Unicom TIC Management System - Project Report

1. Project Overview

This project developed a comprehensive desktop management system for Unicom TIC using C# Windows Forms and SQLite for data persistence. The system provides intuitive graphical user interfaces for various administrative and operational tasks, with a strong focus on modularity and role-based access control. This will be a helpful computer program for a college or institute.

1.1 Key Features Implemented:

- **Secure Login System:** User authentication with distinct roles (Admin, Staff, Lecturer, Student).
- **Main Dashboard:** Centralized access point with dynamic navigation based on logged-in user's role.
- **Management:** (Admin-only) Full CRUD operations for user accounts, including assigning roles.
- **Course & Subject Management:** CRUD operations for defining courses and their associated subjects.
- **Student Management:** Full CRUD operations for student records.
- **Exam Management:** CRUD operations for scheduling and managing exams, linking them to specific subjects.
- **Marks Management:** CRUD operations for recording student scores for specific exams.
- **Timetable Management:**
    1. **Room Allocation:** CRUD operations for managing computer labs and lecture halls, including room types and capacities.
    2. **Timetable Entry:** CRUD operations for scheduling classes, linking subjects, rooms, days, and times.
- **Role-Based Access Control (RBAC):** Granular permissions implemented across all modules, restricting UI elements (buttons, input fields) and data visibility based on the user's assigned role.

2. Technologies Used:

Programming Language: C#

Framework: .NET Framework (Windows Forms Application)

Database: SQLite (lightweight, file-based relational database)

Database Access: System.Data.SQLite NuGet package for ADO.NET interaction.

Architectural Pattern: Model-View-Controller (MVC) pattern for structured and maintainable code organization.

3. Challenges Faced and Solutions :

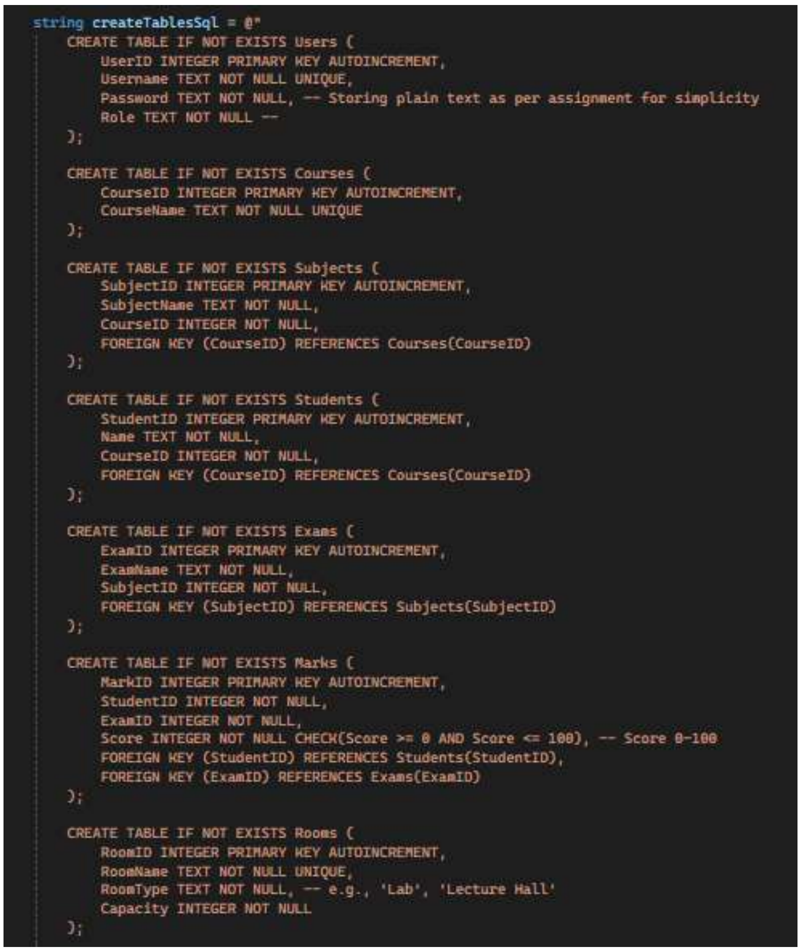
Challenge 1: When adding new tables (Rooms, TimetableEntries) or columns (Role to Users), previous database files (created with older schemas) would cause runtime "Specified cast is not valid" errors or "NOT NULL constraint failed" during initialization, especially for the Capacity and Role columns.

Solution: Implemented CREATE TABLE IF NOT EXISTS clauses, but the most reliable solution during development was explicit manual deletion of the unicomtic.db file from the bin/Debug/.Net directory, followed by a Clean and Rebuild of the solution. This forced a complete recreation of the database with the updated schemas and initial data.

Challenge 2: Systematically restricting UI elements (buttons, textboxes) and filtering data displayed in DataGridViews across multiple forms based on the logged-in user's role.

Solution: Established a clear Role-Based Permissions Matrix for Admin, Staff, Lecturer, and Student roles. Ensured User model included a Role property and database was updated with a Role column. Passed the authenticated User object (_currentUser) down the entire form from LoginForm to MainForm, then from MainForm to hub forms, and from hub forms to individual module forms.

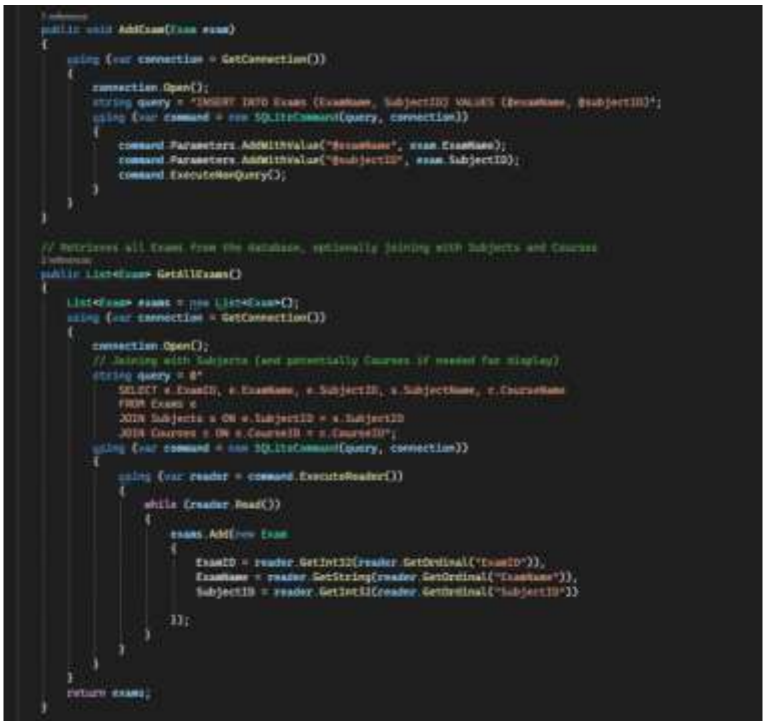
4. Screenshot of code and explainations :



```
string createTablesSql = @"
    CREATE TABLE IF NOT EXISTS Users (
        UserID INTEGER PRIMARY KEY AUTOINCREMENT,
        Username TEXT NOT NULL UNIQUE,
        Password TEXT NOT NULL, -- Storing plain text as per assignment for simplicity
        Role TEXT NOT NULL --
    );

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

    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,
        Name TEXT NOT NULL,
        CourseID INTEGER NOT NULL,
        FOREIGN KEY (CourseID) REFERENCES Courses(CourseID)
    );

    CREATE TABLE IF NOT EXISTS Exams (
        ExamID INTEGER PRIMARY KEY AUTOINCREMENT,
        ExamName TEXT NOT NULL,
        SubjectID INTEGER NOT NULL,
        FOREIGN KEY (SubjectID) REFERENCES Subjects(SubjectID)
    );

    CREATE TABLE IF NOT EXISTS Marks (
        MarkID INTEGER PRIMARY KEY AUTOINCREMENT,
        StudentID INTEGER NOT NULL,
        ExamID INTEGER NOT NULL,
        Score INTEGER NOT NULL CHECK(Score >= 0 AND Score <= 100), -- Score 0-100
        FOREIGN KEY (StudentID) REFERENCES Students(StudentID),
        FOREIGN KEY (ExamID) REFERENCES Exams(ExamID)
    );

    CREATE TABLE IF NOT EXISTS Rooms (
        RoomID INTEGER PRIMARY KEY AUTOINCREMENT,
        RoomName TEXT NOT NULL UNIQUE,
        RoomType TEXT NOT NULL, -- e.g., 'Lab', 'Lecture Hall'
        Capacity INTEGER NOT NULL
    );
```

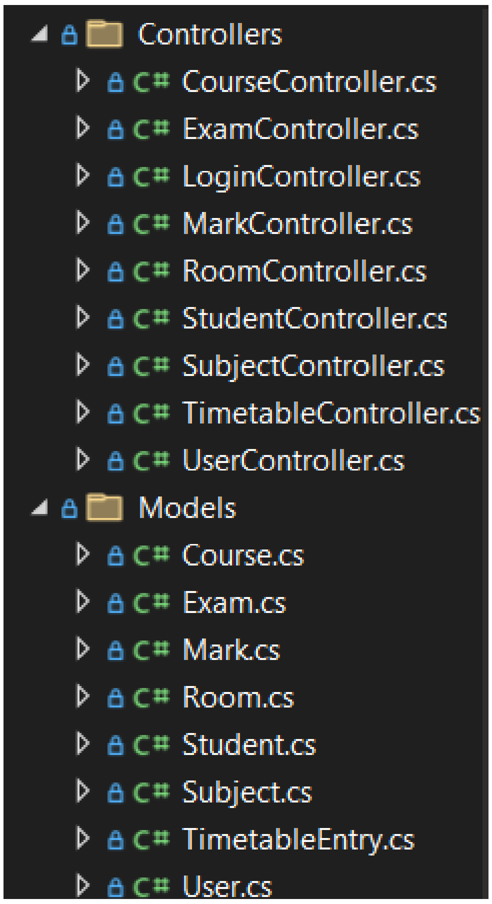
I put all sql query of the table creation together. It is easy an in the starting point when I call the connection part. It will create easily.

figure 1



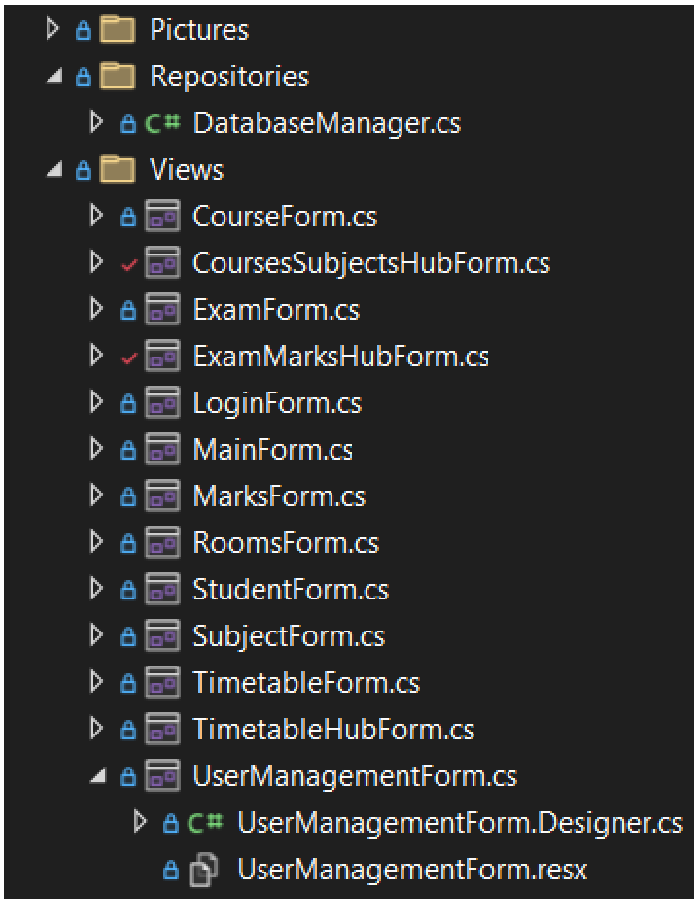
```
1 reference
public void AddExam(Exam exam)
{
    using (var connection = GetConnection())
    {
        connection.Open();
        string query = "INSERT INTO Exams (ExamName, SubjectID) VALUES (@examName, @subjectID)";
        using (var command = new SQLiteCommand(query, connection))
        {
            command.Parameters.AddWithValue("@examName", exam.ExamName);
            command.Parameters.AddWithValue("@subjectID", exam.SubjectID);
            command.ExecuteNonQuery();
        }
    }
}

// Retrieves all Exams from the database, optionally joining with Subjects and Courses
1 reference
public List<Exam> GetAllExams()
{
    List<Exam> exams = new List<Exam>();
    using (var connection = GetConnection())
    {
        connection.Open();
        // Joining with Subjects (and potentially Courses if needed for display)
        string query = @"
            SELECT e.ExamID, e.ExamName, e.SubjectID, s.SubjectName, c.CourseName
            FROM Exams e
            JOIN Subjects s ON e.SubjectID = s.SubjectID
            JOIN Courses c ON s.CourseID = c.CourseID";
        using (var command = new SQLiteCommand(query, connection))
        {
            using (var reader = command.ExecuteReader())
            {
                while (reader.Read())
                {
                    exams.Add(new Exam
                    {
                        ExamID = reader.GetInt32(reader.GetOrdinal("ExamID")),
                        ExamName = reader.GetString(reader.GetOrdinal("ExamName")),
                        SubjectID = reader.GetInt32(reader.GetOrdinal("SubjectID"))

                    });
                }
            }
        }
    }
    return exams;
}
```

The very actions that want to happen when press the button was wrote in databasemanager.cs as method



```
Controllers
    CourseController.cs
    ExamController.cs
    LoginController.cs
    MarkController.cs
    RoomController.cs
    StudentController.cs
    SubjectController.cs
    TimetableController.cs
    UserController.cs
Models
    Course.cs
    Exam.cs
    Mark.cs
    Room.cs
    Student.cs
    Subject.cs
    TimetableEntry.cs
    User.cs
```

figure 3



```
Pictures
Repositories
    DatabaseManager.cs
Views
    CourseForm.cs
    CoursesSubjectsHubForm.cs
    ExamForm.cs
    ExamMarksHubForm.cs
    LoginForm.cs
    MainForm.cs
    MarksForm.cs
    RoomsForm.cs
    StudentForm.cs
    SubjectForm.cs
    TimetableForm.cs
    TimetableHubForm.cs
    UserManagementForm.cs
        UserManagementForm.Designer.cs
        UserManagementForm.resx
```

figure 4

These figure 3 and 4 are my files with code. I tried and do my best and I learn MVC architecture and I did it as how it was on the required document.

5. Review

Developing this School Management System of Unicom TIC using C#, .NET, WinForms, and SQLite was a rewarding experience. I gained practical understanding of object-oriented programming concepts such as classes, methods, and constructors, and how to apply them effectively in real-world applications. I learned how to structure a Windows Forms application, manage data using SQLite, and implement role-based login functionality. Working on this project improved my confidence in designing user interfaces, handling events, and performing database operations. Overall, this project helped me connect theoretical knowledge with practical implementation and strengthened my skills in desktop application development using the .NET platform.