UNICOM TIC REPORT

REALLY GREAT SITE

JUNE 2025

1.Project Overview

2.Features

1.🛠️ Technologies Used

2.Challenges & Solutions

3.Code Samples

Password

Admin =username (Admin)
password (Admin123)

Lecturer=username (Lecturer)
password (Lecturer123)

Student =username (Student)
password (Student123)

Staff=username (Staff)
password (Staff123)

Project Overview

- Unicom TIC Management System is a C# Windows Forms application designed to simplify college academic operations. It primarily focuses on timetable scheduling and internal marks management. The system features role-based access control, providing distinct functionalities for Admin, Lecturer, Student, and Staff users. Admins can manage courses, subjects, users, timetables, and staff accounts. Lecturers can enter and update student marks and view class schedules. Students can access their own timetables and marks records. Staff members handle college administrative and academic responsibilities within the system. SQLite is used for data storage, and BCrypt ensures secure password hashing. This system facilitates smooth academic workflows while enhancing data transparency and accessibility.

Features

1. Admin Dashboard

Description:

The Admin Dashboard serves as the central hub for administrators. Here, they can comprehensively manage Courses, Subjects, Users, Timetables, and Staff all in one place.

Examples:

- Add new courses
- Assign subjects to lecturers
- Create and update accounts for students and lecturers
  Modify timetable schedules

2. Lecturer and Student Logins

Description:

Lecturers and Students log in with unique usernames and passwords. This ensures secure access to their personal information and marks.
Examples:

- Lecturers can view their class schedules and internal marks pages
- Students can only view their own marks and timetables

3. Create, View, and Edit Timetables

Description:

Admins have the ability to create and modify timetables. Timetables are built with details such as class, day, time, room, and lecturer.
Examples:

- Add a new timetable for a class
- Edit existing timetables
- Students and lecturers can view the finalized timetable

4. Assign Subjects to Lecturers

Description:

Functionality to assign subjects to lecturers, defining who teaches which subject.
Examples:

- Admin assigns the subject "Mathematics" to "Mr. Kumar"
- Mr. Kumar can then enter marks for his classes

5. Enter and View Internal Marks
Description:
Lecturers can input internal marks for students, while students can securely view their marks.
Examples:

- Lecturer adds marks for an internal exam
- Student accesses their marks online

6. Password Hashing using BCrypt

Description:

A security mechanism to safely store user passwords. Passwords are hashed using BCrypt, preventing data theft.
Examples:

- User passwords are encrypted using BCrypt before storage
- Login validation compares the hashed password securely
-

7. SQLite Lightweight Local Database

Description:

SQLite is a lightweight, file-based database ideal for fast, simple data storage and access on a local machine.
Examples:

- All data related to students, lecturers, courses, timetables, and marks is stored in SQLite
- Provides quick data retrieval and updates

2.🛠️ Technologies Used

1. Programming Language

- C# (.NET Framework)
- ➤ Windows Forms Application (WinForms)

2.UI Framework

- Windows Forms (WinForms)
- ➤ Desktop-based graphical UI design.
- ➤ Multiple forms like LoginForm, SubjectForm, StaffView, etc.
-

3. Database

- SQLite
- ➤ Lightweight, file-based RDBMS
- ➤ Tables: Users, Subjects, Marks, TimeTable
- ➤ Connection handled in DB.cs
-

4. Security

- BCrypt.Net
- ➤ Password hashing and verification
- ➤ Prevents storing plain-text passwords

5. External Libraries (via NuGet)

- System.Data.SQLite
- ➤ .NET bindings for SQLite
- BCrypt.Net-Next
- ➤ For secure password hashing
-

6. Architecture

- MVC-inspired structure (manual, not full ASP.NET MVC)
  - Controllers/ — Logic
  - Model/ — POCO classes (e.g., User.cs, Mark.cs)
  - Views/ — UI Forms (.cs + .Designer.cs)

7.Development Tools

- Visual Studio (Recommended IDE)
- ➤ UI design + SQLite integration
- ➤ Debugging & Build

Challenges & Solutions

1. Secure User Authentication

- Challenge:

Storing passwords in plain text is insecure and vulnerable to attacks.

- Solution:

You used the BCrypt.Net library to securely hash passwords before storing them in the database.
During login, the system uses BCrypt.Verify() to compare the user's entered password with the hashed password stored in the database, ensuring secure authentication.

2. Database Locking Error (SQLite)

- Challenge:

The error database is locked occurs in SQLite when multiple operations try to access the database at the same time.

- Solution:

1. Properly use using blocks to ensure connections are automatically closed after operations.
2. Minimize connection open-close time to reduce transaction overload and avoid long locks.

3. Role-based Access Control

- Challenge:

Different users like Admin, Staff, and Student require separate UI logic and access permissions.

- Solution:

You added a Role field to the Users table.
After login, you used logic like:
csharp
CopyEdit
if (role == "Admin")
else if (role == "Staff")
else if (role == "Student")

4. Data Validation

- Challenge:
-

The application was crashing when users submitted forms with empty fields or invalid data (e.g., incorrect marks) due to the lack of input validation.

- Solution:
-

Validation logic was added to check form inputs before processing. For example:

Code Samples

1.Role-based UI Navigationt

```
if (role == "Admin")
{
    Adminview adminForm = new Adminview(username, role, this);
    adminForm.FormClosed += (s, args) => this.Close();
    adminForm.Show();
}
else if (role == "Lecturer")
{
    TimetableController controller = new TimetableController();
    DataTable dt = controller.GetLecturersAsDataTable();

    Lecturerview lecturerForm = new Lecturerview(dt);
    lecturerForm.FormClosed += (s, args) => this.Close();
    lecturerForm.Show();
}
else if (role == "Student")
{
    Students studentsForm = new Students(username, this);
    StudentViewForm viewForm = new StudentViewForm(studentsForm);
    viewForm.FormClosed += (s, args) => this.Close();
```

uses if-else if statements to control program flow based on a role variable.
If role is "Admin", an Adminview form is created, passing username, role, and this (presumably the current form). A FormClosed event handler is added to close the current form when adminForm closes, and then adminForm is displayed.
Similarly, if role is "Lecturer", a TimetableController fetches data, and a Lecturerview form is shown, closing the current form upon Lecturerview closure.
For "Student" and "Staff" roles, corresponding Students and Staffview forms are instantiated and displayed, with similar FormClosed event handling to close the parent form. An else block is present for unhandled roles.

```
if (reader.Read())
{
    string hashedPassword = reader["Password"].ToString();
    string inputPassword = userpassword.Text.Trim();


    Console.WriteLine($"Hashed DB pwd: {hashedPassword}");
    Console.WriteLine($"Input pwd: {inputPassword}");
```

- This retrieves the hashed password stored in the database using a DataReader.
- Assumes you've already called reader.Read();.
- Gets the user's entered password from a TextBox (userpassword) and removes any leading/trailing spaces using .Trim().

```
    return;
}

// ☑ Verify bcrypt password
if (BCrypt.Net.BCrypt.Verify(inputPassword, hashedPassword))
{
    string role = reader["Role"].ToString();
    string username = reader["UserName"].ToString();

    this.Hide();
```

This code securely authenticates a user by verifying their entered password using the BCrypt algorithm. Upon successful authentication, it retrieves user details (role and username) from the database and hides the current form to transition to the next screen.

```
private DataTable lecturerData;

public Lecturerview(DataTable data)
{
    InitializeComponent();
    lecturerData = data;
    this.Load += Lecturerview_Load;
}

private void Lecturerview_Load(object sender, EventArgs e)
{
    lecrviwe.DataSource = lecturerData;
}
```

Your paraWhen creating the Lecturerview form, you need to pass a DataTable.
When loading the form, you bind that data to a DataGridView to display it.
This is a useful structure for transferring data between forms.
graph text

```
ublic DataTable GetAllLecturers()

  DataTable dt = new DataTable();

  foreach (DataGridViewColumn column in lecrviwe.Columns)
  {
      dt.Columns.Add(column.Name);
  }

  foreach (DataGridViewRow row in lecrviwe.Rows)
  {
      if (!row.IsNewRow)
      {
          DataRow dr = dt.NewRow();
          foreach (DataGridViewColumn column in lecrviwe.Columns)
          {
              dr[column.Name] = row.Cells[column.Name].Value ?? DBNull.Value;
          }
          dt.Rows.Add(dr);
      }
  }
```

This method creates a DataTable by copying all columns and non-empty rows from a DataGridView named lecrviwe. It first adds columns to the DataTable, then loops through each existing row, skips new rows, copies cell values into a new DataRow, and adds it to the table.

```
public void AddCourse(Course course)
{
    try
    {
        using (var con = DB.GetConnection())
        {
            var command = new SQLiteCommand("INSERT INTO Course(CourseName) VALUES(@name)", con);
            command.Parameters.AddWithValue("@name", course.CourseName);
            command.ExecuteNonQuery();
        }
    }
    catch (SQLiteException ex)
    {

        Console.Error.WriteLine($"Database error inserting course: {ex.Message}");

    }
    catch (Exception ex)
    {

        Console.Error.WriteLine($"Unexpected error inserting course: {ex.Message}");

    }
}
```

Your paraThe AddCourse method adds a new course to the database. It opens a connection, prepares an SQL insert command, binds the course name securely using parameters, and executes the query. It uses try-catch blocks to handle SQLite-specific and general exceptions, ensuring the application handles errors gracefully and securely.
graph text