A
Project Report
On

“UnicomTIC Management System”

Miss.N.Nirushana.
UT010560

Guidance of
Mr.Kathir

Date: 23.06.2025

Table of Contents

Introduction

The UnicomTIC Management System is a desktop-based application developed using C# (Windows Forms) and SQLite to streamline academic and administrative tasks within an educational institution. The system is designed to support role-based access for different users such as administrator, students, lecturers and staff allowing them to access only the features relevant to their specific roles.

This project aims to improve efficiency in handling student records, exam mark entries, timetable management, and subject/course allocation. By automating these processes, the system reduces manual errors, improves data accuracy, and saves time. The user-friendly interface ensures that all users can interact with the system easily without requiring advanced technical skills.

The application is built following object-oriented programming principles and adopts a modular architecture using the MVC (Model-View-Controller) design pattern. This structure ensures better code organization, scalability, and ease maintenance, making the system suitable for real-world deployment in academic environments.

Acknowledgement

I owe a great many thanks to a great many people who helped and supported me, during the Making of this Project.

Firstly, I am truly grateful to my project supervisor, Mr. Kathir, for his constant support, clear instructions, and valuable feedback throughout the development of this system. His guidance helped me understand both theoretical and practical aspects of software development, and his encouragement kept me motivated at every stage. I also extend my appreciation to the administrative staff and lab instructors for providing the necessary resources, infrastructure, and learning environment that played a crucial role in the success of this project.

Special thanks go to my classmates and friends for their input and support during testing and brainstorming phases. Lastly, I am deeply thankful to my family for their unwavering support, patience, and encouragement during the entire duration of this project. I would also thank the Institution and the faculty members without whom this project would have been a distant reality. I also extend my heartfelt thanks to well-wishers.

Current System

Currently, many educational institutions manage student, staff, and academic information manually using tools such as paper records, Excel spreadsheets, or disconnected software systems. These traditional methods are often time-consuming, error-prone, and inefficient.

For instance, exam results are recorded manually, timetables are shared through printed notices or informal channels, and student details are maintained in scattered files. This leads to several issues.

Students often face challenges in checking their marks, attendance, or class schedules without approaching administrative staff directly. Likewise, lecturers and administrative personnel spend excessive time managing routine academic tasks, which affects productivity.

This project was undertaken to address these challenges and introduce an efficient, centralized, and user-friendly system that digitizes and simplifies academic operations. So, I aimed to build a system that reduces manual effort and enhances the overall academic management process.

Problem Definition

Most educational institutions still use manual methods or spreadsheets to manage students, staff, exams, and timetables. This leads to:

- Data entry errors
- Delays in accessing information
- No role-based access control
- Difficulty for students to view marks or timetables
- Poor data security

To solve these issues, an automated system is needed that offers secure, role-based access, faster data handling, and easy access to academic information.

The UnicomTIC Management System is designed to meet this need efficiently.

Customer Requirements

To ensure the Unico**m**TIC Management System effectively serves its users, the following requirements were identified:

1. Functional Requirements

- **Secure Login & Role Assignment**
  - Each user (Admin, Lecturer, Staff, Student) must log in using a username and password.
  - Users are granted access based on their assigned roles.
- **Customized Dashboards**
  - Admin: Full access to manage users, subjects, courses, marks, exams, timetables, and rooms.
  - Lecturer: Can view and manage only their assigned subjects and class schedules.
  - Staff: Access to specific administrative data relevant to their duties.
  - Student: Limited access to their own marks, **room allocation** and timetable.
- **Data Management Features**

- Ability to add, update, delete, and view information for:
- Students, Staff, Lecturers
- Courses and Subjects
- Exam Marks and Timetables
- Staff and Lecturer profiles include essential fields like name, role, ID, and contact info.
- Restricted Access Views
- Students and Lecturers can only view (not edit) academic data assigned to them.

2. Non-Functional Requirements

- **Security**
  - Role-based access control to protect data and prevent unauthorized actions.
- **System Performance**
  - The system should respond quickly and perform smoothly even when multiple users are logged in simultaneously**.**

Technology Used

Components	Technology
Language	C#
Framework	.NET(WinForms)
Database	SQLite
Architecture	MVC(Model View Controller)
Tools	Visual Studio, DB for SQLite

Feasibility Study

1. Technical Feasibility

- Developed using C# (WinForms) and SQLite.
- Tools are easy to use, available, and well-supported.
- Suitable for building desktop-based applications.

2. Economic Feasibility

- No extra cost for software or tools.
- Uses free/open-source platforms.
- Affordable for academic project needs.

3. Operational Feasibility

- Simple and user-friendly interface.
- Role-based access makes it easy to use for all users.
- Requires minimal training for students and staff.

4. Schedule Feasibility

- Project completed within the academic timeframe.
- All modules were delivered as per planned schedule.
- Time management was effective throughout development.

Project Schedule

The development of the UnicomTIC Management System was planned and completed in the following stages:

1. Requirement Analysis

- Duration: 4-5 days
- Activities: Understanding user needs, defining system features, and identifying key modules.

2. System Design

- Duration: 1 Day
- Activities: Designing database structure, UI layout, and module flow using the MVC model.

3. Development

- Duration: 2–3 Weeks
- Activities:
- Coding login and user roles

- Implementing student, lecturer, course, subject, and timetable modules
- Database connection and CRUD operations

4. Testing

- Duration: 5 Days
- Activities: Debugging, validating inputs, fixing issues, and improving user experience.

5. Final Report

- Duration: 3 Days
- Activities: Preparing documentation, screenshots, viva questions

ER Diagram (Text Version)

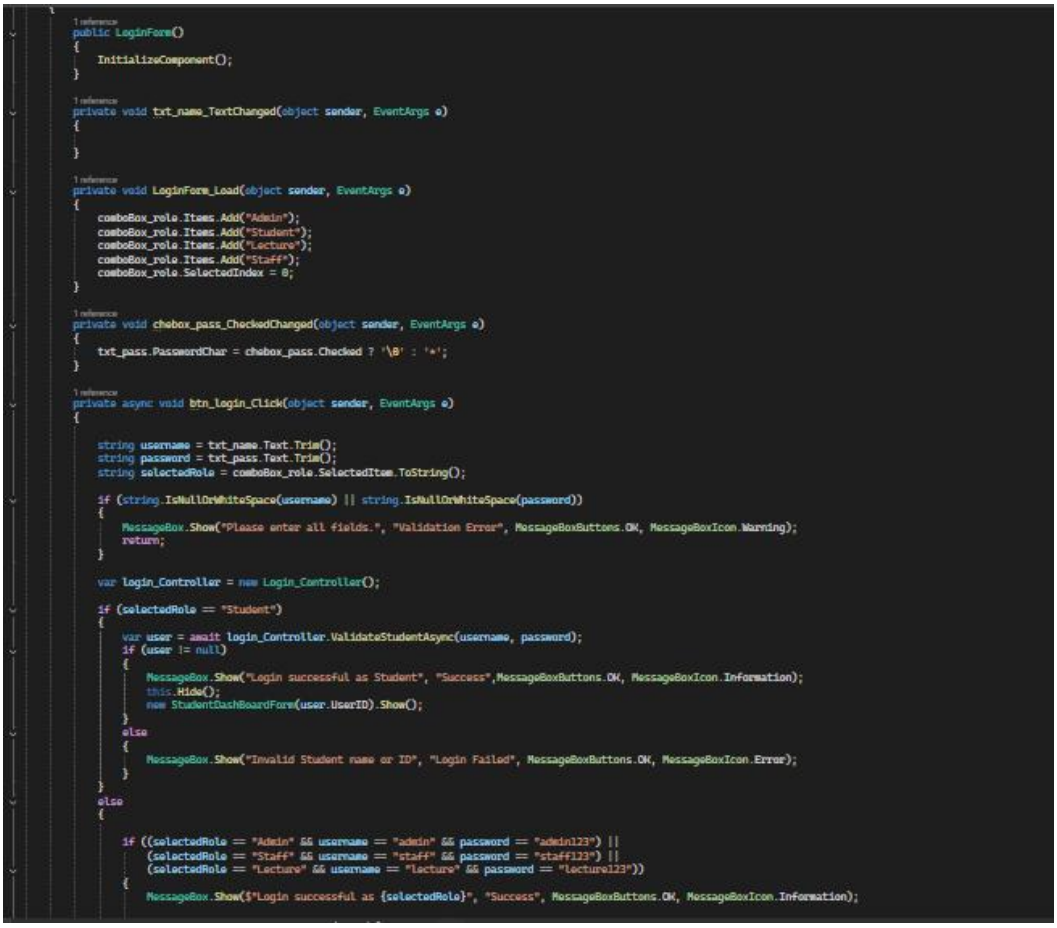
Entities and Attributes

- User
  - UserID (PK)
  - Name
  - Password
  - Role
- Student
  - StudentID (PK)
  - StudentName
  - CourseID (FK)
- Staff
  - StaffID (PK)
  - StaffName
  - Gender
  - Status
- Course
  - CourseID (PK)
  - CourseName
  - StartDate
  - EndDate
- Lecture
  - LectureID (PK)
  - LectureName

    - Address
- Exam
    - ExamID (PK)
    - ExamName
    - SubjectID (FK)
- Room
    - RoomID (PK)
    - RoomName
    - RoomType
- Marks
    - MarkID (PK)
    - StudentID (FK)
    - ExamID (FK)
    - Score
- Timetable
    - TimetableID (PK)
    - SubjectID (FK)
    - RoomID (fk)
    - Timeslot
- Subject
    - SubjectID (PK)
    - SubjectName
    - CourseID (FK)

Code Screenshots and Explanation

- Login Form design and Login Form.cs



```
public LoginForm()
{
    InitializeComponent();
}

private void txt_name_TextChanged(object sender, EventArgs e)
{

}

private void LoginForm_Load(object sender, EventArgs e)
{
    comboBox_role.Items.Add("Admin");
    comboBox_role.Items.Add("Student");
    comboBox_role.Items.Add("Lecture");
    comboBox_role.Items.Add("Staff");
    comboBox_role.SelectedIndex = 0;
}

private void chebox_pass_CheckedChanged(object sender, EventArgs e)
{
    txt_pass.PasswordChar = chebox_pass.Checked ? '\0' : '*';
}

private async void btn_login_Click(object sender, EventArgs e)
{

    string username = txt_name.Text.Trim();
    string password = txt_pass.Text.Trim();
    string selectedRole = comboBox_role.SelectedItem.ToString();

    if (string.IsNullOrWhiteSpace(username) || string.IsNullOrWhiteSpace(password))
    {
        MessageBox.Show("Please enter all fields.", "Validation Error", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        return;
    }

    var login_Controller = new Login_Controller();

    if (selectedRole == "Student")
    {
        var user = await login_Controller.ValidateStudentAsync(username, password);
        if (user != null)
        {
            MessageBox.Show("Login successful as Student", "Success",MessageBoxButtons.OK, MessageBoxIcon.Information);
            this.Hide();
            new StudentDashBoardForm(user.UserID).Show();
        }
        else
        {
            MessageBox.Show("Invalid Student name or ID", "Login Failed", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }
    else
    {

        if ((selectedRole == "Admin" && username == "admin" && password == "admin123") ||
            (selectedRole == "Staff" && username == "staff" && password == "staff123") ||
            (selectedRole == "Lecture" && username == "lecture" && password == "lecture123"))
        {
            MessageBox.Show($"Login successful as {selectedRole}", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
```

Role Selection**: Users choose their role from a dropdown (comboBox_role).**

Input Validation**: Ensures the username and password are not empty.**

Role-Specific Login**:**

For Students**: Validates credentials using a database call (`ValidateStudentAsync`). If successful, opens the `StudentDashBoardForm`.**

For Other Roles**: Checks against hardcoded credentials (e.g., Admin: `admin/admin123`). If successful, opens the relevant dashboard form (`AdminDashBoardForm`, etc.).**

Password Visibility**: Toggles the password field visibility using a checkbox.**

Error Handling**: Displays messages for invalid inputs or failed login attempts.**

- # AdminDashboard Form.cs

```
public partial class AdminDashBoardForm : Form
{
    public AdminDashBoardForm()
    {
        InitializeComponent();
        LoadForm(new DashboardForm());
    }

    public void LoadForm(object formObj)
    {
        if (formObj is Form form)
        {
            if (this.mainPanel.Controls.Count > 0)
            {
                this.mainPanel.Controls.RemoveAt(0);
            }

            //Form form = formObj as Form;
            form.TopLevel = false;
            form.Dock = DockStyle.Fill;
            this.mainPanel.Controls.Add(form);
            this.mainPanel.Tag = form;
            form.Show();
        }
    }

    private void btn_mang_stu_Click(object sender, EventArgs e)
    {

        LoadForm(new StudentForm());
    }

    private void btn_mang_lec_Click(object sender, EventArgs e)
    {

        LoadForm(new LectureForm());
    }

    private void btn_mang_cou_Click(object sender, EventArgs e)
    {

        LoadForm(new CourseForm());
    }

    private void btn_mang_timetable_Click(object sender, EventArgs e)
    {

        LoadForm(new TimetableForm());
    }

    private void btn_manag_exam_Click(object sender, EventArgs e)
    {

        LoadForm(new ExamForm());
    }

    private void btn_manag_mark_Click(object sender, EventArgs e)
    {
```

```
        LoadForm(new LectureForm());
    }

    private void btn_mang_cou_Click(object sender, EventArgs e)
    {

        LoadForm(new CourseForm());
    }

    private void btn_mang_timetable_Click(object sender, EventArgs e)
    {

        LoadForm(new TimetableForm());
    }

    private void btn_manag_exam_Click(object sender, EventArgs e)
    {

        LoadForm(new ExamForm());
    }

    private void btn_manag_mark_Click(object sender, EventArgs e)
    {

        LoadForm(new MarkForm());
    }

    private void btn_manag_room_Click(object sender, EventArgs e)
    {

        LoadForm(new RoomForm());
    }

    private void btn_close_Click(object sender, EventArgs e)
    {
        Application.Exit();
    }

    private void btn_manag_sub_Click(object sender, EventArgs e)
    {
        LoadForm(new SubjectForm());
    }

    private void btn_manag_staff_Click(object sender, EventArgs e)
    {
        LoadForm(new StaffForm());
    }
}
}
```

Default View: When the form loads, the (`DashboardForm`) is displayed in the (`mainPanel`) by default.

Dynamic Form Loading:

- The (`LoadForm`) method removes any existing content in the (`mainPanel`) and embeds a specified form.
- Ensures only one form is displayed at a time.

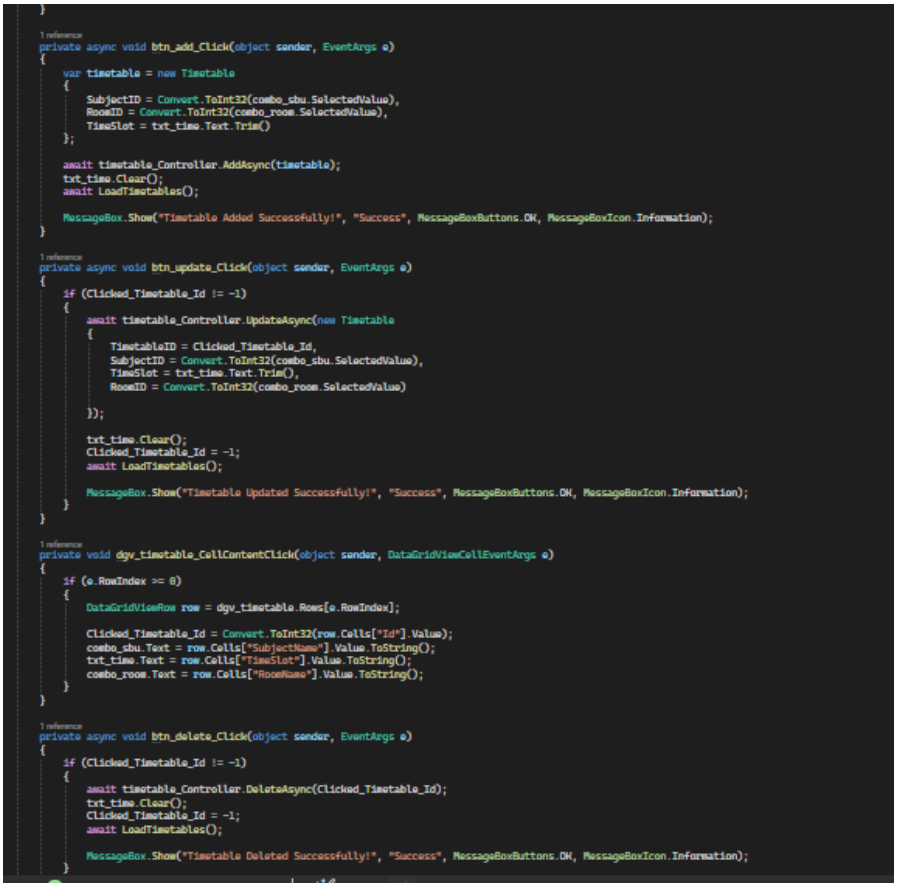
Button Actions: Each button corresponds to a specific management task:

- Manage Students: Loads (`StudentForm`) for managing student data.
- Manage Lectures: Loads (`LectureForm`) for managing lecture information.
- Manage Courses: Loads (`CourseForm`) for managing course details.
- Similar actions for Timetables, Exams, Marks, Rooms, Subjects, and Staff.

Close Application: The "Close" button exits the application by calling (`Application.Exit()`).

Seamless Navigation: Admins can switch between tasks (e.g., Students to Courses) by clicking the respective buttons, dynamically updating the (`mainPanel`) content.

- # Timetable Form.cs



```
}

1 reference
private async void btn_add_Click(object sender, EventArgs e)
{
    var timetable = new Timetable
    {
        SubjectID = Convert.ToInt32(combo_sbu.SelectedValue),
        RoomID = Convert.ToInt32(combo_room.SelectedValue),
        TimeSlot = txt_time.Text.Trim()
    };

    await timetable_Controller.AddAsync(timetable);
    txt_time.Clear();
    await LoadTimetables();

    MessageBox.Show("Timetable Added Successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
}

1 reference
private async void btn_update_Click(object sender, EventArgs e)
{
    if (Clicked_Timetable_Id != -1)
    {
        await timetable_Controller.UpdateAsync(new Timetable
        {
            TimetableID = Clicked_Timetable_Id,
            SubjectID = Convert.ToInt32(combo_sbu.SelectedValue),
            TimeSlot = txt_time.Text.Trim(),
            RoomID = Convert.ToInt32(combo_room.SelectedValue)

        });

        txt_time.Clear();
        Clicked_Timetable_Id = -1;
        await LoadTimetables();

        MessageBox.Show("Timetable Updated Successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
}

1 reference
private void dgv_timetable_CellContentClick(object sender, DataGridViewCellEventArgs e)
{
    if (e.RowIndex >= 0)
    {
        DataGridViewRow row = dgv_timetable.Rows[e.RowIndex];

        Clicked_Timetable_Id = Convert.ToInt32(row.Cells["Id"].Value);
        combo_sbu.Text = row.Cells["SubjectName"].Value.ToString();
        txt_time.Text = row.Cells["TimeSlot"].Value.ToString();
        combo_room.Text = row.Cells["RoomName"].Value.ToString();
    }
}

1 reference
private async void btn_delete_Click(object sender, EventArgs e)
{
    if (Clicked_Timetable_Id != -1)
    {
        await timetable_Controller.DeleteAsync(Clicked_Timetable_Id);
        txt_time.Clear();
        Clicked_Timetable_Id = -1;
        await LoadTimetables();

        MessageBox.Show("Timetable Deleted Successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
```

Purpose: Manages timetables, including adding, updating, deleting, and viewing them.

Features:

1. Add Timetable: Captures subject, room, and timeslot, then saves it.
2. Update Timetable: Updates selected timetable with new details.
3. Delete Timetable: Deletes the selected timetable.
4. Delete All: Removes all timetables and resets IDs.
5. Dynamic Loading:
    - Populates subjects (`combo_sbu`) and rooms (`combo_room`) dropdowns.
    - Displays timetables in a grid (`dgv_timetable`).

Workflow:

- On load: Subjects, rooms, and timetables are fetched and displayed.
- Clicking a timetable row populates its details into the fields.
- Buttons trigger corresponding actions (Add, Update, Delete).

- # Room Form.cs

```
1 reference
private async void btn_add_Click(object sender, EventArgs e)
{
    string roomName = txt_rname.Text.Trim();
    string roomType = cmb_type.SelectedItem?.ToString();

    if (!string.IsNullOrWhiteSpace(roomName) && !string.IsNullOrWhiteSpace(roomType))
    {
        await room_Controller.AddAsync(new Room { RoomName = roomName, RoomType = roomType });
        txt_rname.Clear();
        await LoadRoom();

        MessageBox.Show("Room Added Successfully!", "Success", MessageBoxButtons.OK,MessageBoxIcon.Information);
    }
    else
    {
        MessageBox.Show("Please enter room name and select type.","Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

1 reference
private async void btn_update_Click(object sender, EventArgs e)
{
    if (Clicked_RoomId != -1)
    {
        await room_Controller.UpdateAsync(new Room
        {
            RoomID = Clicked_RoomId,
            RoomName = txt_rname.Text.Trim(),
            RoomType = cmb_type.SelectedItem?.ToString()
        });

        txt_rname.Clear();
        Clicked_RoomId = -1;
        await LoadRoom();

        MessageBox.Show("Room Updated Successfully!", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }
    else
    {
        MessageBox.Show("Please enter room name and select type to update.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}
```

```
1 reference
private async void btn_delete_Click(object sender, EventArgs e)
{
    if (Clicked_RoomId != -1)
    {
        await room_Controller.DeleteAsync(Clicked_RoomId);
        txt_rname.Clear();
        Clicked_RoomId = -1;
        await LoadRoom();

        MessageBox.Show("Room deleted Successfully.", "Success", MessageBoxButtons.OK, MessageBoxIcon.Information);
    }

    else
    {
        MessageBox.Show("Please enter room name and select type to delete.", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
}

1 reference
private void dgv_room_CellContentClick(object sender, DataGridViewCellEventArgs e)
{
    if (e.RowIndex >= 0)
    {
        var row = dgv_room.Rows[e.RowIndex];
        Clicked_RoomId = Convert.ToInt32(row.Cells["RoomID"].Value);
        txt_rname.Text = row.Cells["RoomName"].Value.ToString();
        cmb_type.SelectedItem = row.Cells["RoomType"].Value.ToString();
    }
}

2 references
private async void RoomForm_Load(object sender, EventArgs e)
{
    cmb_type.Items.AddRange(new[] { "Lab", "Hall" });
    cmb_type.SelectedIndex = 0;
    await LoadRoom();
}

5 references
private async Task LoadRoom()
{
    dgv_room.DataSource = await room_Controller.GetAllAsync();
}

1 reference
private async void btn_del_all_Click(object sender, EventArgs e)
{
```

Purpose: Provides functionality to manage rooms, including adding, updating, deleting, and viewing room details.

Key Features:

1. Add Room:
    - Collects room name and type from the user.
    - Saves the room to the database if both fields are provided.
2. Update Room:
    - Updates details of the selected room using its `RoomID`.
3. Delete Room:
    - Deletes the selected room by its `RoomID`.
4. Delete All Rooms:
    - Deletes all rooms and resets their IDs after user confirmation.
5. Dynamic Data Loading:
    - Fetches and displays room data in a `DataGridView` (`dgv_room`).
    - Populates room types (e.g., "Lab", "Hall") in a dropdown (`cmb_type`).
6. Row Selection:
    - Allows the user to click a row in the grid to populate the fields for updating or deleting.

Workflow:

1. Initialization:
    - On form load (`RoomForm_Load`):
        - Room types are added to the dropdown (`cmb_type`).
        - Existing rooms are fetched and displayed in the grid.
2. Adding a Room:
    - The user enters a room name and selects a type.
    - Clicks "Add" to save the room.
    - Data grid is updated with the new room.
3. Updating a Room:
    - The user selects a room row from the grid, which populates the fields.
    - Updates the details and clicks "Update."
    - Data grid reflects the changes.
4. Deleting a Room:
    - The user selects a room row and clicks "Delete."
    - The room is removed, and the grid updates.
5. Deleting All Rooms:
    - The user clicks "Delete All" and confirms the prompt.
    - All rooms are deleted, and the grid clears.

Key Methods:

1. `LoadRoom()`:
   - **Fetches all rooms using `room_Controller.GetAllAsync()` and binds the data to `dgv_room`.**
2. `btn_add`:
   - **Adds a room after validating inputs (room name and type).**
3. `btn_update`:
   - **Updates the selected room based on its `RoomID`.**
4. `btn_delete`:
   - **Deletes the selected room by its `RoomID`.**
5. `btn_del_all`:
   - **Deletes all rooms and resets IDs via `ResetRoomDataAsync()`.**

References

- **Microsoft Docs – Windows Forms Documentation**
- **SQLite Documentation**
- **C# Programming Guide – Microsoft**
- **ChatGPT**
- **W3Schools C# and SQL Tutorials**
- **Support from Mr.Kathir(Lecturer) and Lab instructor.**

Conclusion

The UnicomTic Management System effectively addresses the challenges faced by educational institutions in managing academic operations such as session scheduling, lecture and lab hall allocation, and subject management. By digitizing these processes, the system reduces manual effort, minimizes human errors, and ensures data consistency across departments.

Built using C# with a WinForms interface and SQLite as the backend, the system offers a user-friendly platform that allows Admins, Lecturers, and Students to access and manage relevant information according to their roles. The modular design and role-based access control improve both usability and security.

Through this project, we demonstrated how automation can improve the efficiency and transparency of university academic planning. The system is scalable and can be enhanced in the future with additional features like attendance analytics, notification systems, and mobile app integration, making it a comprehensive solution for university management needs.