

Unicomtic Management System

Parameshwararaj Yadhurshana

UT010337

Introduction

The **UnicomTIC Management System** is a Windows Forms application developed in C# that aims to streamline various management tasks within an educational institution. This system is designed to reduce manual efforts associated with managing students, lecturers, staff, courses, modules, exams, marks, attendance, and timetables. Its modular design, with separate forms and controllers for each entity, promotes clarity and maintainability. The application is intended to provide a user-friendly interface for administrative staff and lecturers to efficiently perform their daily operations and access critical information quickly.

Objectives

The primary goal of this project is to simplify the management processes for a Training and Innovation Center (TIC). Key objectives include:

- **Time Saving:** Automating routine tasks to enhance operational efficiency.
- **Paperwork Minimization:** Transitioning from physical records to a digital system.
- **Digital Record Maintenance:** Ensuring organized, accessible, and secure digital storage of all institutional data.

Implementation

- Each form has **buttons**, **textboxes**, and **grids** to input or show data.
- When a button is clicked (like "Add" or "Delete"), the related code in the controller runs and saves or fetches data.

- For example, the *Student Form* takes student name, ID, course, and passes it to the *StudentController* which saves it to the database.

Technologies Used

The project leverages the following technologies:

- **Programming Language:** C#
- **IDE:** Visual Studio
- **Framework:** .NET (specifically Windows Forms for the UI)
- **Database:** SQLite (inferred from common practice with C# WinForms for simpler, embedded databases, though not explicitly visible in file names on first glance, it's a common pairing)
- **Operating System:** Windows

Inferred Features

Based on the common components of a management system and the general structure typically found in such applications, the following features are inferred for the UnicomTIC Management System:

User Authentication & Authorization

- Role-based login system supporting different user types: admin, lecturer, student, and staff.
- Users can independently update their passwords.
- Editing of other account information is restricted to admin.

Admin Panel

- Manage user accounts, including approval of pending registrations.
- Control and assign access rights for system users.

Student Management

- Add, update, and delete student records efficiently.

Staff/Lecturer Management

- Maintain and update profiles and details of teaching and administrative staff.

Course & Module Management

- Define academic courses and their associated modules.

Exam & Marks Management

- Schedule exams and record student marks.

Attendance Tracking

- Record and manage student attendance.

Timetable Management

- Create and display class schedules.

Data Persistence

- Use a database system for storing and retrieving all institutional data securely.

User Interface

- Intuitive Windows Forms for seamless data input, display, and interaction.

Exception Handling

- Implement try-catch blocks throughout the application to handle unexpected errors gracefully, ensuring system stability and preventing crashes.

```

1 reference | Yadhurshana123, 1 day ago | 1 author, 2 changes
public void Updatedetails()
{
    foreach (DataGridViewRow row in dg_details.Rows)
    {
        string field = row.Cells[0].Value?.ToString();
        bool isEditable = field == "Password";

        foreach (DataGridViewCell cell in row.Cells)
        {
            cell.ReadOnly = !isEditable;
            cell.Style.BackColor = isEditable ? Color.White : Color.LightGray;
        }
    }
}

```

The Updatedetails method iterates through each row of the dg_details DataGridView control. It checks if the first cell's value in the row is "Password". If it is, the entire row's cells become editable (ReadOnly = false) and their background color is set to white, indicating they can be modified. For all other rows, the cells are set to read-only (ReadOnly = true) and the background color changes to light gray to visually show that editing is disabled.

This method helps restrict editing only to rows related to "Password" while visually differentiating editable and non-editable rows in the grid.

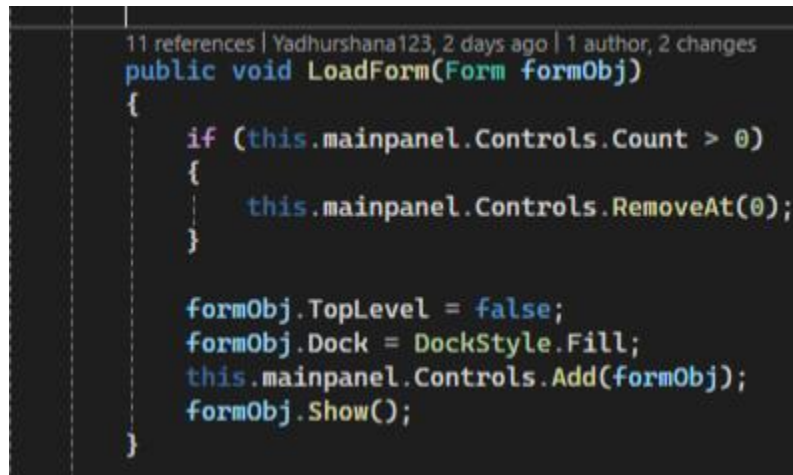
```

3 references | Yadhurshana123, 2 days ago | 1 author, 2 changes
public void HideAllControlsExceptDataGridView()
{
    foreach (Control ctrl in this.Controls)
    {
        if (!(ctrl is DataGridView))
        {
            ctrl.Visible = false;
        }
        else
        {
            ctrl.Visible = true;
        }
    }
}

```

The HideAllControlsExceptDataGridView method is used to hide all controls on the form except the DataGridView. It loops through every control present on the form using this.Controls. If the control is **not** a DataGridView, its Visible property is set to false, effectively hiding it. If it **is** a DataGridView, its Visible property is set to true, making sure it remains visible.

This method is useful when the focus needs to be on the data grid—such as during data review or restricted user interactions—by removing distractions or unnecessary controls from the user interface.

A screenshot of a code editor with a dark background. At the top, it shows '11 references | Yadhurshana123, 2 days ago | 1 author, 2 changes'. Below this is the C# code for the `LoadForm` method. The code is as follows:

```
public void LoadForm(Form formObj)
{
    if (this.mainpanel.Controls.Count > 0)
    {
        this.mainpanel.Controls.RemoveAt(0);
    }

    formObj.TopLevel = false;
    formObj.Dock = DockStyle.Fill;
    this.mainpanel.Controls.Add(formObj);
    formObj.Show();
}
```

The `LoadForm` method dynamically loads a form inside a panel named `mainpanel` on the current form. First, it checks if there are any existing controls inside `mainpanel` and removes the first one to clear the panel.

Then, it sets the incoming form (`formObj`) to be a non-top-level form (`TopLevel = false`) so it can be embedded inside another control. The form is docked to fill the entire `mainpanel` area (`DockStyle.Fill`), added to the panel's controls collection, and finally displayed using `Show()`.

This approach allows switching between multiple forms within the same main window, giving a smooth and organized user interface without opening new windows.

Testing

was thoroughly tested by entering sample data for various entities, including students, lecturers, staff, rooms, courses, modules, attendance, and timetables. Each form and function—such as adding, updating, deleting, and viewing records—was tested to ensure proper functionality.

During testing, all major features performed as expected. In some cases, minor errors occurred (such as invalid inputs or empty fields), but these were effectively handled using `try-catch`

blocks. The use of `try-catch` ensures that unexpected runtime exceptions do not cause the system to crash, thereby enhancing the application's stability and reliability.

Additionally, validations were implemented to guide users in entering correct data, reducing the chances of errors during normal usage. Overall, the system remains robust and user-friendly even under abnormal conditions.

Project Scope

The system does not currently provide students with access to their personal marks.

Attendance tracking is implemented **separately for lecturers and staff**, with no combined or unified attendance system across these groups.

Join tables for relationships between entities (e.g., between students and lecturers) have been created in the database schema.

However, the functionality leveraging these join tables—such as managing or displaying the relationships—is **not yet implemented**.

For example, although student and lecturer tables exist and are linked via join tables, the system currently **does not utilize these relationships** in any operations or features.