UnicomTic Management System

1. Project Overview

The UnicomTic Management System (UMS) is a role-based desktop application designed to manage academic and administrative workflows within an educational institution. The system supports four user roles — Admin, Student, Staff, and Lecturer — with customized dashboards, secure login, and access-controlled functionalities. It streamlines the management of courses, subjects, users, exams, marks, and timetables in a centralized platform.

2. Features

Role-Based Login & Dashboards

- Secure login logic with first-time Admin Sign-Up.
- After the first admin is registered, login functionality is enabled for users.
- Each user role is directed to their respective dashboard with unique access:

    - **Admin Dashboard:**
        - Access to full management: Users, Courses, Subjects, Exams, Marks, Rooms, Timetable
        - Can create and manage all user types
        - Visibility of all buttons and full permissions

    - **Lecturer Dashboard:**
        - View profile
        - Assign marks to students
        - View timetable
        - view exams
        - change Password

- **Staff Dashboard:**
    - Manage Marks
    - Manage exams (add/edit/delete)
    - View timetable
    - View Profile
    - change password

- **Student Dashboard:**
    - View profile (read-only)
    - View their own marks only
    - View timetable
    - view Exams
    - change password

Logout Flow

- Logout button on every dashboard
- When clicked, a message box appears:
    - "Do you want to login again or exit the application?"
    - Based on selection, it either redirects to LoginForm or exits the application

Note : While this prompt may not be ideal in a real-world production environment, it has been intentionally included in this version **for testing and demonstration purposes** to allow easy switching between user roles during development.

User Creation & Management

- Admin can create Students, Lecturers, Staff, and other Admins
- Usernames and passwords are auto-generated (except for Admins)
- If course or subject is not created yet, system will redirect Admin to Manage Course/Subject form before allowing user creation

- Manage User panels allow editing or deleting user records
- Passwords are visible in edit forms (**for testing convenience**)

Course & Subject Management

- Admin can add, update, or delete courses and subjects
- Subjects are linked to courses using ComboBox
- Required step before creating related user roles(student , Lecturer)

Exam & Mark Management

- Exams are created with associated subjects and dates
- Marks assigned by:
    - Lecturers: for all students
    - Staff: for all students
    - Students: view only their own marks

Timetable & Room Management

- Admin can schedule classes
- Rooms created with name and type
- Timetable includes Date, Time slots, subjects, and rooms

Technologies Used

- Programming Language: C#
- UI Framework: Windows Forms
- Database: SQLite
- Architecture: MVC (Model-View-Controller)

Challenges Faced & Solutions

• Ensuring students only see their own profile and marks

Solution: I used the logged-in student's UserID to fetch only their marks and profile details through SQL queries. I also hid any unrelated UI elements on their dashboard to keep it clean and private.

• Implementing Role-Based Dashboard Visibility

Since I reused the Admin dashboard for all roles (Student, Lecturer, Staff), controlling visibility was a must.
Solution: I wrote conditional logic using the logged-in user's **Role** to show only the buttons and options relevant to their role, ensuring a secure and role-specific experience.

• Preventing subject/course mismatches during user creation

At first, users were created before any subjects or courses existed, which led to confusion.
Solution: I made sure courses and subjects are created *first*. If not available, the app redirects to the Course Management form to set them up before proceeding with user creation.

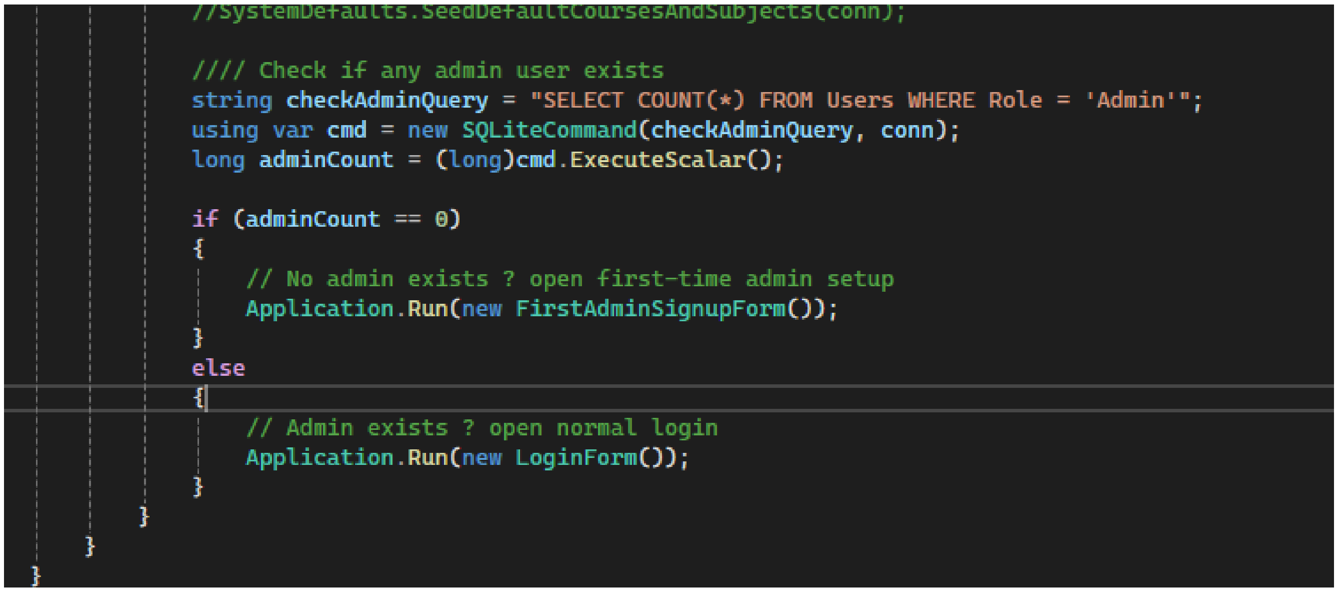
• Mark Entry validation (no duplicates, correct mapping)

Allowing users to enter marks without checking conditions caused errors.
Solution: I used ComboBoxes that dynamically loaded subjects relevant only to the selected student. I also added validation to prevent duplicate mark entries for the same subject and student.

• Managing passwords securely

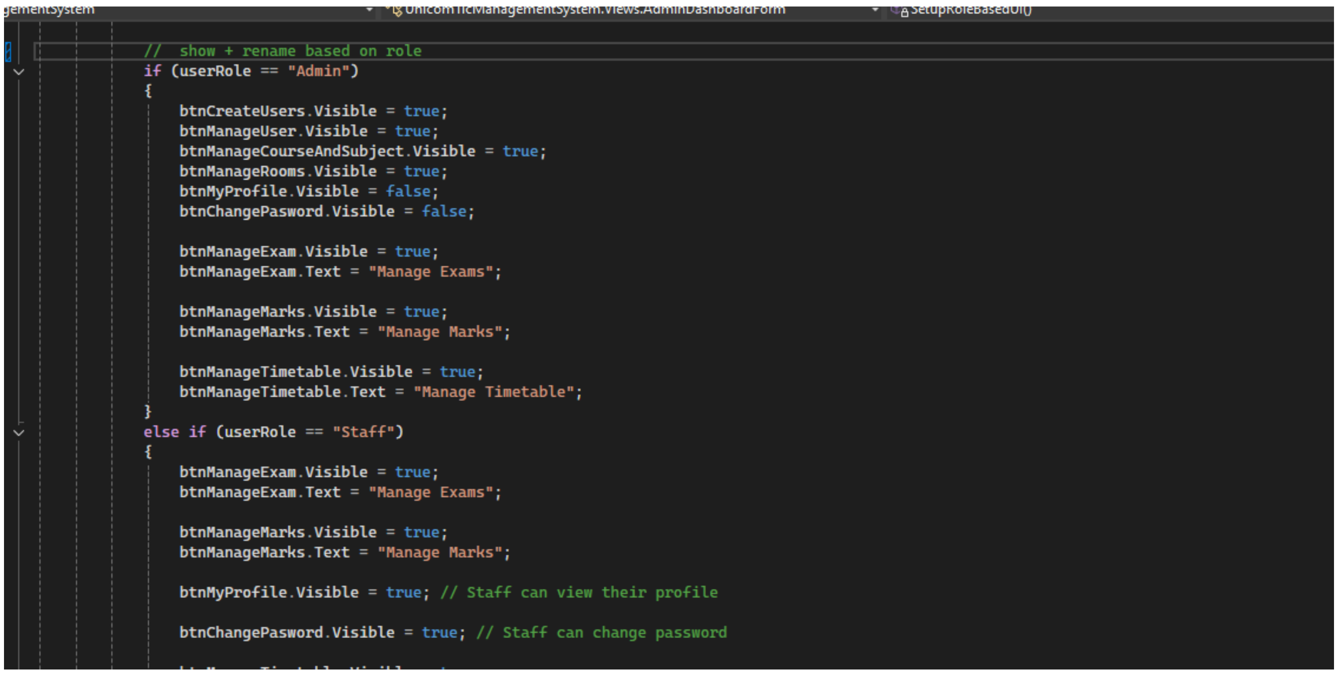
Auto-generated passwords were hard to remember for users.
Solution: I added a "Change Password" feature available on each user's dashboard.
Rules include:

- Must be 6–12 characters
- Must contain letters + numbers
- Old and new passwords cannot be the same
- Username remains unchangeable to prevent duplicate accounts



```
//SystemDefaults.SeedDefaultCoursesAndSubjects(conn);

//// Check if any admin user exists
string checkAdminQuery = "SELECT COUNT(*) FROM Users WHERE Role = 'Admin'";
using var cmd = new SQLiteCommand(checkAdminQuery, conn);
long adminCount = (long)cmd.ExecuteScalar();

if (adminCount == 0)
{
    // No admin exists ? open first-time admin setup
    Application.Run(new FirstAdminSignupForm());
}
else
{
    // Admin exists ? open normal login
    Application.Run(new LoginForm());
}
        }
    }
}
```

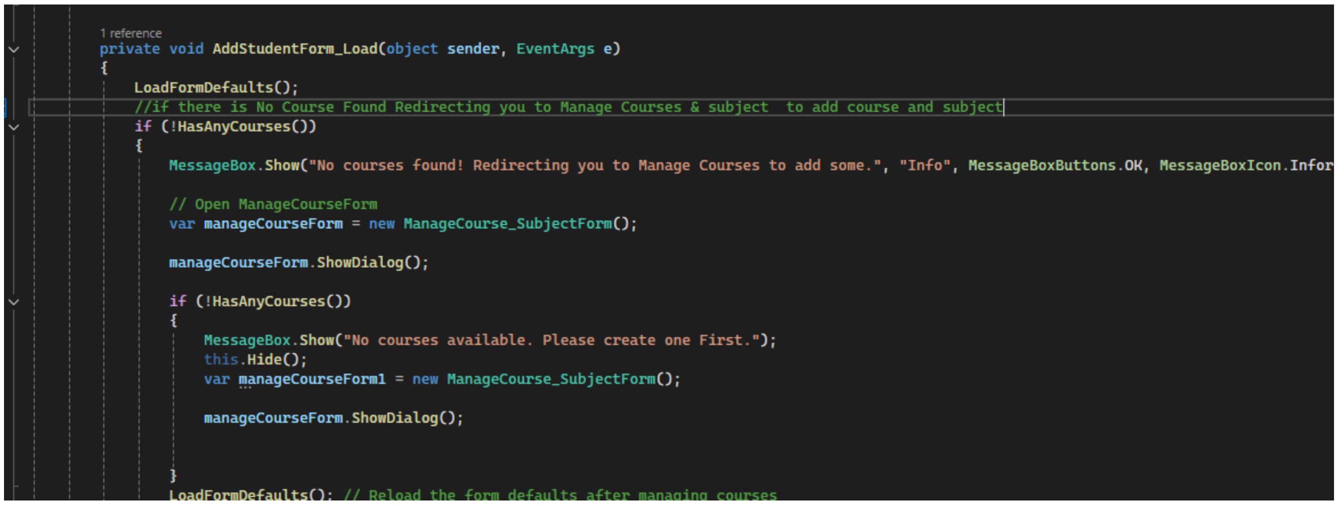
First Admin Sign Up – When the project is launched for the first time, the system detects if there are no users and shows the *First Admin SignUp* form. This form appears only once. After that, login form is shown by default. Includes **password validation**: 6–12 characters, mix of letters and numbers.



```
//  show + rename based on role
if (userRole == "Admin")
{
    btnCreateUsers.Visible = true;
    btnManageUser.Visible = true;
    btnManageCourseAndSubject.Visible = true;
    btnManageRooms.Visible = true;
    btnMyProfile.Visible = false;
    btnChangePasword.Visible = false;

    btnManageExam.Visible = true;
    btnManageExam.Text = "Manage Exams";

    btnManageMarks.Visible = true;
    btnManageMarks.Text = "Manage Marks";

    btnManageTimetable.Visible = true;
    btnManageTimetable.Text = "Manage Timetable";
}
else if (userRole == "Staff")
{
    btnManageExam.Visible = true;
    btnManageExam.Text = "Manage Exams";

    btnManageMarks.Visible = true;
    btnManageMarks.Text = "Manage Marks";

    btnMyProfile.Visible = true; // Staff can view their profile

    btnChangePasword.Visible = true; // Staff can change password
```

Dashboard Visibility Based on Role

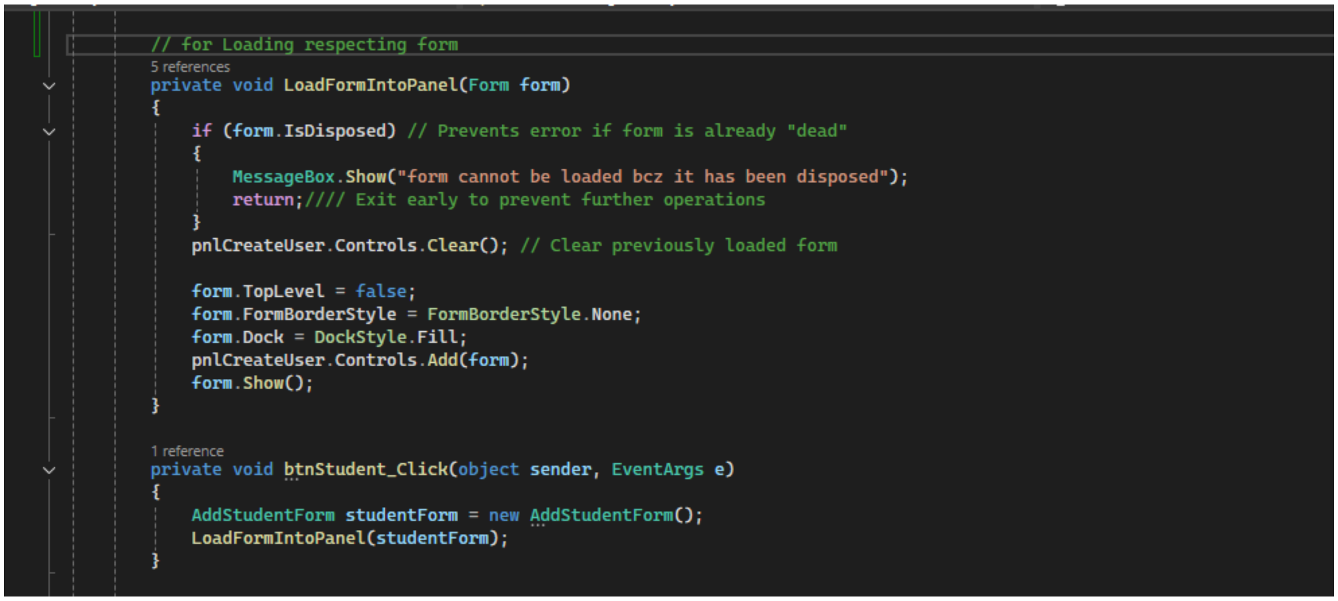
The dashboard dynamically adjusts its visible buttons depending on the logged-in user's role.
For example, Students only see profile and marks buttons, while Admins see all management panels.
This ensures clarity, reduces UI clutter, and improves overall security by restricting access.



```
1 reference
private void AddStudentForm_Load(object sender, EventArgs e)
{
    LoadFormDefaults();
    //if there is No Course Found Redirecting you to Manage Courses & subject  to add course and subject
    if (!HasAnyCourses())
    {
        MessageBox.Show("No courses found! Redirecting you to Manage Courses to add some.", "Info", MessageBoxButtons.OK, MessageBoxIcon.Infor

        // Open ManageCourseForm
        var manageCourseForm = new ManageCourse_SubjectForm();

        manageCourseForm.ShowDialog();

        if (!HasAnyCourses())
        {
            MessageBox.Show("No courses available. Please create one First.");
            this.Hide();
            var manageCourseForm1 = new ManageCourse_SubjectForm();

            manageCourseForm.ShowDialog();


        }
        LoadFormDefaults(); // Reload the form defaults after managing courses
```

Redirecting to Course & Subject Management if No Course Exists

Before creating users like Students or Lecturers, the system checks if any course exists using HasanyCourses().If no course is found, it automatically opens the ManageCourse_SubjectForm() to prompt the admin to create one.This ensures proper data flow and prevents assigning users to undefined courses.

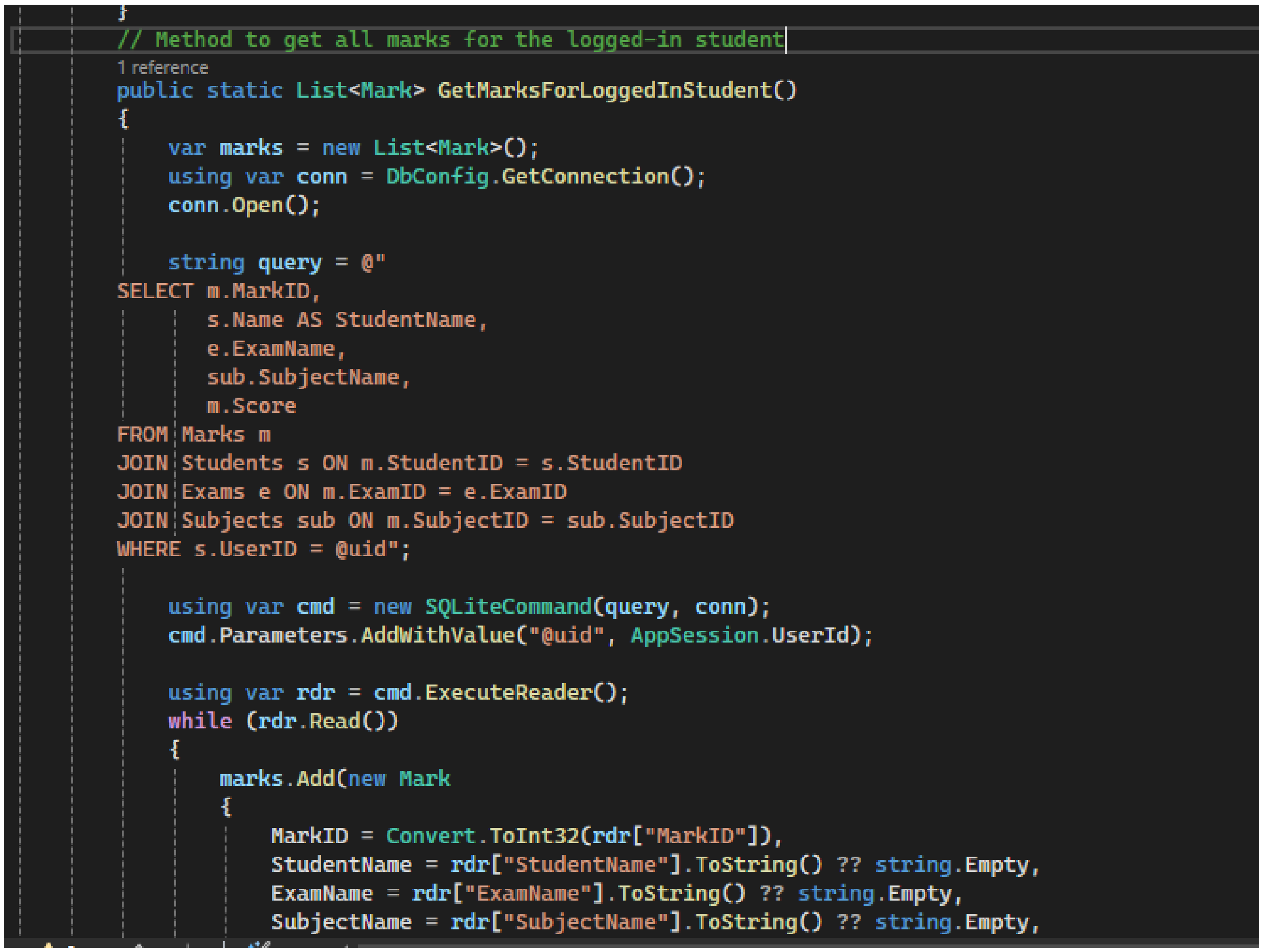


```
        // for Loading respecting form
        5 references
        private void LoadFormIntoPanel(Form form)
        {
            if (form.IsDisposed) // Prevents error if form is already "dead"
            {
                MessageBox.Show("form cannot be loaded bcz it has been disposed");
                return;//// Exit early to prevent further operations
            }
            pnlCreateUser.Controls.Clear(); // Clear previously loaded form

            form.TopLevel = false;
            form.FormBorderStyle = FormBorderStyle.None;
            form.Dock = DockStyle.Fill;
            pnlCreateUser.Controls.Add(form);
            form.Show();
        }

        1 reference
        private void btnStudent_Click(object sender, EventArgs e)
        {
            AddStudentForm studentForm = new AddStudentForm();
            LoadFormIntoPanel(studentForm);
        }
```

Dynamic Form Loading on Button Click

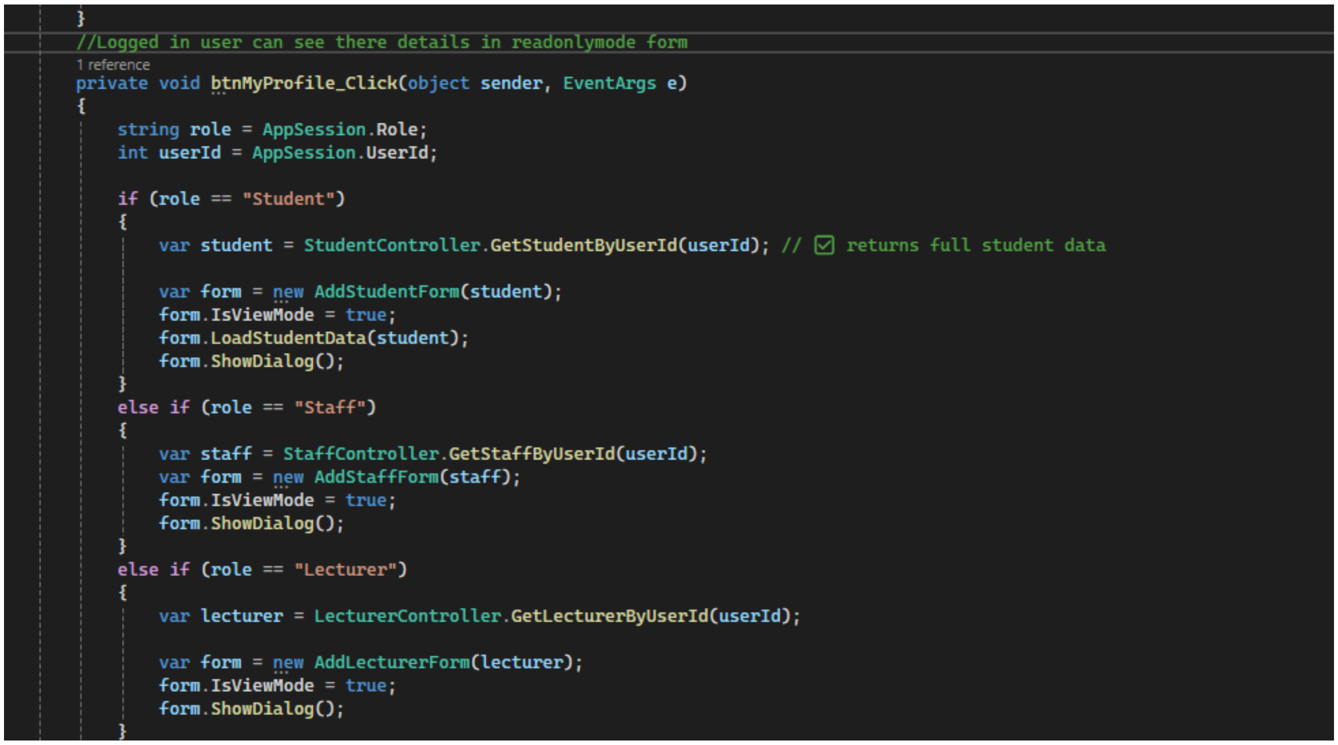
The system uses a method LoadFormIntoPannel() to dynamically embed forms inside a panel, ensuring a single-window experience.

For example, in CreateUserPannel(), clicking on “Student” loads AddStudentForm() into the main panel. This improves UI consistency and modularity.



```
}
// Method to get all marks for the logged-in student
1 reference
public static List<Mark> GetMarksForLoggedInStudent()
{
    var marks = new List<Mark>();
    using var conn = DbConfig.GetConnection();
    conn.Open();

    string query = @"
SELECT m.MarkID,
       s.Name AS StudentName,
       e.ExamName,
       sub.SubjectName,
       m.Score
FROM Marks m
JOIN Students s ON m.StudentID = s.StudentID
JOIN Exams e ON m.ExamID = e.ExamID
JOIN Subjects sub ON m.SubjectID = sub.SubjectID
WHERE s.UserID = @uid";

    using var cmd = new SQLiteCommand(query, conn);
    cmd.Parameters.AddWithValue("@uid", AppSession.UserId);

    using var rdr = cmd.ExecuteReader();
    while (rdr.Read())
    {
        marks.Add(new Mark
        {
            MarkID = Convert.ToInt32(rdr["MarkID"]),
            StudentName = rdr["StudentName"].ToString() ?? string.Empty,
            ExamName = rdr["ExamName"].ToString() ?? string.Empty,
            SubjectName = rdr["SubjectName"].ToString() ?? string.Empty,
```

Fetching Logged-In Student's Marks Securely
This method retrieves all marks for the currently logged-in student by filtering results using their userId from session data.
It joins multiple related tables (`Marks`, `Students`, `Exams`, and `Subjects`) to display rich details such as subject name, exam name, and score.
This ensures students only view their own academic records, maintaining both privacy and clarity.



```
        }
        //Logged in user can see there details in readonlymode form
        1 reference
        private void btnMyProfile_Click(object sender, EventArgs e)
        {
            string role = AppSession.Role;
            int userId = AppSession.UserId;

            if (role == "Student")
            {
                var student = StudentController.GetStudentByUserId(userId); // ✅ returns full student data

                var form = new AddStudentForm(student);
                form.IsViewMode = true;
                form.LoadStudentData(student);
                form.ShowDialog();
            }
            else if (role == "Staff")
            {
                var staff = StaffController.GetStaffByUserId(userId);
                var form = new AddStaffForm(staff);
                form.IsViewMode = true;
                form.ShowDialog();
            }
            else if (role == "Lecturer")
            {
                var lecturer = LecturerController.GetLecturerByUserId(userId);

                var form = new AddLecturerForm(lecturer);
                form.IsViewMode = true;
                form.ShowDialog();
            }
```

View Profile in Read-Only Mode for Logged-In User
This method allows the logged-in user (Student, Staff, or Lecturer) to view their own profile using the shared Add/Edit form, with all input fields locked via isViewMode= true

Note: After creating a user make sure to note down the username and password exactly as shown. They are case sensitive and will be needed for login. This help with testing and avoids confusion later.