Unicom TIC Management System - Project Report

1. Project Overview

✨ Key Features Implemented

- • Role-based login system for Admin, Staff, Students, and Lecturers
- • Course & Subject management (add/edit/delete courses like "BSc Computer Science")
- • Student records with enrollment in courses
- • Exam marks tracking for students
- • Timetable scheduling with room allocation (computer labs or lecture halls)
- • Simple database using SQLite to store all information

🛠️ Technologies Used

- • C# with Windows Forms (.NET Framework)
- • SQLite database (with System.Data.SQLite NuGet package)
- • MVC-like architecture (separating Models, Views, and Controllers)
- • Visual Studio 2022 as our development environment

😅 Challenges & Solutions

Problem	How We Solved It
Making the login system work	Created a Users table and validated username/password (plain text for now)
Showing different dashboards for different roles	Used if-else to show/hide buttons based on user role
Connecting to SQLite database	Made a DatabaseManager class to handle all database operations
Room allocation in timetable	Added a dropdown to select between labs and lecture halls

2. Code Samples (Screenshots)

Login Verification

```
1 reference
private void btnLogin_Click(object sender, EventArgs e)
{
    string username = txtUsername.Text.Trim();
    string password = txtPassword.Text;

    // Admin hardcoded login
    if (username == "admin" && password == "Admin@123")
    {
        MessageBox.Show("Admin login successful!");
        var adminDashboard = new AdminDashboardForm();
        adminDashboard.Show();
        this.Hide();
        return;
    }

    // Try student login
    Student student = AuthenticateStudent(username, password);
    if (student != null)
    {
        var connection = new SQLiteConnection(connectionString);
        connection.Open();

        this.Hide();
        var studentDashboard = new StudentDashboardForm(student, connection);
        studentDashboard.Show();
        return;
    }

    // Try lecturer login
    Lecturer lecturer = AuthenticateLecturer(username, password);
    if (lecturer != null)
    {
        var connection = new SQLiteConnection(connectionString);
        connection.Open();

        this.Hide();
        var lecturerDashboard = new LecturerDashboardForm(lecturer, connection);
        lecturerDashboard.Show();
        return;
    }

    // Try staff login
    Staff staff = AuthenticateStaff(username, password);
    if (staff != null)
```

- • Role-checking logic
- • Secure form transitions
- • Plain-text password validation (for beginner project)

Adding a New Course

```
4 references
private void LoadCourses()
{
    dgvCourses.DataSource = null;
    dgvCourses.DataSource = courseController.GetAllCourses();
}

3 references
private void ClearInputs()
{
    txtCourses.Clear();
}

0 references
private void btnAdd_Click(object sender, EventArgs e)
{
    if (!string.IsNullOrWhiteSpace(txtCourses.Text))
    {
        var course = new Course
        {
            CourseName = txtCourses.Text.Trim()
        };
        courseController.AddCourse(course);
        LoadCourses();
        ClearInputs();
    }
}

0 references
private void btnUpdate_Click(object sender, EventArgs e)
{
    if (dgvCourses.SelectedRows.Count > 0)
    {
        var course = new Course
        {
            CourseID = Convert.ToInt32(dgvCourses.SelectedRows[0].Cells["CourseID"].Value),
            CourseName = txtCourses.Text.Trim()
        };
        courseController.UpdateCourse(course);
        LoadCourses();
        ClearInputs();
```

- • Input validation (empty checks)
- • Database binding with CourseController
- • Instant UI refresh after submission

Timetable with Room Allocation

```
4 references
private void LoadTimetables()
{
    var timetables = timetableController.GetAllTimetables();

    // For displaying related names, join with loaded courses, subjects, lecturers
    var table = new DataTable();
    table.Columns.Add("TimetableID", typeof(int));
    table.Columns.Add("Course", typeof(string));
    table.Columns.Add("Subject", typeof(string));
    table.Columns.Add("Lecturer", typeof(string));
    table.Columns.Add("Date", typeof(string));
    table.Columns.Add("TimeSlot", typeof(string));
    table.Columns.Add("Location", typeof(string));

    foreach (var t in timetables)
    {
        var courseName = courses.FirstOrDefault(c => c.CourseID == t.CourseID)?.CourseName ?? ""
        var subjectName = subjects.FirstOrDefault(s => s.SubjectID == t.SubjectID)?.SubjectName
        var lecturerName = lecturers.FirstOrDefault(l => l.UserID == t.LecturerID)?.Username ??

        table.Rows.Add(t.TimetableID, courseName, subjectName, lecturerName,
            t.Date.ToString("yyyy-MM-dd"), t.TimeSlot, t.Location);
    }

    dgvTimetables.DataSource = table;

    dgvTimetables.Columns["TimetableID"].Visible = false; // hide PK
}

// When Course changes, reload Subjects for that Course
0 references
private void cmbCourse_SelectedIndexChanged(object sender, EventArgs e)
{
    if (cmbCourse.SelectedIndex >= 0)
    {
        int selectedCourseId = (int)cmbCourse.SelectedValue;
        LoadSubjects(selectedCourseId);
    }
    else
    {
        cmbSubject.DataSource = null;
    }
}
```

- • Dynamic dropdowns for room selection
- • DataGridView joins with course/lecturer data
- • Clear room type display (📄 Lab / 🏛️ Hall)

Database Connection (SQLite)

```
private const string DatabaseFile = "DBconnection.db";  // Your DB file name
private const string ConnectionString = "Data Source=" + DatabaseFile + ";Version=3;";

1 reference
public static string GetConnectionString()
{
    return ConnectionString;
}

// Returns an OPEN SQLiteConnection
23 references
public static SQLiteConnection GetConnection()
{
    var connection = new SQLiteConnection(ConnectionString);
    connection.Open();
    return connection;
}
```

- • Singleton pattern for connections
- • using statements for auto-disposal
- • Connection string management

Student Dashboard

```
    1 reference
    private void StudentDashboardForm_Load(object sender, EventArgs e)
    {
        LoadMarks();
        LoadTimetable();

    }

    1 reference
    private void LoadMarks()
    {
        var marksRepo = new MarksRepository(_connection);
        List<Marks> marks = marksRepo.GetMarksByStudentId(_student.StudentId);
        dgvMarks.DataSource = marks;
    }

    1 reference
    private void LoadTimetable()
    {
        var timetableRepo = new TimetableRepository(_connection);
        List<Timetable> timetable = timetableRepo.GetTimetableByCourseId(_student.CourseId);
        dgvTimetable.DataSource = timetable;
    }
}
```

- • Marks display with grade calculation

- • Timetable sorting (by day/timeslot)
- • Student-specific data filtering

Input Validation

```
// Try student login
Student student = AuthenticateStudent(username, password);
if (student != null)
{
    var connection = new SQLiteConnection(connectionString);
    connection.Open();

    this.Hide();
    var studentDashboard = new StudentDashboardForm(student, connection
    studentDashboard.Show();
    return;
}

// Try lecturer login
Lecturer lecturer = AuthenticateLecturer(username, password);
if (lecturer != null)
{
    var connection = new SQLiteConnection(connectionString);
    connection.Open();

    this.Hide();
    var lecturerDashboard = new LecturerDashboardForm(lecturer, connect
    lecturerDashboard.Show();
    return;
}

// Try staff login
Staff staff = AuthenticateStaff(username, password);
if (staff != null)
{
    var connection = new SQLiteConnection(connectionString);
    connection.Open();

    this.Hide();
    var staffDashboard = new StaffDashboardForm(staff, connection);
    staffDashboard.Show();
    return;
}
```

- • Null/empty field handling
- • Score range validation (0–100)

- • User-friendly error messages

Final Thoughts

- • This was our first time building a desktop application with C# and SQLite!
- • We learned how to design forms using Windows Forms.
- • We understood basic database operations (add, edit, delete records).
- • We saw the importance of organizing code (MVC pattern).
- • We implemented role-based permissions.
- • Future improvements include adding password hashing and room conflict checking.

🔗 GitHub Repository: [https://github.com/Divikshan/Divikshan-School-management-system.git]