View Trainer Details

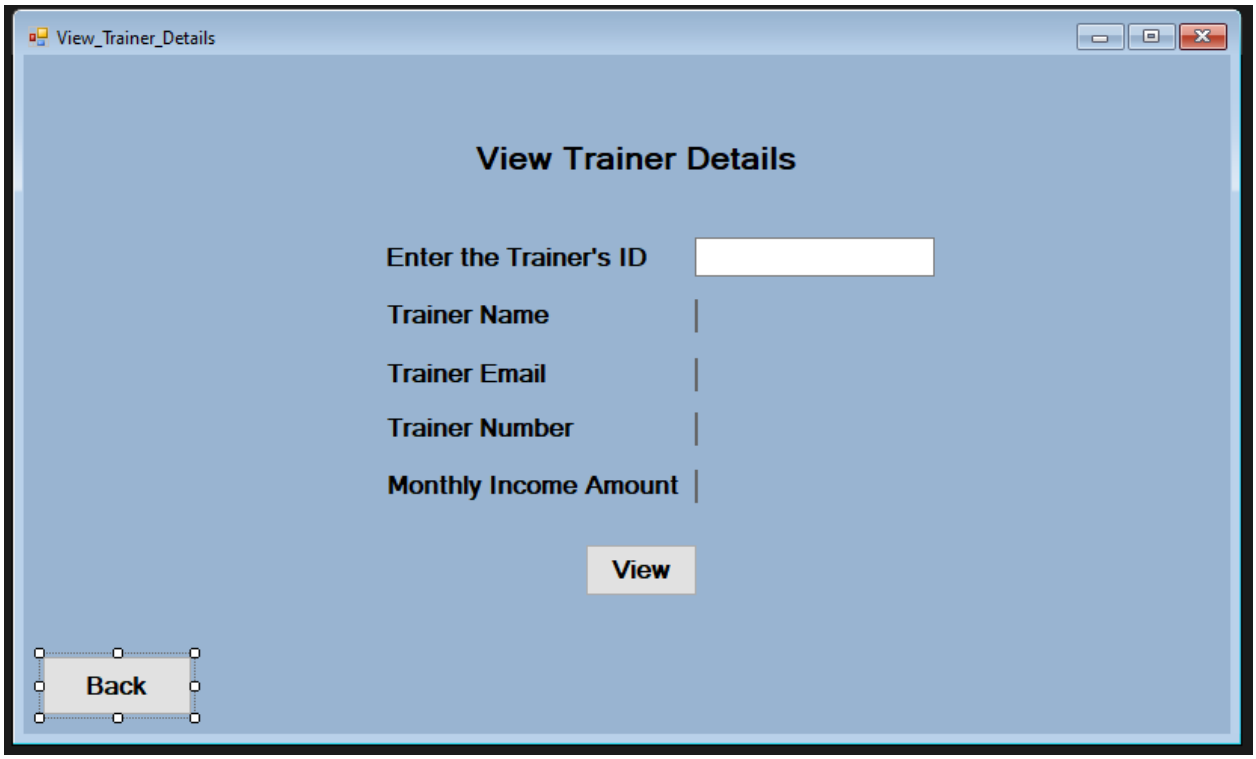


Storyboard Description

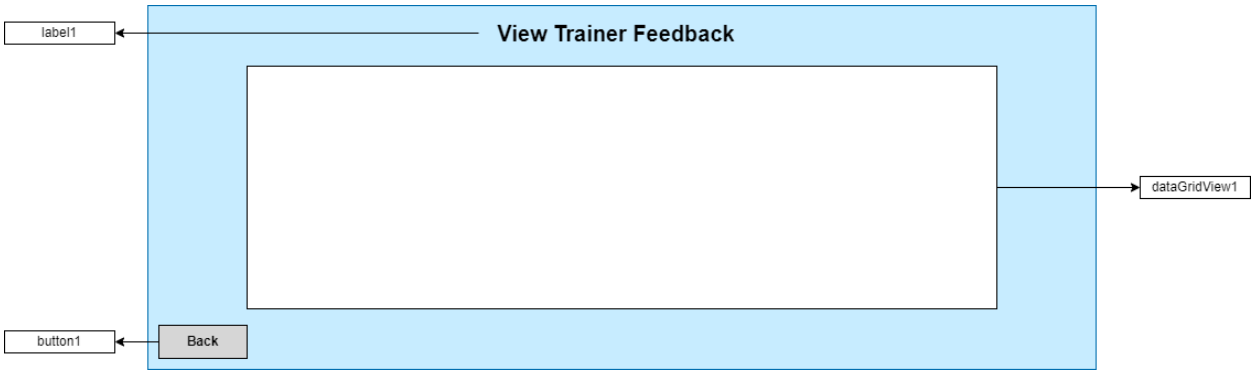
Control	Control Name	Description
label1	lbl_TrainerDetails	To label form header
label2	lbl_TrainerID	To label the related controls
label3	lbl_TrainerName	
label4	lbl_TrainerEmail	
label5	lbl_TrainerNumber	
label6	lbl_MonthlyIncomeAmount	
label7	lbl_TrainerNameDisp	To Display the trainer's name
label8	lbl_TrainerEmailDisp	To Display the trainer's email

label9	lbl_TrianerNumberDisp	To Display the trainer's number
label10	lbl_MonthlyIncomeAmountDisp	To Display the trainer's Monthly income amount.
textbox1	txt_TrainerID	To allow admin to enter the Trianer's ID which will be used to retrieve data from the database accordingly.
button1	btn_View	To allow admin to view the data about a specific trainer that is stored in the database.
button2	btn_Back	To allow the admin to go back to options form.

User Interface for View Trainer Details



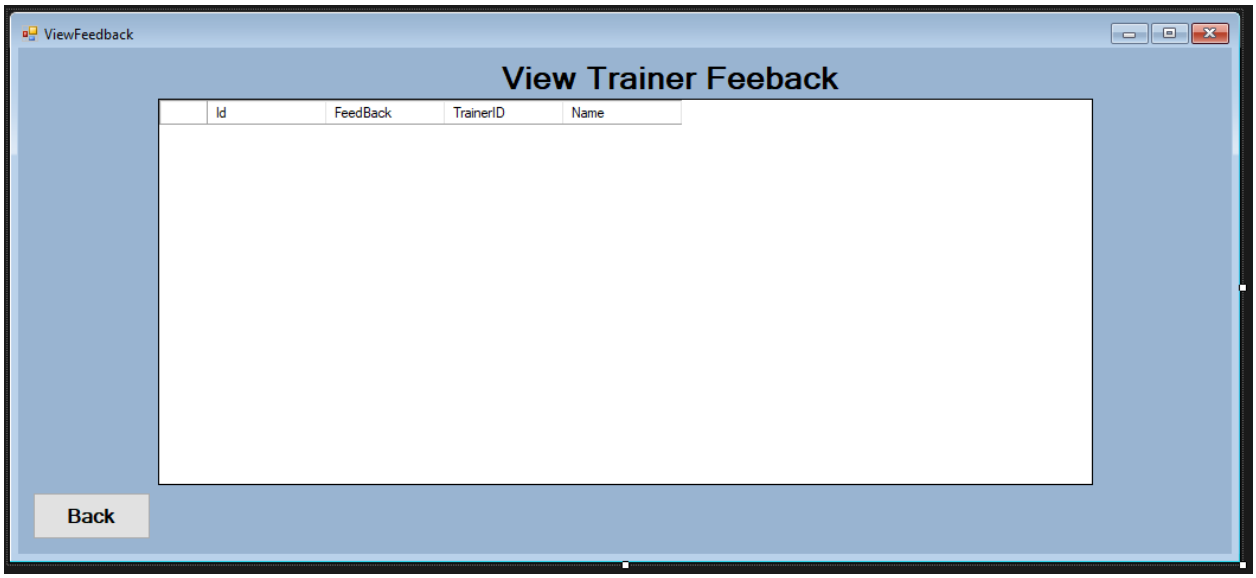
Storyboard for View Trainer Feedback



Storyboard Description

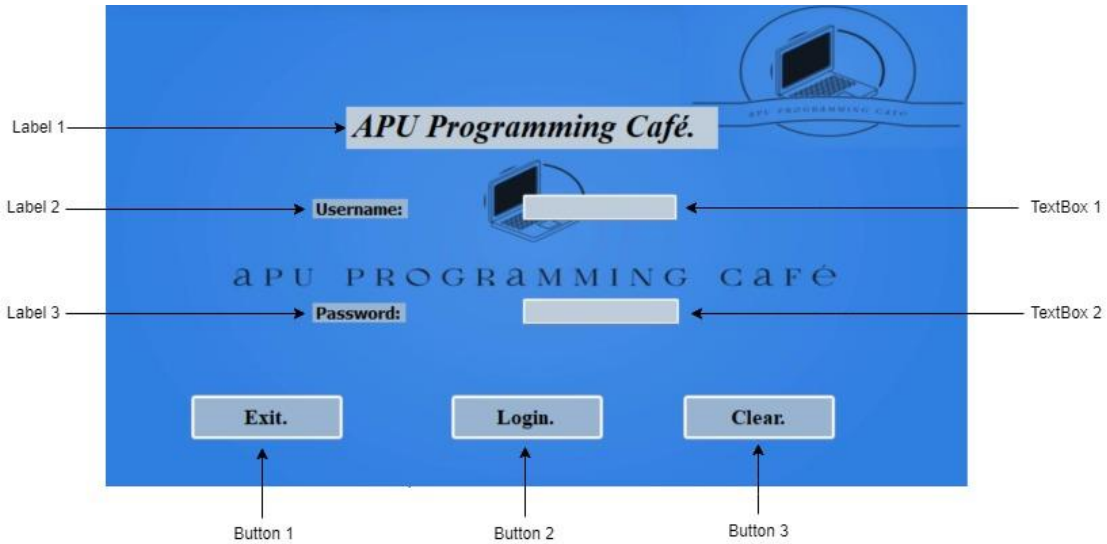
Control	Control Name	Description
label1	lbl_ViewTrainerFeedback	To label form header
dataGridView1	dataGridView1	To display the trainer's name, ID and the Feedback given by the trainers.
button1	btn_Back	To allow the admin to go back to options form.

User Interface for View Trainer Feedback



1.2 Trainer (Abdulrahman Gamil Mohammed Ahmed TP07012)

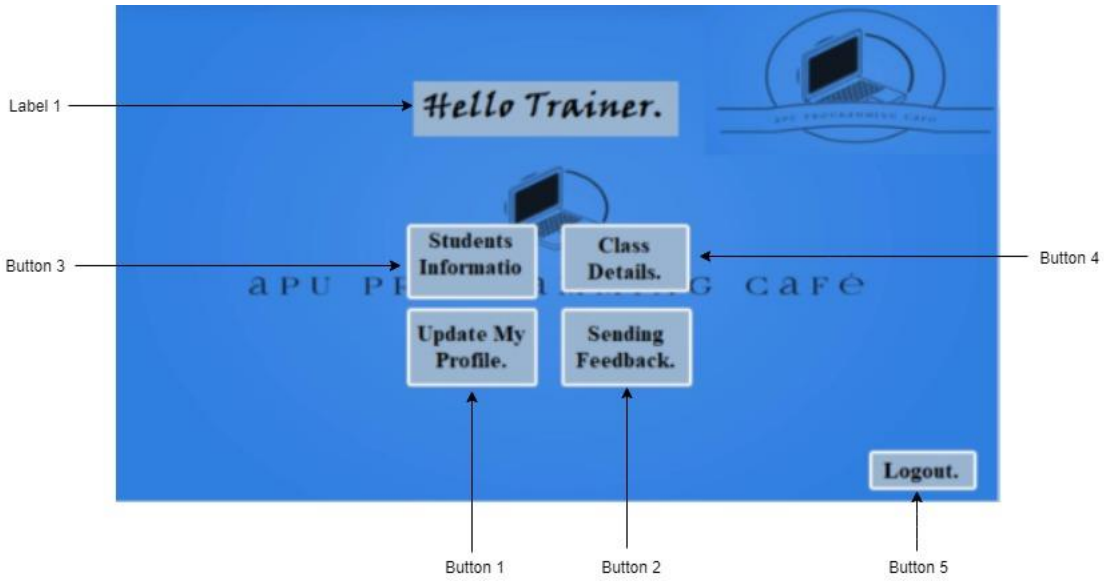
Login Form.



Storyboard Description

Control	Control Name	Description
Label 1	lbl_Hello	Welcoming the user.
Label 2	lbl_Username	Inform the user to write a username in the front textbox.
Label 3	lbl_Password	Inform the user to write the password in the front textbox.
Button 1	btn_Exit	Close the application.
Button 2	btn_Login	Make the user access the application.
Button 3	btn_Clear	Clear the text boxes.
TextBox 1	txtBox_Username	To take the username and connected it with the database.
TextBox 2	txtBox_Password	To take the password and connected with the database

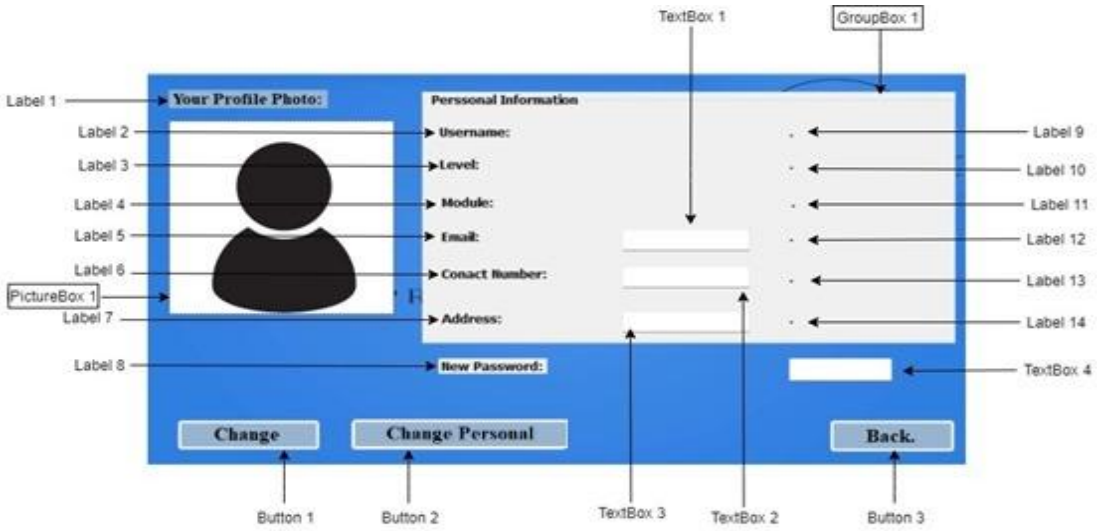
Main Page Form.



Storyboard Description

Control	Control Name	Description
Label 1	lbl_Hello	Welcoming the trainer.
Button 1	btn_Update_My_Profile	Transfer the trainer to the updateb my profile form.
Button 2	btn_Send_Feedback	Transfer the trainer to sending feedback form.
Button 3	btn_Students_information	Transfer the trainer to students information form.
Button 4	btn_Class_D	Transfer the trainer to the Class details form.
Button 5	btn_Logout	Transfer the trainer to the login page.

Update Profile Form.



Storyboard Description

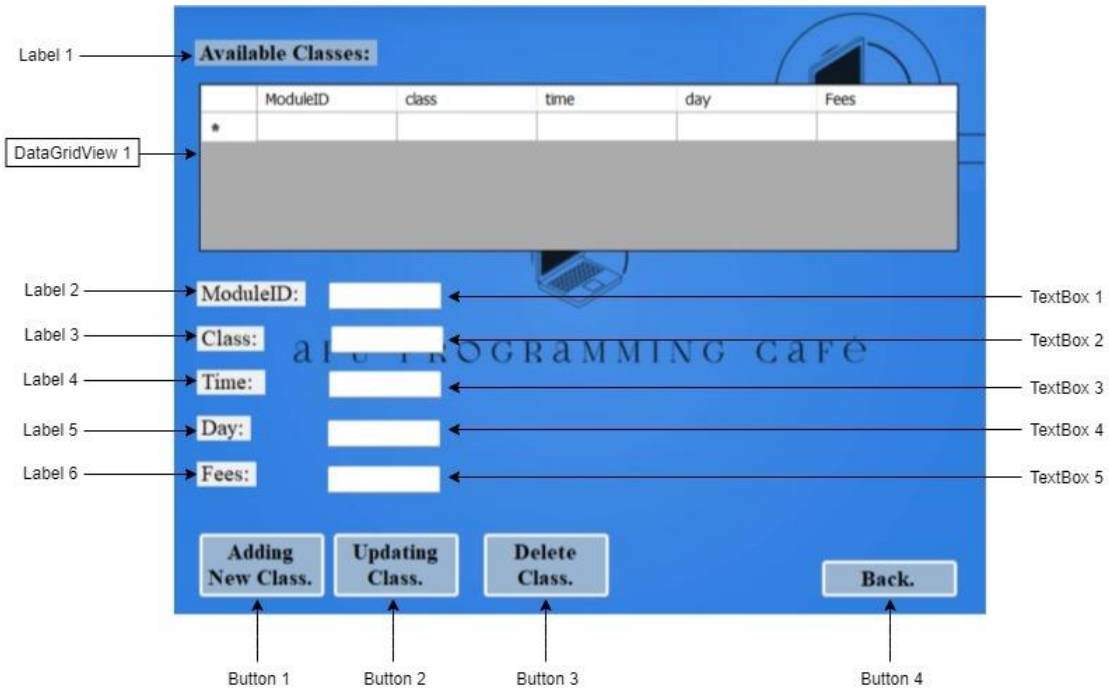
Control	Control Name	Description
Label 1	lbl_Photo_Profile	To illustrate that the following photo is for the trainer.
Label 2	lbl_show_Username	Inform the trainer what the front label will contain.
Label 3	lbl_Level_Show	Inform the trainer what the front label will contain.
Label 4	lbl_Module_Show	Inform the trainer what the front label will contain.

Label 5	lbl_Show_Email	Inform the trainer what the front label will contain.
Label 6	lbl_ContactNumber_Show	Inform the trainer what the front label will contain.
Label 7	lbl_Address_Show	Inform the trainer what the front label will contain.
Label 8	lbl_New_Password	Inform the trainer what the front label will contain.
Label 9	lbl_Name_	Show the username of the trainer from the database.
Label 10	lbl_Level	Show the level of the trainer from the database.
Label 11	lbl_Module	Show the module of the trainer from the database.
Label 12	lbl_Email	Show the email of the trainer from the database.

Label 13	lbl_ContactNumber	Show the contact number of the trainer from the database.
Label 14	lbl_Address	Show the address of the trainer from the database.
Button 1	btn_Change_Password	Change only the password of the trainer depending on the new password on the textbox.
Button 2	btn_Chamge_Perssonal	Change the email and the address and the number of the trainer depending on the three textboxes.
Button 3	btn_Back	Show the main page.
Group Box1	groupBox_Perssonal_Information	Contain all the personal information of the trainer.
PictureBox1	pictureBox_Profile	The photo of the trainer.
TextBox 1	txtBox_Email	The new email that the trainer wanted to be in his profile.

TextBox 2	txtBox_Address	The new address that the trainer wanted to be in his profile.
TextBox 3	txtBox_ContactNumber	The new contact number that the trainer wanted to be in his profile.
TextBox 4	txtBox_Password	The new password that the user wants to update it.

Classes Details Form.

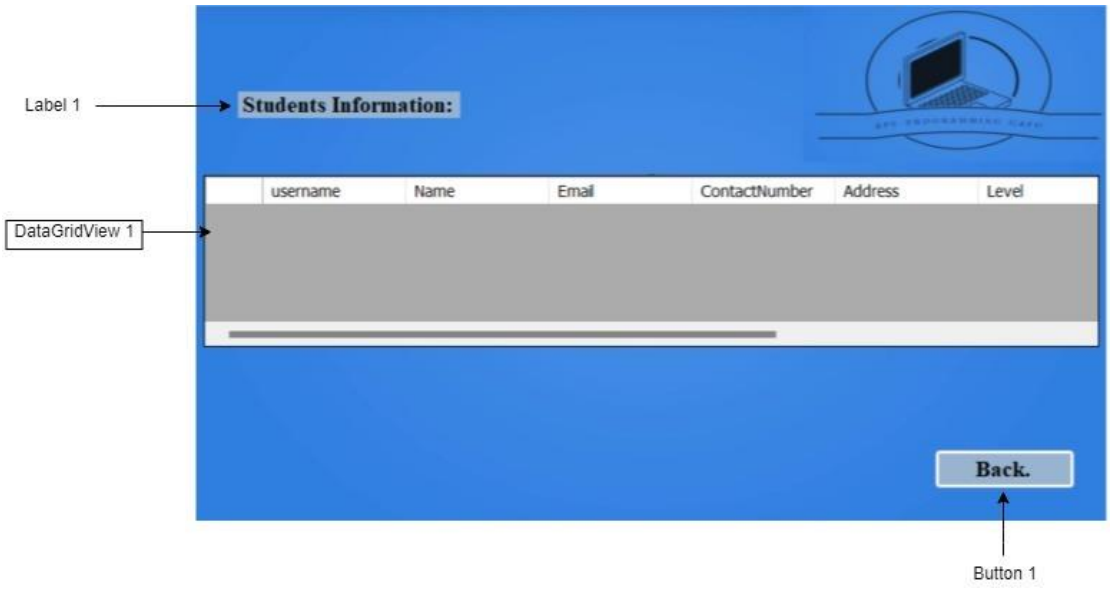


Storyboard Description

Control	Control Name	Description
Label 1	lbl_Classes	Inform the trainer that the following data grid view displays all the classes.
Label 2	lbl_ModuleID	Tell the trainer to write the ModuleID in the front textbox.
Label 3	lbl_Class	Tell the trainer to write the Class in the front textbox
Label 4	lbl_Time	Tell the trainer to write the Time in the front textbox

Label 5	lbl_Day	Tell the trainer to write the Day in the front textbox
Label 6	lbl_Fees	Tell the trainer to write the Fees in the front textbox
Button 1	btn_Adding_New_Class	Add new class,
Button 2	btn_Update_Class	Update the class details based on the ModuleID.
Button 3	btn_Delete_Class	Delete the class based on the ModuleID.
Button 4	btn_Back	Back to the main page.
DataGridView 1	dataGridView_Class	Show all the classes that are stored in the class table in the database.
TextBox 1	txtBox_ModuleID	
TextBox 2	txtBox_Class	Take input (Class) from the trainer.
TextBox 3	txtBox_Time	Take the proper (Time) input from the trainer.
TextBox 4	txtBox_Day	Take input (Day) from the trainer.
TextBox 5	txtBox_Fees	Take input from the trainer, that illustrates the fees.

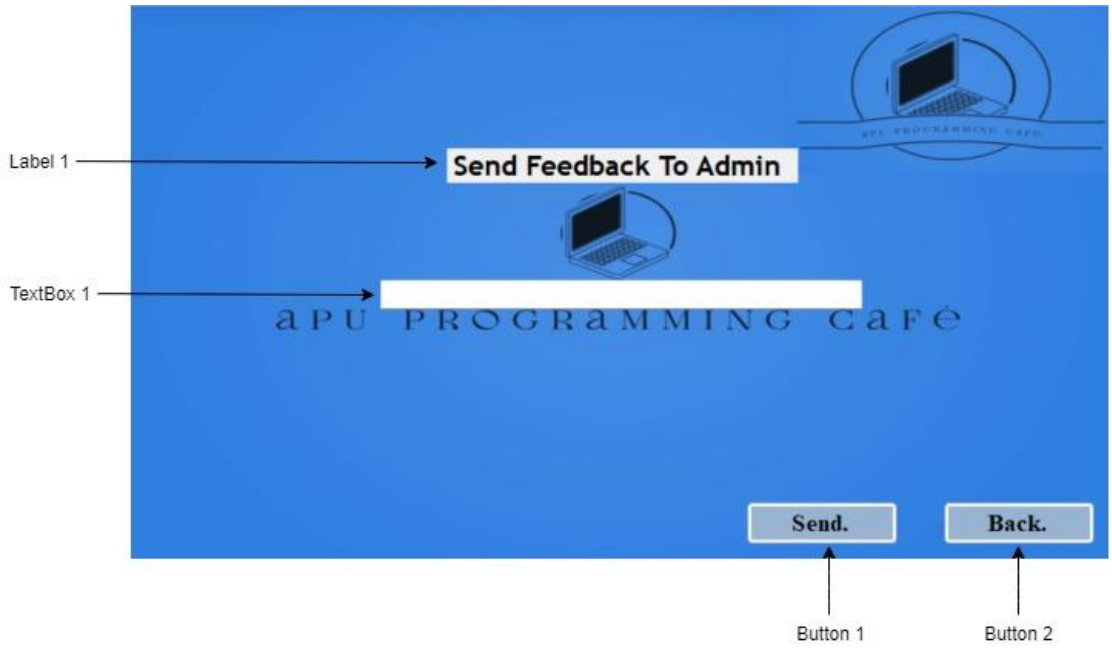
Students Information Form.



Storyboard Description

Control	Control Name	Description
Label 1	lbl_Student_Info	Inform the trainer that the following data grid view will show the students' information.
DataGridView 1	dataGridView_Students_Information	Show the entire data from the table of the students that is stored in the database.
Button 1	btn_Back	Back the trainer to the main page.

Send Feedback Form.

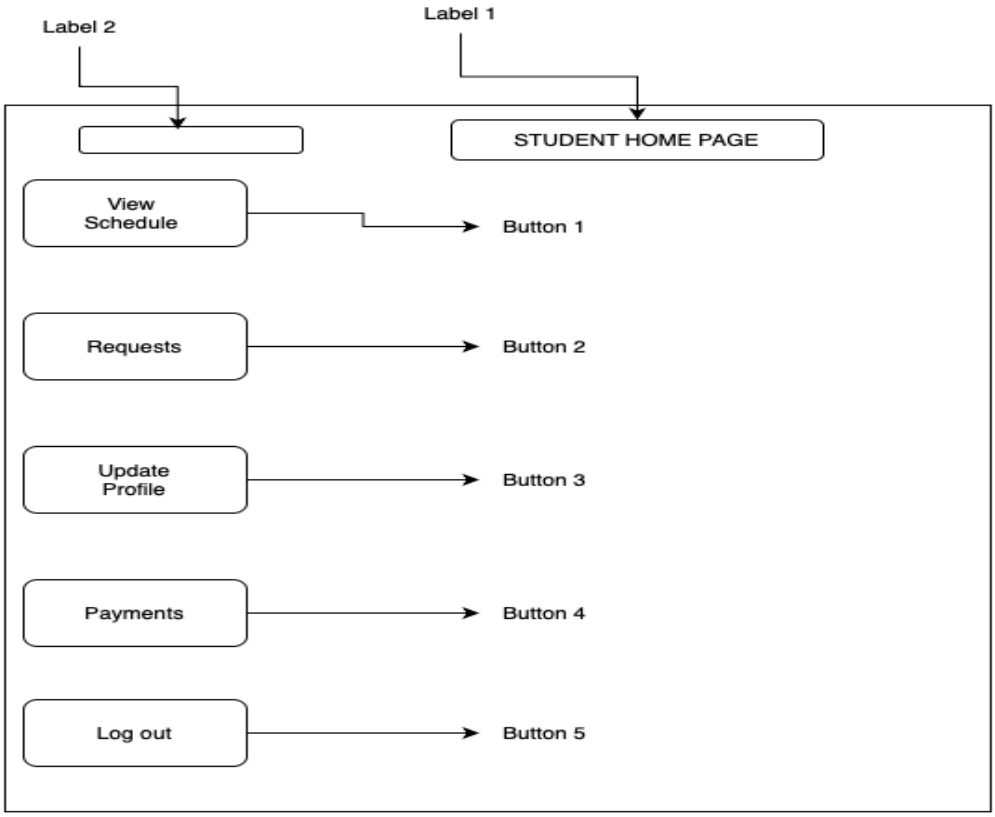


Storyboard Description

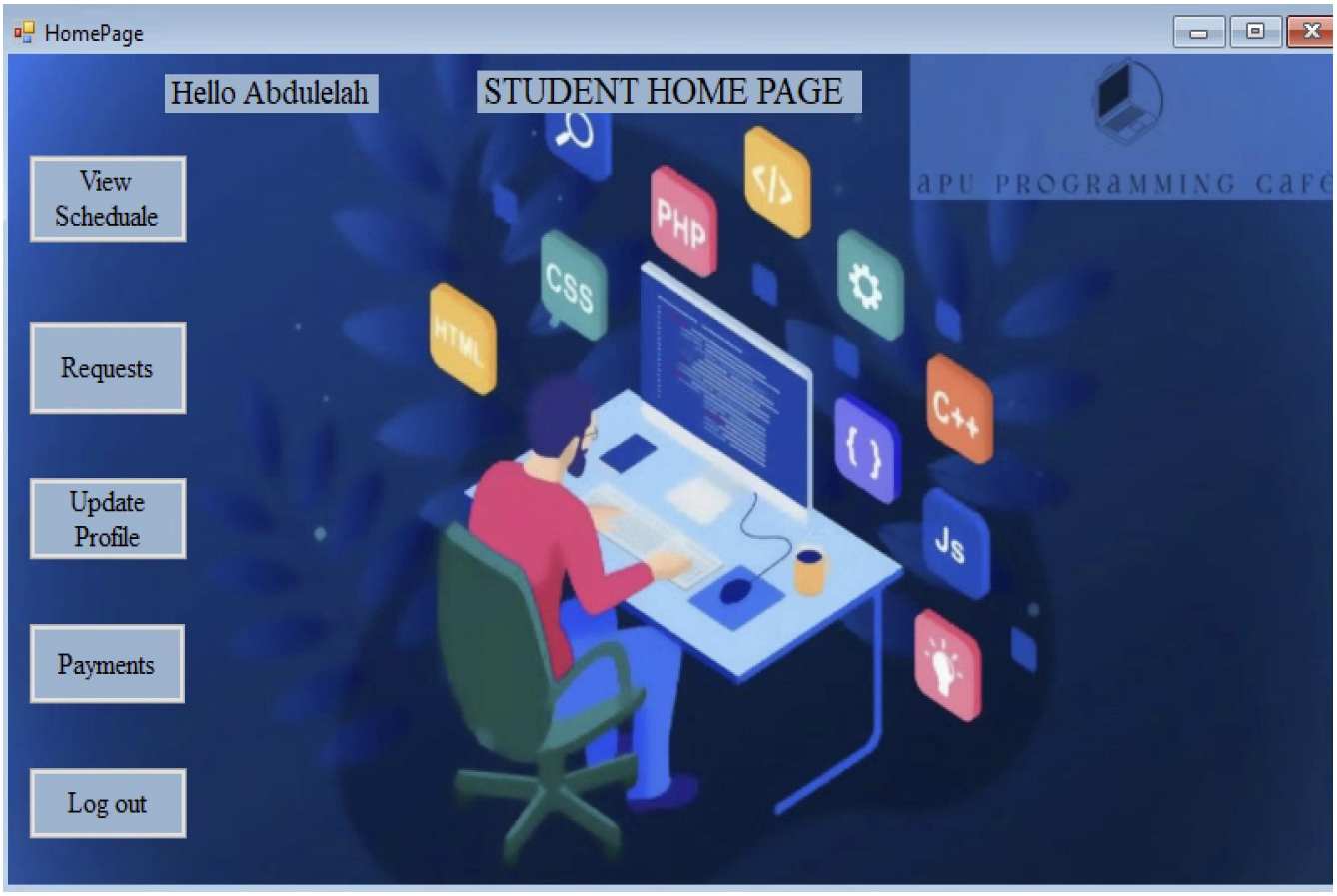
Control	Control Name	Description
Label 1	lbl_Send_Feedback	Inform the trainer that the text in the textbox will send to the admin.
TextBox 1	txtBox_Feedback	Write the feedback to the admin.
Button 1	btn_Send	Insert the text in the textbox into the database so that the admin can see it.
Button 2	btn_Back	Back to the main page.

1.3 Student (Abdulelah Abdulrehman Al-Kaf TP069319)

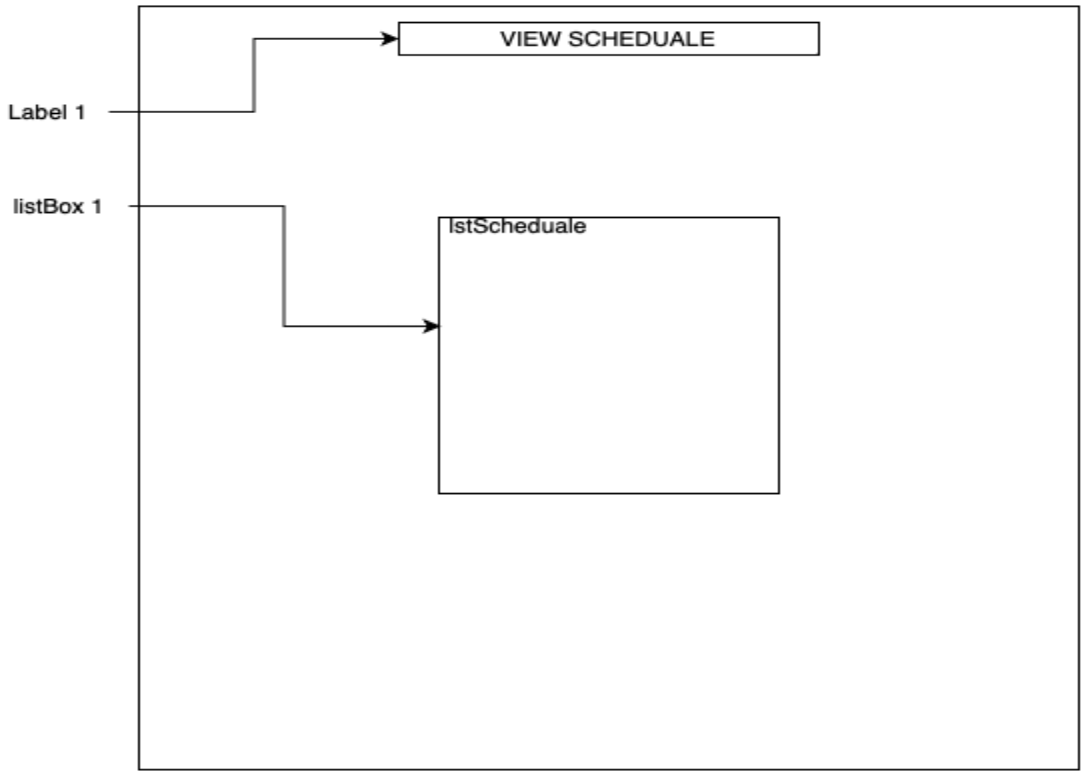
Homepage



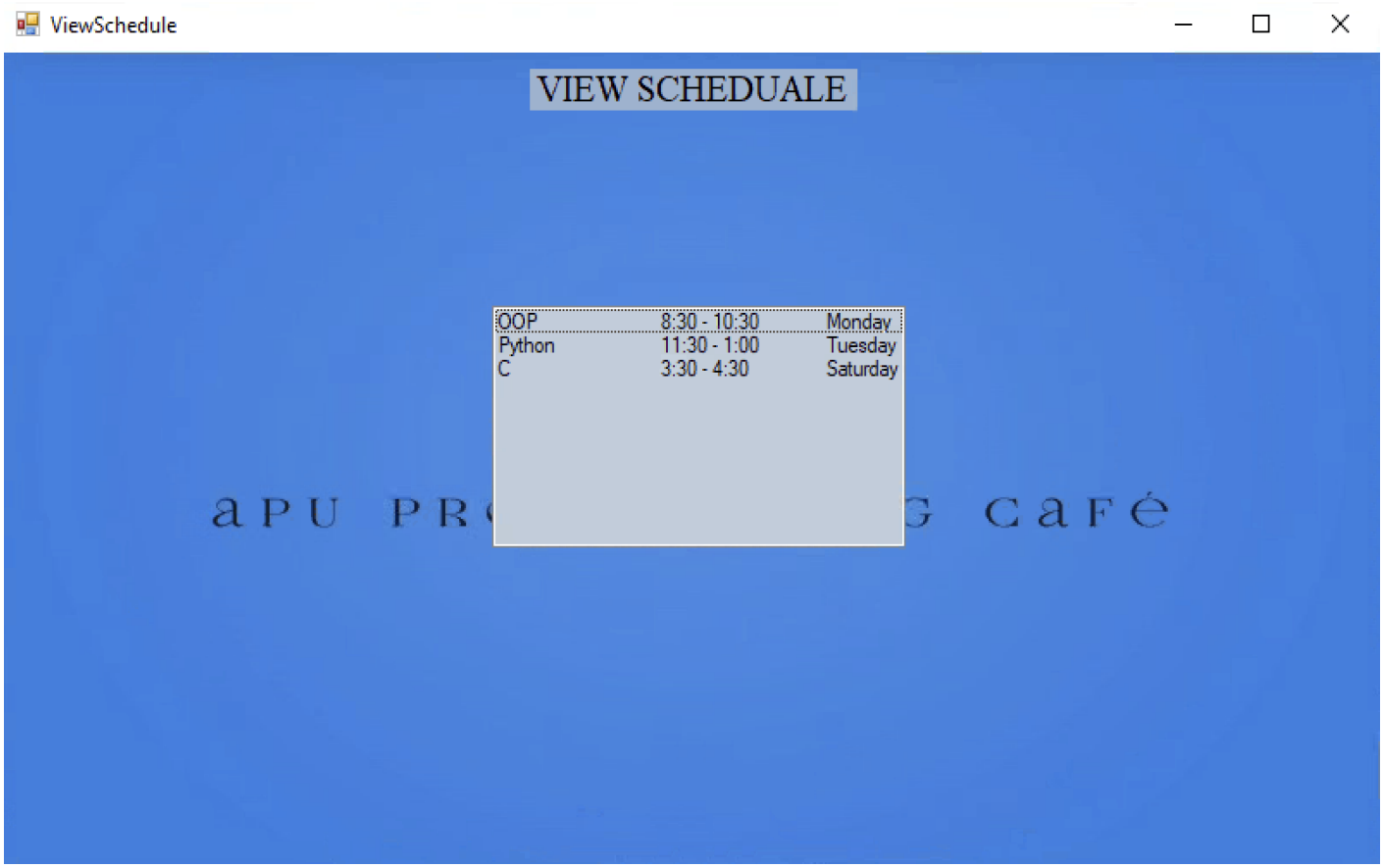
Control	Control Name	Description
Label 2	lblStudentID	After login it will appear student name
Button 1	btnViewScheduale	To open view schedule page
Button 2	btnRequests	To open view requests page
Button 3	Update Profile	To open update profile page
Button 4	Payments	To open payment page
Button 5	Log out	To go back to login page


HomePage
Hello Abdulelah
STUDENT HOME PAGE
View Scheduale
Requests
Update Profile
Payments
Log out
PHP
CSS
HTML
C++
Js

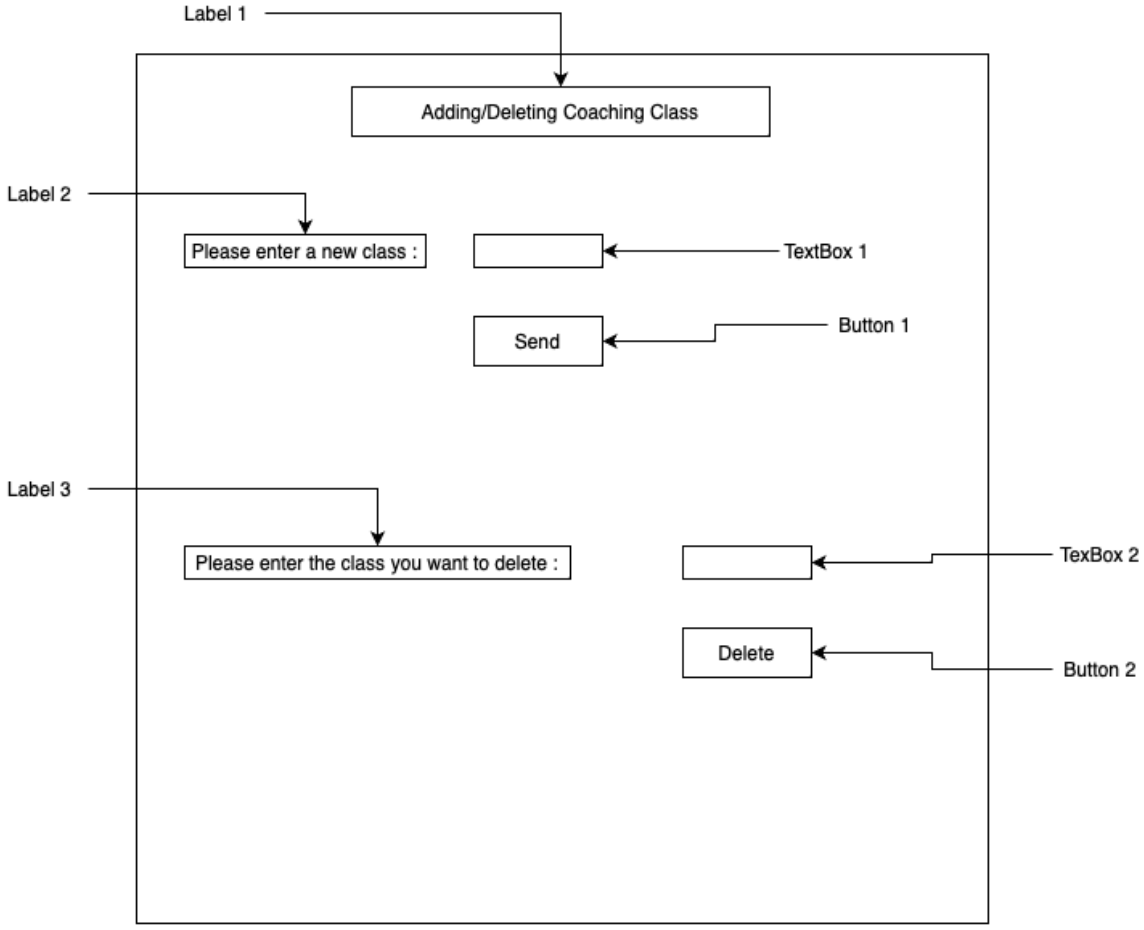
View Schedule:



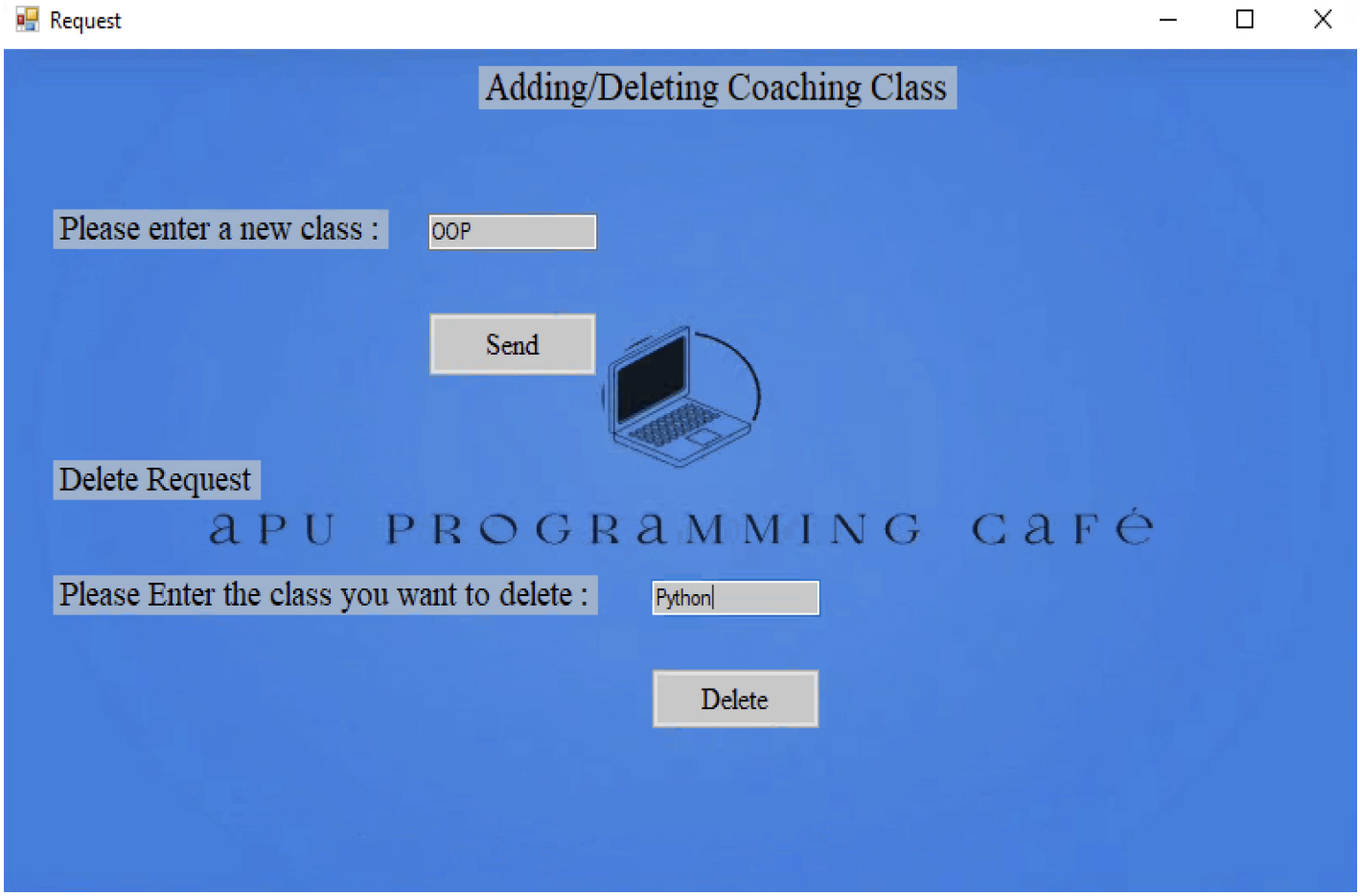
Control	Control Name	Description
Listbox 1	lstScheduale	After the student press view schedule button it will appear his schedule

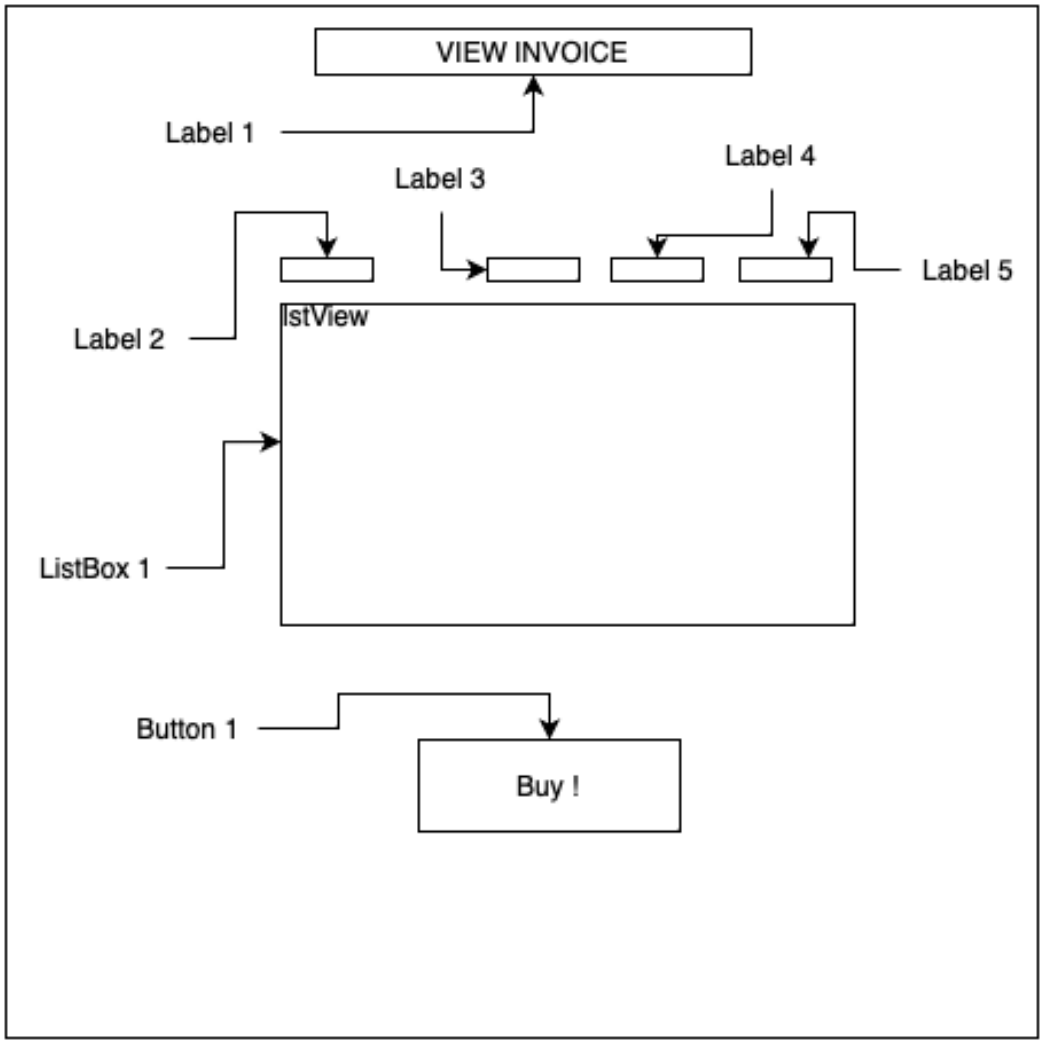

ViewSchedule
VIEW SCHEDUALE
OOP 8:30 - 10:30 Monday
Python 11:30 - 1:00 Tuesday
C 3:30 - 4:30 Saturday
aPU PR
G CaFé

Adding/Deleting Coaches Class

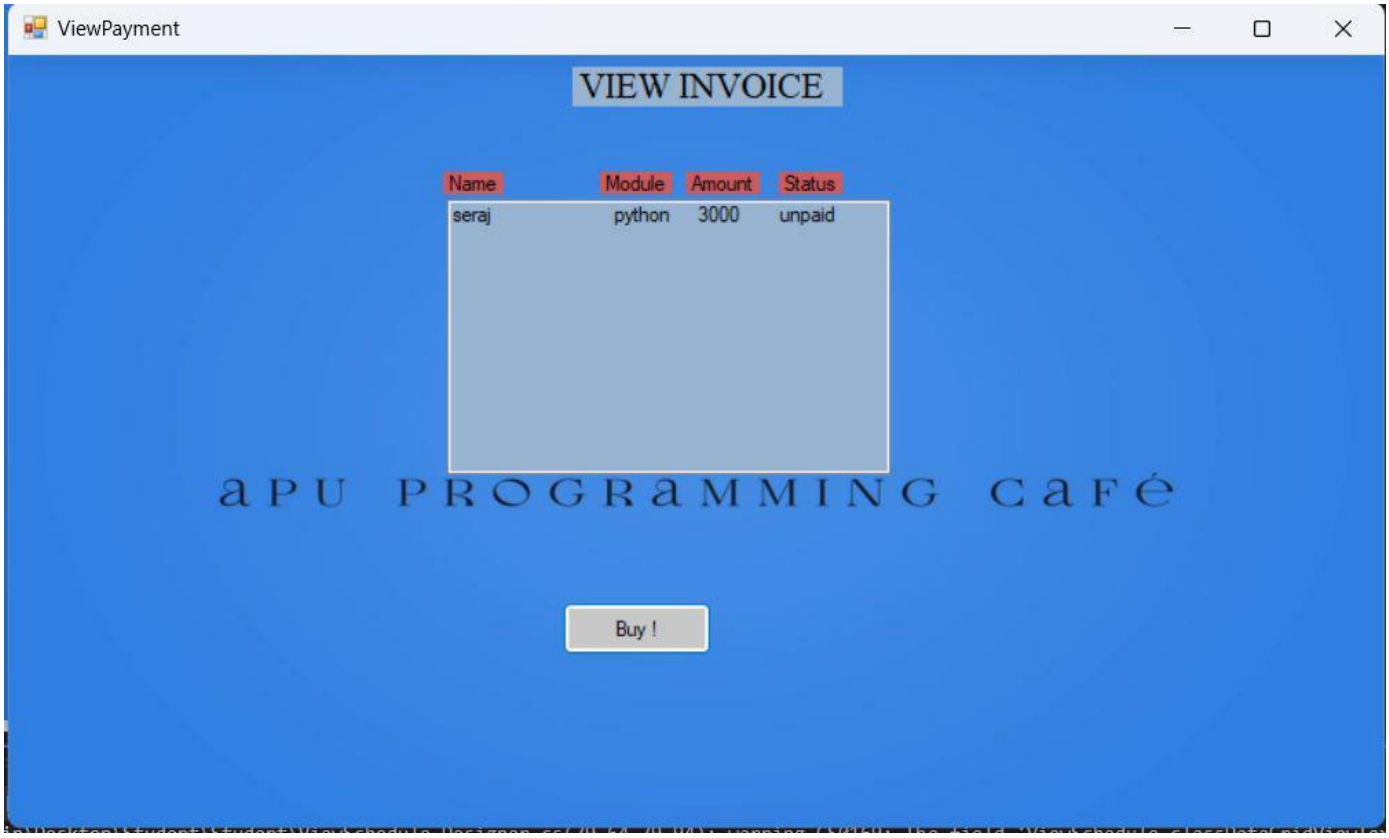


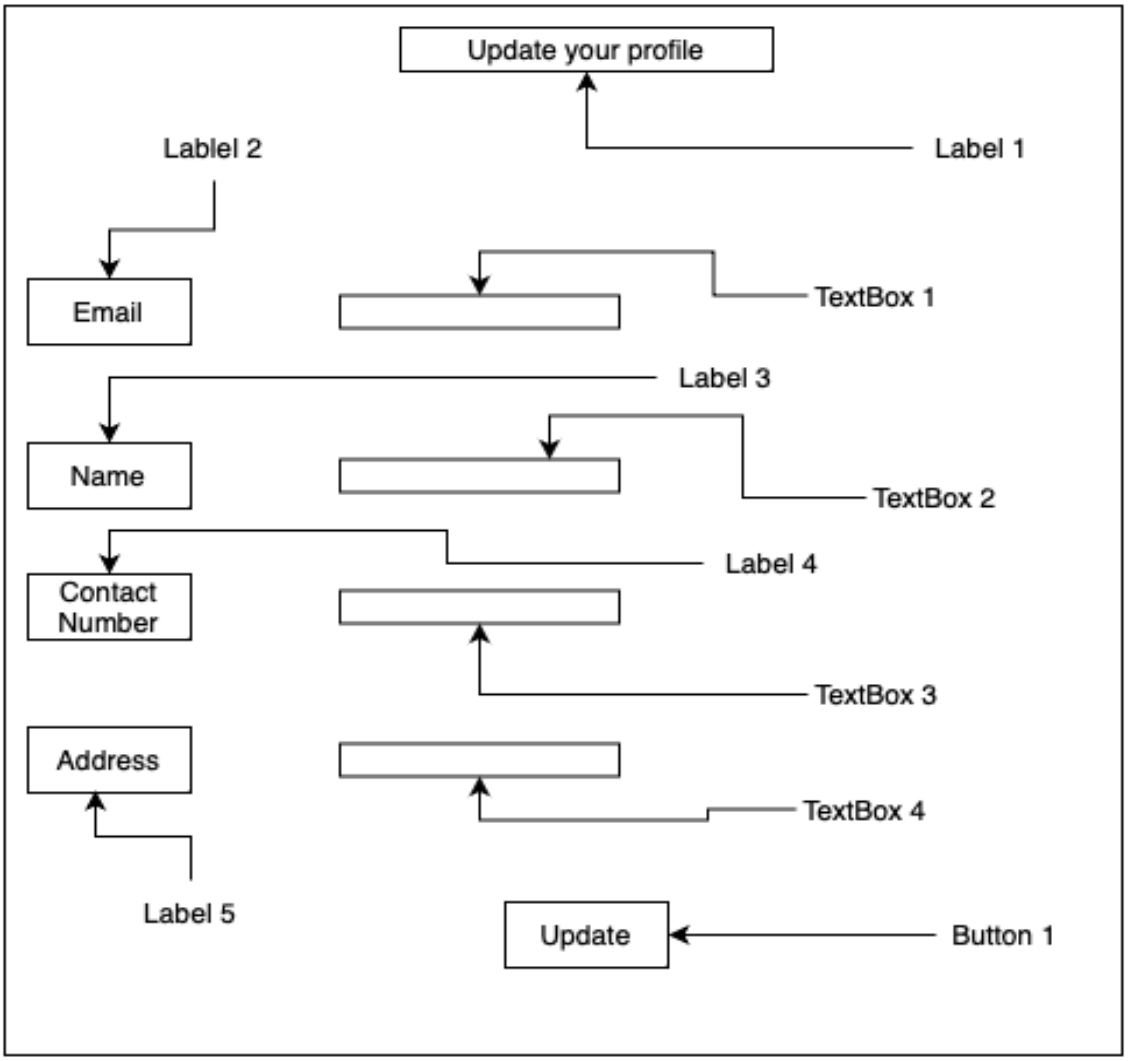
Control	**Control Name**	**Description**
TextBox 1	txtNew	To allow user to enter a new class
TextBox 2	txtDelete	To allow user to enter a class
Button 1	btnSend	To allow user to send the class entered
Button 2	btnDelete	To allow user to delete the class entered


Request
Adding/Deleting Coaching Class
Please enter a new class :
OOP
Send
Delete Request
aPU PROGRaMMING CaFé
Please Enter the class you want to delete :
Python
Delete



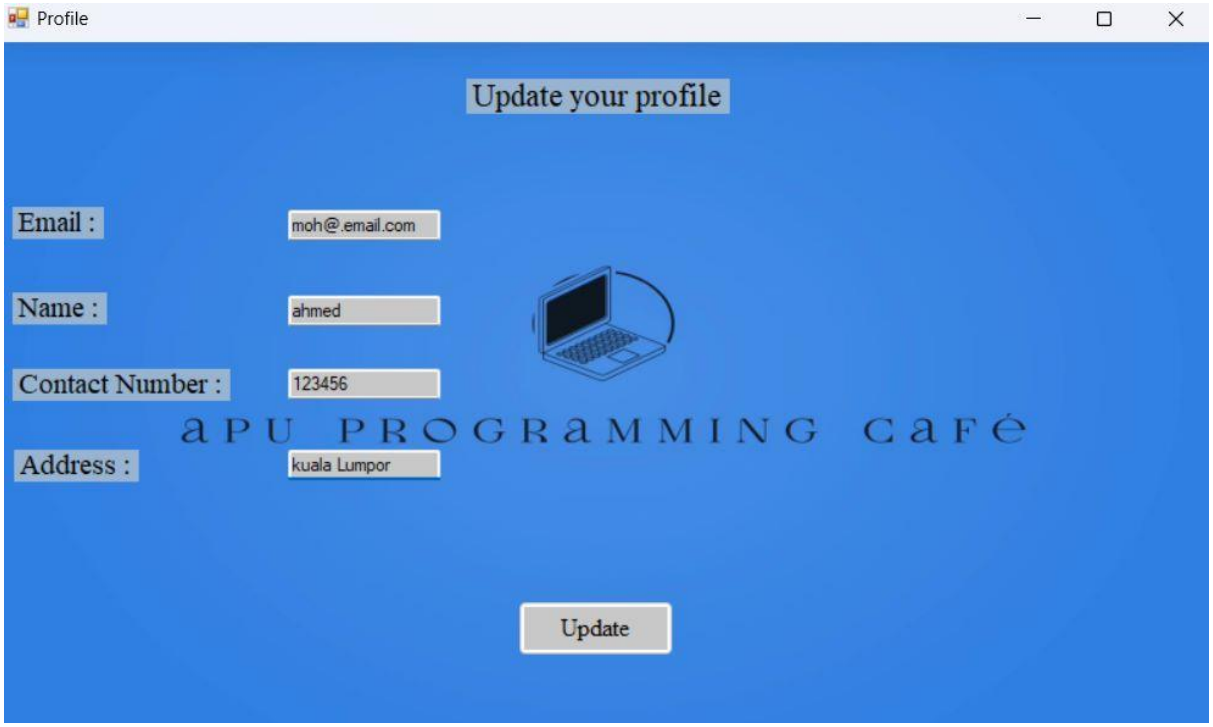
Control	**Control Name**	**Description**
ListBox 1	lstView	To View the invoice for the student
Button 1	btnBuy	To allow the student to proceed with payment


ViewPayment
VIEW INVOICE
Name
Module
Amount
Status
seraj
python
3000
unpaid
aPU PROGRaMMING CaFé
Buy !



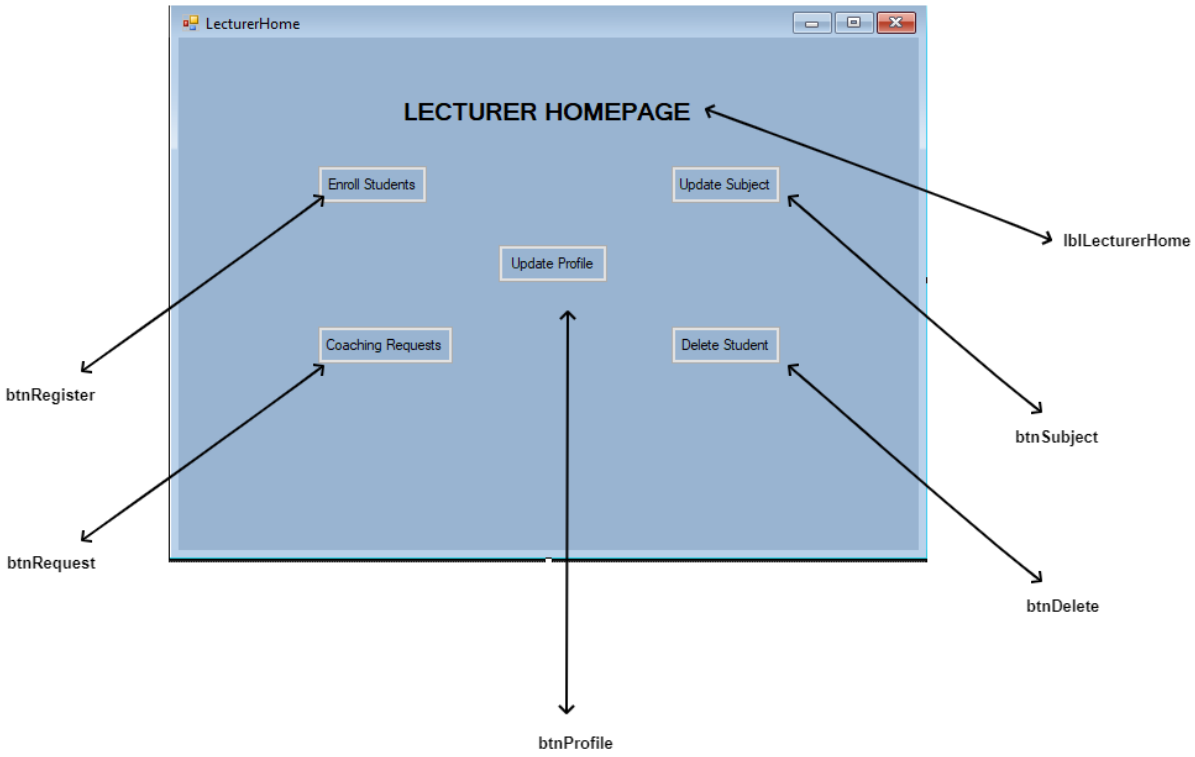
Control	**Control Name**	**Description**
TextBox 1	txtEmail	To allow user to enter new email
TextBox 2	txtName	To allow user to enter new name
TextBox 3	txtContact	To allow user to enter new contact number

TextBox 4	txtAddress	To allow user to enter new address
Button 1	btnUpdate	To allow student to update the new information



1.4 Lecturer (Amanullah Ghauri TP071215)

1. LECTURER_HOMEPAGE FORM:

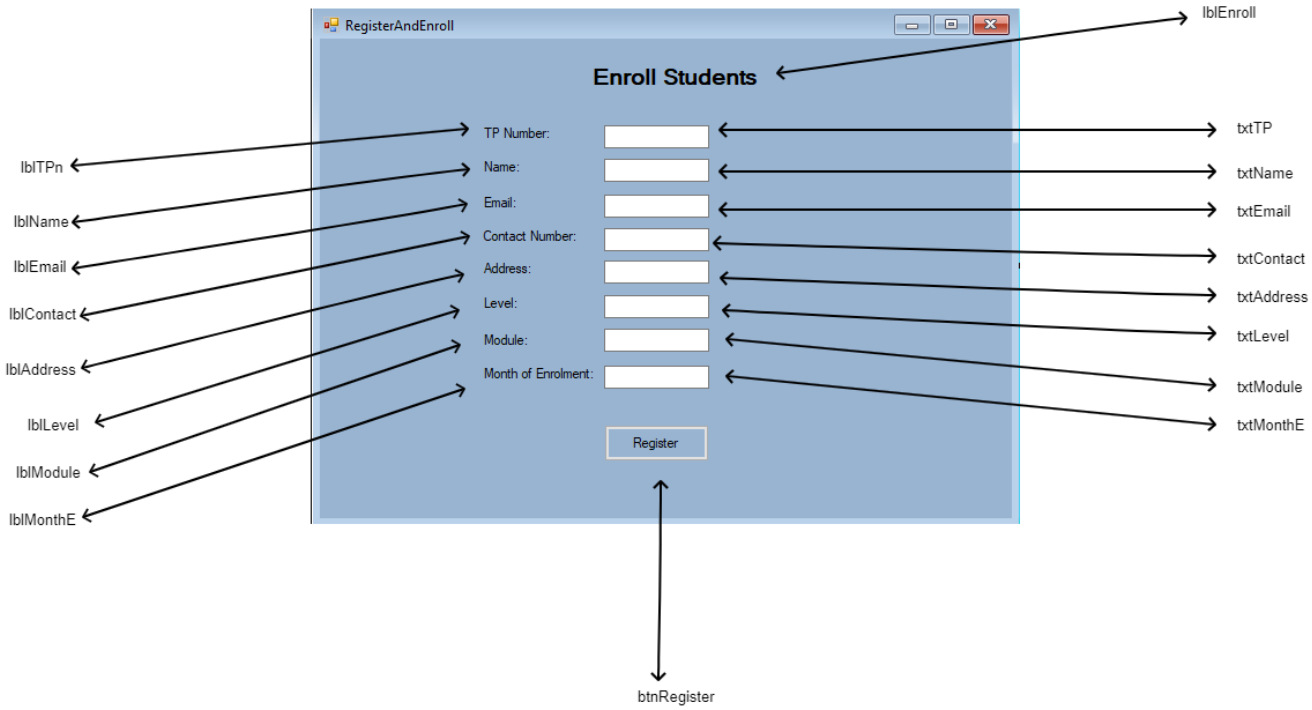


Storyboard Description

Control	Control Name	Description
Label 1	btnRegister	Takes the user to the lecturer page
Label 2	btnRequest	Takes the user to the Coaching Request page
Label 3	btnProfile	Takes the user to the Update Profile page

Label 4	btnDelete	Takes the user to the Delete Student page
Label 5	btnSubject	Takes the user to the Update Subject page
Label 6	lblLecturerHome	Displays "Lecturer Homepage" text

2. RegisterAndEnroll FORM:

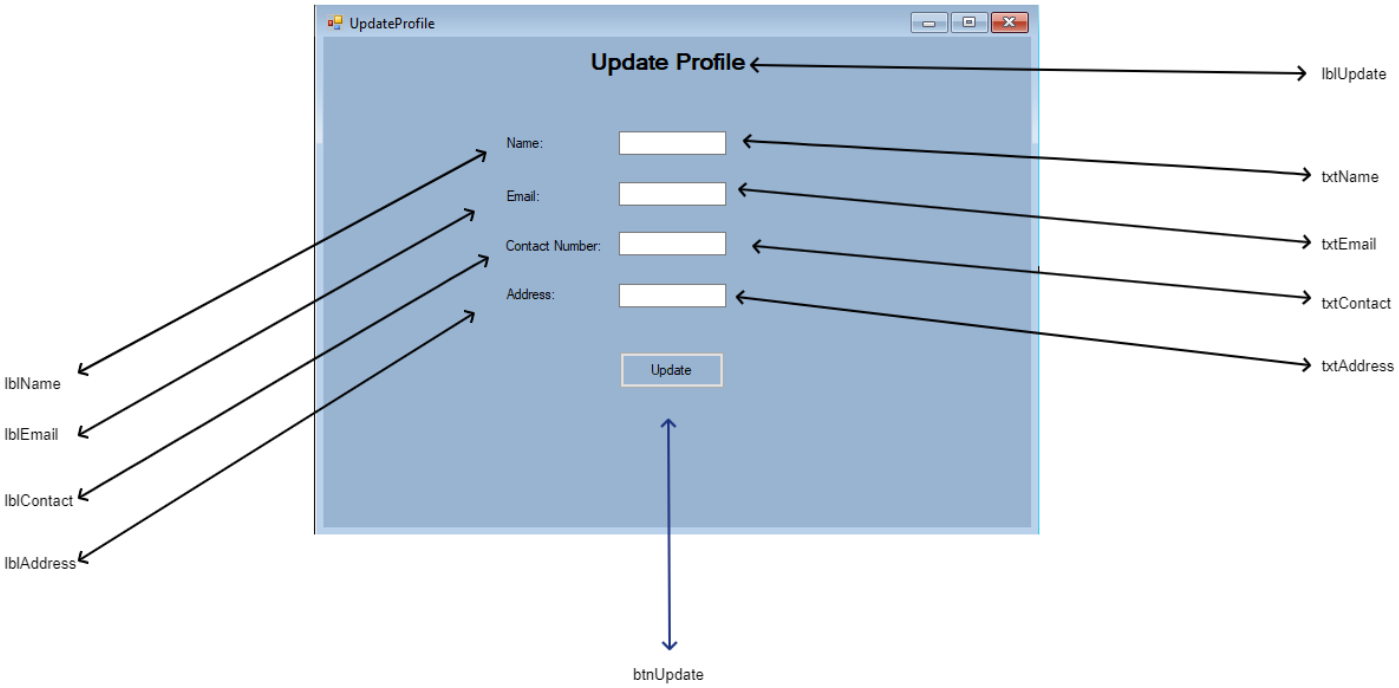


Storyboard Description

Control	Control Name	Description
Label 1	lblTPn	Displays "TP Number:" text
Label 2	lblName	Displays "Name:" text
Label 3	lblEmail	Displays "Email:" text
Label 4	lblContact	Displays "Contact Number:" text
Label 5	lblAddress	Displays "Address" text
Label 6	lblLevel	Displays "Level:" text
Label 7	lblModule	Displays "Module:" text
Label 8	lblMonthE	Displays "Month of Enrolment:" text
Label 9	btnRegister	Registers the student
Label 10	txtMonthE	Allows user to enter Month of Enrolment
Label 11	txtModule	Allows user to enter Module name
Label 12	txtLevel	Allows user to enter Level of student

Label 13	txtAddress	Allows user to enter Address of student
Label 14	txtContact	Allows user to enter Contact Number of Student
Label 15	txtEmail	Allows user to enter Email of Student
Label 16	txtName	Allows user to enter Name of student
Label 17	txtTP	Allows user to enter TP number of the student
Label 18	lblEnroll	Displays “Enroll Students” text

3. UpdateProfile FORM:

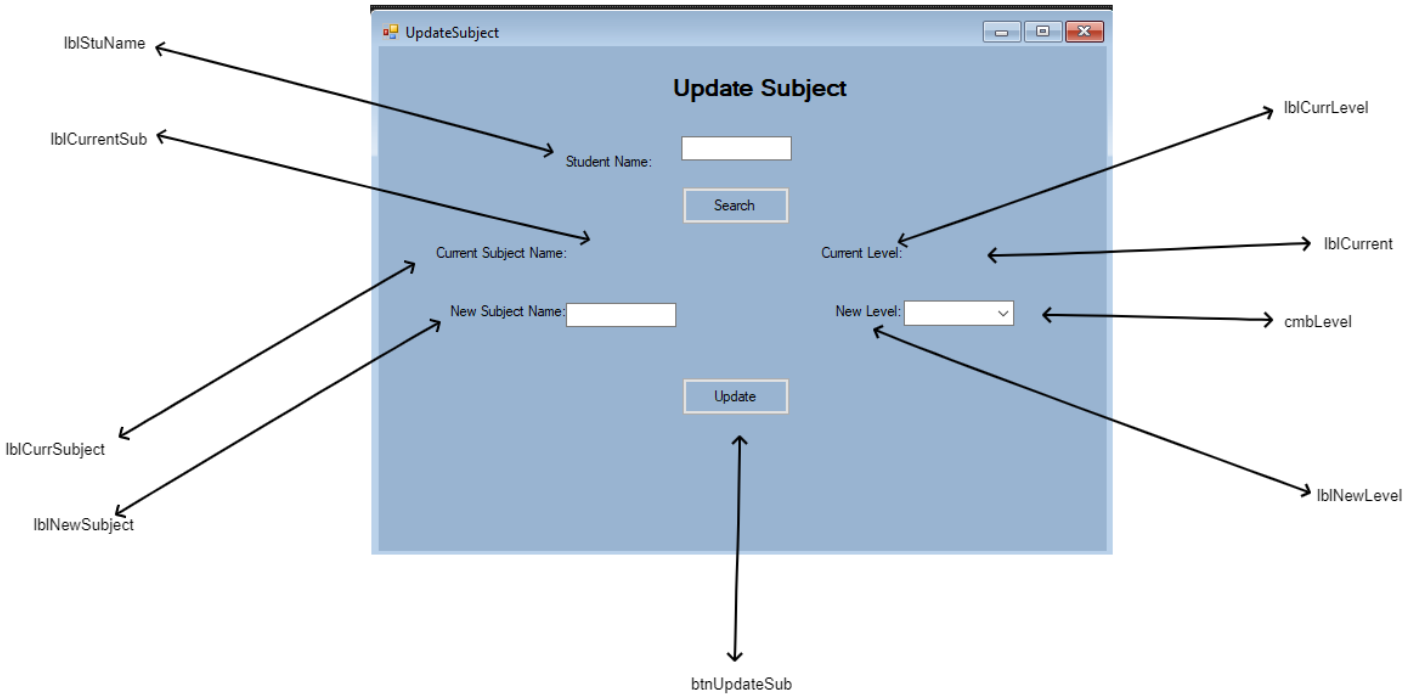


Storyboard Description

Control	Control Name	Description
Label 1	lblName	Displays “Name:” text
Label 2	lblEmail	Displays “Email:” text
Label 3	lblContact	Displays “Contact Number:” text
Label 4	lblAddress	Displays “Address” text
Label 5	btnUpdate	Updates Lecturers information

Label 13	txtAddress	Allows user to enter Address of the lecturer
Label 14	txtContact	Allows user to enter Contact Number of the lecturer
Label 15	txtEmail	Allows user to enter Email of the lecturer
Label 16	txtName	Allows user to enter Name of the lecturer
Label 17	txtTP	Allows user to enter TP number of the lecturer
Label 18	lblUpdate	Displays “Update Profile” text

4. UpdateSubject FORM:

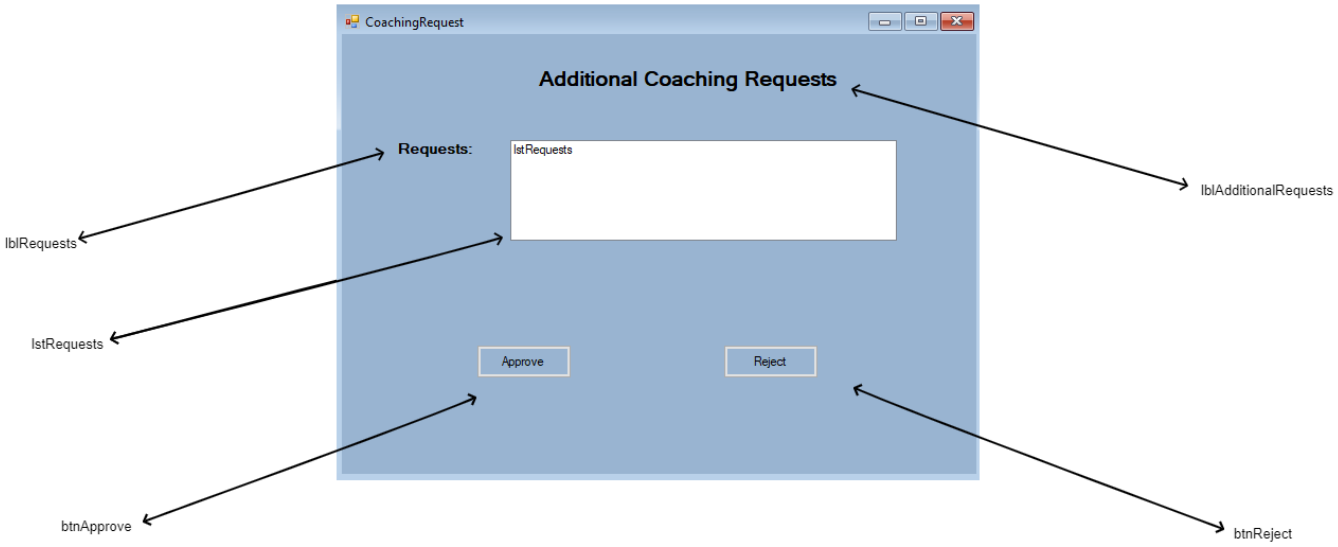


Storyboard Description

Control	Control Name	Description
Label 1	lblStuName	Displays "Student Name:" text
Label 2	lblCurrentSub	Displays the current subject the student is enrolled in after student name is entered
Label 3	lblCurrSubject	Displays "Current Subject Name:" text

Label 4	lblNewSubject	Displays "New Subject Name:" text
Label 5	btnUpdateSub	Updates Subject and Level of student
Label 6	lblNewLevel	Displays "New Level:" text
Label 7	cmbLevel	Beginner, Intermediate and Advance options are shown to the user for selection
Label 15	lblCurrent	Shows the current level of student after student name is entered
Label 16	lblCurrLevel	Displays "Current Level:" text

5. AdditionalCoachingRequests FORM:

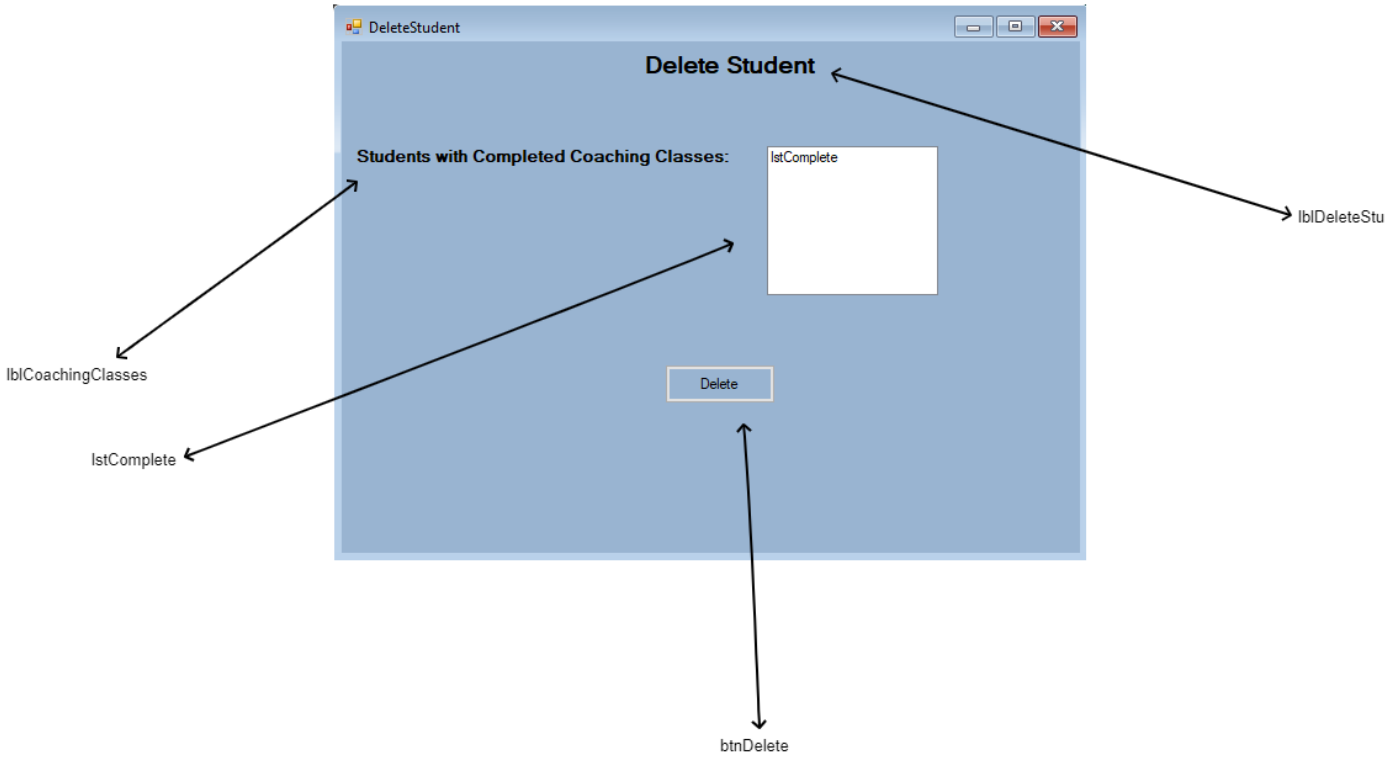


Storyboard Description

Control	Control Name	Description
Label 1	lblRequests	Displays "Requests:" text
Label 2	lstRequests	Shows list of students who have requested to join a coaching class
Label 3	btnApprove	Approves the request of student
Label 4	btnReject	Rejects the request of student

Label 5	lblAdditionalRequests	Displays "Additional Coaching Requests" text

6. DeleteStudent FORM:

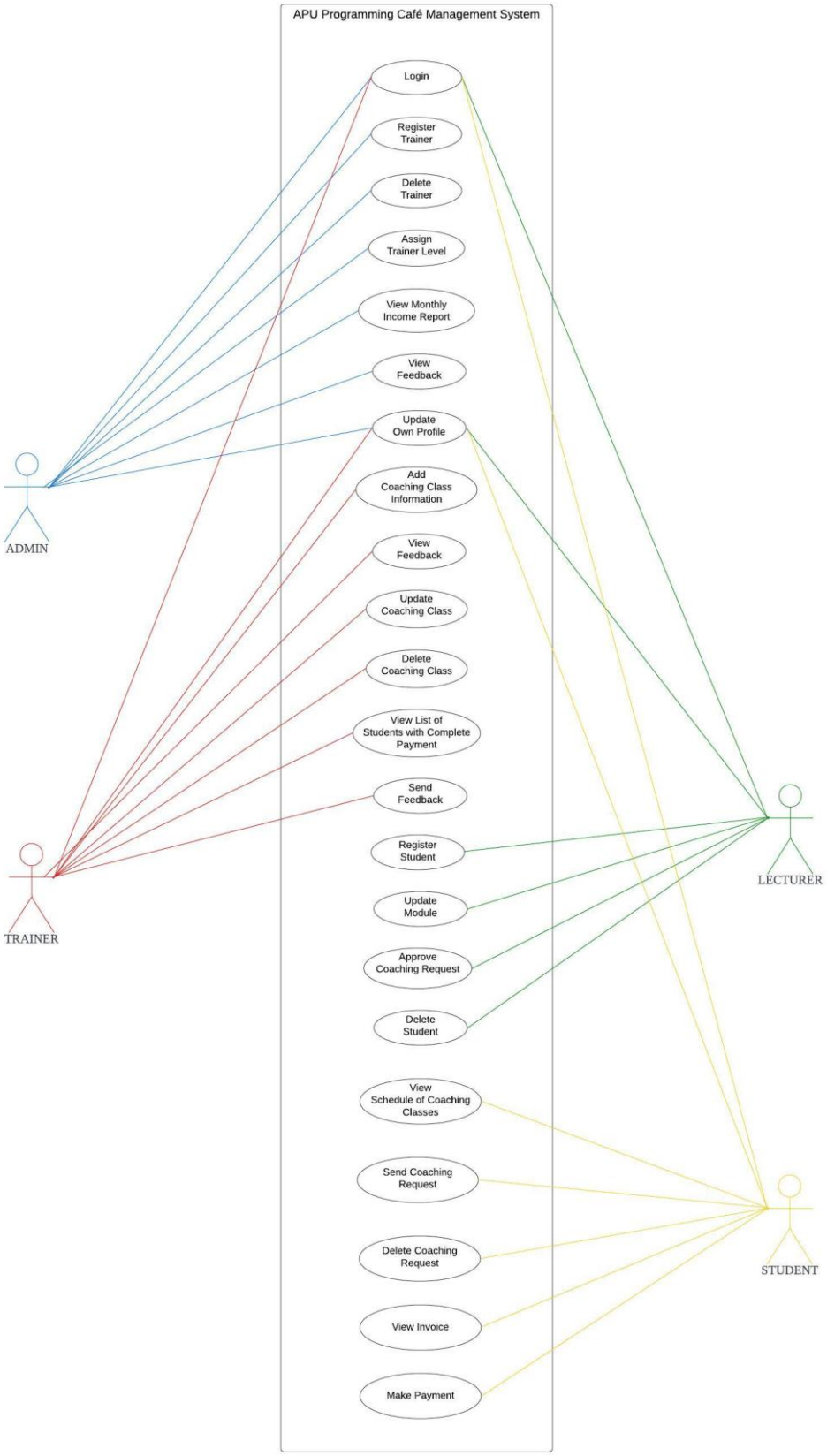


Storyboard Description

Control	Control Name	Description
Label 1	lblCoachingClasses	Displays "Students with Completed Coaching Classes" text
Label 2	lstComplete	Shows list of students who have completed their coaching classes
Label 3	btnDelete	Deletes the student record from the database
Label 4	lblDeleteStu	Displays "Delete Student" text

7.

2.0 Use-Case Diagram



3.0 Class Diagram

3.1 Admin (IBRAHEEM MOHAMMED IMADELDIN AWAD TP070765)

Admin
public string AdminName; public string UserID; public string Number; public string Email;
public Admin (string Name, string ContactNumber, string ID, string email) public string UpdateProfile ()

Trainer
public string Module1 {get; set;} public string Level1 {get; set;} public string TrainerID1 {get; set;} public string Name1 {get; set;} public string Password1 {get; set;} public string Number1 {get; set;} public string Email1 {get; set;}
public string RegisterAndEnroll()

3.2 Trainer (ABDULRAHMAN GMAIL MOHMMED AHMED TP071012)

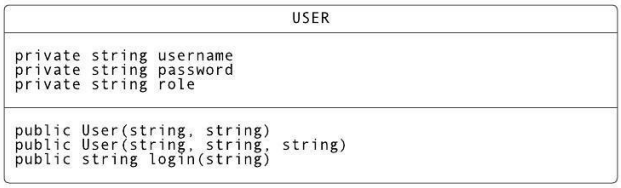
LoginManager
private SqlConnection conn;
public bool Authenticate(string username, string password) public LoginManager(string connectionString)

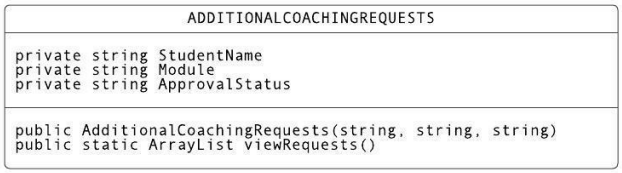
Class_Details
private SqlConnection conn;
public Class_Details(string connectionString) public void AddNewClass(string moduleID, string className, string time, string day, string fees) public void UpdateClass(string moduleID, string className, string time, string day, string fees) public void DeleteClass(string moduleID)

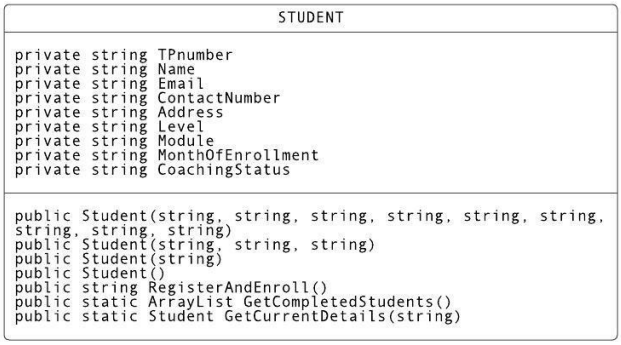
3.3 Student (ABDULELAH HUSSEIN ABDULRAHMAN AL-KAF TP: 069319)

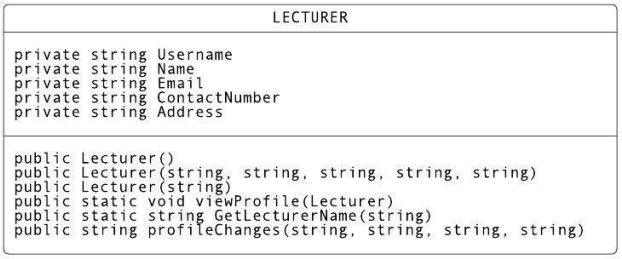
Class : Student	**Class**: ReqestCoaching
public Student(string u, string p) public Student() public string Login(string un) public static ArrayList ViewAll() public static ArrayList ViewPay() public string UpdateProfile public static string name	public RequestCoaching() public string UpdateRequest() public string deleteRequest()
private string username; private string currName; private string password; private string name; private string email; private string contNum; private string address;	

3.3 Lecturer (Amanullah Ghauri TP: 071215)









4.0 Explanation of the codes implemented in the system where OOP concepts were used.

IBRAHEEM MOHAMMED IMADELDIN AWAD (TP070765):

Admin:

- **Class**

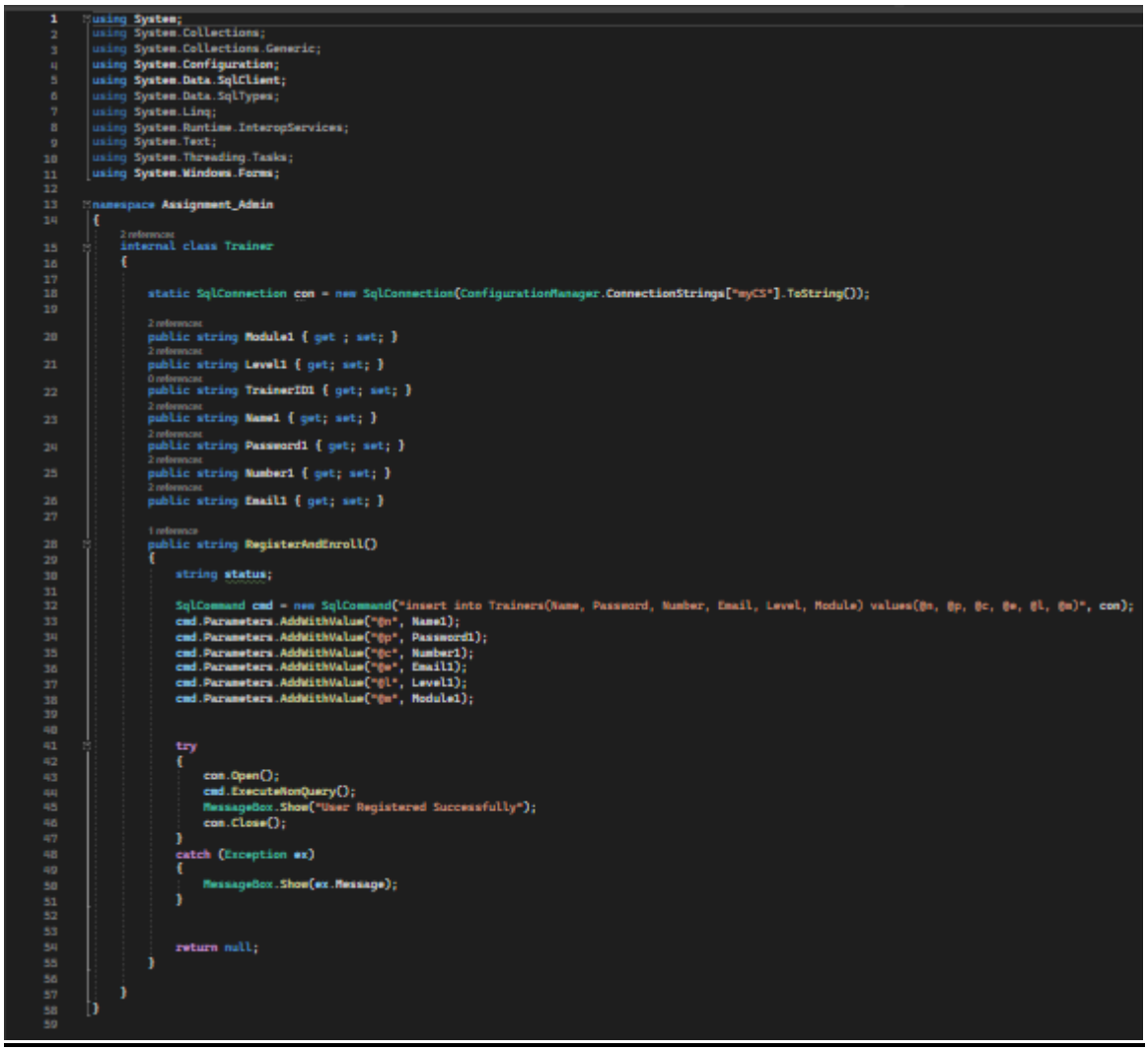
1. Admin Class

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data.SqlClient;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Assignment_Admin
{
    3 references
    internal class Admin
    {
        public string AdminName;
        public string UserID;
        public string Number;
        public string Email;

        static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());

        1 reference
        public Admin(string Name, string ContactNumber, string ID, string email)
        {
            AdminName = Name;
            UserID = ID;
            Number = ContactNumber;
            Email = email;
        }

        1 reference
        public string UpdateProfile()
        {
            string status;

            SqlCommand cmd = new SqlCommand("UPDATE Admin_information SET Name = '" + AdminName + "', Number = '" + Number + "',Email = '"+Email+"' Where Admin_Id = '" + UserID + "'", con);
            try
            {
                con.Open();
                cmd.ExecuteNonQuery();
                con.Close();
                MessageBox.Show("Updated Successfully");


            }
            catch
            {
                MessageBox.Show("Something Went Wrong\n Try again\n");
            }
            return string.Empty;
        }

    }
```

An administrator object containing the values AdminName, UserID, Number, and Email is included within the Admin class. Through a connection string called myCS, it connects to a SQL database. The properties are initialised with supplied values by a constructor. The admin's database information may be changed by using the UpdateProfile() function. In order to match the Admin_Id and UserID, it conducts a SQL UPDATE operation, changing the Name, Number, and Email fields in the Admin_information table. A try-catch block is used to protect this process and handle possible exceptions. If the process is successful, the message "Updated Successfully" is shown; if not, the message "Something Went Wrong" is presented. It says,

"Try it again." This class efficiently encapsulates both the update process and the administrative data.

2. Trainer Class



```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Configuration;
using System.Data.SqlClient;
using System.Data.SqlTypes;
using System.Linq;
using System.Runtime.InteropServices;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Assignment_Admin
{
    2 references
    internal class Trainer
    {

        static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());

        2 references
        public string Module1 { get ; set; }
        2 references
        public string Level1 { get; set; }
        0 references
        public string TrainerID1 { get; set; }
        2 references
        public string Name1 { get; set; }
        2 references
        public string Password1 { get; set; }
        2 references
        public string Number1 { get; set; }
        2 references
        public string Email1 { get; set; }

        1 reference
        public string RegisterAndEnroll()
        {
            string status;

            SqlCommand cmd = new SqlCommand("insert into Trainers(Name, Password, Number, Email, Level, Module) values(@n, @p, @c, @e, @l, @m)", con);
            cmd.Parameters.AddWithValue("@n", Name1);
            cmd.Parameters.AddWithValue("@p", Password1);
            cmd.Parameters.AddWithValue("@c", Number1);
            cmd.Parameters.AddWithValue("@e", Email1);
            cmd.Parameters.AddWithValue("@l", Level1);
            cmd.Parameters.AddWithValue("@m", Module1);


            try
            {
                con.Open();
                cmd.ExecuteNonQuery();
                MessageBox.Show("User Registered Successfully");
                con.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }


            return null;
        }
    }
}
```

With fields like Module1, Level1, TrainerID1, Name1, Password1, Number1, and Email1, the Trainer class represents a trainer object. Using a connection string called myCS, the class connects to a SQL database. This class's RegisterAndEnroll() method facilitates the addition of fresh trainer data to the database's Trainers table. A try-catch block surrounds the activities in RegisterAndEnroll(), making sure that any execution-related exceptions are properly

handled. "User Registered Successfully" is the message that appears after successful procedures.

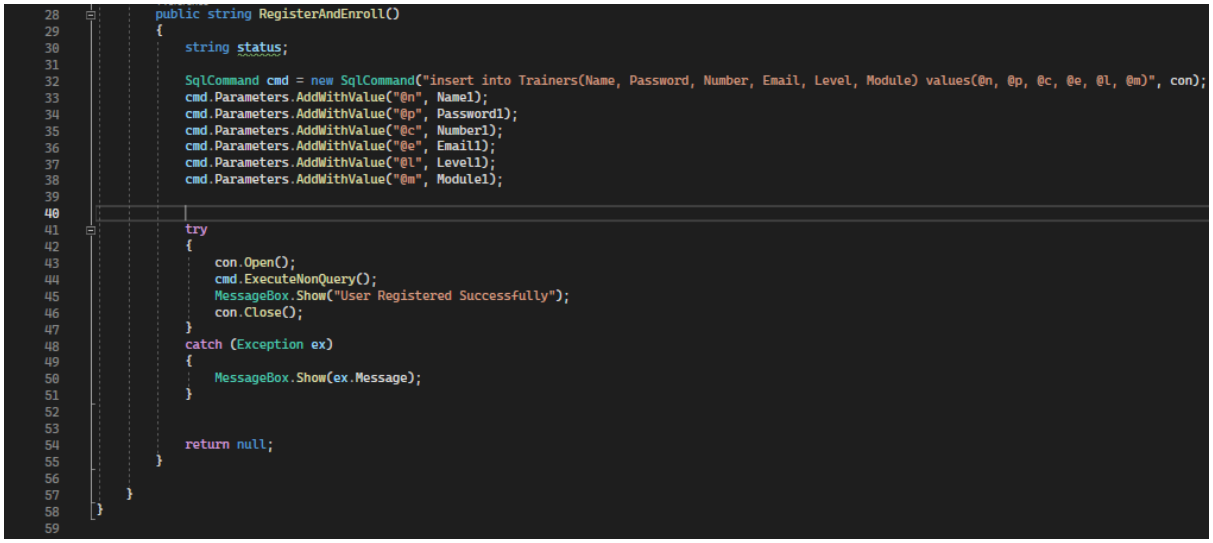
- **Methods**

1. public string UpdateProfile()

The Update Profile function is intended to use SQL to update the admin's profile in a database. When Admin_Id coincides with UserID, it updates the Name, Number, and Email fields of the Admin_information database with new values. A try-catch block, which manages potential errors, contains this action. A notification box that reads "Updated Successfully" appears after a successful execution. A notification box that reads "Something Went Wrong" appears if an error occurs. Reply again.

```
30          public string UpdateProfile()
31          {
32              string status;
33
34              SqlCommand cmd = new SqlCommand("UPDATE Admin_information SET Name = '" + AdminName + "', Number = '" + Number + "',Email = '"+Email+"' Where Admin_Id = '" + UserID + "'", con);
35              try
36              {
37                  con.Open();
38                  cmd.ExecuteNonQuery();
39                  con.Close();
40                  MessageBox.Show("Updated Successfully");
41
42
43              }
44              catch
45              {
46                  MessageBox.Show("Something Went Wrong\n Try again\n");
47              }
48              return string.Empty;
49          }
50
51      }
52  }
53
```

2. public string RegisterAndEnroll()



```
        public string RegisterAndEnroll()
        {
            string status;

            SqlCommand cmd = new SqlCommand("insert into Trainers(Name, Password, Number, Email, Level, Module) values(@n, @p, @c, @e, @l, @m)", con);
            cmd.Parameters.AddWithValue("@n", Name1);
            cmd.Parameters.AddWithValue("@p", Password1);
            cmd.Parameters.AddWithValue("@c", Number1);
            cmd.Parameters.AddWithValue("@e", Email1);
            cmd.Parameters.AddWithValue("@l", Level1);
            cmd.Parameters.AddWithValue("@m", Module1);


            try
            {
                con.Open();
                cmd.ExecuteNonQuery();
                MessageBox.Show("User Registered Successfully");
                con.Close();
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message);
            }


            return null;
        }

    }
}

```

The RegisterAndEnroll function uses SQL to add a new trainer to a database. It adds new values to the Name, Password, Number, Email, Level, and Module fields in the Trainers database. By using arguments, this approach avoids SQL injection. The notification "User Registered Successfully" appears if the operation is successful. It displays the error message if there is one. To deal with any errors, this procedure is carried out inside a try-catch block.

Objects

Objects in the Admin class:

UserID, number, email, and admin name: These are public string fields that display the administrator's information. a static SQLConnection object that creates a connection to the database is a drawback. Admin: With the supplied arguments, this constructor initialises the class fields. An admin's database profile can be updated using the UpdateProfile() function. It produces a SqlCommand called cmd, runs it, deals with any potential exceptions, and informs the user if the action was successful or unsuccessful. These items represent the information and actions related to an administrator.

Objects in the Trainer Class:

The database connection is represented by a static SqlConnection object. Public properties providing trainer information: Module1, Level1, TrainerID1, Name1, Password1, Number1, Email1. Using the RegisterAndEnroll() function, a new trainer is added to the database. To avoid SQL injection, it creates a SqlCommand, cmd, with a parameterized query. A try-catch block is used to perform the command while handling exceptions and showing success or error signals. Together, these objects capture information and behaviors pertaining to a Trainer.

ABDULRAHMAN GAMIL MOHAMMED AHMED (TP071012):

Trainer:

- **Classes**

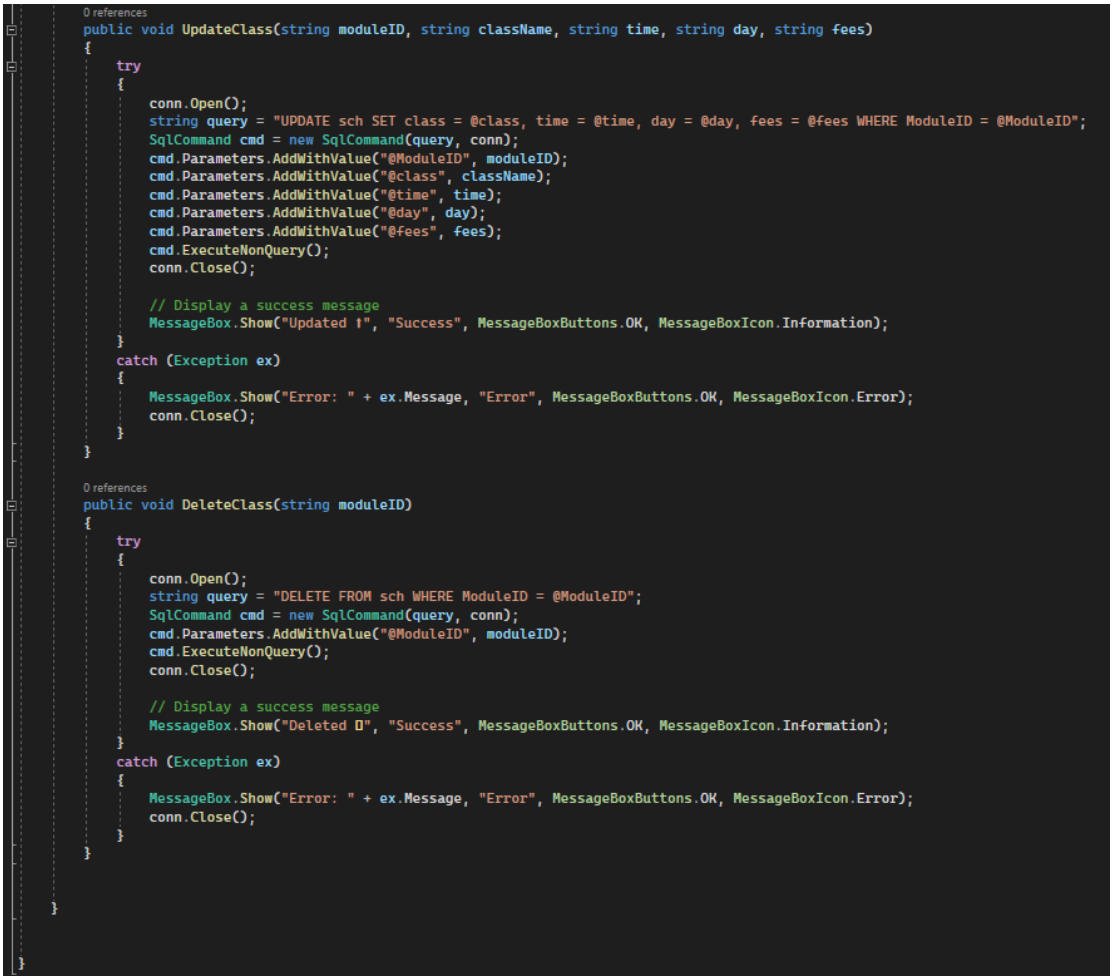
1. **LoginManager**

```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Data;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Trainer_final
{
    3 references
    public class LoginManager
    {
        private SqlConnection conn;

        // Constructor for LoginManager class that takes the connection string as a parameter
        1 reference
        public LoginManager(string connectionString)
        {
            // Create a new instance of SqlConnection using the provided connection string
            conn = new SqlConnection(connectionString);
        }

        // Method to authenticate a user given the username and password
        1 reference
        public bool Authenticate(string username, string password)
        {
            try
            {
                conn.Open();

                // SQL query to select records from the Login table where the username and password match the provided values
                string query = "SELECT * FROM Login WHERE Username = @Username AND Password = @Password AND role = 'Trainer'";
                SqlCommand cmd = new SqlCommand(query, conn);
                cmd.Parameters.AddWithValue("@Username", username);
                cmd.Parameters.AddWithValue("@Password", password);

                // Use SqlDataAdapter to fill a DataTable with the results of the query
                SqlDataAdapter sda = new SqlDataAdapter(cmd);
                DataTable dtable = new DataTable();
                sda.Fill(dtable);

                conn.Close();

                // Return true if there is at least one row in the DataTable, indicating a successful authentication
                return (dtable.Rows.Count > 0);
            }
            catch (Exception ex)
            {
                // Display an error message if an exception occurs during the authentication process
                MessageBox.Show("Error");
                conn.Close();
                return false;
            }
        }
    }
}
```

The Trainer_final application relies on the Class_Details class to handle important tasks related to managing class information. Specifically, this class is responsible for adding, updating, and deleting records for each class in the connected database's sch table.

2. Class_Details

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Data.SqlClient;
using System.Security.AccessControl;
using System.Configuration;
using System.Reflection;


namespace Trainer_final
{
    3 references
    internal class Class_Details
    {
        private SqlConnection conn;

        1 reference
        public Class_Details(string connectionString)
        {
            conn = new SqlConnection(connectionString);
        }

        1 reference
        public void AddNewClass(string moduleID, string className, string time, string day, string fees)
        {
            try
            {
                conn.Open();
                string query = "INSERT INTO sch (ModuleID, class, time, day, fees) VALUES (@ModuleID, @class, @time, @day, @fees)";
                SqlCommand cmd = new SqlCommand(query, conn);
                cmd.Parameters.AddWithValue("@ModuleID", moduleID);
                cmd.Parameters.AddWithValue("@class", className);
                cmd.Parameters.AddWithValue("@time", time);
                cmd.Parameters.AddWithValue("@day", day);
                cmd.Parameters.AddWithValue("@fees", fees);
                cmd.ExecuteNonQuery();
                conn.Close();

                // Display a success message
                MessageBox.Show("Recorded ☑", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                conn.Close();
            }
        }
```



```
        0 references
        public void UpdateClass(string moduleID, string className, string time, string day, string fees)
        {
            try
            {
                conn.Open();
                string query = "UPDATE sch SET class = @class, time = @time, day = @day, fees = @fees WHERE ModuleID = @ModuleID";
                SqlCommand cmd = new SqlCommand(query, conn);
                cmd.Parameters.AddWithValue("@ModuleID", moduleID);
                cmd.Parameters.AddWithValue("@class", className);
                cmd.Parameters.AddWithValue("@time", time);
                cmd.Parameters.AddWithValue("@day", day);
                cmd.Parameters.AddWithValue("@fees", fees);
                cmd.ExecuteNonQuery();
                conn.Close();

                // Display a success message
                MessageBox.Show("Updated !", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                conn.Close();
            }
        }

        0 references
        public void DeleteClass(string moduleID)
        {
            try
            {
                conn.Open();
                string query = "DELETE FROM sch WHERE ModuleID = @ModuleID";
                SqlCommand cmd = new SqlCommand(query, conn);
                cmd.Parameters.AddWithValue("@ModuleID", moduleID);
                cmd.ExecuteNonQuery();
                conn.Close();

                // Display a success message
                MessageBox.Show("Deleted !", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            catch (Exception ex)
            {
                MessageBox.Show("Error: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                conn.Close();
            }
        }


    }


}
```

Authenticating users is facilitated through employment of the LoginManager class. Its function is to verify submitted usernames and passwords against those registered in a databases' "Login" table.

- **Methods**

1. **public Class_Details(string connectionString)**

```
public Class_Details(string connectionString)
{
    conn = new SqlConnection(connectionString);
}
```

For successful communication with the database, it is mandatory to pass a valid 'connectionString' while initializing an object of Class_Details class via its constructor.

2. **public void AddNewClass(string moduleID, string className, string time, string day, string fees)**

```
public void AddNewClass(string moduleID, string className, string time, string day, string fees)
{
    try
    {
        conn.Open();
        string query = "INSERT INTO sch (ModuleID, class, time, day, fees) VALUES (@ModuleID, @class, @time, @day, @fees)";
        SqlCommand cmd = new SqlCommand(query, conn);
        cmd.Parameters.AddWithValue("@ModuleID", moduleID);
        cmd.Parameters.AddWithValue("@class", className);
        cmd.Parameters.AddWithValue("@time", time);
        cmd.Parameters.AddWithValue("@day", day);
        cmd.Parameters.AddWithValue("@fees", fees);
        cmd.ExecuteNonQuery();
        conn.Close();

        // Display a success message
        MessageBox.Show("Recorded ☑", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        conn.Close();
    }
}
```

To effectively add a new class record to the sch table, one should utilize the AddNewClass method. Within this method lies parameters for moduleID, className, time, day and fees. Through opening a database connection and constructing an SQL query complete with parameterized values; SqlCommand executes this query before closing out the connection. Depending on said operation's functionality - either a success or error message will be displayed in response.

3. **public void UpdateClass(string moduleID, string className, string time, string day, string fees)**

```
public void UpdateClass(string moduleID, string className, string time, string day, string fees)
{
    try
    {
        conn.Open();
        string query = "UPDATE sch SET class = @class, time = @time, day = @day, fees = @fees WHERE ModuleID = @ModuleID";
        SqlCommand cmd = new SqlCommand(query, conn);
        cmd.Parameters.AddWithValue("@ModuleID", moduleID);
        cmd.Parameters.AddWithValue("@class", className);
        cmd.Parameters.AddWithValue("@time", time);
        cmd.Parameters.AddWithValue("@day", day);
        cmd.Parameters.AddWithValue("@fees", fees);
        cmd.ExecuteNonQuery();
        conn.Close();

        // Display a success message
        MessageBox.Show("Updated !", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        conn.Close();
    }
}
```

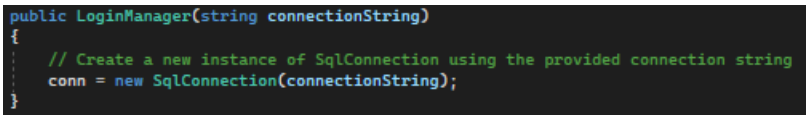
To update an existing class record in the sch table, make use of the UpdateClass method. This method shares the same parameters as AddNewClass and executes comparable database operations. Essentially, it creates an SQL update query using parameterized values and updates the sought-after record that corresponds with the moduleID supplied. This method is also programmed to exhibit a success message upon successful completion; however, it may show an error message if there was a malfunction.

4. **public void DeleteClass(string moduleID)**

```
public void DeleteClass(string moduleID)
{
    try
    {
        conn.Open();
        string query = "DELETE FROM sch WHERE ModuleID = @ModuleID";
        SqlCommand cmd = new SqlCommand(query, conn);
        cmd.Parameters.AddWithValue("@ModuleID", moduleID);
        cmd.ExecuteNonQuery();
        conn.Close();

        // Display a success message
        MessageBox.Show("Deleted □", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    catch (Exception ex)
    {
        MessageBox.Show("Error: " + ex.Message, "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
        conn.Close();
    }
}
```

In order to delete a specific class record from the sch table, you can rely on the DeleteClass method which requires only the moduleID parameter. With this information, a delete query is built and executed using parameterization for security and efficiency reasons. Upon completion, it provides helpful feedback in either an error message or success message.

5. **public LoginManager(string connectionString)**



In order to create a seamless and effective communication link between your application and the database, you might consider utilizing the constructor LoginManager. Additionally, this method has an essential parameter named connectionString, which represents your database's location data. During operation execution in code execution sequence, when you feed this object-oriented constructor with connection string data - it forms SQL Connection instance immediately.

6. **public bool Authenticate(string username, string password)**

```
public bool Authenticate(string username, string password)
{
    try
    {
        conn.Open();

        // SQL query to select records from the Login table where the username and password match the provided values
        string query = "SELECT * FROM Login WHERE Username = @Username AND Password = @Password AND role = 'Trainer'";
        SqlCommand cmd = new SqlCommand(query, conn);
        cmd.Parameters.AddWithValue("@Username", username);
        cmd.Parameters.AddWithValue("@Password", password);

        // Use SqlDataAdapter to fill a DataTable with the results of the query
        SqlDataAdapter sda = new SqlDataAdapter(cmd);
        DataTable dtable = new DataTable();
        sda.Fill(dtable);

        conn.Close();

        // Return true if there is at least one row in the DataTable, indicating a successful authentication
        return (dtable.Rows.Count > 0);
    }
    catch (Exception ex)
    {
        // Display an error message if an exception occurs during the authentication process
        MessageBox.Show("Error");
        conn.Close();
        return false;
    }
}
```

Verifying an individual's identity is achieved by implementing the Authenticate method which demands for them to enter their username and password as parameters. Once these details are received, conn.Open() then establishes an instant connection.

After this stage comes creating a SQL query whose objective is selecting records from "Login" based on matching values of provided credentials by applying parameterized queries that significantly reduce risks of SQL injection attacks.

Objects

1 Objects in LoginManager class:

To successfully interact with a SQL Server database establishing an error free connection is essential. Thats where the SqlConnection class comes in; it sets up a dependable communication channel between your application and the targeted database server.

A critical tool in managing SQL Server databases is using the SqlCommand object correctly. This utility helps develop and execute various types of sql statements and stored procedures required for manipulating database data. When formulating commands to run against a database with this object it allows for precise identification of which specific sql query or procedure will be executed.

A suitable medium for collecting, organizing, and analyzing large volumes of data is through SqlConnections and SqlDataAdapters. When it comes to populating and refreshing Data tables based on information retrieved through executing an SQL query. One can rely on SqlDataAdapters. This toolkit offers various handy data driven routines that interact seamlessly with databases to update stored information while also providing efficient methods for reading them into local Datatables.

Typically used for storing structured information similar to how one would store data within a database table the DataTable object plays a vital role for developers seeking optimal

organization. At its core this essential tool stores results from SQL queries within an in memory table representation.

2 Objects in Class_Details class:

Efficient communication between a program and an SQL Server database is made possible by the SqlCommand object. With its ability to construct and execute SQL queries. It becomes an indispensable tool for developers working in this space. When leveraging its capabilities properly it streamlines data interactions in real time making applications run smoothly and securely.

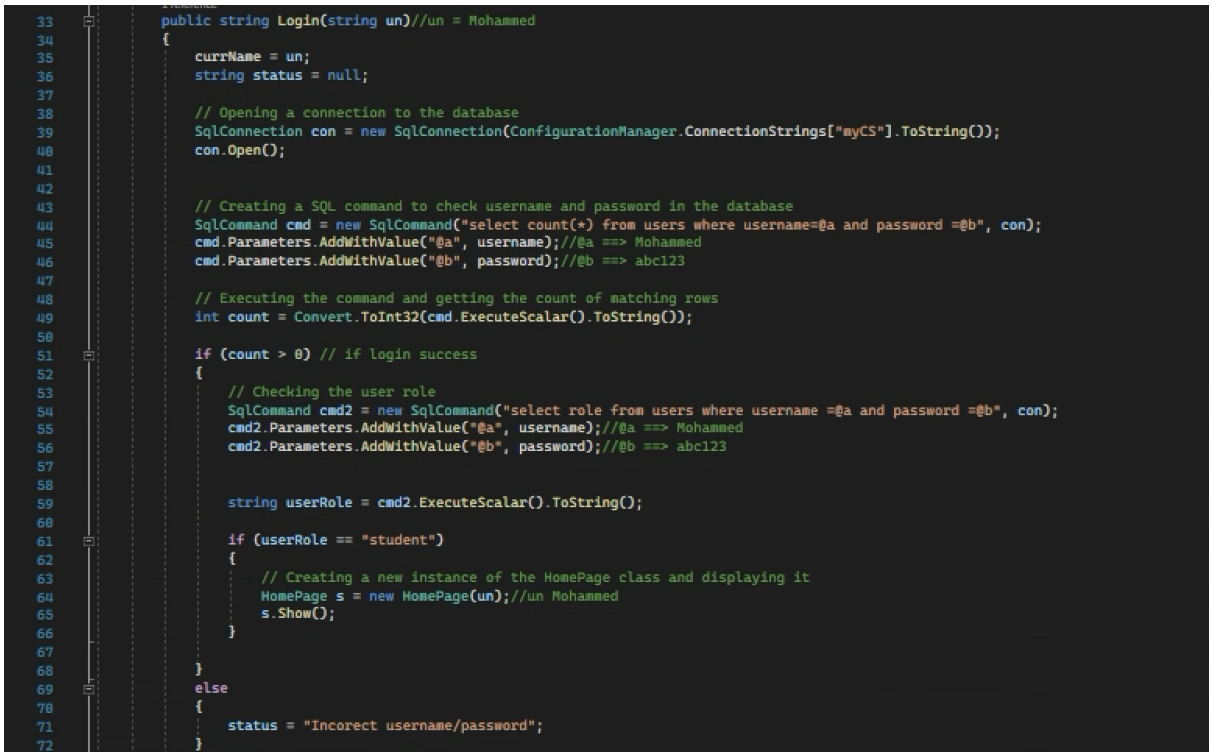
The SqlConnection object serves as an essential means of connecting to a SQL Server database. Its primary function involves establishing the connection to the database and executing SQL statements

ABDULELAH HUSSEIN ABDULRAHMAN AL-KAF (TP069319):

Student:

```
internal class Student
{
    // Declaring private fields for student information
    private string username;
    private string currName;
    private string password;
    private string name;
    private string email;
    private string contNum;
    private string address;

    // Creating a static SqlConnection object using a connection string from configuration
    static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());

    // Parameterized constructor to initialize the username and password
    1 reference
    public Student(string u, string p)
    {
        username = u;
        password = p;
    }
```

The provided code declares the **Student Class** and private fields for student information, including private string fields for username, current lane, password, name, email, contact number, and address. These fields are intended to store specific attributes of a student. Additionally, the code creates a static SqlConnection object named "con" using a connection string retrieved from the configuration. The SqlConnection object is used for database connectivity. The code also includes a parameterized constructor for the Student class, which takes in a username and password as parameters.



```
public string Login(string un)//un = Mohammed
{
    currName = un;
    string status = null;

    // Opening a connection to the database
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());
    con.Open();


    // Creating a SQL command to check username and password in the database
    SqlCommand cmd = new SqlCommand("select count(*) from users where username=@a and password =@b", con);
    cmd.Parameters.AddWithValue("@a", username);//@a ==> Mohammed
    cmd.Parameters.AddWithValue("@b", password);//@b ==> abc123

    // Executing the command and getting the count of matching rows
    int count = Convert.ToInt32(cmd.ExecuteScalar().ToString());

    if (count > 0) // if login success
    {
        // Checking the user role
        SqlCommand cmd2 = new SqlCommand("select role from users where username =@a and password =@b", con);
        cmd2.Parameters.AddWithValue("@a", username);//@a ==> Mohammed
        cmd2.Parameters.AddWithValue("@b", password);//@b ==> abc123


        string userRole = cmd2.ExecuteScalar().ToString();

        if (userRole == "student")
        {
            // Creating a new instance of the HomePage class and displaying it
            HomePage s = new HomePage(un);//un Mohammed
            s.Show();
        }

    }
    else
    {
        status = "Incorect username/password";
    }
```

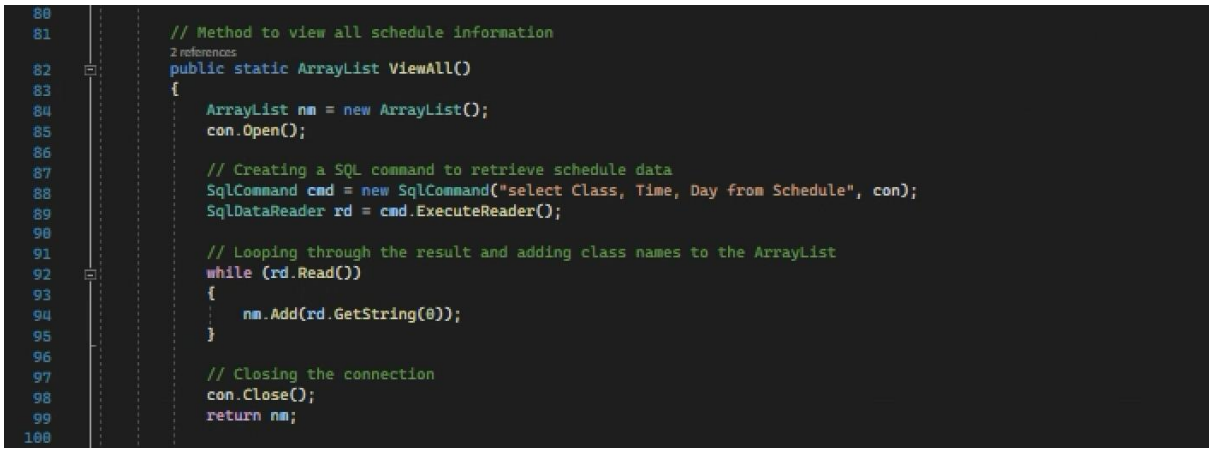
The provided code represents a method named "Login" that takes a username (un) as a parameter and returns a string. Within the method, there are several objects and methods utilized for performing a login operation. The code first assigns the value of the username parameter to the currName field. Then, a SqlConnection object named "con" is created to establish a database connection using a connection string retrieved from the configuration. The connection is opened using the con.Open() method.

Subsequently, a SqlCommand object named "cmd" is created to execute a SQL command that checks the username and password in the database. The command uses parameters to

prevent SQL injection and is designed to retrieve the count of matching rows. The ExecuteScalar() method is called to execute the command and retrieve the count as an integer.

Next, if the count is greater than 0, indicating a successful login, the code proceeds to check the user role by creating another SqlCommand object named "cmd2". This command retrieves the role from the users table based on the username and password. The ExecuteScalar() method is used again to retrieve the role as a string.

If the user role is "student", a new instance of the HomePage class is created with the username passed as an argument, and the Show() method is called to display the home page. Otherwise, if the count is not greater than 0 or the user role is not "student", the status variable is set to "Incorrect username/password".



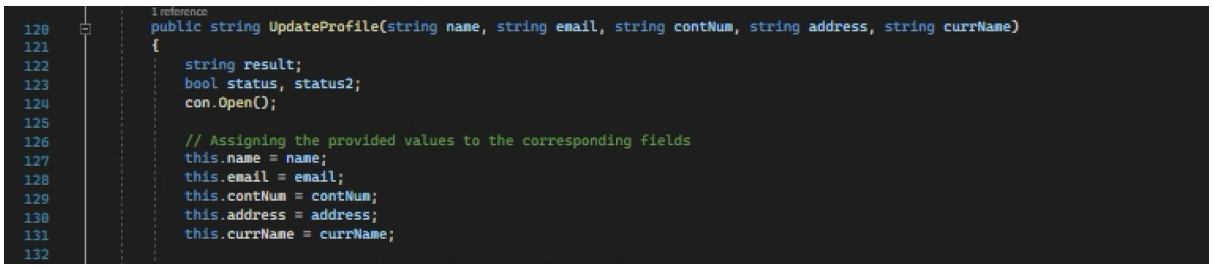
```
80
81          // Method to view all schedule information
            2 references
82          public static ArrayList ViewAll()
83          {
84              ArrayList nm = new ArrayList();
85              con.Open();
86
87              // Creating a SQL command to retrieve schedule data
88              SqlCommand cmd = new SqlCommand("select Class, Time, Day from Schedule", con);
89              SqlDataReader rd = cmd.ExecuteReader();
90
91              // Looping through the result and adding class names to the ArrayList
92              while (rd.Read())
93              {
94                  nm.Add(rd.GetString(0));
95              }
96
97              // Closing the connection
98              con.Close();
99              return nm;
100
```

The provided code includes a method named "ViewALL" that is declared as static and returns an ArrayList. The method aims to retrieve schedule information from the database and populate an ArrayList with the class names.

Within the method, an ArrayList object named "nm" is instantiated to store the class names. The code then opens the database connection using con.Open().

A SqlCommand object named "cmd" is created to execute a SQL command that retrieves the class, time, and day information from the Schedule table. The command is executed using the ExecuteReader() method, and the resulting data is stored in a SqlDataReader object named "rd".

The code enters a while loop, iterating through the records returned by the SqlDataReader. Inside the loop, it retrieves the class name from the first column (index 0) of the current record using the GetString() method and adds it to the ArrayList "nm" using the Add() method.



The code given is for a method called "UpdateProfile" that takes several inputs, such as name, email, contact number, address, and current lane. The output of the method is a string.

Inside the method, there are objects for connecting to the database, like SqlConnection, and methods for doing things with the database. The code starts by using con.make() to make the link to the database.

Then, the this term is used to send the numbers given as arguments to the appropriate parts of the class. The name parameter is put in the name field, the email parameter in the email field, the contNum parameter in the contrum field, the address parameter in the address field, and the currName parameter in the currName field.

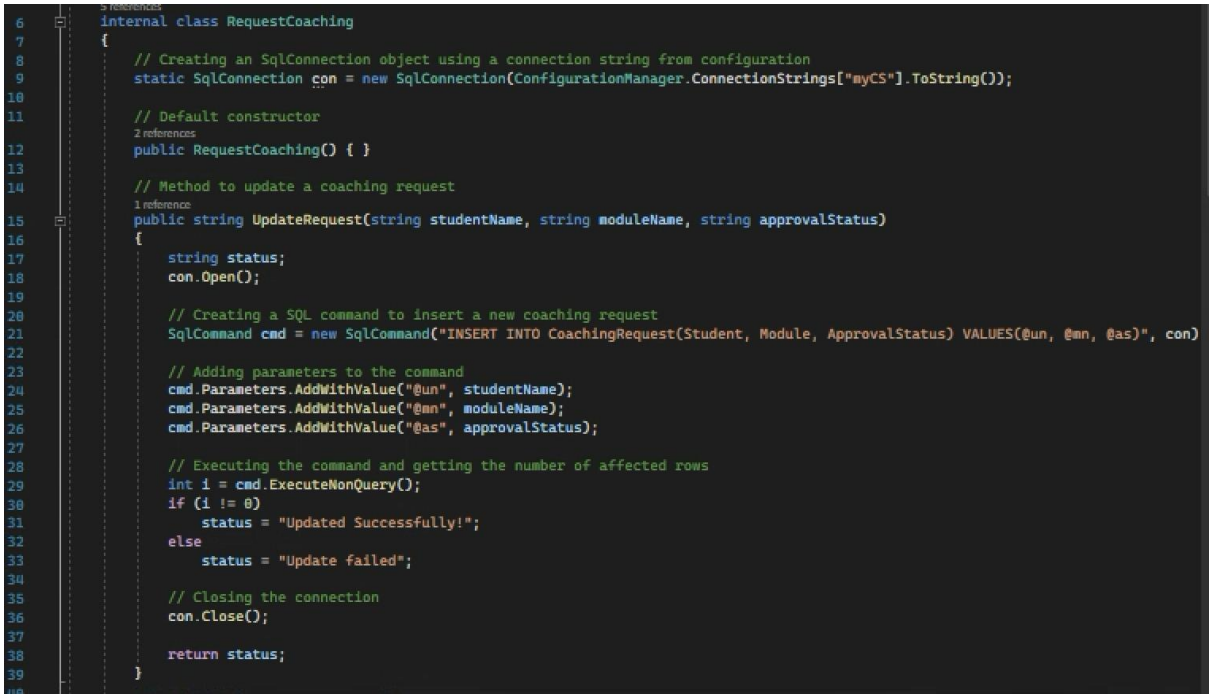
```
138             cmd.Parameters.AddWithValue("@na", this.name);
139             cmd.Parameters.AddWithValue("@em", this.email);
140             cmd.Parameters.AddWithValue("@cn", this.contNum);
141             cmd.Parameters.AddWithValue("@add", this.address);
142             cmd.Parameters.AddWithValue("@cur", this.currName);
143
144             // Executing the command and getting the number of affected rows
145             int i = cmd.ExecuteNonQuery();
146             if (i != 0)
147                 status = true;
148             else
149                 status = false;
150
151             // Closing the connection
152             con.Close();
153
154
155
156             con.Open();
157
158             // Updating the username in the Users table
159             cmd.CommandText = "UPDATE Users SET username = @na Where username = @cur";
160
161
162             //cmd.CommandText = "UPDATE Student SET Email = @em, Name = @na, Address = @add, ContactNumber = @cn Where Name = @cur";
163             int j = cmd.ExecuteNonQuery();
164             if (j != 0)
165                 status2 = true;
166             else
167                 status2 = false;
168
169             // Closing the connection
170             con.Close();
171
172             if (status && status2)
173             {
174                 result = "Updated Succesfully!";
175             }
176             else
177             {
178                 result = "Update Failed!";
179             }
```

The provided code continues from the previous code snippet and represents the rest of the "UpdateProfile" method. The code includes the addition of parameter values to the SqlCommand object "cmd" for updating the corresponding fields in the database table. Specifically, the code adds parameter values for the name field using "@na", email field using "@em", contNum field using "@en", address field using "@add", and currNane field using "@cur". The ExecuteNonQuery() method is then called on the cmd object to execute the command and obtain the number of affected rows, which is stored in the variable "i". Based on the value of "i", the status variable is set to true if it's not equal to 0, indicating a successful update, or false otherwise. The connection is closed using con.Close().

Next, the code uses con to connect to the database again.Open() and uses the current username to change the username in the "Users" table. The update command is run with the ExecuteNonQuery() method, and the value "j" stores the number of rows that will be changed.

In the same way, the value of "j" determines whether the status2 variable is set to true or false. If "j" is not equal to 0, this means that the username change was successful. con.Close() is used to close the link again.

Lastly, the code checks the values of status and status2. If both are true, it sets the result variable to "Updated Successfully!"; otherwise, it sets it to "Update Failed!" Then, the result variable is sent back.

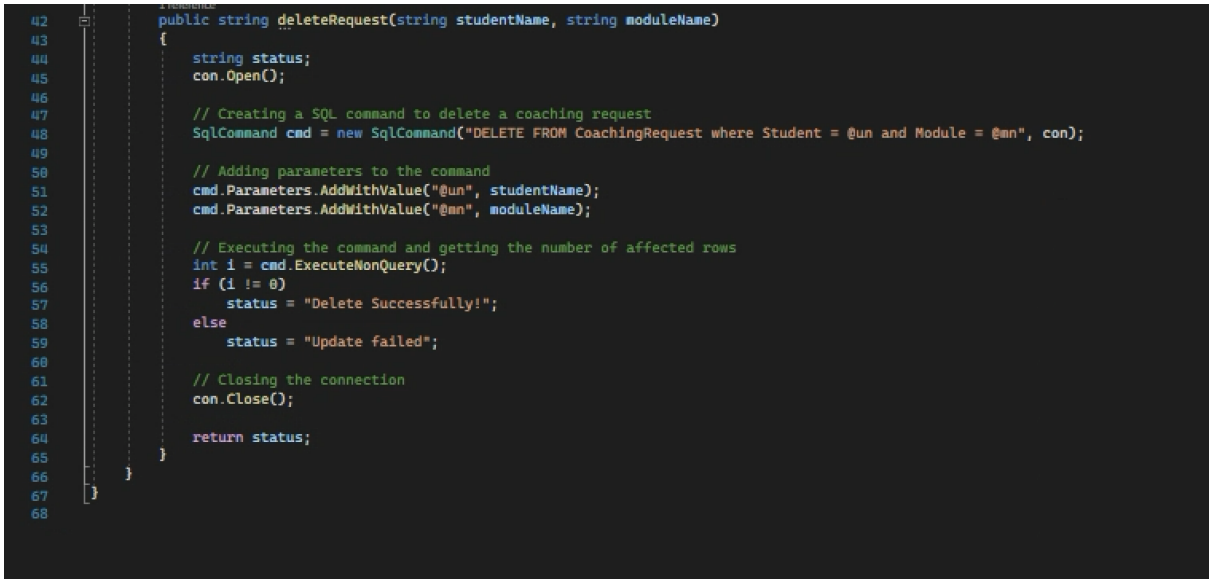


```
internal class RequestCoaching
{
    // Creating an SqlConnection object using a connection string from configuration
    static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());

    // Default constructor
    2 references
    public RequestCoaching() { }

    // Method to update a coaching request
    1 reference
    public string UpdateRequest(string studentName, string moduleName, string approvalStatus)
    {
        string status;
        con.Open();

        // Creating a SQL command to insert a new coaching request
        SqlCommand cmd = new SqlCommand("INSERT INTO CoachingRequest(Student, Module, ApprovalStatus) VALUES(@un, @mn, @as)", con)

        // Adding parameters to the command
        cmd.Parameters.AddWithValue("@un", studentName);
        cmd.Parameters.AddWithValue("@mn", moduleName);
        cmd.Parameters.AddWithValue("@as", approvalStatus);

        // Executing the command and getting the number of affected rows
        int i = cmd.ExecuteNonQuery();
        if (i != 0)
            status = "Updated Successfully!";
        else
            status = "Update failed";

        // Closing the connection
        con.Close();

        return status;
    }
```

The code given has an internal class called "RequestCoaching" and a method called "UpdateRequest" that takes studentName, moduleName, and approvalStatus as inputs. The class also has a database connection object called "con" that is a static SqlConnection object.

In the "UpdateRequest" method, a connection string from the setup is used to make a SqlConnection object. The way then makes an INSERT SQL command to use specified values to add a new coaching request to the "CoachingRequest" database table. CMD is used to add the arguments to the program.Parameters.AddWithValue(). The ExecuteNonQuery() method of the command is used to run the insert command and get the number of rows that will be changed.

Depending on the number of "i", the status variable is set to show whether or not the change worked. con.Close() is used to end the relationship. The state variable is then sent back.



```
42          public string deleteRequest(string studentName, string moduleName)
43          {
44              string status;
45              con.Open();
46
47              // Creating a SQL command to delete a coaching request
48              SqlCommand cmd = new SqlCommand("DELETE FROM CoachingRequest where Student = @un and Module = @mn", con);
49
50              // Adding parameters to the command
51              cmd.Parameters.AddWithValue("@un", studentName);
52              cmd.Parameters.AddWithValue("@mn", moduleName);
53
54              // Executing the command and getting the number of affected rows
55              int i = cmd.ExecuteNonQuery();
56              if (i != 0)
57                  status = "Delete Successfully!";
58              else
59                  status = "Update failed";
60
61              // Closing the connection
62              con.Close();
63
64              return status;
65          }
66      }
67  }
68
```

The code given has a "deleteRequest" method that takes two parameters: studentName and moduleName. The method gives back a string called "status" that shows how the delete action went.

Inside the method, the code uses con.Open() to make a link to the database. Then, it makes a SqlCommand object called "cmd" to run a SQL command that deletes a coaching request from the "CoachingRequest" table based on the given studentName and moduleName. CMD is used to add the arguments to the program.Parameters.AddWithValue().

The cmd object's ExecuteNonQuery() method is used to run the remove command and get the number of rows that will be changed. This number is kept in the "i" variable. Depending on the value of "i", the status variable is set to show whether the delete action was successful or not. By calling con.Close(), the link to the database is stopped, and the state value is returned.

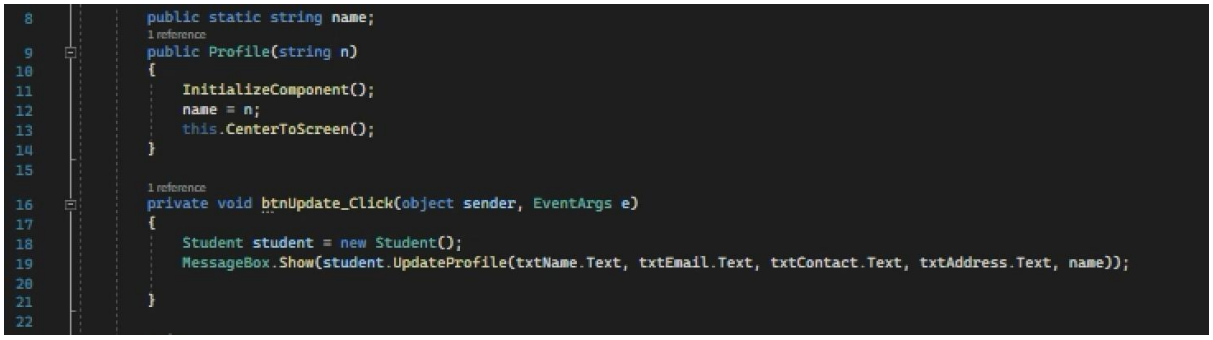
```
1 reference
private void btnViewScheduale_Click(object sender, EventArgs e)
{
    ViewSchedule view = new ViewSchedule();
    view.Show();
}

1 reference
private void btnRequests_Click(object sender, EventArgs e)
{
    Request req = new Request(name);
    req.Show();
}

1 reference
private void btnUpdate_Click(object sender, EventArgs e)
{
    Profile prf = new Profile(name);
    prf.Show();
}

1 reference
private void btnLogout_Click(object sender, EventArgs e)
{
    this.Close();
}

1 reference
private void btnPayments_Click(object sender, EventArgs e)
{
    ViewPayment vp = new ViewPayment();
    vp.Show();
}
```

The code shown above is When the "View Schedule" button is hit, the "btnviewSchedule_Click" event handler is called. It makes an instance of the VienSchedule class and uses the Show() method to show that instance.

When the "Requests" button is hit, the btnRequests_Click event handler is called. It makes an instance of the Request class and uses the Show() method to show it.When the "Update" button is hit, the btnUpdate_Click event driver is called. It makes an instance of the Profile class and uses the Show() method to show that instance.

When the "Logout" button is hit, the btnLogout_Click event handler is called. It uses the Close() method to close the current form.

When the "Payments" button is hit, the btnPayments_Click event handler is called. It makes an instance of the VlemPayment class and uses the Show() method to show that instance.

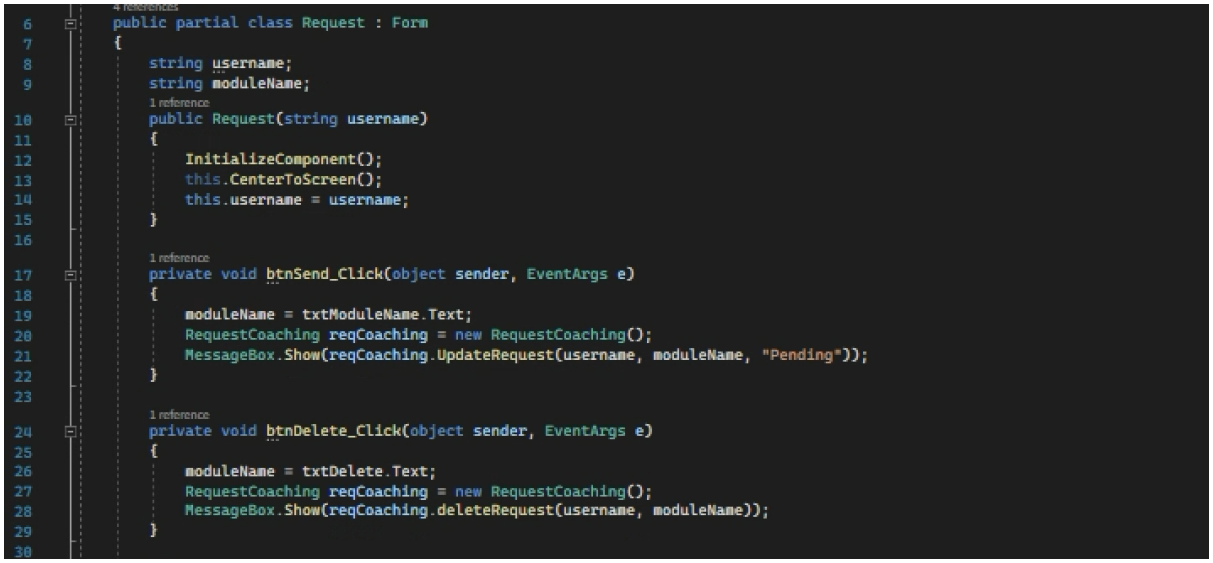


```
8       public static string name;
        1 reference
9       public Profile(string n)
10      {
11          InitializeComponent();
12          name = n;
13          this.CenterToScreen();
14      }
15
        1 reference
16      private void btnUpdate_Click(object sender, EventArgs e)
17      {
18          Student student = new Student();
19          MessageBox.Show(student.UpdateProfile(txtName.Text, txtEmail.Text, txtContact.Text, txtAddress.Text, name));
20
21      }
22
```

The piece of code shown has an event handler called btnUpdate_Click. This handler is called when the "Update" button is clicked in a GUI program. In this event handler, "Student student = new Student();" is used to make an instance of the Student class. This makes a new Student class object, which lets you use its methods and features.

User input is collected in text boxes like txtName, txtEmail, txtContact, and txtAddress so that the student's record can be kept up to date. The new numbers for the student's name, email, contact information, and location are kept in these text boxes.

Then, the MessageBox.Show() method is used to show the person a message box. In this method, the student object's UpdateProfile() method is called, and the changed values from the text boxes and a variable called "name" are sent to it. The UpdateProfile() method is in charge of changing the student's profile based on the numbers that are given.



```
4 references
public partial class Request : Form
{
    string username;
    string moduleName;
    1 reference
    public Request(string username)
    {
        InitializeComponent();
        this.CenterToScreen();
        this.username = username;
    }

    1 reference
    private void btnSend_Click(object sender, EventArgs e)
    {
        moduleName = txtModuleName.Text;
        RequestCoaching reqCoaching = new RequestCoaching();
        MessageBox.Show(reqCoaching.UpdateRequest(username, moduleName, "Pending"));
    }

    1 reference
    private void btnDelete_Click(object sender, EventArgs e)
    {
        moduleName = txtDelete.Text;
        RequestCoaching reqCoaching = new RequestCoaching();
        MessageBox.Show(reqCoaching.deleteRequest(username, moduleName));
    }

```

In the code snippet given, objects and functions are used to handle teaching requests in a program with a graphical user interface (GUI).

The code sets up two word variables: 'username' and'moduleName'. These factors are used to keep track of the information that is important for the teaching requests.

The 'Request' class is presented with a constructor method that uses the value passed as an input to set the 'username' variable. This class is in charge of handling the requests that users make.

The 'InitializeComponent' method is probably made by a GUI creator. It is used to set up the form's components so that they are ready to be shown.

The 'CenterToScreen' way is used to put the form in the middle of the screen, making it easier for the user to see.

There are two event handler methods set up: 'btnsend_Click' and 'btnDelete_Click'When the "btnsend" button is hit, the 'btnsend_Click' method is called.

AMANULLAH GHAURI (TP071215):

Lecturer:

Classes:

In object-oriented programming (OOP), a class is a blueprint or a template for creating objects. It defines a set of attributes and methods that an object of that class will contain. The attributes contain data about the object, while the methods are functions that can act on that data. Classes help in organizing code by allowing the creation of objects that share common attributes and behaviors. (Nishad, 2023)

When developing the functionalities for the Lecturer segment, I've used the concept of classes as they help provide a structured and efficient way to manage your code. I have added the following FOUR classes:

1) **User Class**

```
4 references
internal class User
{
    private string username;
    private string password;
    private string role;

    1 reference
    public User(string uname, string pass)
    {
        username = uname;
        password = pass;
    }

    0 references
    public User(string uname, string pass, string rol)
    {
        username = uname;
        password = pass;
        role = rol;
    }
```

The User class is a critical part of the system as it serves as the blueprint for every user, regardless of their role. It has the attributes that are common to all types of users - username, password, and role. These attributes form the fundamental identity of a user in the system.

The username and password are used to authenticate users when they try to log into the system. The role, on the other hand, describes the access privileges and responsibilities of a user within the system. For instance, a user with the role of 'admin' does not have the same functionalities when compared to a 'student' or 'lecturer'.

One of the key methods in this class is the login () method. The choice to incorporate the login function in the User class ensures the principle of cohesion is maintained, where related operations are encapsulated within a single unit. This not only enhances code readability but also makes the code easier to maintain and debug.

Overall, the User class forms the cornerstone of the system's user management, ensuring efficient and secure handling of user credentials and operations.

2) Lecturer Class

```
10 references
internal class Lecturer
{
    private string Username;
    private string Name;
    private string Email;
    private string ContactNumber;
    private string Address;
    static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());

    1 reference
    public string Name1 { get => Name; set => Name = value; } // Can not be Name and name, cuz pivate string is Name, so has to be Name1
    1 reference
    public string Email1 { get => Email; set => Email = value; }
    1 reference
    public string ContactNumber1 { get => ContactNumber; set => ContactNumber = value; }
    1 reference
    public string Address1 { get => Address; set => Address = value; }

    0 references
    public Lecturer() { }

    0 references
    public Lecturer(string Username, string Name, string Email, string ContactNumber, string Address)
    {
        this.Username = Username;
        this.Name = Name;
        this.Email = Email;
        this.ContactNumber = ContactNumber;
        this.Address = Address;
    }
```

The Lecturer class is a fundamental component of the system as it serves as the blueprint for every lecturer in the educational environment. This class has attributes specific to the lecturer role - Username, Name, Email, ContactNumber, and Address. These attributes are unique to the lecturer and are essential for managing lecturer-related functionalities.

The Username is used to identify the lecturer in the system, whereas Name, Email, ContactNumber, and Address are the lecturer's personal details that are maintained in the system for contact and identification purposes.

The Lecturer class contains several key methods like viewProfile() and profileChanges().

The decision to have these lecturer-related methods within the Lecturer class itself maintains the principle of cohesion, where operations related to a specific context are encapsulated within a single unit. This not only enhances code readability but also improves code maintainability and debugging.

In a nutshell, the Lecturer class is essential for managing lecturer-related operations in the system efficiently and securely. It provides the structure for handling lecturer-specific data and functions, hence playing a pivotal role in the system's overall functionality.

3) Student Class

```
14 references
internal class Student
{
    private string TPnumber;
    private string Name;
    private string Email;
    private string ContactNumber;
    private string Address;
    private string Level;
    private string Module;
    private string MonthOfEnrollment;
    private string CoachingStatus;
    static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());

    1 reference
    public string Module1 { get => Module; set => Module = value; }
    1 reference
    public string Level1 { get => Level; set => Level = value; }

    1 reference
    public Student(string TPnumber, string Name, string Email, string ContactNumber, string Address, string Level, string Module, string MonthOfEnrollmen
    {
        this.TPnumber = TPnumber;
        this.Name = Name;
        this.Email = Email;
        this.ContactNumber = ContactNumber;
        this.Address = Address;
        this.Level = Level;
        this.Module = Module;
        this.MonthOfEnrollment = MonthOfEnrollment;
    }
```

The student class serves as the blueprint for every student within the educational institution. It encompasses attributes that are specifically for the student role - TPnumber, Name, Email, ContactNumber, Address, Level, Module, MonthOfEnrollment, and CoachingStatus. These attributes capture the critical aspects of a student's academic and personal profile within the system.

The TPnumber is a unique identifier for the student, facilitating easy identification within the system. Name, Email, ContactNumber, and Address are personal details of the student, critical for communication and identification purposes. Level, Module, and MonthOfEnrollment are academic details that define the student's current stage of study, whereas CoachingStatus indicates the student's progress in their coaching journey.

The Student class contains several key methods like RegisterAndEnroll(), GetCompletedStudents(), and GetCurrentDetails(). By encapsulating student-specific methods within the student class, the principle of cohesion is once again adhered to, enhancing code readability and maintainability.

In summary, the student class is central to managing student-related data and functions in the system. It provides the structure for handling student-specific operations, playing a critical role in the overall system functionality.

4) AdditionalCoachingRequests Class

```
2 references
internal class AdditionalCoachingRequests
{
    private string StudentName;
    private string Module;
    private string ApprovalStatus;
    static SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());

    0 references
    AdditionalCoachingRequests(string s, string m, string ApprovalStatus = "Pending")
    {
        StudentName = s;
        Module = m;
    }

    1 reference
    public static ArrayList viewRequests()
    {
        ArrayList r = new ArrayList();

        con.Open();
        SqlCommand cmd = new SqlCommand("SELECT Student, Module FROM CoachingRequest WHERE ApprovalStatus IS NULL OR ApprovalStatus = 'Pending'", con);
        SqlDataReader rd = cmd.ExecuteReader();

        while (rd.Read())
        {
            string request = rd.GetString(0) + " has requested to join " + rd.GetString(1) + " module for additional coaching";
            r.Add(request);
        }
        con.Close();
        return r;
    }
}
```

The AdditionalCoachingRequests class is designed to manage the process of students requesting additional coaching for various modules. It has attributes specifically related to these coaching requests, namely StudentName, Module, and ApprovalStatus. Each of these attributes plays a crucial role in efficiently managing and processing the additional coaching requests within the system.

The StudentName attribute represents the name of the student making the request, while Module specifies the academic module for which the student is seeking additional coaching. Together, these two attributes clearly define the context of a particular coaching request.

The ApprovalStatus attribute tracks the current status of the request, which is initially set to 'Pending'. This attribute is critical for managing the life cycle of each coaching request and provides important information for both students and administrators.

The AdditionalCoachingRequests class has a key method, viewRequests().By housing this method within the AdditionalCoachingRequests class, the principle of cohesion is once again upheld - related data and operations are kept within the same class, enhancing the readability and maintainability of the code.

In conclusion, the AdditionalCoachingRequests class forms the backbone for managing additional coaching requests within the system.

Methods:

In object-oriented programming (OOP), a method is a programmed procedure that is defined as part of a class and is available to any object instantiated from that class. Each object can call the method, which runs within the context of the object that calls it. Methods are the actions that perform operations on a variable. A method accepts parameters as arguments, manipulates these, and then produces an output when the method is called on an object. (Moore, 2023)

Here are some of the methods I have added in the Lecturer segment:

1) Login ()

```
public string login(string un) //method recevives username entered in the textbox
{
    string status = null; // to store status of login
    SqlConnection con = new SqlConnection(ConfigurationManager.ConnectionStrings["myCS"].ToString());
    con.Open();

    //SqlCommand cmd = new SqlCommand("select count(*) from users where username = '" + username + "' and password = '" + password + "'", con);
    SqlCommand cmd = new SqlCommand("select count(*) from users where username = @a and password = @b", con); // modified version
    cmd.Parameters.AddWithValue("username", username);
    cmd.Parameters.AddWithValue("@a", username); // @a = username
    cmd.Parameters.AddWithValue("@b", password); // @b = password

    int count = Convert.ToInt32(cmd.ExecuteScalar());

    if (count > 0) // login successfull now find userRole
    {
        SqlCommand cmd2 = new SqlCommand("select role from Users where username = @a and password = @b", con); //cmd2 used cuz can't use same obj name in the same method
        cmd2.Parameters.AddWithValue("@a", username);
        cmd2.Parameters.AddWithValue("@b", password);

        string userRole = cmd2.ExecuteScalar().ToString(); // string userRole = Lecturer. Redirects to the userRoles form
        /*
        if (userRole == "admin")
        {
            AdminHome a = new AdminHome(un);
            a.ShowDialog();
        }
        else if (userRole == "trainer")
        {
            TrainerHome s = new TrainerHome(un);
            s.ShowDialog();
        }
        */
        if (userRole == "Lecturer")
        {
            LecturerHome s = new LecturerHome(un); //passing username to the constuctor of the lecturehome form
            s.ShowDialog();
        }
        /*
        else if (userRole == "student")
        {
        StudentHome s = new StudentHome(un);
        s.ShowDialog();
        }*/
    }
    else
        status = "Incorrect username/password";
    con.Close();

    return status; // status = "Incorrect username/password" if login failed and status = "NULL" if login successfull
}
```

The login method is designed to authenticate a user by checking their provided username and password against stored user credentials in a database. It works as follows:

- The method receives a username as an argument (string un). It initializes a string variable, status, as null. This status will later be used to store and return the status of the login operation.
- The method then establishes a connection (con) to a database using a connection string fetched from the application's configuration file.

- A SQL command (cmd) is then prepared to count the number of records in the users table where the username and password match the ones provided by the user.

- The parameters @a and @b are then assigned the values of username and password, respectively. It is executed, and the result of matching records is stored in the count variable.

- If count is more than 0, this indicates a successful login since a record with matching username and password was found. The method then prepares another SQL command (cmd2) to fetch the role of the user. The role is stored in userRole.

- Next, the method checks the role of the user. If the user is a Lecturer, it creates a new instance of LecturerHome, passing the username to the constructor of the LectureHome form, and displays it using ShowDialog().

- If count is 0, no matching username and password were found, and the status is set to "Incorrect username/password" to indicate a failed login.

- Finally, the method closes the database connection and returns the status. This will be "Incorrect username/password" if the login failed and null if the login was successful.

2) profileChanges ()

```
1 reference
public string profileChanges(string n, string e, string c, string a)
{
    string status;
    con.Open();

    Name = n;
    Email = e;
    ContactNumber = c;
    Address = a;

    SqlCommand cmd = new SqlCommand("UPDATE Lecturer set Name = @n, Email = @e, ContactNumber = @c, Address = @a", con);
    cmd.Parameters.AddWithValue("@n", Name);
    cmd.Parameters.AddWithValue("@e", Email);
    cmd.Parameters.AddWithValue("@c", ContactNumber);
    cmd.Parameters.AddWithValue("@a", Address);

    int i = cmd.ExecuteNonQuery(); // To check if update was successfull. If one query is successfull then the rest are too.
    if (i != 0)
    {
        status = "Updated Successfull!";
    }
    else
        status = "Unable to Update";
    con.Close();
    return status;
}
```

The profileChanges() method allows a user to update their profile information. The method receives four parameters: n, e, c, and a, which represent the name, email, contact number, and address of the user respectively. It works as follows:

- The method opens a connection (con) to the database. Then, it assigns the passed values to the instance variables Name, Email, ContactNumber, and Address.

- Next, the method prepares an SQL UPDATE command (cmd) that will modify the values in the Lecturer table for the current user. The command uses parameters (@n, @e, @c, @a).

- The command is then executed using ExecuteNonQuery(), which is typically used for SQL commands such as INSERT, UPDATE, DELETE.

- If “i” is not 0, it means that the update operation affected at least one row and was successful. Therefore, the status is set to "Updated Successful!". If “i” is 0, no rows were affected, indicating that the update operation failed, and the status is set to "Unable to Update".

- Finally, the method closes the database connection and returns the status to indicate whether the profile update was successful or not

3) RegisterAndEnroll ()

```
1 reference
public string RegisterAndEnroll()
{
    string status;
    con.Open();
    SqlCommand cmd = new SqlCommand("insert into Student(TPnumber, Name, Email, ContactNumber, Address, Level, Module, MonthOfEnrollment, CoachingStatus) values(@tp, @n, @e, @c, @a, @
    cmd.Parameters.AddWithValue("@tp", TPnumber);
    cmd.Parameters.AddWithValue("@n", Name);
    cmd.Parameters.AddWithValue("@e", Email);
    cmd.Parameters.AddWithValue("@c", ContactNumber);
    cmd.Parameters.AddWithValue("@a", Address);
    cmd.Parameters.AddWithValue("@l", Level);
    cmd.Parameters.AddWithValue("@m", Module);
    cmd.Parameters.AddWithValue("@me", MonthOfEnrollment);
    cmd.Parameters.AddWithValue("@cs", "In Progress");

    SqlCommand cmd2 = new SqlCommand("select count(*) from Users where username = @tp", con);
    cmd2.Parameters.AddWithValue("@tp", TPnumber);
    int count = Convert.ToInt32(cmd2.ExecuteScalar());
    if (count == 0)
    {
        SqlCommand cmd3 = new SqlCommand("insert into Users(username, password, role) values(@tp,'5677AB','Student')", con); // Default password set
        cmd3.Parameters.AddWithValue("@tp", TPnumber);

        cmd3.ExecuteNonQuery(); //Record updated in User Table

        int i = cmd.ExecuteNonQuery(); // To check if registration was successfull. If one query is successfull then the rest are too.
        if (i != 0)
        {
            status = "Registration Successfull!";
        }
        else
            status = "Unable to Register";
        con.Close();
        return status;
    }
    else
        status = "Usename Already Exists!";
    con.Close();
    return status;
}
```

The RegisterAndEnroll method is responsible for registering a new student and enrolling them in a course module. It works as follows:

- The method opens a connection (con) to the database. SqlCommand object cmd is created with an SQL INSERT statement to add a new record to the Student table. The inserted data includes information such as the student's TP number, name, email, contact number, address, course level, module, month of enrollment, and coaching status, which is initially set to "In Progress".

- Another SqlCommand object cmd2 is created with a SELECT statement to count the number of existing users with the same TP number, which is being used as the username in this case.

- If the TP number does not exist in the Users table (indicated by count being 0), another SqlCommand object cmd3 is created to insert a new record into the Users table. The new user is assigned a default password of '5677AB' and a role of 'Student'. cmd3.ExecuteNonQuery() is then used to execute this SQL statement, which updates the Users table.

- Next, the method executes the first SQL INSERT statement cmd.ExecuteNonQuery(), storing the result in “i”. If “i” is not 0, it indicates that at least one record was successfully inserted, thus the status is set to "Registration Successful!". If “i” is 0, no records were inserted, and the status is set to "Unable to Register".

- If the TP number already exists in the Users table (indicated by count being not 0), the method skips the registration process and directly sets the status to "Username Already Exists!".

Finally, the connection to the database is closed, and the method returns the status string, indicating the result of the registration attempt.

4) viewRequests ()

```
1 reference
public static ArrayList viewRequests()
{
    ArrayList r = new ArrayList();

    con.Open();
    SqlCommand cmd = new SqlCommand("SELECT Student, Module FROM CoachingRequest WHERE ApprovalStatus IS NULL OR ApprovalStatus = 'Pending'", con);
    SqlDataReader rd = cmd.ExecuteReader();

    while (rd.Read())
    {
        string request = rd.GetString(0) + " has requested to join " + rd.GetString(1) + " module for additional coaching";
        r.Add(request);
    }
    con.Close();
    return r;
}
```

The viewRequests method is designated for viewing all pending coaching requests. It works as follows:

- First, the method creates an ArrayList object r that will store the requests to be returned. It then opens a connection to the database using con.Open(). A SqlCommand object cmd is created with an SQL SELECT statement to retrieve all records from the CoachingRequest table where ApprovalStatus is either 'NULL' or 'Pending'.

- The ExecuteReader method is invoked on the cmd object to execute the SQL statement and return a SqlDataReader object rd.

- A while loop is initiated that continues as long as there are more rows to be read in rd (indicated by rd.Read()). For each row, the student's name (rd.GetString(0)) and module (rd.GetString(1)) are concatenated into a string that reads "Student X has requested to join Module Y for additional coaching". This string is then added to the ArrayList r.

- After all rows have been read and processed, the connection to the database is closed with con.Close(). Finally, the method returns the ArrayList r, which now contains all pending coaching requests in the format described above.

In conclusion, the methods defined in these classes are essential for handling various tasks within the system. By maintaining a clean code through well-defined methods, the system becomes more maintainable and scalable for future development needs.

Objects:

In object-oriented programming (OOP), objects are the fundamental building blocks of a program. An object is a self-contained entity that consists of both data and behavior, and it represents a real-world concept or entity. A class is a blueprint for creating objects, and it defines the properties and methods that all instances of that class will have. (contributors, 2023)

In the development of the lecturer segment, I have made extensive use of objects. Following are some key examples illustrating how I've incorporated them:

1) **Lecturer Object Example**

```
1 reference
private void btnUpdate_Click(object sender, EventArgs e)
{
    Lecturer obj1 = new Lecturer(name);
    MessageBox.Show(obj1.profileChanges(txtName.Text, txtEmail.Text, txtContact.Text, txtAddress.Text));
}
```

In this example, the object obj1 of the Lecturer class is being created. This object is created when the 'Update' button is clicked, and the constructor of Lecturer class is invoked with name as a parameter.

Next, the method profileChanges is called on obj1, passing in the current values from the text fields in the user interface as parameters. These values represent the name, email, contact number, and address that are to be updated in the lecturer's profile.

The result of the profileChanges method is then displayed to the user using a MessageBox.

In summary, this function again demonstrates the utility of objects in managing related data and operations, here specifically for updating a lecturer's profile.

2) User Object Example

```
1 reference
private void btnLogin_Click(object sender, EventArgs e)
{
    string stat; // will hold the status of the login
    User obj1 = new User(txtUsername.Text,txtPassword.Text); // uses constuctor to store the values in textbox
    stat = obj1.login(txtUsername.Text); // calls login method through "obj1" and stores retun value in "stat"
    if(stat != null) // if state != null meaning login failed, so "Incorrect username/password" displayed to user
    {
        MessageBox.Show(stat);
    }
    txtUsername.Text = String.Empty; // Textbox reset
    txtPassword.Text = String.Empty; // Textbox reset
    // If stat = null it just resets the textboxes
}
```

In this example, a User object named obj1 is created when the 'Login' button is clicked, and the constructor of the User class is invoked with the entered username and password as parameters.

Following the creation of the object, the login method of the User class is called on obj1. The entered username is passed as an argument, and the result of the login attempt is stored in the

stat variable. If the login attempt is unsuccessful, a message box appears to inform the user that their username or password was incorrect.

After the login attempt, the text boxes for the username and password fields are reset, preparing the form for the next login attempt.

This function is another example of the utility of objects, demonstrating how they can manage related data and operations in a cohesive, easy-to-manage unit of functionality.

3) Student Object Example

```
1 reference
private void btnRegister_Click(object sender, EventArgs e)
{
    Student obj1 = new Student(txtTP.Text, txtName.Text, txtEmail.Text, txtContact.Text, txtAddress.Text, txtLevel.Text, txtModule.Text, txtMonthE.Text);
    MessageBox.Show(obj1.RegisterAndEnroll()); // shows if registration is successfull or not
}
```

In this example, the object obj1 of the Student class is being created. When the 'Register' button is clicked, a Student object obj1 is instantiated, with its various attributes being set via text from the corresponding text fields in the user interface.

The RegisterAndEnroll() method is then invoked on obj1, which is responsible for handling the registration and enrollment processes. The outcome of this method is then displayed in a MessageBox.

This code illustrates how objects can encapsulate related data like student details and operations like registration into a cohesive, easy-to-manage unit.

5.0 Test Plan & Test Cases

Test Case	Function Name	Test Objective	Expected Result	Actual Result	Remarks
1	Add class.	To add a new class for the students who were assigned to the module that the trainer trains.	Add a new class to the (sch) table	add a new class with details.	None.
2	Delete class.	Delete a specific class with all its details from the database.	Delete an existing class in the (sch) table.	The selected class gets deleted from the database.	None.
3	Update class.	Update a specific class with all its details from the database.	Update an existing class in the (sch) table depending on the ModuleID.	Update the wanted class with its all details in the database.	None.
4	Update trainer profile.	Update the details of the assigned username.	The trainer can change the email and contact number and address in the (trainer)	The trainer can change his personal information in the database easily.	None.

			table depending on the trainer's username.		
5	View the students' information.	To inform the trainer withal the students they have.	Call all the data in the database that is related to the students.	View all the information of the students easily.	None.
6	Send feedback to the administrator.	Send to the administrator feedback messages and show the administrator the sender's username.	Insert the feedback text into the database depending on the username.	send the feedback message to the administrator depending on the trainer's username	None.
7	View Trainer Feedback	To view the feedback sent by every trainer along with their names and ID's..	Call all the data in the database under the feedback table and display the feedback along with the name and the ID of the trainer.	All the trainers along with their ID's and Names are displayed in the grid view accurately.	None.

8	Update the admin profile	Update the Phone number, Email, and Name.	The admin should be able to update their phone number, email, and name by entering the information into a textbox and after clicking the update button then the information is updated in the database.	The admin updates their name, contact number, and email and it updates correctly in the database.	None
9	Delete Trainer	Delete all the trainer details from the database.	The admin should be able to delete trainers by selecting them from the grid view and then clicking the delete causing that trainer's	The admin can delete the trainer's information without any issues.	None

			information to be deleted completely from the database		
10	Register New Trainer	Register a new trainer into the system and add their information in the database.	The admin should be able to register new trainers into the database by inserting information into the text boxes in the admin registration form.	The admin is able is to add information accurately into when the register button is clicked.	None.
11	View Trainer Details	View all the trainer details and information that is stored in the database.	The admin should be able to view all the trainers along with their details that are stored in the database when their ID's are	The admin is able to view all the trainers along with their details whenever the trainer ID is entered in the textbox.	None.

			entered in the text box.		
12	Update student	Update email, name, contact number, Address	The student sfould be able to update the information given	As expected	None
13	View payment	Display non- paid fees.	Subjects that have not been paid is displayed	As expected	None
14	Login	identify the role	System will identify the role	As expected	None
15	Add request	Request a new subject	System will receive the request	As expected	None

6.0 Conclusion

STRENGTHS

- Students can update information throw the system.
- The trainer can view the list of all the students who enrolled in any module.
- trainer has the ability to communicate with the administrator directly.

WEAKNESSES

- Students can't mark their attendance throw the system
- trainer doesn't have the ability to message the students.

RECOMMENDATIONS

- It is recommended that an attendance system should be implemented within the student system. Such a system would provide students with greater attendance flexibility and would benefit both students and instructors.

-give the ability to all the people with different roles and responsibilities to communicate with each others by implementing chat options.

7.0 References

How to retrieve data from a SQL Server database in C#? - Stack Overflow

How To Connect SQL Database In ASP.NET Using C# And Insert And View The Data Using

Visual Studio 2015 (c-sharpcorner.com)

contributors, M. (2023, April 2). *Object-oriented programming*. Retrieved from developer.mozilla: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/Object-oriented_programming

Moore, K. (2023, June 11). *Methods (OOP)*. Retrieved from brilliant: https://brilliant.org/wiki/methods-oop/

Nishad, A. (2023). *Classes and Objects In Object Oriented Programming*. Retrieved from enjoy algorithms: https://www.enjoyalgorithms.com/blog/classe-and-object-in-oops

8.0 Workload Matrix

No.	Assigned Task & Brief Description	Assigned Member Name	Completion Status / Comment
1.	- Administrator user with its section. - Storyboard for Administrator project. - Class Diagram for Administrator. - Explanation codes for Administrator. - Test Plan. - Conclusion.	Ibraheem Mohammed Imadeldin Awad (TP070765).	Completed.
2.	- Trainer user with its section. - Storyboard for Trainer project. - Class Diagram for Trainer. - Explanation codes for Trainer. - Test Plan. - Conclusion.	Abdulrahman Gamil Mohammed Ahmed (TP071012).	Completed.
3.	- Student user with its section - Storyboard for student project - Class Diagram for student - Explanation codes for Student - Test Plan - Conclusion	ABDULELAH HUSSEIN ABDULRAHMAN AL – KAF(TP:069319).	Completed.
4.	- Lecturer user with its section - Storyboard for Lecturer Project - Class Diagram for Lecturer - Explanation codes for Lecturer - Use-case Diagram	AMANULLAH GHAURI	Completed