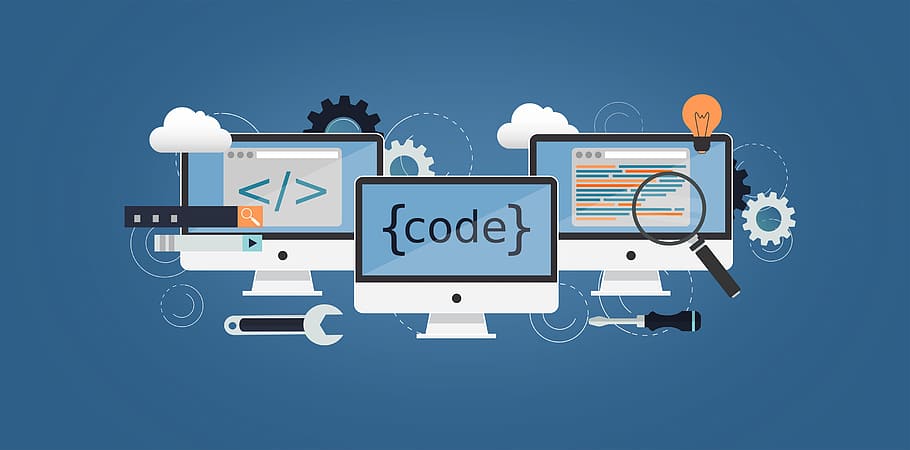
****

**Unicom *TIC* Management System**

****

**Project Submission Report-2025**

**J.Kirushanth.**

***Group B***

**Project Title** : **Unicom TIC School Management System**

**🔹 Key Features Implemented:**

1. **Ole-Based Login System** **(Admin, Staff, Lecturer & Student)**

* **Multiple User Roles** implemented: Admin, Staff, Lecturer, and Student.
* **Secure Login Validation**: Each user must enter valid username and password to access the system.
* **Role-Specific Access Control**:
* **Admin**: Full access to manage users, students, lecturers, courses, and timetable.
* **Staff**: Limited access to student and exam management.
* **Lecturer**: Can view and manage marks and assigned subjects.
* **Student**: Can view their marks and personal timetable.
* **Centralized Users Table**: All user credentials are stored in one table with Role and UserID for easy management.
* **Dynamic Redirection**: After login, each user is redirected to their respective dashboard (e.g., AdminDashboardForm, StudentDashboardForm).
* **Invalid Login Handling**: Shows error messages for incorrect username/password or inactive users.
* **Reusability and Maintenance**: Login logic is centralized and easily extendable for future roles if needed.

### Key Points for ****User Management via AdminPermissionForm****:

* **Centralized User Management**: One form (AdminPermissionForm) allows the admin to manage **Admins, Lecturers, Staff**, and **Students** from a single interface.
* **Full CRUD Operations**:
  + **Create**: Add new users by selecting role and entering user details.
  + **Read**: View all users with filtering based on role (Admin, Staff, etc.).
  + **Update**: Edit user details such as name, phone number, or credentials.
  + **Delete**: Remove user records safely with confirmation prompts.
* **Role-Based Entry Logic**: Each added user is stored in the Users table along with their role and UserID, and linked to their role-specific tables.
* **Search/Filter Functionality**: Users can be filtered by role using a search box or dropdown, making large data easier to manage.
* **Auto-Generated IDs**: Each user is assigned a unique UserID automatically to avoid duplication.
* **Validation & Error Handling**: Proper input checks and error messages to prevent incomplete or invalid data entries.
* **DataGridView Integration**: User records are displayed in a clean grid with row selection to support easy updates and deletions.

### Key Points for ****Course & Subject Management with Course-Subject Linkage****:

* **Course & Subject Modules**: Separate forms created to manage **Courses** and their respective **Subjects**.
* **Course-Subject Relationship**: Each **Subject** is linked to a **Course** using a CourseID foreign key, ensuring data consistency and structured hierarchy.
* **Add / Update / Delete** Support:.
* Admins can easily create, modify, or delete both Courses and Subjects
* **Dropdown Integration**: While adding a Subject, the associated Course can be selected from a ComboBox, which loads data dynamically.
* **Auto Display of Course Name**: When Subjects are displayed in a grid, their corresponding Course name is shown (via JOIN queries).
* **Validation Checks**: Prevents adding a Subject without selecting a Course and avoids duplicates.
* **DataGridView UI**: Courses and Subjects are shown in organized tables with sorting and selection features for easier management.
* **Back-End Logic**: Proper use of **controllers** to handle database interactions, ensuring maintainability and separation of concerns.

### ****Role-Based on Dashboards - Key Points****

### Admin Dashboard

* **Full User Management Access** (Admins, Lecturers, Staff, Students) via AdminPermissionForm.
* Can manage **Courses**, **Subjects**, **Timetables**, **Rooms**, and **Marks**.
* **Overview Table/Grids** for all modules with Add, Update, Delete functionality.
* Access to every part of the system for configuration and maintenance.
* Admin-exclusive features like reports, permission settings, and data overview.

### Staff Dashboard

* Focused on **Marks Management** and **Exam Entry**.
* Staff can enter, update, and delete marks for students.
* Can view **Timetables** to check scheduled classes or exams.
* Read-only access to Students and Subjects to support accurate mark entry.

### Lecturer Dashboard

* Can view **Assigned Subjects**, **Timetable**, and relevant course data.
* See when and where lectures are scheduled (lab/hall).
* Future expansion: could include attendance, notes upload, etc.
* Limited write-access (only view-related operations for current implementation).

### Student Dashboard

* Personalized view showing **own Marks** and **Timetable** only.
* Timetable shows Subject, Room, RoomType, and TimeSlot.
* Marks display includes Subject-wise performance.
* Highly restricted access to ensure student sees only self-related data.

**🔹 Technology Used:**

* **Language**: C# (.NET Framework)
* **UI**: WinForms (Windows Forms Application)
* **Database**: SQLite
* **Architecture**: MVC (Models, Views, Controllers)
* **External Tools**: GitHub for version control, Visual Studio

**🔹 Challenges Faced & Solutions – Key Points**

### Forign Key Constraint Error :

* **Issue**: Foreign key violations during insert/update (e.g., SubjectID or RoomID not found).
* **Solution**: Enabled SQLite constraints by using PRAGMA foreign\_keys = ON; inside every controller.

### Subject Not adding Properly :

* **Issue**: Subject added successfully message appeared, but record was not saved.
* **Solution**: Fixed SQL syntax & ensured parameters matched column types. Added detailed exception messages for debugging.

### Timetable Combobox not Loarding Subjects/Rooms :

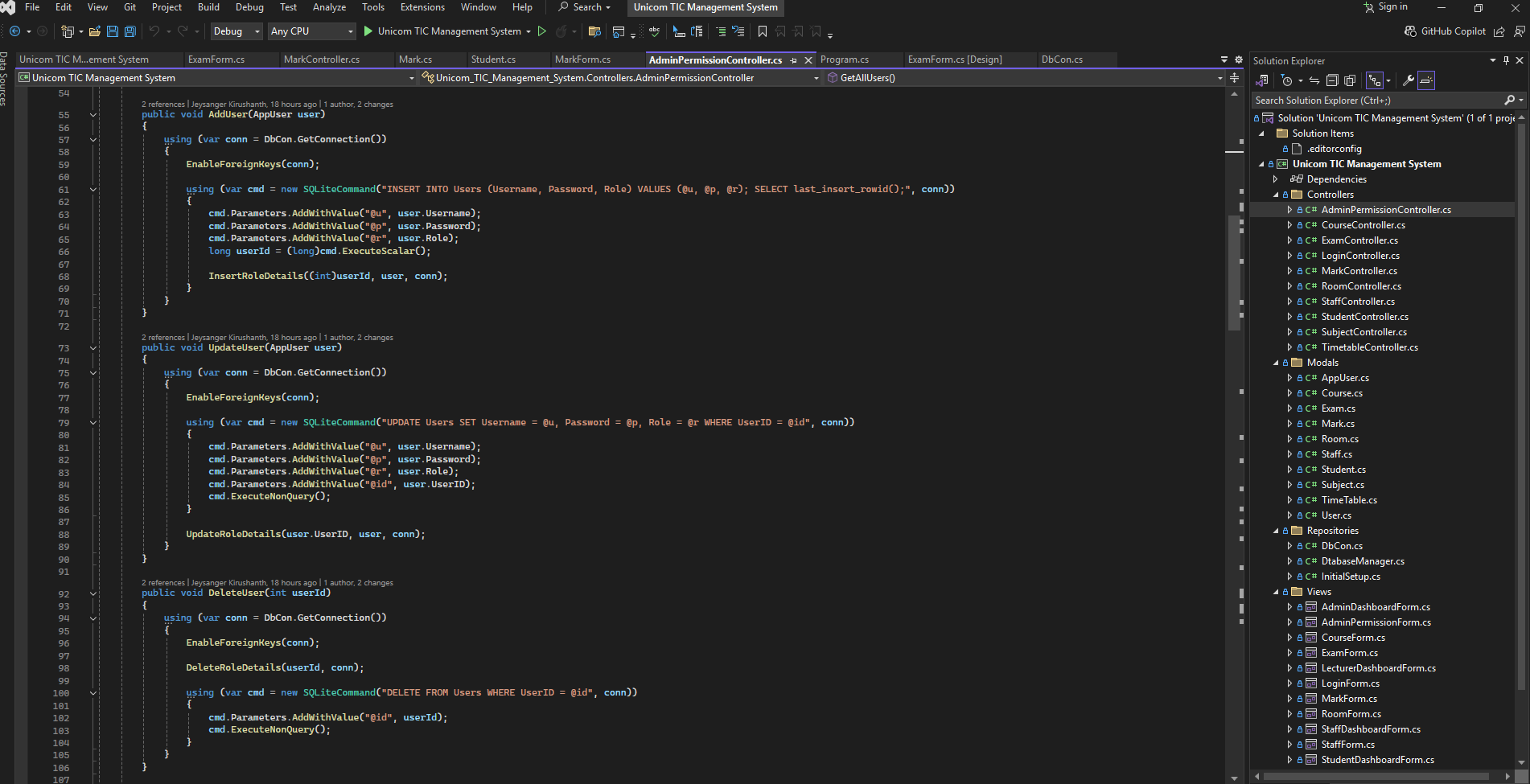
* **Issue**: Subject and Room dropdowns were hardcoded and didn't reflect database updates.
* **Solution**: Implemented dynamic loading from **Subjects** and **Rooms** tables using controller methods.

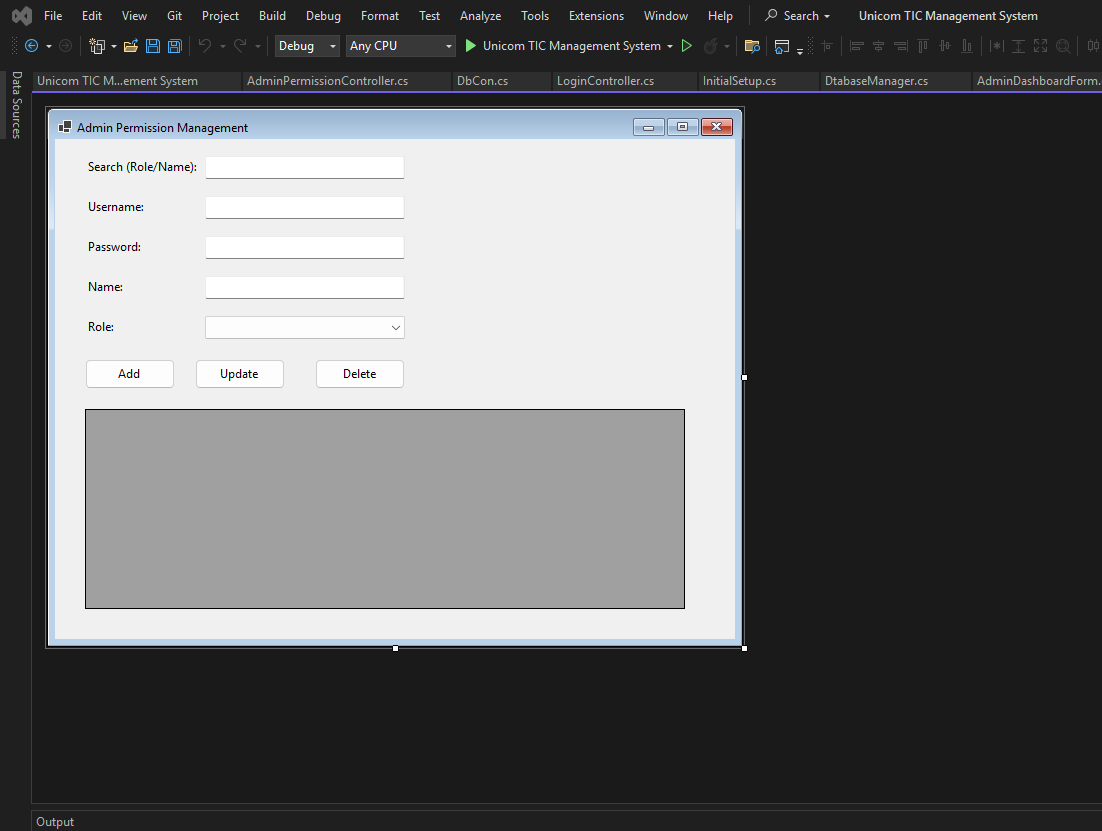
### Subject Not adding Properly :

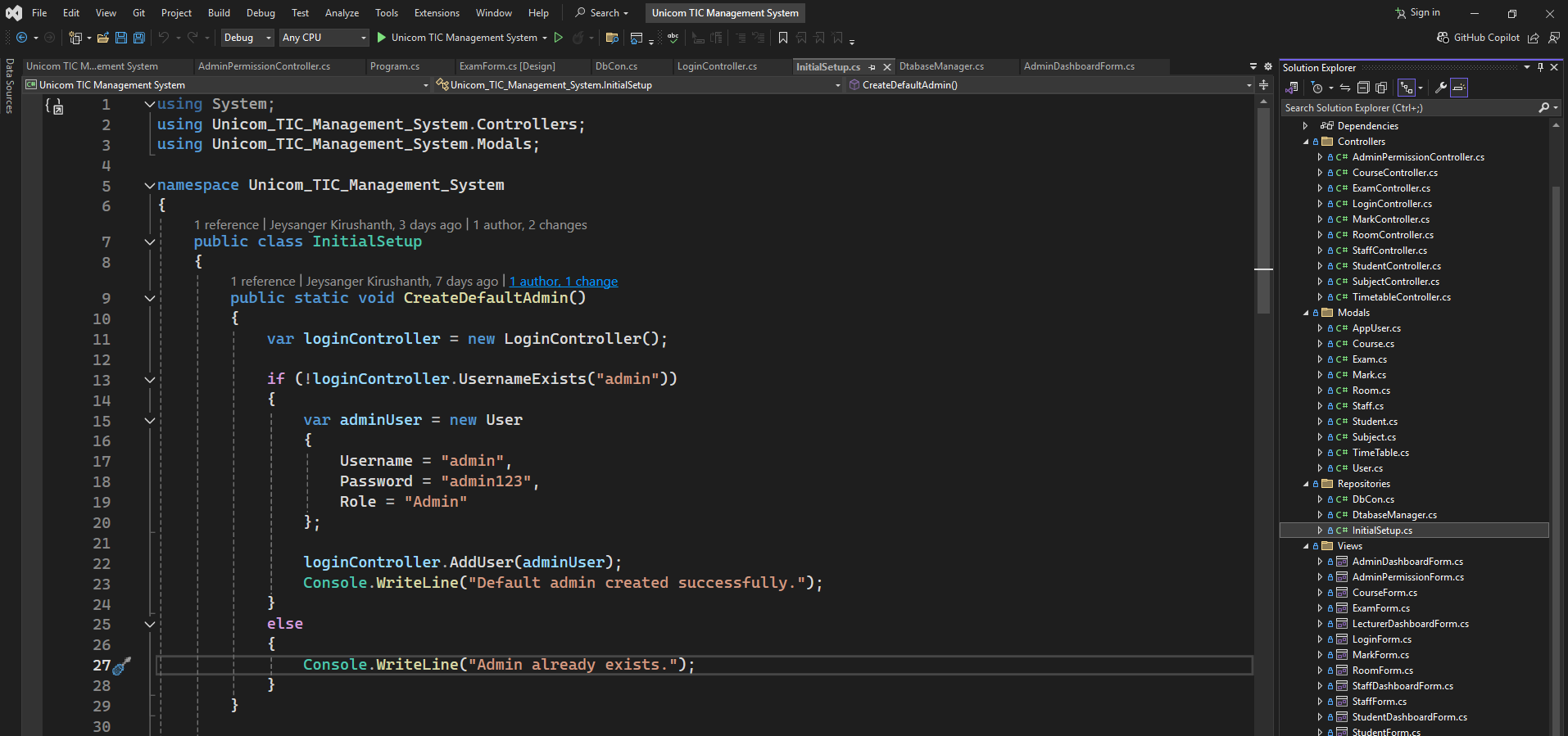
* **Issue**: Selected row values were not populating input fields for update/delete actions.
* **Solution**: Properly used SelectionChanged event and accessed SelectedRows[0].Cells[...] for accurate data binding.

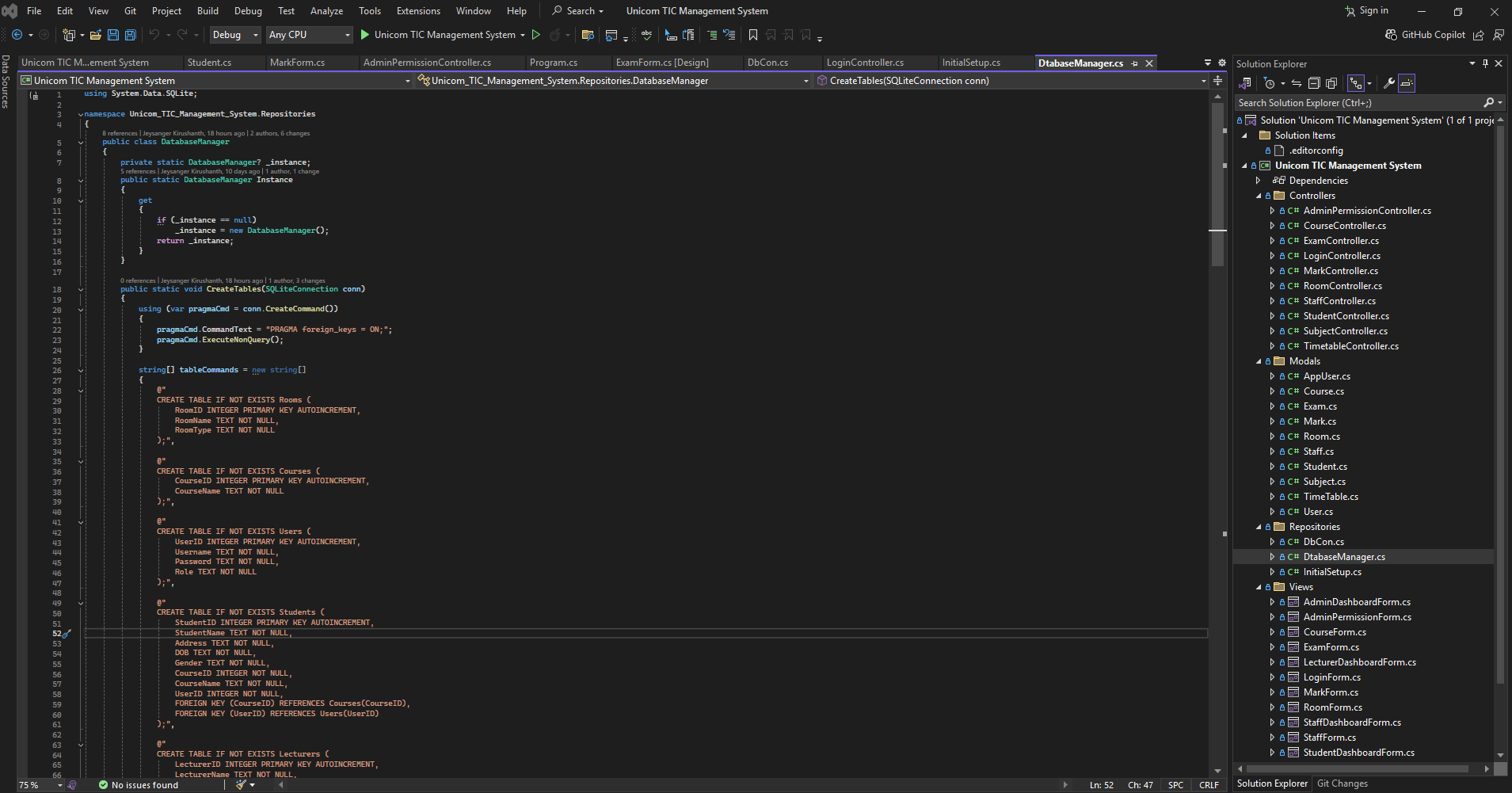
**🔹 Code Samples (ScreenShots):**

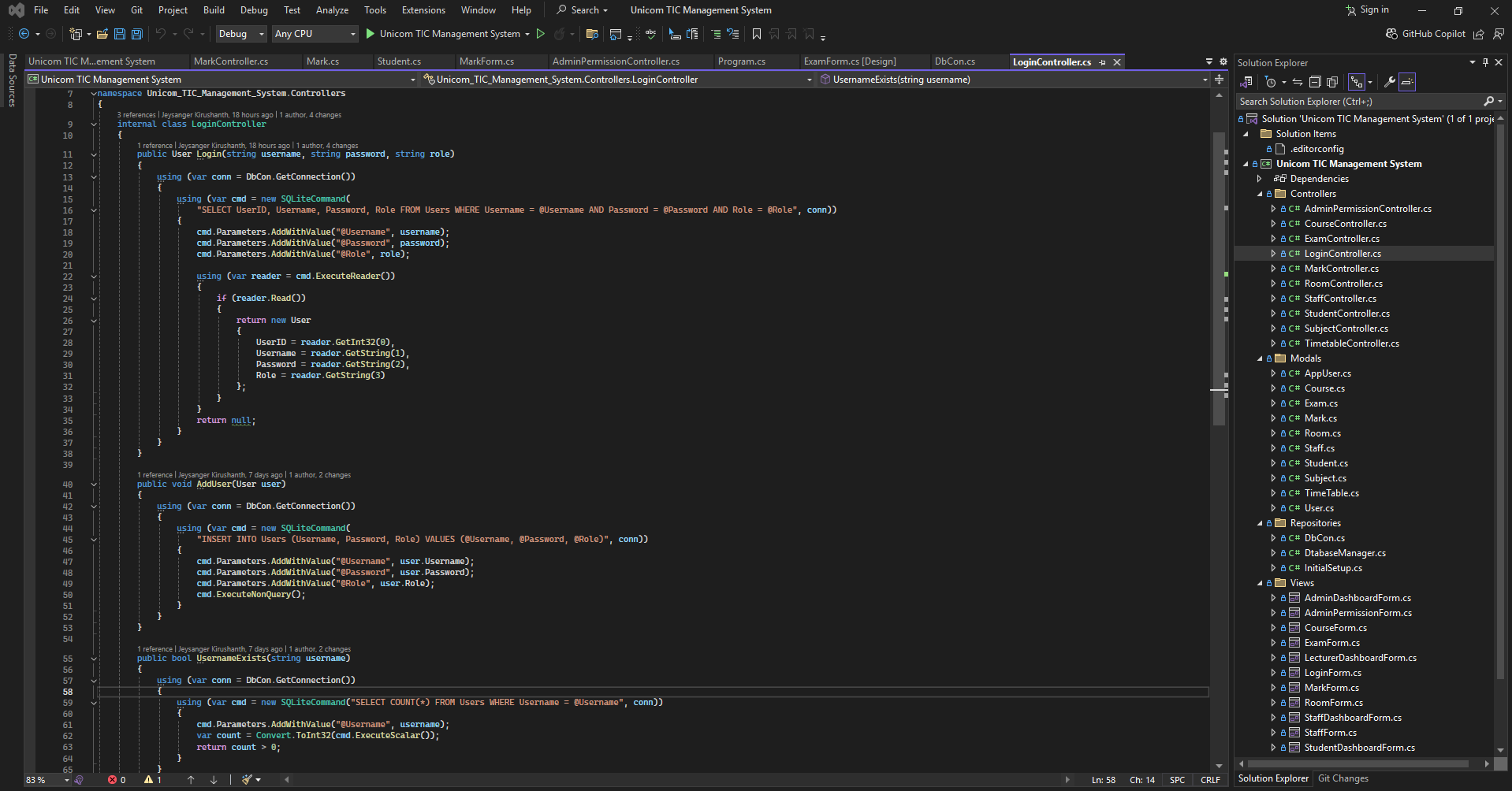
**AdminPermissionController**

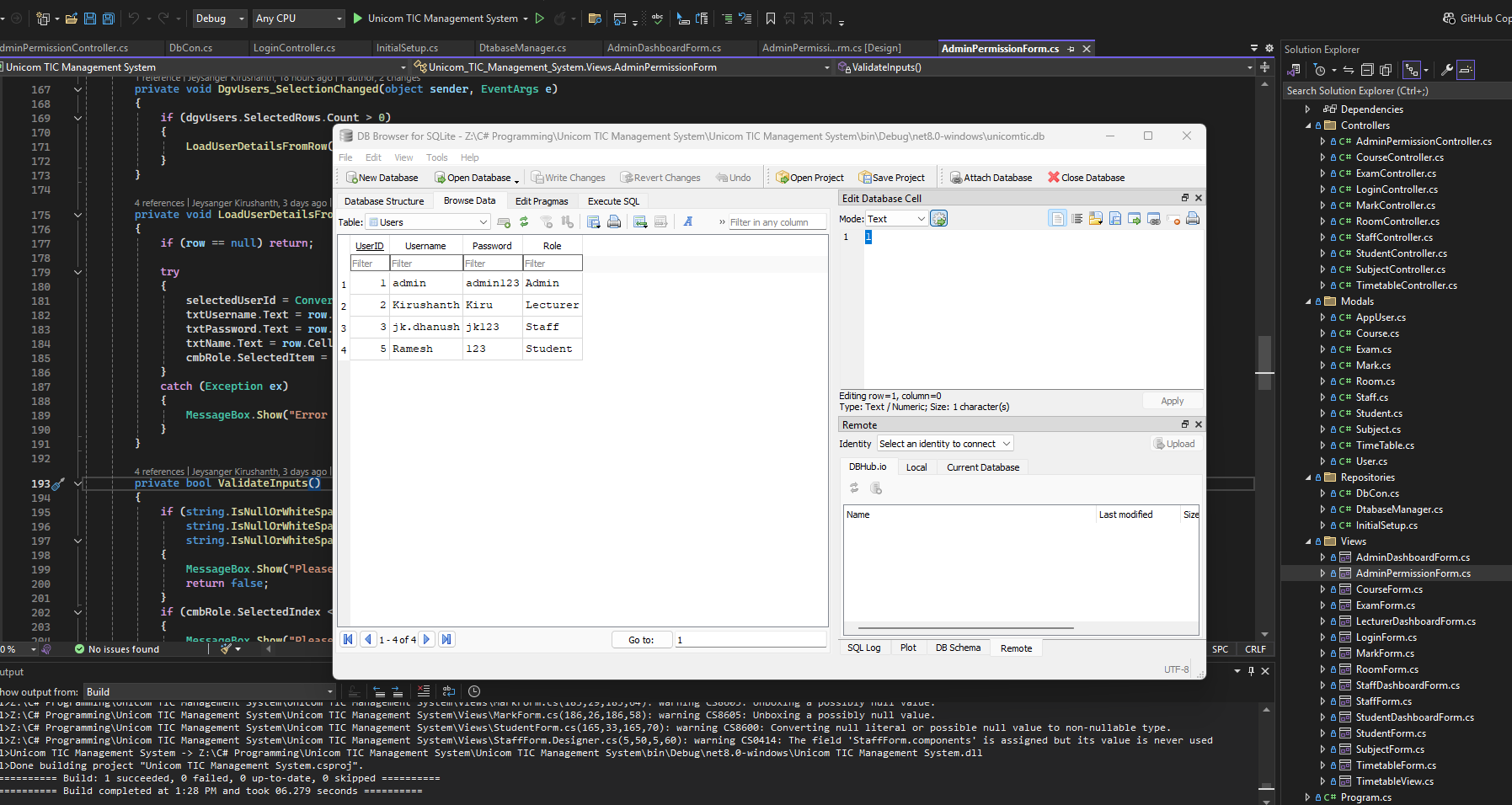
****

**AdminPermissionForm**

**AdminPasswordInitalSetup**

**Database manager**

**LoginController**

**SQLite table**