Project Title:

Unicom TIC Management System (UMS) Project

Type:

C# Windows Forms Desktop Application
Built with the **MVC (Model-View-Controller)** architecture using a **SQLite** database.

Project Overview:

The **Unicom TIC Management System (UMS)** is a desktop-based educational management platform designed to support multiple user roles such as **Admin, Lecturer, Staff, and Student** with tailored interfaces and access control. The system enables smooth handling of academic data, such as courses, subjects, users, exams, marks, and timetables. It ensures secure login and role-based navigation.

Key Features Implemented:

User Login & Role-Based Dashboards

- Secure login system with role-based access.
- First-run logic: only Admin can register at first run, login enabled afterward.
- Role-based dashboards:
    - **Admin**: Full access to all modules. ○ **Lecturer**: View profile, manage marks for assigned subjects, view timetable. ○ **Staff**: View profile, manage exams, all student marks, and timetable.
    - **Student**: View profile, view own marks and full timetable.
- **Logout button** returns user to login screen.

User Account Linking (via ComboBox)

- Admin can assign login credentials (from Users table) to:
    - Students ○ Lecturers ○ Staff
- Each management form (Student, Lecturer, Staff) includes a **ComboBox** to:
    - Show only available users of the corresponding role.
    - Automatically hide users already assigned to prevent duplication.

Student Management

- Add/Edit/Delete student records with:
    - Personal details (name, contact, email, etc.) ○ **Course selection** via ComboBox ○ **User account selection** via ComboBox
- Students can:
    - View their profile ○ View their own marks
    - View full timetable

Lecturer Management

- Add/Edit/Delete lecturer records with:
    - Personal details ○ **User account selection** via ComboBox
- Each subject is **assigned to a lecturer** in Subject Management.
- Lecturers can:
    - View their profile ○ Manage marks (add/edit/delete) **only for their assigned subjects** ○ View timetable

Staff Management

- Add/Edit/Delete staff records with:
    - Personal details ○ **User account selection** via ComboBox
- Staff can:
    - View profile ○ Manage **Exams** (Add/Edit/Delete) ○ Manage **all student marks** ○ View timetable **Course & Subject Management**

- Admin can manage:
    - **Courses** (Add/Edit/Delete)
    - **Subjects** (Add/Edit/Delete)

- Each subject is assigned to a course and a lecturer.

- Uses ComboBoxes to associate courses and lecturers.

Exams & Marks Management

- **Exam Management** (by Staff/Admin):
  - Link exams to subjects.
- **Marks Management**: ○ **Lecturers**: Manage marks for their subjects. ○ **Staff**: Manage marks for all students.
  - **Students**: View only their own marks.

Timetable Management

- Admin can create timetables.
- **Time slot** is entered manually via **TextBox**
- Timetables can include:
  - Subject ○ Room
  - Time (manually typed)
- All roles can **view** the timetable.
- Students, Staff and lecturers have **read-only access** to the timetable in their dashboards.

Technologies Used:

- **C# WinForms** (UI and controls)
- **SQLite** (relational database)
- **MVC Architecture** (for separation of concerns)

Additional Functionalities:

- **Filtered ComboBoxes**:
  - Only users with the correct role are shown.
  - Assigned users are hidden from selection to prevent multiple assignments.

- **Dynamic Role-Based UI**: ○ Each role sees only what they need, ensuring clarity and security.
- **Reusable Control Design**: ○ Admin dashboard reused with visibility changes for different roles.

- **Clean Navigation Flow**:
    - Clear transitions between login, dashboard, and module screens.
- Clear Button:
    - Each management module includes a **Clear** button.
    - It allows users to **quickly reset all input fields** (TextBoxes, ComboBoxes, etc.) to their default state.
    - Improves user experience by preventing manual deletion and reducing form entry error
- Password validation:
    - The system enforces a **minimum password length of 6 characters** during registration and user creation
- Logout Flow
    - Logout button on every dashboard which leads the user back to login page

---

Challenges & Solutions:

- Assigning users to Students, Lecturers, and Staff - I had to ensure that only valid users were assigned to each role. To solve this, I used **ComboBoxes** that displayed only users based on their selected roles (e.g: only "Student" role users for student management). Additionally, once a user was assigned to a role, I **hid them from the ComboBox** so they couldn't be assigned again.

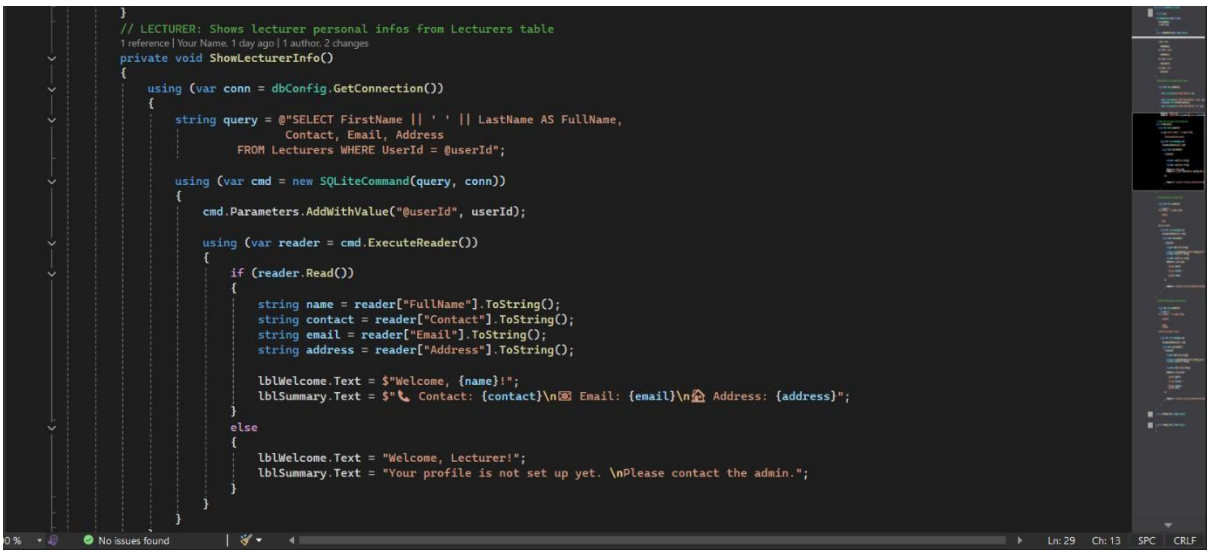
- Displaying correct marks for Students and Lecturers - It was difficult to show only the relevant marks. I solved this using the logged-in user's **UserId and Role**: **Students** see only their own marks by matching their UserId.
  Lecturers manage marks for only the **subjects assigned to them**, also linked via UserId.

- **Hiding buttons from the Admin dashboard for other roles** - Since I reused the Admin dashboard UI for all roles, I implemented **role-checking logic using the Role property** to dynamically **hide or disable buttons** that weren't relevant for the logged-in user. This made the dashboard clean and role-specific.

- **Writing important SQL queries** - Some queries; especially those fetching marks, subject assignments, or timetable data were tricky. I used the UserId and foreign key relationships to **write accurate SQL queries** and avoided showing irrelevant data.

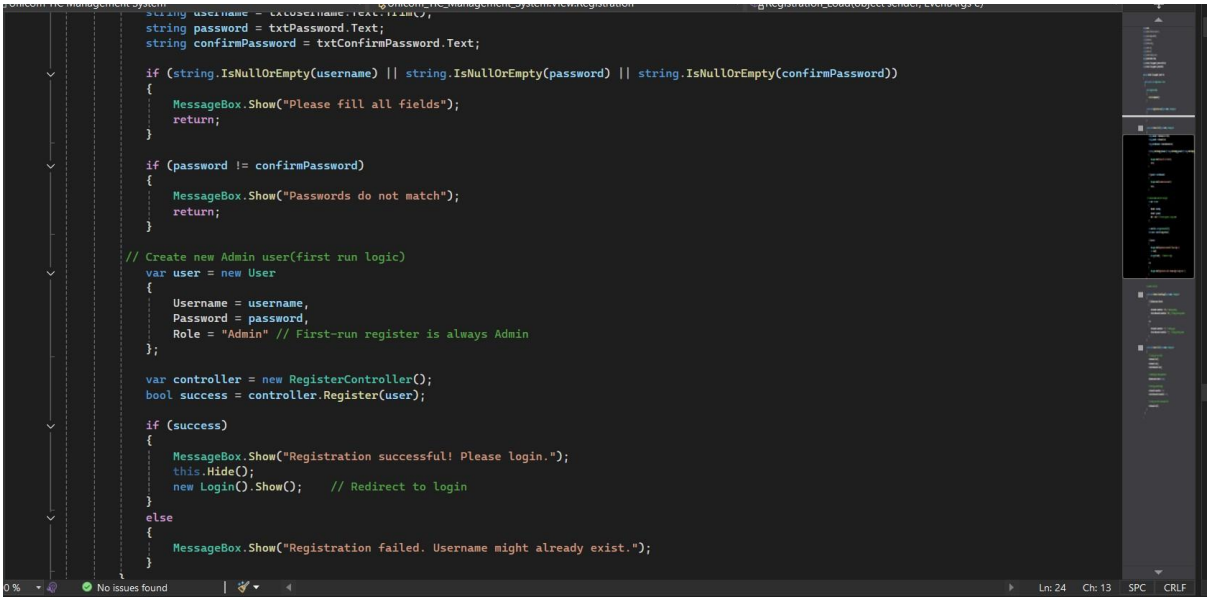
```
}

// Load marks for the currently logged-in stvdent
1 reference | Your Name, 1 day ago | 1 author, 2 changes
private void LoadMarksForStudent(int userId)
{
    using (var conn = dbConfig.GetConnection())
    {
        string getStudentIdQuery = "SELECT StudentId FROM Students WHERE UserId = @userId";
        using (var cmd = new SQLiteCommand(getStudentIdQuery, conn))
        {
            cmd.Parameters.AddWithValue("@userId", userId);
            var result = cmd.ExecuteScalar();

            if (result != null)
            {
                int studentId = Convert.ToInt32(result);

                //Get marks for that StudentId
                string getMarksQuery = @"
            SELECT m.Score, e.ExamName, s.SubjectName
            FROM Marks m
            JOIN Exams e ON m.ExamId = e.ExamId
            JOIN Subjects s ON e.SubjectId = s.SubjectId
            WHERE m.StudentId = @studentId";

                using (var marksCmd = new SQLiteCommand(getMarksQuery, conn))
                {
                    marksCmd.Parameters.AddWithValue("@studentId", studentId);
                    using (var reader = marksCmd.ExecuteReader())
                    {
                        DataTable dt = new DataTable();
                        dt.Load(reader);
                        dgvMarks.DataSource = dt;
                    }
                }
```

This C# function, `LoadMarksForStudent`, retrieves and displays a student's marks from a database. It takes a `userId` as input, establishes a database connection, and first fetches the corresponding `StudentId`. Using this `StudentId`, it then retrieves the student's marks, including scores, exam names, and subject names. The results are loaded into a `DataTable` and displayed in a data grid view, providing a clear overview of the student's academic performance.



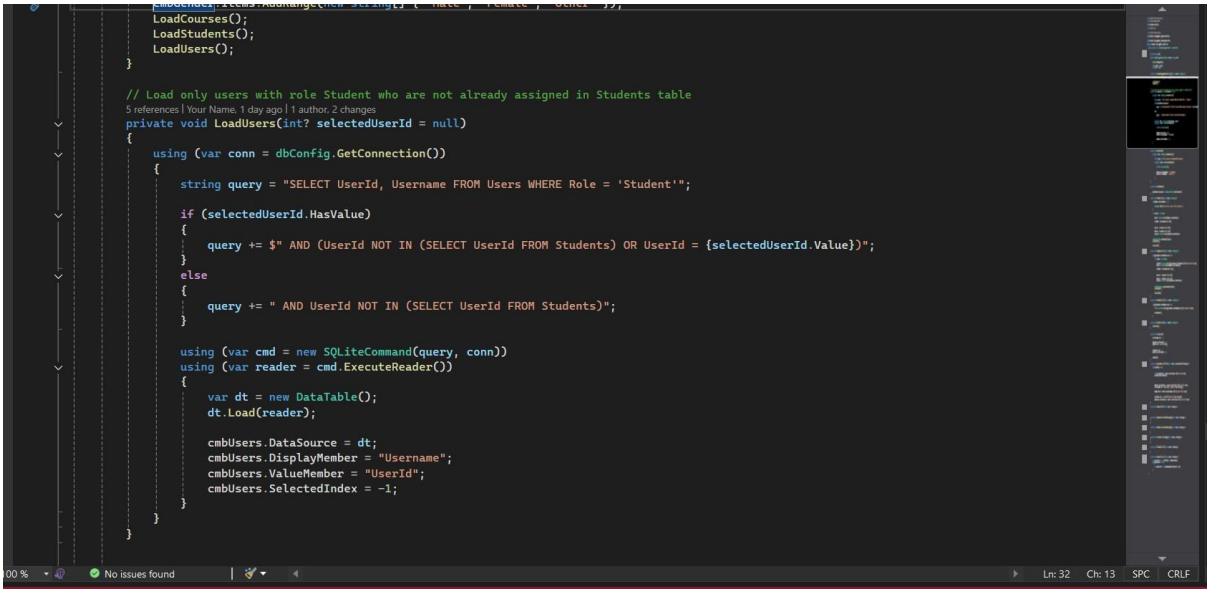
```
}
// LECTURER: Shows lecturer personal infos from Lecturers table
1 reference | Your Name, 1 day ago | 1 author, 2 changes
private void ShowLecturerInfo()
{
    using (var conn = dbConfig.GetConnection())
    {
        string query = @"SELECT FirstName || ' ' || LastName AS FullName,
                        Contact, Email, Address
                FROM Lecturers WHERE UserId = @userId";

        using (var cmd = new SQLiteCommand(query, conn))
        {
            cmd.Parameters.AddWithValue("@userId", userId);

            using (var reader = cmd.ExecuteReader())
            {
                if (reader.Read())
                {
                    string name = reader["FullName"].ToString();
                    string contact = reader["Contact"].ToString();
                    string email = reader["Email"].ToString();
                    string address = reader["Address"].ToString();

                    lblWelcome.Text = $"Welcome, {name}!";
                    lblSummary.Text = $"📞 Contact: {contact}\n✉ Email: {email}\n🏠 Address: {address}";
                }
                else
                {
                    lblWelcome.Text = "Welcome, Lecturer!";
                    lblSummary.Text = "Your profile is not set up yet. \nPlease contact the admin.";
                }
            }
        }
    }
```

the ShowLecturerInfo function is implemented to display personal information of lecturers. This C# function connects to the database and retrieves details such as the lecturer's full name, contact information, email, and address using the provided userId. The retrieved data is then used to update the user interface, welcoming the lecturer and displaying their contact details. If no data is found, default messages are shown to indicate that the profile is not set up and to prompt the user to contact the admin.



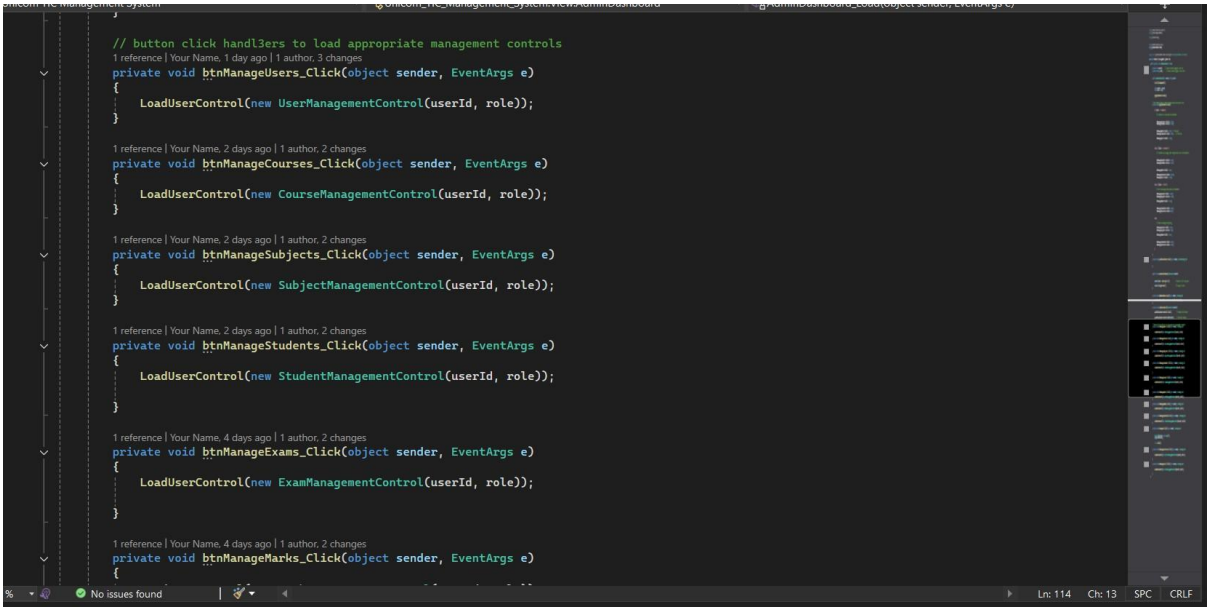
```
        string username = txtUsername.Text.Trim();
        string password = txtPassword.Text;
        string confirmPassword = txtConfirmPassword.Text;

        if (string.IsNullOrEmpty(username) || string.IsNullOrEmpty(password) || string.IsNullOrEmpty(confirmPassword))
        {
            MessageBox.Show("Please fill all fields");
            return;
        }

        if (password != confirmPassword)
        {
            MessageBox.Show("Passwords do not match");
            return;
        }

    // Create new Admin user(first run logic)
        var user = new User
        {
            Username = username,
            Password = password,
            Role = "Admin" // First-run register is always Admin
        };

        var controller = new RegisterController();
        bool success = controller.Register(user);

        if (success)
        {
            MessageBox.Show("Registration successful! Please login.");
            this.Hide();
            new Login().Show();    // Redirect to login
        }
        else
        {
            MessageBox.Show("Registration failed. Username might already exist.");
        }
    }
```

the registration logic is implemented to handle the creation of new admin users. This C# code snippet captures the username, password, and password confirmation from the respective text fields. It performs validation to ensure all fields are filled and that the password and confirmation password match. Upon successful validation, it creates a new User object with the provided details and sets the role to "Admin". The RegisterController is then used to attempt the registration. If successful, a success message is displayed, and the user is redirected to the login page. If the registration fails, an appropriate error message is shown.



```
        LoadCourses();
        LoadStudents();
        LoadUsers();
    }

    // Load only users with role Student who are not already assigned in Students table
    5 references | Your Name, 1 day ago | 1 author, 2 changes
    private void LoadUsers(int? selectedUserId = null)
    {
        using (var conn = dbConfig.GetConnection())
        {
            string query = "SELECT UserId, Username FROM Users WHERE Role = 'Student'";

            if (selectedUserId.HasValue)
            {
                query += $" AND (UserId NOT IN (SELECT UserId FROM Students) OR UserId = {selectedUserId.Value})";
            }
            else
            {
                query += " AND UserId NOT IN (SELECT UserId FROM Students)";
            }

            using (var cmd = new SQLiteCommand(query, conn))
            using (var reader = cmd.ExecuteReader())
            {
                var dt = new DataTable();
                dt.Load(reader);

                cmbUsers.DataSource = dt;
                cmbUsers.DisplayMember = "Username";
                cmbUsers.ValueMember = "UserId";
                cmbUsers.SelectedIndex = -1;
            }
        }
    }
```

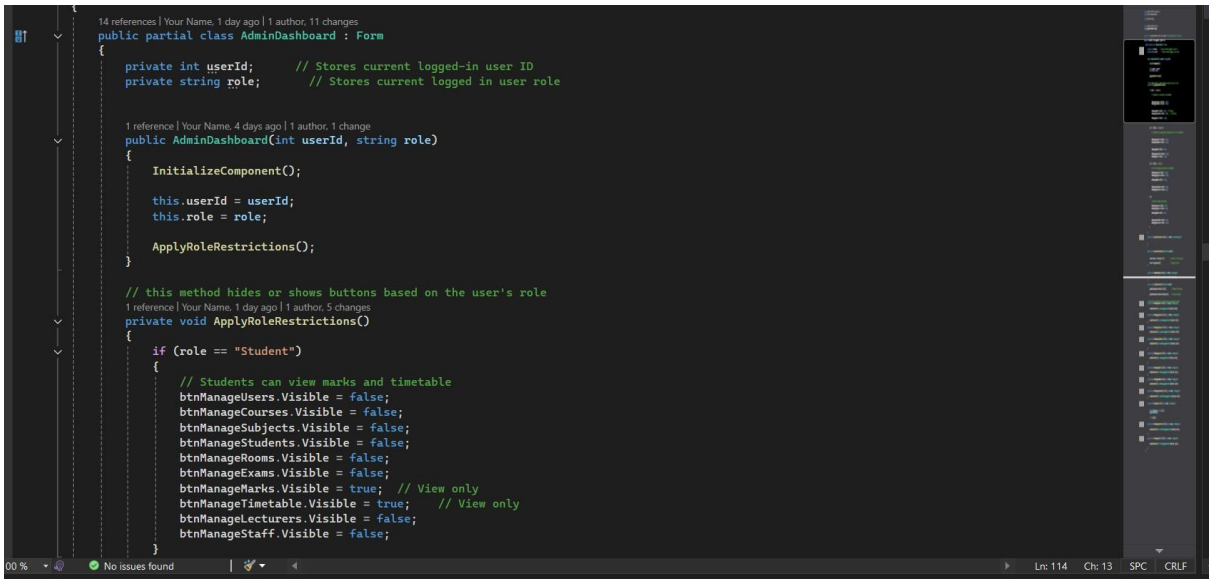
the LoadUsers function is implemented to load users with the role "Student" who are not already assigned in the Students table. This C# function establishes a database connection and constructs a SQL query to select the relevant users. If a specific selectedUserId is provided, it excludes that user from the results. The query ensures that only users not already present in the Students table are selected. The results are loaded into a DataTable, which is then used as the data source for a combo box, displaying the usernames and using the user IDs as values.



```
// button click handl3ers to load appropriate management controls
1 reference | Your Name, 1 day ago | 1 author, 3 changes
private void btnManageUsers_Click(object sender, EventArgs e)
{
    LoadUserControl(new UserManagementControl(userId, role));
}

1 reference | Your Name, 2 days ago | 1 author, 2 changes
private void btnManageCourses_Click(object sender, EventArgs e)
{
    LoadUserControl(new CourseManagementControl(userId, role));
}

1 reference | Your Name, 2 days ago | 1 author, 2 changes
private void btnManageSubjects_Click(object sender, EventArgs e)
{
    LoadUserControl(new SubjectManagementControl(userId, role));
}

1 reference | Your Name, 2 days ago | 1 author, 2 changes
private void btnManageStudents_Click(object sender, EventArgs e)
{
    LoadUserControl(new StudentManagementControl(userId, role));

}

1 reference | Your Name, 4 days ago | 1 author, 2 changes
private void btnManageExams_Click(object sender, EventArgs e)
{
    LoadUserControl(new ExamManagementControl(userId, role));

}

1 reference | Your Name, 4 days ago | 1 author, 2 changes
private void btnManageMarks_Click(object sender, EventArgs e)
{
```

No issues found | Ln: 114 Ch: 13 SPC CRLF

the admin dashboard includes several buttons that allow administrators to manage different aspects of the system. Each button click event handler loads the appropriate user control to perform the respective management tasks. The C# code snippet provided shows the event handlers for managing users, courses, subjects, students, exams, and marks. Each handler calls the LoadUserControl method with a new instance of the respective management control and the necessary parameters.



```
14 references | Your Name, 1 day ago | 1 author, 11 changes
public partial class AdminDashboard : Form
{
    private int userId;      // Stores current logged-in user ID
    private string role;       // Stores current logged in user role


    1 reference | Your Name, 4 days ago | 1 author, 1 change
    public AdminDashboard(int userId, string role)
    {
        InitializeComponent();

        this.userId = userId;
        this.role = role;

        ApplyRoleRestrictions();
    }

    // this method hides or shows buttons based on the user's role
    1 reference | Your Name, 1 day ago | 1 author, 5 changes
    private void ApplyRoleRestrictions()
    {
        if (role == "Student")
        {
            // Students can view marks and timetable
            btnManageUsers.Visible = false;
            btnManageCourses.Visible = false;
            btnManageSubjects.Visible = false;
            btnManageStudents.Visible = false;
            btnManageRooms.Visible = false;
            btnManageExams.Visible = false;
            btnManageMarks.Visible = true;  // View only
            btnManageTimetable.Visible = true;    // View only
            btnManageLecturers.Visible = false;
            btnManageStaff.Visible = false;
        }
```

No issues found | Ln: 114 Ch: 13 SPC CRLF

the AdminDashboard class is responsible for managing the visibility of various administrative buttons based on the user's role. This C# class stores the current loggedin user's ID and role, and initializes these values in its constructor. The ApplyRoleRestrictions method is called to set the visibility of buttons according to the user's role. For instance, if the user is a student, most management buttons are hidden, and only the buttons to view marks and timetable are visible.