Table of Contents

Unicom TIC Management System - Project Submission Report

1. Project Overview (Point-Wise)

Key Features Implemented:

- Login System with role-based access for Admin, Staff, Students, and Lecturers.
- Dashboard with visible modules depending on user role.
- Exam & Marks Management (Admin/Staff/Lecturer: Manage exams and marks; Students: View only).
- Student Management with CRUD operations for Admin.
- Timetable Management with Lab/Hall Allocation (Admin: Add/Edit/Delete; Others: View only).
- SQLite database with relationships between tables.
- MVC Architecture (Models, Views, Controllers separated).
- Role-based dashboards that restrict access to appropriate features.
- Error messages and input validation included.
- Simple and intuitive WinForms user interface.

Technologies Used:

- Programming Language: C#

- Framework: WinForms (.NET Framework)
- Database: SQLite using System.Data.SQLite
- Architecture: MVC (Model–View–Controller)
- IDE: Visual Studio
- Design Patterns: MVC
- UI Elements: Buttons, ComboBoxes, DataGridViews, TextBoxes, Labels,panel

Challenges Faced and Solutions:

- Role-Based Dashboard Visibility: Implemented dashboard UI filtering based on logged-in user role.
- Timetable room assignment: Used Combo Box with room filtering and error-checking logic.
- Login Validation: Added a simple user validation system querying the Users table and managing sessions based on roles.
- Database Table Creation: Implemented Migration.cs to auto-create tables if they don’t exist.
- Navigation Between Forms: Used controller logic to switch panels dynamically upon button clicks instead of opening new forms.

2. Code Samples (Screenshots)

Create table and default admin username: admin and password:admin123

```
namespace UnicomTICManagementSystem.Repositories
{

    1 reference
    internal static class Migration
    {
        1 reference
        public static void CreateTable()
        {
            using (var connect = DatabaseManager.DatabaseConnect())
            {
                // SQL statements to create tables if they do not exist
                string Query = @"
                    -- Users table for authentication
                    CREATE TABLE IF NOT EXISTS Users(
                        ID INTEGER PRIMARY KEY AUTOINCREMENT,
                        UserName TEXT NOT NULL,
                        Password TEXT NOT NULL,
                        Gmail TEXT NOT NULL,
                        CreateDate TEXT DEFAULT CURRENT_TIMESTAMP,
                        UpdateDate TEXT DEFAULT CURRENT_TIMESTAMP
                    );

                    -- Departments table to organize courses and subjects
                    CREATE TABLE IF NOT EXISTS Departments(
                        ID INTEGER PRIMARY KEY AUTOINCREMENT,
                        Name TEXT NOT NULL,
                        Description TEXT NOT NULL,
```

Get Student List Method

```
2 references
public List<Models.Student> GetStudent(int studentId)
{
    List<Models.Student> students = new List<Models.Student>();

    using (SQLiteConnection connect = DatabaseManager.DatabaseConnect())
    {
        string studentQuery = "SELECT * FROM Students WHERE ID = @StudentID;";
        SQLiteDataAdapter adapter = new SQLiteDataAdapter(studentQuery, connect);
        adapter.SelectCommand.Parameters.AddWithValue("@StudentID", studentId);

        DataTable datatable = new DataTable();
        adapter.Fill(datatable);

        foreach (DataRow row in datatable.Rows)
        {
            Models.Student student = new Models.Student
            {
                ID = Convert.ToInt32(row["ID"]),
                FirstName = row["FirstName"].ToString(),
                LastName = row["LastName"].ToString(),
                DateOfBirth = Convert.ToDateTime(row["DateOfBirth"]),
                Gender = row["Gender"].ToString(),
                Nationality = row["Nationality"].ToString(),
                NICno = row["NICno"].ToString(),
                Gmail = row["Gmail"].ToString(),
                PhoneNumber = row["PhoneNumber"].ToString(),
                Address = row["Address"].ToString(),
                AdmissionDate = Convert.ToDateTime(row["AdmissionDate"]),
                FatherName = row["FatherName"].ToString(),
                MotherName = row["MotherName"].ToString(),
                ParentsPhoneNumber = row["ParentsPhoneNumber"].ToString(),
                DepartmentsID = Convert.ToInt32(row["DepartmentsID"]),
                CoursesID = Convert.ToInt32(row["CoursesID"]),
                UsersID = Convert.ToInt32(row["UsersID"])
            };
            students.Add(student);
        }
    }
    return students;
}
```

Update Student Method

```
// Update existing student record in database
1 reference
public void UpdateStudentInDatabase(Models.Student student)
{
    using (var connect = DatabaseManager.DatabaseConnect())
    {
        string updateQuery = @"
            UPDATE Students SET
                FirstName = @FirstName,
                LastName = @LastName,
                DateOfBirth = @DateOfBirth,
                Gender = @Gender,
                Nationality = @Nationality,
                NICno = @NICno,
                Gmail = @Gmail,
                PhoneNumber = @PhoneNumber,
                Address = @Address,
                FatherName = @FatherName,
                MotherName = @MotherName,
                ParentsPhoneNumber = @ParentsPhoneNumber
            WHERE ID = @ID";

        using (var command = new SQLiteCommand(updateQuery, connect))
        {
            command.Parameters.AddWithValue("@FirstName", student.FirstName);
            command.Parameters.AddWithValue("@LastName", student.LastName);
            command.Parameters.AddWithValue("@DateOfBirth", student.DateOfBirth);
            command.Parameters.AddWithValue("@Gender", student.Gender);
            command.Parameters.AddWithValue("@Nationality", student.Nationality);
            command.Parameters.AddWithValue("@NICno", student.NICno);
            command.Parameters.AddWithValue("@Gmail", student.Gmail);
            command.Parameters.AddWithValue("@PhoneNumber", student.PhoneNumber);
            command.Parameters.AddWithValue("@Address", student.Address);
            command.Parameters.AddWithValue("@FatherName", student.FatherName);
            command.Parameters.AddWithValue("@MotherName", student.MotherName);
            command.Parameters.AddWithValue("@ParentsPhoneNumber", student.ParentsPhoneNumber);
            command.Parameters.AddWithValue("@ID", student.ID);

            command.ExecuteNonQuery();
        }
    }
}
```

Add New Subject

```
1 reference
public int CreateSubject(Subject subject)
{
    // Check if essential fields are filled
    if (!string.IsNullOrWhiteSpace(subject.Name) && !string.IsNullOrWhiteSpace(subject.Credit))
    {
        using (SQLiteConnection connect = DatabaseManager.DatabaseConnect())
        {
            // SQL insert query to add new subject
            string subjectQuery = @"INSERT INTO Subjects(Name, Credit, DepartmentsID)
                                    VALUES(@name, @credit, @departmentsid);";
            try
            {
                using (SQLiteCommand command = new SQLiteCommand(subjectQuery, connect))
                {
                    // Bind parameters safely to avoid SQL injection
                    command.Parameters.AddWithValue("@name", subject.Name);
                    command.Parameters.AddWithValue("@credit", subject.Credit);
                    command.Parameters.AddWithValue("@departmentsid", subject.DepartmentsID);

                    // Execute insert command
                    command.ExecuteNonQuery();

                    MessageBox.Show($"Subject {subject.Name} Registered Successfully.");

                    // Retrieve last inserted row ID for this subject
```

```
                    command.CommandText = "SELECT last_insert_rowid();";
                    command.Parameters.Clear();
                    return Convert.ToInt32(command.ExecuteScalar());
                }
            }
            catch (Exception ex)
            {
                // Show error message on failure
                MessageBox.Show($"Error saving subject:\n{ex.Message}", "Subject Save Failed",
                return -1;
            }
        }
    }
    else
    {
        // Alert user if required fields are missing
        MessageBox.Show("Fill Subject Details");
        return -1;
    }
}

1 reference
```

ExamController.cs

```
1 reference
public List<Exam> GetExamsBySubjectId(int subjectId)
{
    var exams = new List<Exam>();
    using (SQLiteConnection connect = DatabaseManager.DatabaseConnect())
    {
        string query = "SELECT * FROM Exams WHERE SubjectsID = @SubjectsID";
        using (SQLiteCommand command = new SQLiteCommand(query, connect))
        {
            command.Parameters.AddWithValue("@SubjectsID", subjectId);
            using (SQLiteDataReader reader = command.ExecuteReader())
            {
                while (reader.Read())
                {
                    exams.Add(new Exam
                    {
                        ID = Convert.ToInt32(reader["ID"]),
                        ExamName = reader["ExamName"].ToString(),
                        SubjectsID = Convert.ToInt32(reader["SubjectsID"])
                    });
                }
            }
        }
    }
    return exams;
```

Delete Student Method

```
// Delete student record and associated user
1 reference
public void DeleteStudent(Models.Student student)
{
    using (SQLiteConnection connect = DatabaseManager.DatabaseConnect())
    {
        using (var transaction = connect.BeginTransaction())
        {
            try
            {
                // Delete from Students table
                string deleteStudentQuery = "DELETE FROM Students WHERE ID = @Id";
                SQLiteCommand deleteStudentCmd = new SQLiteCommand(deleteStudentQuery, connect);
                deleteStudentCmd.Parameters.AddWithValue("@Id", student.ID);
                deleteStudentCmd.ExecuteNonQuery();

                // Delete from Users table
                string deleteUserQuery = "DELETE FROM Users WHERE ID = @UserId";
                SQLiteCommand deleteUserCmd = new SQLiteCommand(deleteUserQuery, connect);
                deleteUserCmd.Parameters.AddWithValue("@UserId", student.UsersID);
                deleteUserCmd.ExecuteNonQuery();

                transaction.Commit();

                MessageBox.Show("Student and associated user deleted successfully.");
            }
            catch (Exception ex)
            {
                transaction.Rollback();
                MessageBox.Show("Error while deleting: " + ex.Message);
            }
        }
    }
}
```

3. How the App Works

Login System: Authenticates users based on their role (Admin, Staff, Student, Lecturer).

Role Dashboards: Different dashboards are shown depending on the user role.

Data Operations: Add/Edit/Delete supported for Admin; restricted view-only for other roles.

Rooms & Timetable: Admin, Lecturer, Staff allocates labs/halls while scheduling; Students can only view.

Exam & Marks: Staff and Lecturers can update marks; students can only view their own.

4. Features

Filter (Courses, Subjects, Students, Exams, Rooms in dropdown list)

Error Messages (e.g., “Please select a room”, “Check your username or password”)

Input Validation (e.g., Score must be 0–100)

Descriptive Mappings for Viewing Data:

- Student name | Course name
- Exam name | Subject name
- Timetable – Subject name | Room name | Time slot
- Room name | Room type
- Subject name | Course name

5. Folder Structure Overview

UnicomTICManagementSystem/

- Models
  - Course.cs
  - Subject.cs
  - Student.cs
  - Exam.cs
  - Mark.cs
  - Room.cs
  - Timetable.cs
  - User.cs
  - Admin.cs
  - CourseSubject.cs
  - Department.cs
  - Lecturer.cs
  - LecturerSubject.cs
  - RoomSubject.cs
  - Staff.cs
  - Student.cs
  - StudentLecture.cs
  - StudentSubject.cs
  - Lecturer.cs

- Views
  - AdminRegisterForm.cs
  - CourseRegisterForm.cs
  - CreateEducationForm.cs
  - CreateUser.cs
  - Dashboard.cs
  - DepartmentRegister.cs
  - ExamMark.cs
  - ExamRegisterForm.cs
  - LectureRegisterForm.cs
  - Markregister.cs
  - RoomRegister.cs
  - Staffregister.cs
  - StudentRegisterForm.cs
  - SubjectRegisterForm.cs
  - UserRegisterForm.cs
  - UserSigninForm.cs
  - ViewExam.cs
  - ViewLectureDetails.cs
  - Viewmark.cs
  - ViewStaffDetails.cs
  - ViewStudentDetails.cs
  - ViewSubject.cs

- Controllers
  - LoginController.cs
  - AdminController.cs
  - CourseController.cs
  - CourseSubjectController.cs
  - DepartmentController.cs
  - ExamController.cs
  - LecturerController.cs
  - LecturerSubjectController.cs
  - MarkController.cs
  - RoomController.cs
  - RoomSubjectController.cs
  - StaffController.cs
  - StudentController.cs
  - StudentLecturerController.cs
  - StudentSubjectController.cs
  - SubjectController.cs
  - TimetableController.cs
  - UserController.cs
- Repositories
  - DatabaseManager.cs
  - Migration.cs
- Program.cs
- Service
  -

6. Conclusion

This project helped in learning:

- How to implement an MVC architecture in a desktop application.
- SQLite CRUD operations in C#.
- Role-based access control with a simple UI.
- Structuring a school management system from scratch.