Project Guide

Introduction to Unicom Management System

The Unicom Management System is a considerable software solution designed to efficiently manage and streamline the day-to-day administrative and academic operations of a college. This system supports the organization in handling essential functions related to students, lecturers, and staff, and integrates several key modules such as:

- Course and Subject Management
- Class, Examination and Marks Management
- Timetable and Hall Allocation
- Attendance Tracking
- Student Information Management

Each module is interconnected to ensure seamless data flow and operational efficiency, reducing manual workload and improving accuracy across departments. With user-friendly interfaces and secure role-based access.

Functioning of System

The system includes a centralized dashboard accessible by the Admin, which provides access to all system features. Students have restricted access, allowing them to view only their personal details within the Student Management module. Lecturers can access modules related to Exams and Marks, Attendance Management, and Student Information for all students. Staff members have access to Timetable and Hall Management as well as Attendance Management.

The system operation begins with the Admin, who is responsible for creating and managing user accounts by assigning specific roles. This ensures that only registered users can access the system. When a new user is created with an assigned role by the Admin, they can then register their account and receive a username and password. Upon logging in, the system automatically detects the user's role and grants access permissions accordingly, ensuring that each user can only access features relevant to their role.

Steps of Function in the Unicom Management System

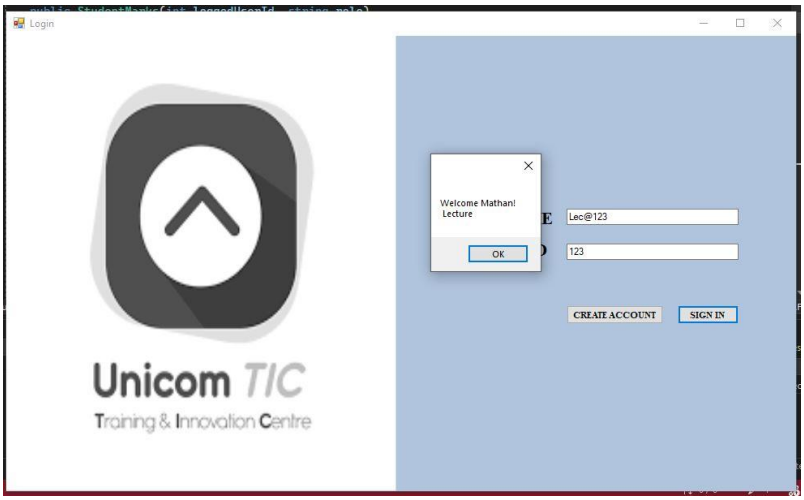
1. **User Registration by Role**
   The Admin registers users by assigning roles (Student, Lecturer, Staff), ensuring role-based access to system features.
2. **Course and Subject Creation**
   Courses are created and linked with relevant subjects to organize academic structure.
3. **Add Exams and Classes**
   Exams and classes are added and assigned to subjects for academic planning and assessment.
4. **Hall Allocation**
   Halls are scheduled and allocated based on sessions and course requirements.
5. **Marks Entry for Students**
   Lecturers input student marks for each exam under the appropriate subject.
6. **Timetable Creation**
   Timetables are generated based on lecturers, halls, dates, and time slots to manage class schedules efficiently.
7. **Attendance Management**
   Attendance is recorded for students, lecturers, and staff for each session.
8. **Student Access to Information**
   Students can securely view their **timetable**, **exam marks**, and **attendance records** from their dashboard.

Key Features Implemented

Inserted Role based notification when User logged in to the System.

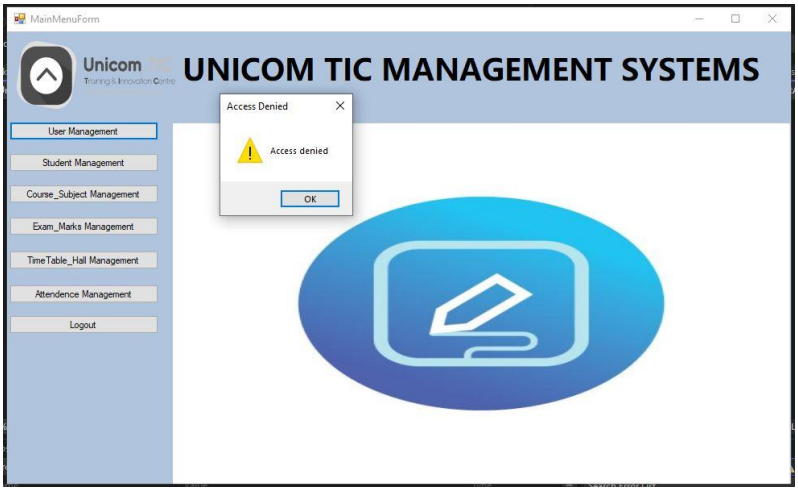
I have implemented role-based notifications that appear when a user logs into the system, displaying a personalized welcome message and informing them of their available features and access level based on their assigned role (Admin, Lecturer, Student, or Staff).

UserName : Admin
PassWord : Admin@123



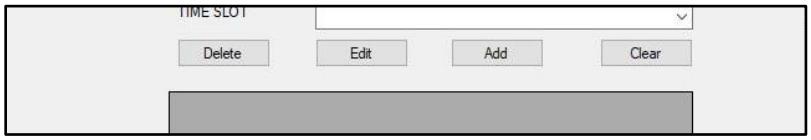
Given User based Access. It is Allowing Modules Based on the Features:

The system provides user-based access control, allowing each user to access only the modules and features permitted by their assigned role



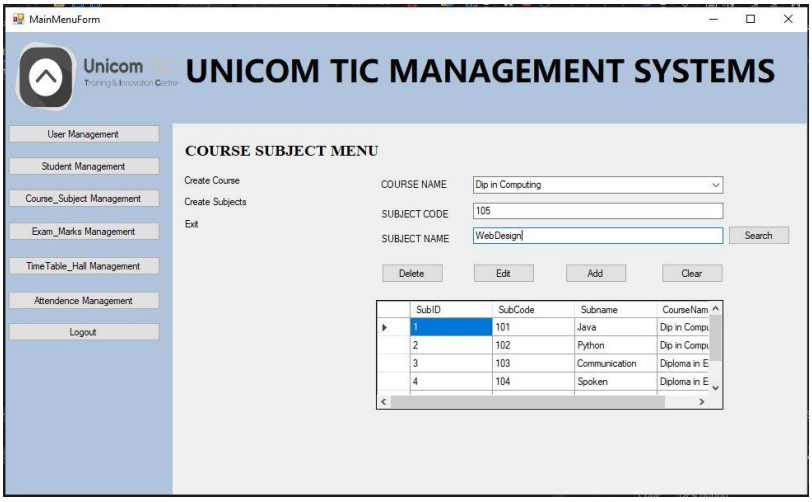
Given CRUD operation features for all Forms (Add,Edit,Delete,Search and Clear)

All forms in the system support full CRUD operations, including Add, Edit, Delete, Search, and Clear functionalities, ensuring efficient data management across all modules.



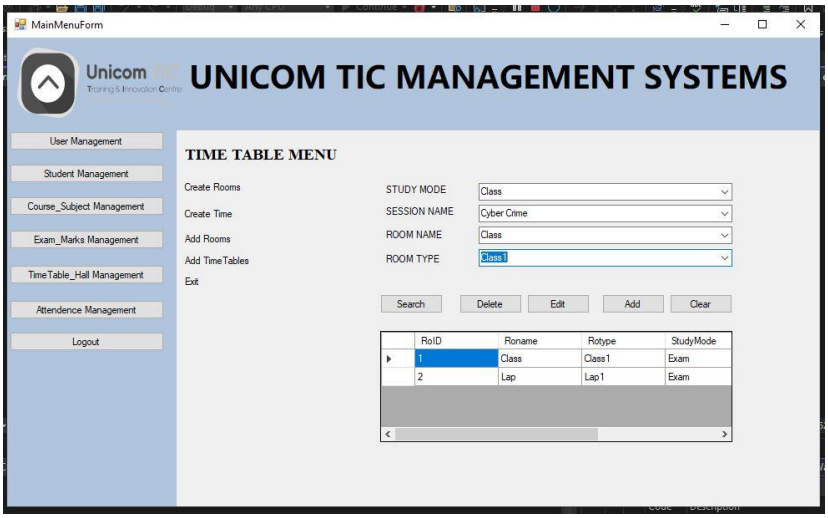
I have given Add features for subjects based on the Course Selection:

The system allows subjects to be added based on the selected course, ensuring proper organization and alignment of subjects under their respective courses.



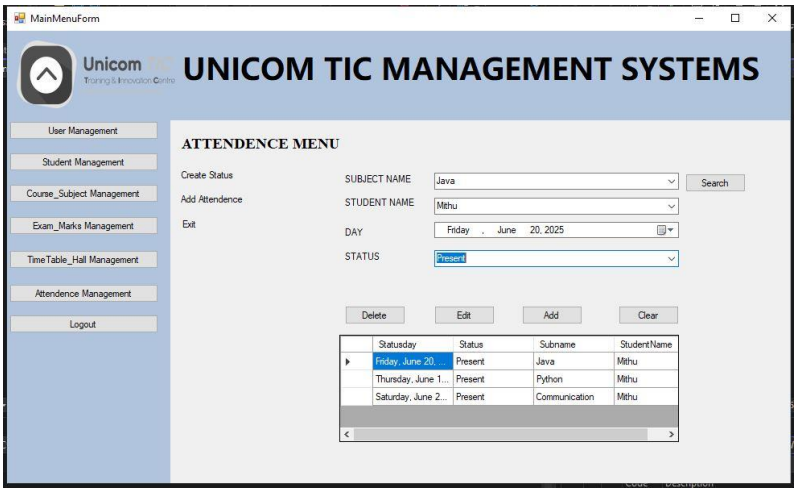
Allocate Class room and laps according to the required Exam or Class Session

The system allocates classrooms and Laps based on the requirements of scheduled exams or class sessions, ensuring efficient room utilization and session planning.



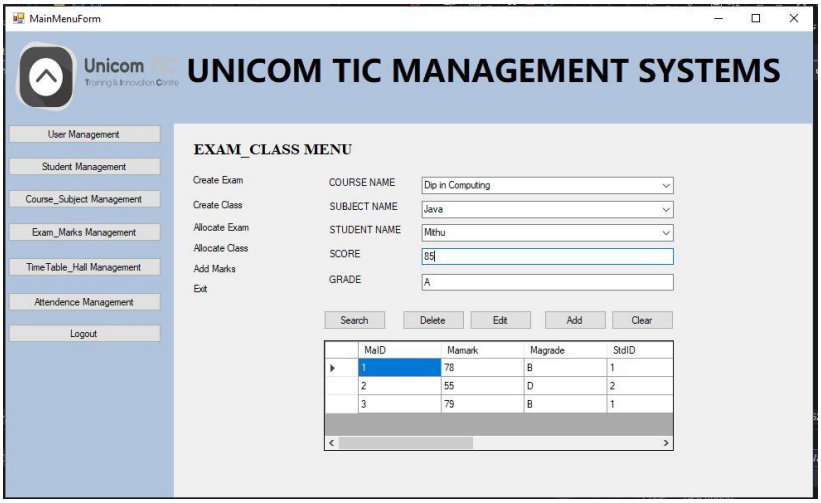
Add Attendance based on the Student, Subject and Date with Status

Attendance is recorded based on the selected student, subject, and date, along with their attendance status, ensuring accurate tracking of individual attendance records.



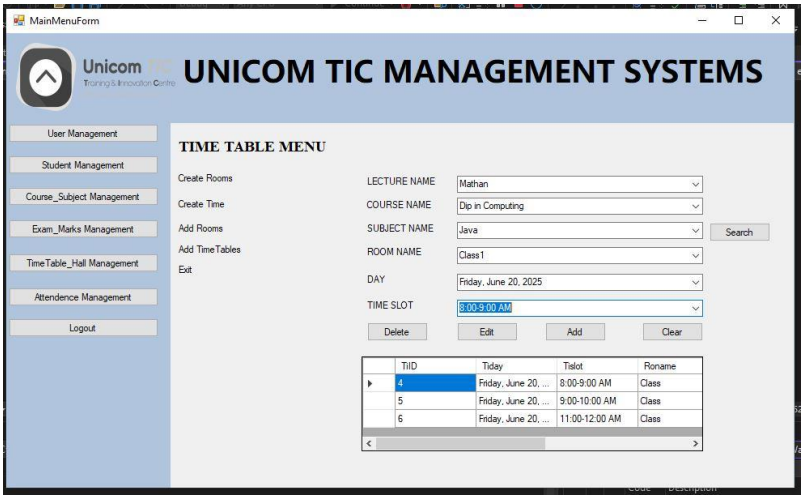
Add Marks based on Student name, Subject and Mark. Show grade and store.

Marks are added based on the student's name and subject. Upon entering the mark, the system automatically calculates and displays the corresponding grade, which is then stored along with the record



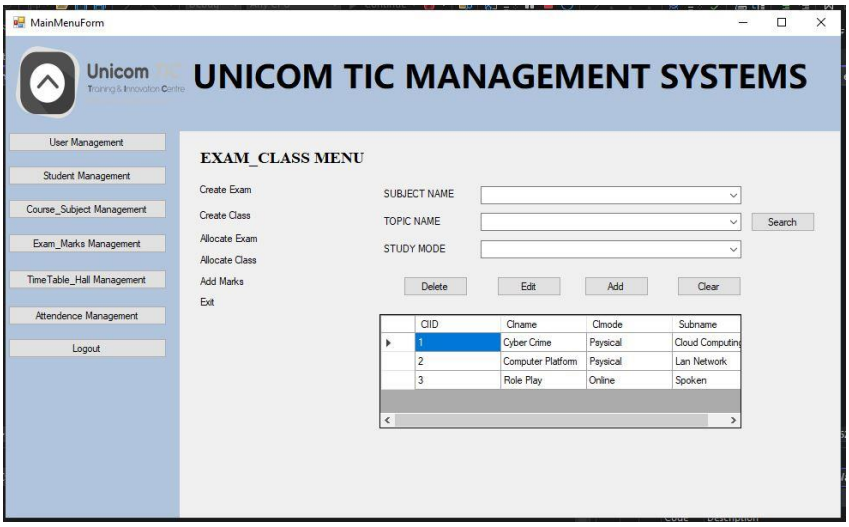
Add Timetable lecture Name, Course name, Subject name, Room, day and Time.

The timetable is created using the lecturer's name, course name, subject name, room, day, and time. The system prevents duplicate time allocations, ensuring that a classroom cannot be assigned to more than one lecture at the same time.



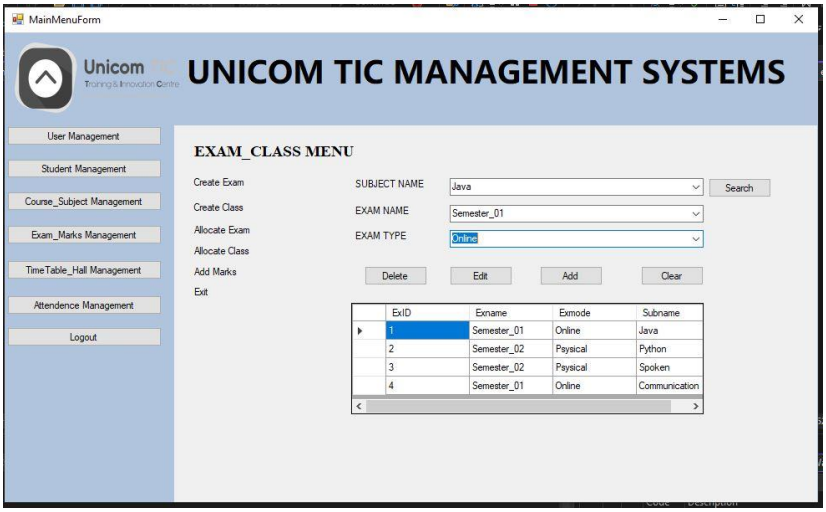
Allocate Class With Topic name

Classes are allocated with specific topic names to organize and manage the academic sessions effectively



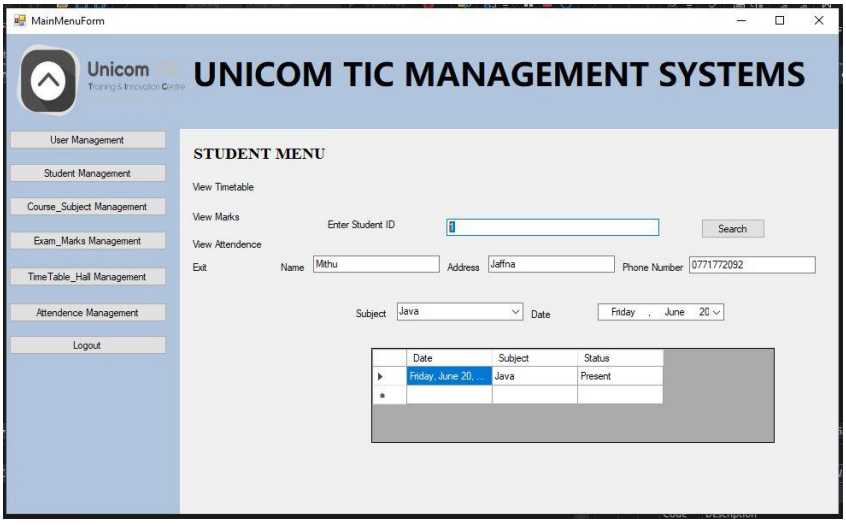
Allocate Exam For Specific Subject

Exams are allocated to specific subjects to organize assessments accurately and streamline exam management.



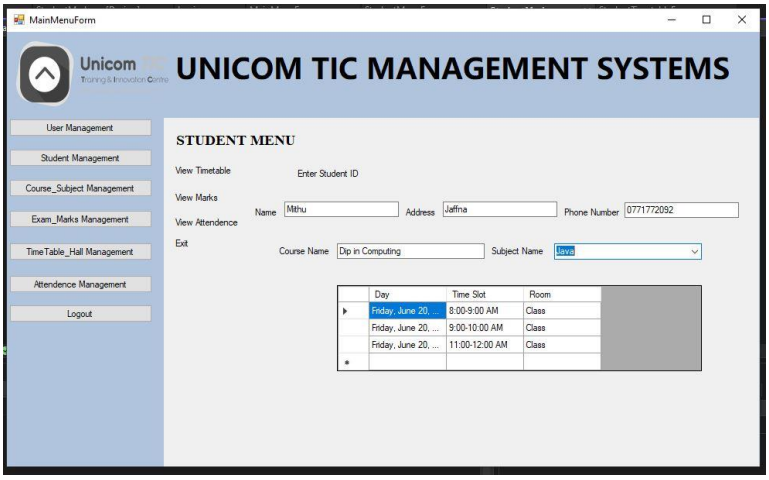
View Timetable for Admin, Staff and Lecture by enter Student Id

Admin, Staff, and Lecturers can view a student's timetable, marks, and attendance by entering the student ID, enabling comprehensive monitoring of student performance and schedules.



View Timetable for Student by enter username and password.

Students can view their timetable, marks, and attendance by logging in with their username and password, providing secure access to their academic information.



Technologies used

Environment:

Visual Studio with C# WinForms.

Database:

SQLite, using System.Data.SQLite.Core (install via NuGet).

User Interface:

WinForms with Form Designer, using Button, TextBox, ComboBox, DataGridView,Date timePicker.

OOP:

Encapsulation: Use private fields and public properties (e.g., private string name; with public string Name { get; set; }).

Inheritance (optional): Use a Person class for Student if you want to share Name.

Design Pattern:

MVC: Keep data, forms, and logic separate.

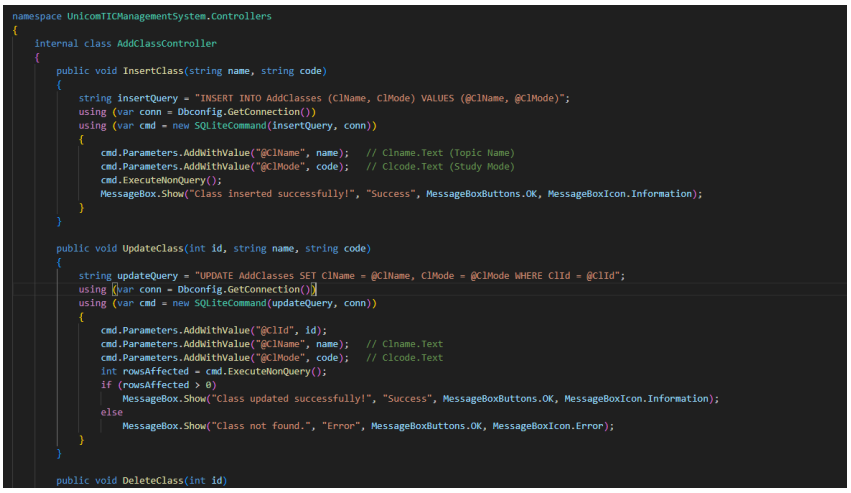
Error Handling:

Check inputs (e.g., Phone number is 10 digits).

Show messages (e.g., “Wrong User name and password” or “Select Subject”).

Handle database errors (e.g., “Database is locked , No such column name m.score ”).

Codes for Best Work

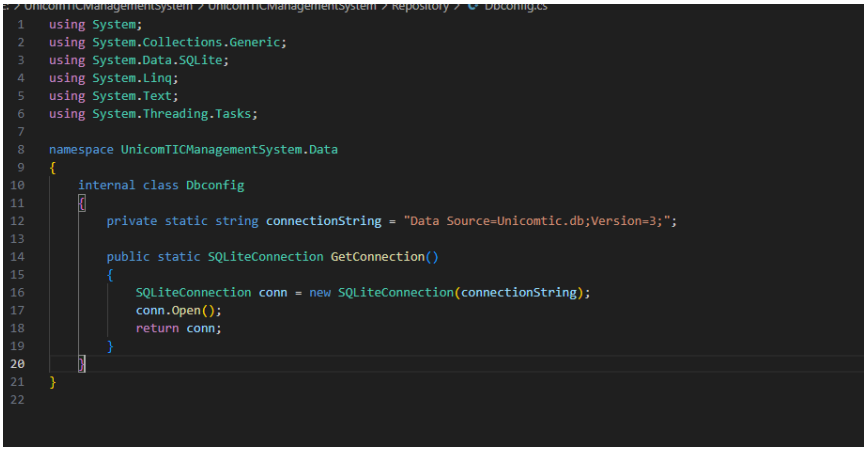


```
namespace UnicomTICManagementSystem.Controllers
{
    internal class AddClassController
    {
        public void InsertClass(string name, string code)
        {
            string insertQuery = "INSERT INTO AddClasses (ClName, ClMode) VALUES (@ClName, @ClMode)";
            using (var conn = Dbconfig.GetConnection())
            using (var cmd = new SQLiteCommand(insertQuery, conn))
            {
                cmd.Parameters.AddWithValue("@ClName", name);   // Clname.Text (Topic Name)
                cmd.Parameters.AddWithValue("@ClMode", code);   // Clcode.Text (Study Mode)
                cmd.ExecuteNonQuery();
                MessageBox.Show("Class inserted successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }

        public void UpdateClass(int id, string name, string code)
        {
            string updateQuery = "UPDATE AddClasses SET ClName = @ClName, ClMode = @ClMode WHERE ClId = @ClId";
            using (var conn = Dbconfig.GetConnection())
            using (var cmd = new SQLiteCommand(updateQuery, conn))
            {
                cmd.Parameters.AddWithValue("@ClId", id);
                cmd.Parameters.AddWithValue("@ClName", name);   // Clname.Text
                cmd.Parameters.AddWithValue("@ClMode", code);   // Clcode.Text
                int rowsAffected = cmd.ExecuteNonQuery();
                if (rowsAffected > 0)
                    MessageBox.Show("Class updated successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
                else
                    MessageBox.Show("Class not found.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        public void DeleteClass(int id)
```

CRUD Operations



```
using System;
using System.Collections.Generic;
using System.Data.SQLite;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace UnicomTICManagementSystem.Data
{
    internal class Dbconfig
    {
        private static string connectionString = "Data Source=Unicomtic.db;Version=3;";

        public static SQLiteConnection GetConnection()
        {
            SQLiteConnection conn = new SQLiteConnection(connectionString);
            conn.Open();
            return conn;
        }
    }
}
```

Database Connection

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace UnicomTICManagementSystem.Models
{
    internal class Attendence
    {
        public int AttendID { get; set; }
        public string Statusday { get; set; }
        public int StatusID { get; set; }
        public string Status { get; set; }
        public int SubID { get; set; }
        public string Subname { get; set; }
        public int StudentID { get; set; }
        public string StudentName { get; set; }
    }
}
```

Properties Method

```
cmd.CommandText = @"
            RoomId INTEGER NOT NULL,
            CourseID INTEGER NOT NULL,
            SubID INTEGER NOT NULL,
            LecID INTEGER NOT NULL,
            FOREIGN KEY (RoomId) REFERENCES Rooms(RoomId),
            FOREIGN KEY (CourseID) REFERENCES Courses(CouId),
            FOREIGN KEY (SubID) REFERENCES Subjects(SubjectId),
            FOREIGN KEY (LecID ) REFERENCES lectures(LecId)
        );

            CREATE TABLE IF NOT EXISTS Attendances (
            AttenId INTEGER PRIMARY KEY AUTOINCREMENT,
            Date Text NOT NULL,
            StdId INTEGER NOT NULL,
            SubjectId INTEGER NOT NULL,
            StatusId INTEGER NOT NULL,
            FOREIGN KEY (StdId) REFERENCES Students(StdId),
            FOREIGN KEY (StatusId) REFERENCES AddStatus(StatusId),
            FOREIGN KEY (SubjectId) REFERENCES Subjects(SubjectId)

        );

            CREATE TABLE IF NOT EXISTS Marks (
            MarksId INTEGER PRIMARY KEY AUTOINCREMENT,
            MarkScore INTEGER NOT NULL,
            MarksGrade TEXT NOT NULL,
            StdId INTEGER NOT NULL,
            CourseId INTEGER NOT NULL,
            SubjectId INTEGER NOT NULL,
            FOREIGN KEY (StdId) REFERENCES Students(StdId),
            FOREIGN KEY (CourseId) REFERENCES Courses(CouId),
            FOREIGN KEY (SubjectId) REFERENCES Subjects(SubjectId)
```

Foreign Key for Create Tables

```
        );

            INSERT INTO Roles (RoleCode, RoleName)
            SELECT 'R001', 'Admin'
            WHERE NOT EXISTS (SELECT 1 FROM Roles WHERE RoleName = 'Admin');

            INSERT INTO Users (UserName, UserPass, UserRole)
            SELECT 'Admin', 'Admin@123', 'Admin'
            WHERE NOT EXISTS (SELECT 1 FROM Users WHERE UserName = 'Admin');

            INSERT INTO Staffs (StaffName, StaffPhone, StaffAddress, UserId)
            SELECT 'Default Admin', '0000000000', 'Admin Office', UserId
            FROM Users
            WHERE UserName = 'Admin'
            AND NOT EXISTS (
            SELECT 1 FROM Staffs
            WHERE UserId = (SELECT UserId FROM Users WHERE UserName = 'Admin')
        );
    ";

    cmd.ExecuteNonQuery();

}
```

Default username Password for Admin

```
{
    string userQuery = "SELECT * FROM Users WHERE UserName = @username AND UserPass = @password";
    using (var userCmd = new SQLiteCommand(userQuery, conn))
    {
        userCmd.Parameters.AddWithValue("@username", username);
        userCmd.Parameters.AddWithValue("@password", password);

        using (var reader = userCmd.ExecuteReader())
        {
            if (reader.Read())
            {
                int userId = Convert.ToInt32(reader["UserId"]);
                string role = reader["UserRole"].ToString();

                string fullName = "";

                if (role == "Student")
                {
                    fullName = GetNameFromTable(conn, "Students", "StdName", userId);
                }
                else if (role == "Staff")
                {
                    fullName = GetNameFromTable(conn, "Staffs", "StaffName", userId);
                }
                else if (role == "Lecture")
                {
                    fullName = GetNameFromTable(conn, "Lectures", "LecName", userId);
                }
                else
                { fullName = "Admin";
                }

                MessageBox.Show($"Welcome {fullName}!\n {role}");
```

Log in Role and name Display

```
public List<Timetable> GetAllTimetables()
{
    var list = new List<Timetable>();
    using (var conn = Dbconfig.GetConnection())
    {
        var cmd = new SQLiteCommand(@"
    SELECT t.TimeId, t.TimeDay, t.TimeSlot,
           t.RoomId, r.RooomName,
           t.CourseID, c.CouName,
           t.SubID, s.SubjectName,
           t.LecID, l.LecName
    FROM TimeTables t
    LEFT JOIN Rooms r ON t.RoomId = r.RoomId
    LEFT JOIN Courses c ON t.CourseID = c.CouId
    LEFT JOIN Subjects s ON t.SubID = s.SubjectId
    LEFT JOIN lectures l ON t.LecID = l.LecId", conn);

        using (var reader = cmd.ExecuteReader())
        {
            while (reader.Read())
            {
                list.Add(new Timetable
                {
                    TiID = reader.GetInt32(0),
                    Tiday = reader.GetString(1),
                    Tislot = reader.GetString(2),
                    RoID = reader.GetInt32(3),
                    Roname = reader.IsDBNull(4) ? "" : reader.GetString(4),
                    CourseID = reader.GetInt32(5),
                    CourseName = reader.IsDBNull(6) ? "" : reader.GetString(6),
                    SubID = reader.GetInt32(7),
                    Subname = reader.IsDBNull(8) ? "" : reader.GetString(8),
                    LecID = reader.GetInt32(9),
                    LecName = reader.IsDBNull(10) ? "" : reader.GetString(10)
                });
```

Return List Method

```
private void Ssearch_Click(object sender, EventArgs e)
{
    string searchSubject = SelectcomboBox.Text.Trim();

    if (!string.IsNullOrEmpty(searchSubject))
    {
        using (var conn = Dbconfig.GetConnection())
        {
            string query = @"
        SELECT t.TimeId, t.TimeDay, t.TimeSlot, t.RoomId, r.RooomName, s.SubjectId, s.SubjectName
        FROM TimeTables t
        LEFT JOIN Rooms r ON t.RoomId = r.RoomId
        LEFT JOIN Subjects s ON t.SubID = s.SubjectId
        WHERE s.SubjectName LIKE @SubjectName
        LIMIT 1";

            using (var cmd = new SQLiteCommand(query, conn))
            {
                cmd.Parameters.AddWithValue("@SubjectName", $"%{searchSubject}%");

                using (var reader = cmd.ExecuteReader())
                {
                    if (reader.Read())
                    {
                        selectedTimeId = Convert.ToInt32(reader["TimeId"]);
                        TiDaycomboBox.Text = reader["TimeDay"].ToString();
                        TiSlotcomboBox.Text = reader["TimeSlot"].ToString();

                        int roomId = Convert.ToInt32(reader["RoomId"]);
                        TimecomboBox.SelectedValue = roomId;

                        int subjectId = Convert.ToInt32(reader["SubjectId"]);
                        SelectcomboBox.SelectedValue = subjectId;
```

Search Button Function

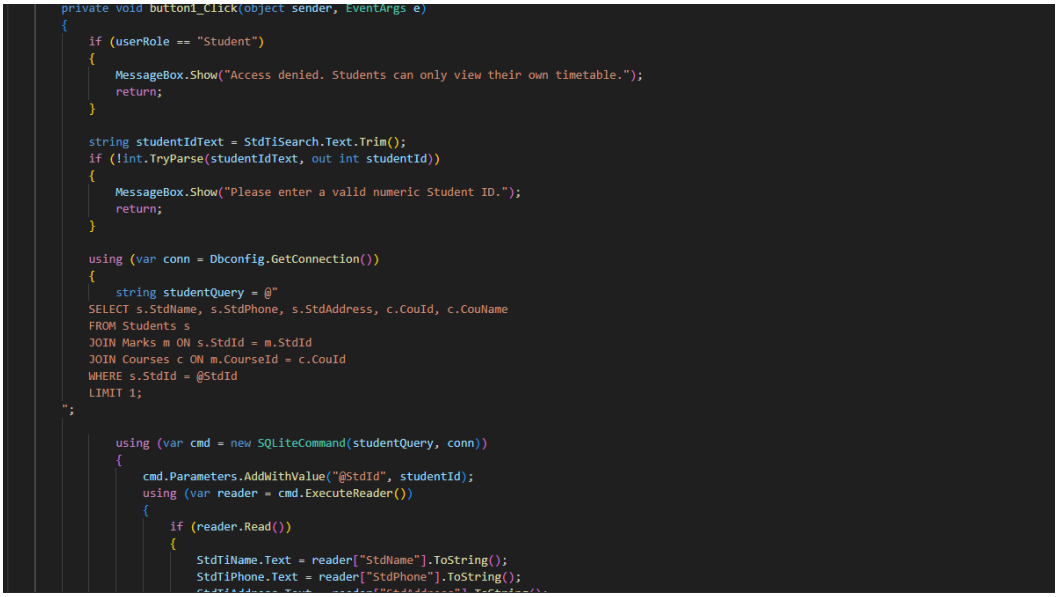
```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using UnicomTICManagementSystem.Models;

namespace UnicomTICManagementSystem
{
    public partial class MainMenuForm : Form
    {
        private readonly string userRole;
        private readonly int loggedInUserId;

        public MainMenuForm(int userId, string role)
        {
            InitializeComponent();
            loggedInUserId = userId;
            userRole = role;
        }

        public void loadform(object Form)
        {
            if (this.mainpanel.Controls.Count > 0)
                this.mainpanel.Controls.RemoveAt(0);
            Form f = Form as Form;
            f.TopLevel = false;
            f.Dock = DockStyle.Fill;
            this.mainpanel.Controls.Add(f);
            this.mainpanel.Tag = f;
            f.Show();
        }
        private void MainMenuForm_Load(object sender, EventArgs e)
```

User Information Transfer Through Form Constructors



```
private void button1_Click(object sender, EventArgs e)
{
    if (userRole == "Student")
    {
        MessageBox.Show("Access denied. Students can only view their own timetable.");
        return;
    }

    string studentIdText = StdTiSearch.Text.Trim();
    if (!int.TryParse(studentIdText, out int studentId))
    {
        MessageBox.Show("Please enter a valid numeric Student ID.");
        return;
    }

    using (var conn = Dbconfig.GetConnection())
    {
        string studentQuery = @"
    SELECT s.StdName, s.StdPhone, s.StdAddress, c.CouId, c.CouName
    FROM Students s
    JOIN Marks m ON s.StdId = m.StdId
    JOIN Courses c ON m.CourseId = c.CouId
    WHERE s.StdId = @StdId
    LIMIT 1;
";

        using (var cmd = new SQLiteCommand(studentQuery, conn))
        {
            cmd.Parameters.AddWithValue("@StdId", studentId);
            using (var reader = cmd.ExecuteReader())
            {
                if (reader.Read())
                {
                    StdTiName.Text = reader["StdName"].ToString();
                    StdTiPhone.Text = reader["StdPhone"].ToString();
```

User based Access To view Details

```
private void MarkScore_TextChanged(object sender, EventArgs e)
{
    if (int.TryParse(MarkScore.Text.Trim(), out int score))
    {
        Markgrade.Text = GetGradeFromScore(score);
    }
    else
    {
        Markgrade.Text = string.Empty;
    }
}

private void LoadCourses()
{
    var Courses = CourseControll.GetAllCourse();
    CoursecomboBox.DataSource = Courses;
    CoursecomboBox.DisplayMember = "CourseName";
    CoursecomboBox.ValueMember = "CourseID";
}
private void LoadSubjects()
{
    var subjects = subjectController.GetAllSubject();
    SelectcomboBox.DataSource = subjects;
    SelectcomboBox.DisplayMember = "Subname";
    SelectcomboBox.ValueMember = "SubID";
}

private void LoadStudents()
{
    using (var conn = Dbconfig.GetConnection())
    {
        string query = "SELECT StdId, StdName FROM Students";
        using (var cmd = new SQLiteCommand(query, conn))
        using (var adapter = new SQLiteDataAdapter(cmd))
```

Load Data From Controller methods

```
private void Sadd_Click(object sender, EventArgs e)
{
    if (string.IsNullOrWhiteSpace(TiDaycomboBox.Text) || string.IsNullOrWhiteSpace(TiSlotcomboBox.Text))
    {
        MessageBox.Show("Please Select All Values.");
        return;
    }

    if (TimecomboBox.SelectedValue == null || CoursecomboBox.SelectedValue == null || SelectcomboBox.SelectedValue == null)
    {
        MessageBox.Show("Please select All Values.");
        return;
    }

    using (var conn = Dbconfig.GetConnection())
    {
        var checkCmd = new SQLiteCommand(@"
                        SELECT COUNT(*) FROM TimeTables
                        WHERE RoomId = @RoomId AND TimeDay = @TimeDay AND TimeSlot = @TimeSlot", conn);


        checkCmd.Parameters.AddWithValue("@RoomId", TimecomboBox.SelectedValue);
        checkCmd.Parameters.AddWithValue("@TimeDay", TiDaycomboBox.Text.Trim());
        checkCmd.Parameters.AddWithValue("@TimeSlot", TiSlotcomboBox.Text.Trim());

        var count = Convert.ToInt32(checkCmd.ExecuteScalar());

        if (count > 0)
        {
            MessageBox.Show("A timetable entry for this Lecture in the same Room, Day, and Time Slot already exists!", "Duplicate Entry", MessageBoxButtons.OK, Messag
            return; // Stop insertion
        }
```

Duplicate validation Fro Time table Entry

Challenges Faced and Methods to Solved

- Handling Database was little difficult at the beginning
- Code Writing Environment was new and hard to understand the format