UnicomTic management system project

User Role-Based Authentication:

- Admin, Teacher, and Student login with role-based UI control.
- Admin has full CRUD access to all data.

CRUD Operations for All Entities:

- Add, Update, Delete, and View for:
    - Students
    - Teachers
    - Admins
    - Courses
    - Subjects
    - Exams

Database Management with SQLite:

- Relational database structure with foreign key constraints.
- Transaction-based safe inserts and updates.

Inheritance for Cleaner Code

- A shared userbase class is used for all user types to reduce redundancy.
- MVC-like structure:
    - Forms (UI)
    - Models (Data classes) Controllers (DB logic)

1.Challenge: Role-Based User Login

- Implementing different dashboards and UI behavior for Admin, Teacher, and Student based on login.

2. Challenge: SQL Query Errors

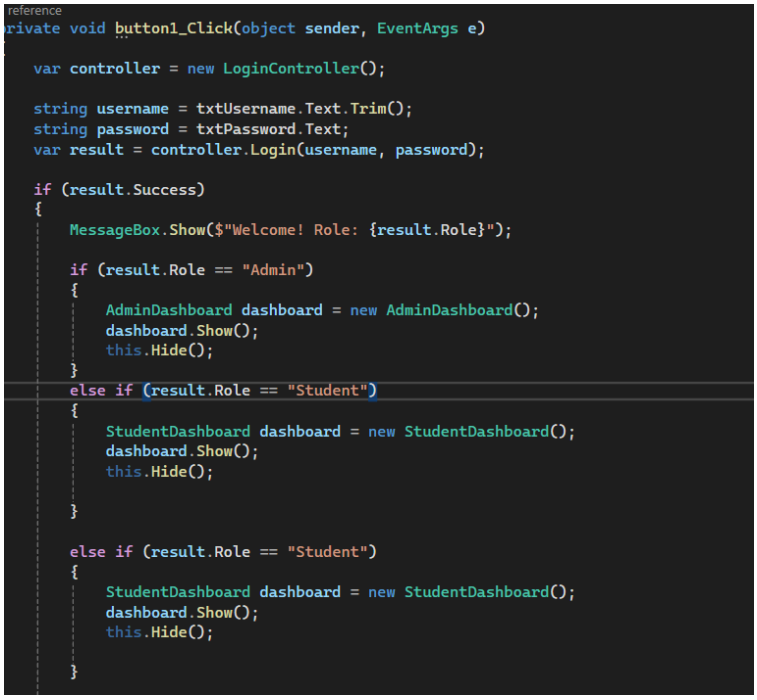
- Incorrect joins between `Users`, `Students`, `Teachers`, `Subjects`, etc.
- Foreign key mismatch or null reference errors.

3.Challenge: Windows Forms UI Not Reflecting Backend

- DataGridView not refreshing after insert/update.
- ComboBox not showing database values.
- `ExecuteNonQuery()` not working due to open connection or missing transactions.

Code Sample

1. **Role-Based Login Logic**
   - **Role-based login verifying credentials and directing to appropriate dashboard (Admin, Teacher, or Student).**



```
reference
rivate void button1_Click(object sender, EventArgs e)
{
    var controller = new LoginController();

    string username = txtUsername.Text.Trim();
    string password = txtPassword.Text;
    var result = controller.Login(username, password);

    if (result.Success)
    {
        MessageBox.Show($"Welcome! Role: {result.Role}");

        if (result.Role == "Admin")
        {
            AdminDashboard dashboard = new AdminDashboard();
            dashboard.Show();
            this.Hide();
        }
        else if (result.Role == "Student")
        {
            StudentDashboard dashboard = new StudentDashboard();
            dashboard.Show();
            this.Hide();

        }

        else if (result.Role == "Student")
        {
            StudentDashboard dashboard = new StudentDashboard();
            dashboard.Show();
            this.Hide();

        }
```

   2. Inheritance in Models (User.cs → Student.cs, Teacher.cs)
   - Using inheritance to reduce code duplication and maintain shared user properties.

```
namespace UnicomTicManagement.Model
{
    9 references
    public class Student : User
    {
        4 references
        public int StudentId { get; set; }
        7 references
        public string StudentNumber { get; set; }
        7 references
        public int CourseId { get; set; }
        1 reference
        public string CourseName { get; set; }
    }
}
```

```
namespace UnicomTicManagement.Model
{
    5 references
    public class User
    {
        13 references
        public int UserId { get; set; }
        22 references
        public string Name { get; set; }
        22 references
        public string Username { get; set; }
        22 references
        public string Password { get; set; }
        22 references
        public string Email { get; set; }
        14 references
        public string Role { get; set; }
    }
}
```

3. StudentController: Add, Update, Delete

```
0 references
public string AddStudent(Student student)
{
    try
    {
        using (var conn = DbCon.GetConnection())
        using (var transaction = conn.BeginTransaction())
        {

            string insertUserQuery = @"
            INSERT INTO Users (Name, Username, Password, Email, Role)
            VALUES (@Name, @Username, @Password, @Email, @Role);
            SELECT last_insert_rowid();";

            long userId;
            using (var cmd = new SQLiteCommand(insertUserQuery, conn))
            {
                cmd.Parameters.AddWithValue("@Name", student.Name);
                cmd.Parameters.AddWithValue("@Username", student.Username);
                cmd.Parameters.AddWithValue("@Password", student.Password);
                cmd.Parameters.AddWithValue("@Email", student.Email);
                cmd.Parameters.AddWithValue("@Role", student.Role);

                userId = (long)cmd.ExecuteScalar();
            }
```

- **StudentController with full CRUD operations using SQLite transactions and foreign keys.**

4..ER Diagram Relationship with database

```
1 reference
public static void CreateTables()
{
    using (var conn = DbCon.GetConnection())
    {
        var cmd = conn.CreateCommand();
        cmd.CommandText = @"
    CREATE TABLE IF NOT EXISTS Users (
        UserId INTEGER PRIMARY KEY AUTOINCREMENT,
        Name TEXT NOT NULL,
        Username TEXT UNIQUE NOT NULL,
        Password TEXT NOT NULL,
        Email TEXT,
        Role TEXT NOT NULL
    );

    CREATE TABLE IF NOT EXISTS Courses (
        CourseId INTEGER PRIMARY KEY AUTOINCREMENT,
        CourseName TEXT NOT NULL
    );

    CREATE TABLE IF NOT EXISTS Subjects (
        SubjectId INTEGER PRIMARY KEY AUTOINCREMENT,
        SubjectName TEXT NOT NULL,
        CourseId INTEGER NOT NULL,
        FOREIGN KEY(CourseId) REFERENCES Courses(CourseId)
    );

    CREATE TABLE IF NOT EXISTS Students (
        StudentId INTEGER PRIMARY KEY AUTOINCREMENT,
        UserId INTEGER NOT NULL,
        StudentNumber TEXT UNIQUE NOT NULL,
        CourseId INTEGER,
        FOREIGN KEY(UserId) REFERENCES Users(UserId),
```