A

Project Report

On

“Unicom TIC Management System”

Miss.P.Inthusa

UT010953

Guidance of

Mr.Kathir

Date: 23.06.2025

Table of Contents

Introduction & Acknowledgement

- ***Introduction:***

The UnicomTIC Management System is a Windows-based desktop application developed to efficiently manage various academic and administrative activities within an educational institution. This system provides role-based access for students, lecturers, staff, and administrators, ensuring that each user can only access the features and information relevant to their role.

The primary goal of this project is to simplify tasks such as student record management, exam mark entry, timetable viewing, and course/subject assignment through a centralized and user-friendly interface. The system reduces paperwork, minimizes errors, and enhances the speed of data processing using a well-structured C# (Windows Forms) application backed by a SQLite database.

This application was built by applying object-oriented programming principles and separating functionality into models, controllers, and forms, making the system modular, scalable, and maintainable

- ***Acknowledgement:***

I would like to express my sincere gratitude to all those who supported and guided me throughout the development of the UnicomTIC Management System.

First and foremost, I am deeply thankful to my project guide, our Lecturer Mr. Kathir for his continuous support, valuable guidance, and clear teaching of the core project concepts. His thoughtful feedback, practical insights, and encouragement helped me shape this project effectively from start to finish. The project idea and its implementation became understandable only because of his clear explanation and structured teaching.

I also extend my heartfelt thanks to the Department of Computer Science for providing the opportunity, facilities, and a learning environment that helped me improve my technical knowledge and project development skills.

Finally, I would like to thank my family and friends for their constant support, motivation, and belief in my efforts during the course of this project.

Current System

At present, most schools and colleges manage student, staff, and exam information using files or Excel sheets. This method takes more time, leads to mistakes, and makes it hard to find or update data. There is no proper way for students to check their marks or timetable on their own. Staff and lecturers also face difficulties in managing or viewing academic information.

This project is done as part of our academic work in the Department of Computer Science with the guidance of Mr. Kathir, Lecturer, who supported us with helpful ideas and continuous encouragement. The aim is to create an easy and efficient system to manage students, staff, exams, timetables, and marks.

Problem Definition

In many educational institutions, managing students, lecturers, staff, exams, and timetables is still done manually or using spreadsheets. This creates several problems such as:

- Data entry mistakes
- Time delays in updating or accessing information
- Difficulty for students to view their own marks and class timetables
- No role-based access, meaning everyone cannot get the right information they need
- Lack of security and user tracking in manual systems

Because of these issues, there is a strong need for an automated system that:

- Allows each user (Admin, Student, Lecturer, Staff) to log in separately
- Shows a personalized dashboard for each role
- Enables students to view only their own timetable and marks
- Helps lecturers and staff manage their assigned tasks easily
- Stores data safely using a structured database

The UnicomTIC Management System is designed to solve all of these problems in a simple, user-friendly way.

Customer Requirements

To ensure the UnicomTIC Management System meets the needs of its users, both functional and non-functional requirements were identified:

1. Functional Requirements:

- User Authentication
    - Users must log in with their username and password.
    - Each user is assigned a specific role (Admin, Student, Lecturer, Staff).
- Role-Based Dashboards
    - Admin: Can manage courses, subjects, students, exams, marks, rooms, timetables, and users.
    - Student: Can view only their own marks and timetable.
    - Lecturer: Can view subject and timetable details related to them.
    - Staff: Can access information based on their job title.
- Data Management Modules
    - CRUD operations for Courses, Subjects, Students, Exams, Marks, Rooms, and Timetables.
    - Staff and Lecturer records are also maintained with basic info like name, ID, address, and role.
- Read-Only Access for Students and Lecturers where needed, especially in viewing marks and timetables.

2. Non-Functional Requirements:

- Security

  Prevent unauthorized access by verifying role and login credentials.
- Performance

  System should load and respond to actions quickly, even with multiple users.
- Data Storage

  Data is stored in SQLite for simplicity and offline access.
- Maintainability

  Modular code using Controllers and Models for easy updates and scalability.

Feasibility Study

A feasibility study was conducted to check if the UnicomTIC Management System project can be developed and used successfully. It includes different types of analysis:

1. Technical Feasibility
    - The system is built using C# (Windows Forms) and SQLite.
    - All the tools used are well-supported, easy to install, and run on standard Windows computers.
    - No special hardware or software is required.
2. Economic Feasibility
    - The project uses free and open-source tools like SQLite and Visual Studio Community Edition.
    - There is no extra cost for licensing or hosting.
    - Ideal for small institutions with limited budgets.
3. Operational Feasibility
    - The system is user-friendly with clear menus and role-based access.
    - Admin, Staff, Lecturers, and Students can easily navigate and use the features after simple training.
    - No complex operations or commands are required.
4. Legal Feasibility
    - The software does not use any copyrighted or paid libraries.
    - All the development tools and technologies used are legally available for educational purposes.

Design

The design of the UnicomTIC Management System focuses on simplicity, role-based access, and easy navigation. The system is built using object-oriented programming principles and follows a structured controller-based pattern.

A. System Architecture

- The project is organized into four main layers:

  - Models: Handle data (Student, Lecturer, Subject, Course, etc.)

  -Controllers: Handle logic and data operations (CRUD)

  -Forms (UI): Handle user interaction via Windows Forms

  -Database (SQLite): Stores all application data in tables

UnicomTICManagementSystem

```
├── Controllers
│   ├ CourseController.cs
│   ├StudentController.cs
│   ├ SubjectController.cs
│   ├ MarksController.cs
│   ├ ExamController.cs
│   ├RoomController.cs
│   ├ TimetableController.cs
│   ├ UserController.cs
│
├── Data
│   ├ DBConfig.cs
│   ├ Migration.cs
│
├── Models
│   ├ Course.cs
│   ├ Student.cs
│   ├ Subject.cs
│   ├ Exam.cs
│   ├ Mark.cs
│   ├ Room.cs
│   ├ Timetable.cs
│   ├ User.cs
│   ├ Lecturer.cs
│   ├ Staff.cs
│
├── Forms
│   ├ LoginForm.cs
│   ├ AdminDashboardForm.cs
│   ├ StudentDashboardForm.cs
│   ├ StaffDashboardForm.cs
│   ├ LecturerDashboardForm.cs
│   ├ CourseForm.cs
│   ├ SubjectForm.cs
│   ├ StudentForm.cs
│   ├ ExamForm.cs
│   ├ MarksForm.cs
│   ├ RoomForm.cs
│   ├ TimetableForm.cs
│   ├ StudentMarksForm.cs
│   ├ ReadOnlyTimetableForm.cs
│   ├ LecturerForm.cs
│
└── Program.cs
```

○ FORMS OVERVIEW

1. LoginForm.cs
   Purpose: Provides login functionality for all user roles (Admin, Student, Lecturer, Staff).
   How it works: Validates credentials and redirects to the correct dashboard based on the selected role.

2. AdminDashboardForm.cs
   Purpose: Admin's main interface.
   Features:
   View/manage courses, students, subjects, exams, marks.
   Handle room and timetable setup.
   Full system access.

3. StudentDashboardForm.cs
   Purpose: Student's home screen after login.
   Features:
   View personal marks, View timetable.
   No editing/deleting capabilities (read-only access).

4. LecturerDashboardForm.cs
   Purpose: Dashboard for lecturers.
   Features:
   View own timetable.
   Access related subject/course info.
   Secure logout access.

5. StaffDashboardForm.cs
   Purpose: Dashboard for staff users.
   Features:
   View system components like timetable and student details (if needed).
   Usually read-only or limited privileges.

6. CourseForm.cs
   Purpose: Manage course records.
   Features:
   Add, update, delete course entries.
   Display course list in a data grid.

7. SubjectForm.cs
   Purpose: Manage subjects under each course.
   Features:
   Assign subjects to courses.
   CRUD operations.

8. StudentForm.cs
   Purpose: Add and manage student records.
   Features:
   Assign students to courses.
   View and update student info.

9. ExamForm.cs
   Purpose: Create/manage exams for subjects.
   Features:
      Assign exams to subjects.
      Set exam names and other details.

10. MarksForm.cs
   Purpose: Enter/update student exam scores.
   Features:
      Assign marks for each exam per student.
      Edit or delete scores.

11. RoomForm.cs
   Purpose: Manage rooms and room types (lab/classroom).
   Features:
      Assign rooms for timetable slots.

12. TimetableForm.cs
   Purpose: Create and manage class timetables.
   Features:
      Assign subjects to timeslots and rooms.
      Full CRUD support.

13. StudentMarksForm.cs
   Purpose: Display student-specific marks in a read-only format.
   Features:
      Used inside the student dashboard.
      Auto-loads the student's name-related scores.

14. ReadOnlyTimetableForm.cs
   Purpose: Show the timetable to logged-in students or lecturers.
   Features:
      Shows timetable based on user ID.
      No update/delete option.

15. LecturerForm.cs
   Purpose: Manage lecturer data.
   Features:
      Add/update lecturer profile.
      Assign subjects/courses.

- CONTROLLERS LOGIC

1. UserController.cs

Purpose:

Handles login validation for different user roles like Student, Lecturer, and Staff.

Key Functions:

ValidateStudentAsync(name, id) → Checks if a student with that name and ID exists.

ValidateLecturerAsync(name, id) → Validates a lecturer login.

ValidateStaffAsync(name, id) → Validates a staff login.

Used in: LoginForm.cs

2. CourseController.cs

Purpose:

Handles CRUD (Create, Read, Update, Delete) operations for courses.

Key Functions:

GetAllAsync() → List all courses.

AddAsync(course) → Add a new course.

UpdateAsync(course) → Update existing course info.

DeleteAsync(courseId) → Delete a course by ID.

Used in: CourseForm.cs

3. SubjectController.cs

Purpose:

Manages subjects and their links to courses.

Key Functions:

GetAllAsync() → View all subjects.

AddAsync(subject) → Add subject.

UpdateAsync(subject) → Update subject.

DeleteAsync(id) → Delete subject.

Used in: SubjectForm.cs

4. StudentController.cs

Purpose:

Handles all operations for student data.

Key Functions:

GetAllAsync() → Show all students.

AddAsync(student) → Add a student.

UpdateAsync(student) → Update a student.

DeleteAsync(id) → Delete student.

Used in: StudentForm.cs

5. ExamController.cs

Purpose:

Manages exam information like exam name and subject.

Key Functions:

GetAllAsync() → View all exams.

AddAsync(exam) → Add exam.

UpdateAsync(exam) → Update exam.

DeleteAsync(id) → Delete exam.

Used in: ExamForm.cs

6. MarksController.cs

Purpose:

Handles marks entry and student-specific mark reports.

Key Functions:

GetAllAsync() → All marks.

GetMarksByStudentIdAsync(id) → Show one student's marks.

GetStudentMarksDetailedAsync(studentId) → Show detailed info: course, subject, exam, score.

AddAsync(mark) → Add a mark.

UpdateAsync(mark) → Update a mark.

DeleteAsync(id) → Delete a mark.

Used in: MarksForm.cs, StudentMarksForm.cs

7. RoomController.cs

Purpose:

Manages data about rooms used for classes or exams.

Key Functions:

GetAllAsync() → List rooms.

AddAsync(room) → Add new room.

UpdateAsync(room) → Change room info.

DeleteAsync(id) → Delete room.

Used in: RoomForm.cs

8. TimetableController.cs

Purpose:

Handles timetable creation and reading.

Key Functions:

GetAllAsync() → All timetable entries.

GetAllTimetablesAsync() → Detailed view with subject, course, room.

GetTimetableForStudentAsync(studentId) → Student's timetable only.

AddAsync(timetable) → Add timetable slot.

UpdateAsync(timetable) → Change timetable.

DeleteAsync(id) → Delete slot.

Used in: TimetableForm.cs, ReadOnlyTimetableForm.cs

B. Use Case Design

Each user role has its own set of actions:

User Role	Features
Admin	Manage students, lecturers, staff, subjects, courses, exams, timetable
Lecture	View assigned timetable
Staff	View and manage basic academic operations (no delete/update for student)
Student	View own marks and personal timetable only

C. Database Design

Key Tables:

- Users: Stores login info (username, password, role)
- Students, Lecturers, Staffs: Role-specific info
- Subjects, Courses: Academic data
- Exams, Marks, Rooms, Timetables: Support exam and scheduling

All tables are linked using foreign keys to maintain data consistency

D. ER Diagram Summary (Text View)

[Users]
- UserID (PK)
- Username
- Password
- Role

[Students]
- StudentID (PK)
- Name
- CourseID (FK → Courses.CourseID)
  ↳ Linked to Users.UserID (Login)

[Lecturers]
- LecturerID (PK)
- Name
- Address
- CourseID (FK → Courses.CourseID)
- SubjectID (FK →Subjects.SubjectID)
  ↳ Linked to Users.UserID (Login)

[Staffs]
- StaffID (PK)
- Name
- Address
- JobTitle
  ↳ Linked to Users.UserID (Login)

[Courses]
- CourseID (PK)
- CourseName

[Subjects]
- SubjectID (PK)
- SubjectName
- CourseID (FK → Courses.CourseID)

[Exams]
- ExamID (PK)
- ExamName
- SubjectID (FK → Subjects.SubjectID)

[Marks]
- MarkID (PK)
- StudentID (FK → Students.StudentID)
- ExamID (FK → Exams.ExamID)
- Score

[Rooms]
- RoomID (PK)
- RoomName
- RoomType

[Timetables]
- TimetableID (PK)
- SubjectID (FK → Subjects.SubjectID)
- RoomID (FK → Rooms.RoomID)
- TimeSlot

Schedule

The development of the UnicomTIC Management System was divided into multiple structured phases to ensure systematic progress and timely completion. Below is the schedule outlining the key stages of development.

- Work Breakdown Structure (WBS):

1. Requirement Analysis

– Understand the roles (Admin, Staff, Lecturer, Student)

– Define functional requirements

– Design basic data structure

2. System Design

– Design database (ER diagram)

– Create user interface mockups (Forms for Login, Dashboard, CRUD)

3. Development

– Implement database connection (SQLite)

– Develop forms and controllers

– Add authentication and role-based dashboards

4. Testing

– Unit testing for forms and validation

– Integration testing of forms and database

– Fix bugs (login redirection, table creation, role access)

5. Finalization & Deployment

– Final cleanup of code

– Prepare project report

– Push to GitHub repository

- Gantt Chart (Timeline Example):

Days	Task
1-3	Requirement gathering & planning
4-6	Database schema design (ER + tables)
7-11	Login form, Role selection logic
12-15	Dashboard and CRUD form development
16-18	Testing, debugging, and improvements
19-21	Report writing and GitHub deployment

Conclusion

The UnicomTIC Management System was developed to provide a comprehensive and user-friendly solution for managing academic operations such as student details, lecturer assignments, course and subject management, exam marks, and timetable scheduling.

This system effectively addresses the limitations of traditional manual processes by introducing a role-based dashboard that offers different functionalities to Admins, Staff, Lecturers, and Students. With features like secure login, CRUD operations, read-only student views, and database connectivity via SQLite, it ensures that academic data is better organized, easily accessible, and protected.

Throughout the development process, real-world application design principles and object-oriented programming concepts were implemented, leading to a modular and scalable solution.

This project has not only improved efficiency in managing academic workflows but also enhanced my technical understanding of full-stack Windows Forms development, database operations, and user authentication systems

Code Screenshots and Explanation

1. LoginForm.cs

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

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

    var userController = new UserController();

    if (selectedRole == "Student")
    {
        var user = await userController.ValidateStudentAsync(username, password);
        if (user != null)
        {
            MessageBox.Show("Login successful as Student", "Success");
            this.Hide();
            new StudentDashbordForm(user.UserId).Show();
        }
        else
        {
            MessageBox.Show("Invalid Student name or ID", "Login Failed");
        }
    }
    else if (selectedRole == "Lecturer")
    {
        var user = await userController.ValidateLecturerAsync(username, password);
        if (user != null)
        {
            MessageBox.Show("Lecturer login successful");
            this.Hide();
            new LecturerDashbordForm(user.UserId).Show();
        }
        else
        {
            MessageBox.Show("Invalid lecturer login");

        }

    }
    else if (selectedRole == "Staff")
    {
        var user = await userController.ValidateStaffAsync(username, password);
        if (user != null)
        {
            MessageBox.Show("Login successful as Staff", "Success");
            this.Hide();
            new StaffDashbordForm(user.UserId).Show();
        }
        else
```

No issues found

This part of the code handles role-based login. Based on the selected role and credentials entered, it validates the user and redirects them to their specific dashboard.

2. UserController.cs

```
private readonly DBConfig config = new DBConfig();

1 reference
public async Task<User> ValidateStudentAsync(string name, string studentId)
{
    using (var conn = DBConfig.GetConnection())
    {
        var cmd = new SQLiteCommand("SELECT * FROM Students WHERE Name = @name AND StudentID = @id", conn);
        cmd.Parameters.AddWithValue("@name", name);
        cmd.Parameters.AddWithValue("@id", studentId);

        using (var reader = await cmd.ExecuteReaderAsync())
        {
            if (await reader.ReadAsync())
            {
                return new User
                {
                    UserId = Convert.ToInt32(reader["StudentID"]),
                    UserName = reader["Name"].ToString(),
                    Password = reader["StudentID"].ToString(),
                    Role = "Student"
                };
            }
        }
    }


    return null;
}


1 reference
public async Task<User> ValidateLecturerAsync(string name, string id)
{
    using (var conn = DBConfig.GetConnection())
    {
        var cmd = new SQLiteCommand("SELECT * FROM Lecturers WHERE Name = @name AND LecturerID = @id", conn);
        cmd.Parameters.AddWithValue("@name", name);
        cmd.Parameters.AddWithValue("@id", id);

        using (var reader = await cmd.ExecuteReaderAsync())
        {
            if (await reader.ReadAsync())
            {
                return new User
                {
                    UserId = Convert.ToInt32(reader["LecturerID"]),
                    UserName = reader["Name"].ToString(),
                    Role = "Lecturer"
                };
            }
        }
    }
```

This controller validates students and lecturers from the database using their name and ID. It ensures only valid users can log in and view their personalized information.

3. StudentMarksForm.cs

```
2 references
private async void StudentMarksForm_Load_1(object sender, EventArgs e)
{
    DataTable marksTable = await marksController.GetStudentMarksDetailedAsync(studentId);
    dgvStudentMarks.DataSource = marksTable;

    dgvStudentMarks.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
    dgvStudentMarks.ReadOnly = true;
    dgvStudentMarks.AllowUserToAddRows = false;
    dgvStudentMarks.AllowUserToDeleteRows = false;




}
```

This code displays marks related to the logged-in student only. It uses a controller method to fetch data securely and shows it in a read-only table.

4. Migration.cs

```
using System.Threading.Tasks;
using UnicomTICManagementSystem.Models;

namespace UnicomTICManagementSystem.Data
{
    2 references
    internal class Migration
    {
        1 reference
        public void Create_Table()
        {
            using (var connection = DBConfig.GetConnection())
            {

                string query = @"
                                CREATE TABLE IF NOT EXISTS Users (UserID INTEGER PRIMARY KEY,Username TEXT,Password TEXT,Role TEXT);

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

                                CREATE TABLE IF NOT EXISTS Subjects (SubjectID INTEGER PRIMARY KEY,SubjectName TEXT,CourseID INTEGER,FOREIGN KEY(CourseID) REFERENCES Courses(CourseID));

                                CREATE TABLE IF NOT EXISTS Students (StudentID INTEGER PRIMARY KEY, Name TEXT,CourseID INTEGER,FOREIGN KEY(CourseID) REFERENCES Courses(CourseID));

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

                                CREATE TABLE IF NOT EXISTS Marks (MarkID INTEGER PRIMARY KEY,StudentID INTEGER,ExamID INTEGER,Score INTEGER,FOREIGN KEY(StudentID) REFERENCES Students(StudentID),FOREIG

                                CREATE TABLE IF NOT EXISTS Rooms (RoomID INTEGER PRIMARY KEY,RoomName TEXT,RoomType TEXT);

                                CREATE TABLE IF NOT EXISTS Timetables (TimetableID INTEGER PRIMARY KEY,SubjectID INTEGER,TimeSlot TEXT,RoomID INTEGER,FOREIGN KEY(SubjectID) REFERENCES Subjects(Subject

                                CREATE TABLE IF NOT EXISTS Lecturers (LecturerID INTEGER PRIMARY KEY,Name TEXT,Address TEXT,CourseID INTEGER,SubjectID INTEGER,FOREIGN KEY(CourseID) REFERENCES Courses(

                                CREATE TABLE IF NOT EXISTS Staffs (StaffID INTEGER PRIMARY KEY,Name TEXT,Address TEXT,JobTitle TEXT );

                                ";
                using (SQLiteCommand cmd = new SQLiteCommand(query, connection))
                {
                    cmd.ExecuteNonQuery();
                }
            }
```

This file defines and creates all required tables for the system (Users, Students, Exams, Marks, etc.). It ensures the database is correctly structured for academic management.

5.ReadOnlyTimetableForm.cs

```
{
    private readonly string role;
    private readonly int userId;
    private readonly TimetableController controller = new TimetableController();




    3 references
    public ReadOnlyTimetableForm(string role, int userId)

    {
        InitializeComponent();
        this.role = role;
        this.userId = userId;
        this.Load += ReadOnlyTimetableForm_Load;


    }

    2 references
    private async void ReadOnlyTimetableForm_Load(object sender, EventArgs e)
    {
        DataTable timetable = null;

        if (role == "Student")
        {
            timetable = await controller.GetTimetableForStudentAsync(userId);
        }
        else if (role == "Lecturer")
        {
            timetable = await controller.GetTimetableByLecturerIdAsync(userId);
        }
        else if (role == "Staff" || role == "Admin")
        {
            timetable = await controller.GetAllTimetablesAsync();
        }

        if (timetable != null)
        {
            dgvTimetable.DataSource = timetable;
            dgvTimetable.ReadOnly = true;
            dgvTimetable.AllowUserToAddRows = false;
            dgvTimetable.AllowUserToDeleteRows = false;
            dgvTimetable.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
        }
        else
        {
            MessageBox.Show("No timetable data available.");
        }


    }
```

This form loads the timetable assigned to a student or lecturer. It is fully read-only and prevents users from making changes, maintaining system integrity.

6.TimetableController.cs

```
1 reference
public async Task<DataTable> GetAllTimetablesAsync()
{
    using (var conn = DBConfig.GetConnection())
    {
        string query = @"
    SELECT
        s.SubjectName,
        c.CourseName,
        t.TimeSlot,
        r.RoomName,
        r.RoomType
    FROM Timetables t
    INNER JOIN Subjects s ON t.SubjectID = s.SubjectID
    INNER JOIN Courses c ON s.CourseID = c.CourseID
    INNER JOIN Rooms r ON t.RoomID = r.RoomID";

        var cmd = new SQLiteCommand(query, conn);
        var adapter = new SQLiteDataAdapter(cmd);
        var table = new DataTable();
        await Task.Run(() => adapter.Fill(table));
        return table;
    }
}
1 reference
public async Task<DataTable> GetTimetableForStudentAsync(int studentId)
{
    using (var conn = DBConfig.GetConnection())
    {
        string query = @"
    SELECT
        s.SubjectName,
        c.CourseName,
        t.TimeSlot,
        r.RoomName,
        r.RoomType
    FROM Timetables t
    INNER JOIN Subjects s ON t.SubjectID = s.SubjectID
    INNER JOIN Courses c ON s.CourseID = c.CourseID
    INNER JOIN Rooms r ON t.RoomID = r.RoomID
    INNER JOIN Students st ON st.CourseID = c.CourseID
    WHERE st.StudentID = @studentId";

        var cmd = new SQLiteCommand(query, conn);
        cmd.Parameters.AddWithValue("@studentId", studentId);

        var adapter = new SQLiteDataAdapter(cmd);
        var table = new DataTable();
        await Task.Run(() => adapter.Fill(table));
        return table;
    }
```

This controller method uses SQL JOIN queries to fetch timetable data including subjects, rooms, and course information. It ensures that the timetable shown is complete and detailed.

References

-Microsoft Docs - Windows Forms Documentation

-SQLite Documentation

-C# Programming Guide - Microsoft

-ChatGPT

-W3Schools C# and SQL Tutorials

-Practical guidance and support from Mr. Kathir(Lecturer),Department of Computer Science