University of Greenwich ID Number: 001340129

FPT Student ID Number: GCD220151

Module Code: COMP1551 – Application Development

Module Assessment Title: Desktop Information System Development

Lecturer Name: Nguyen The Nghia

Submission Date: 12/4/2025

Table of Contents

1. System Description

The Desktop Information System is designed to help the education centre manage personal and role-specific information about Teachers, Administration Staff, and Students in an efficient and organised way. From the user's point of view, the system replaces manual paperwork and separate files with a single, easy-to-use desktop application. The Administration staff, who are the primary users, can quickly view all users in one place and keep their details up to date without searching through different documents.

The system allows the Admin to add new records, update existing information, or delete users when they leave the centre. Each record contains basic personal details such as name, telephone, and email. Depending on the user's role, additional information is included: Teachers have salary and teaching subjects, Admin Staff have salary, full-time or part-time employment status, and weekly working hours, while Students have up to three enrolled subjects. When the Admin selects a role while creating or editing a record, the system only shows the fields that are relevant to that role, which makes data entry clearer and reduces mistakes.

All information is stored in a local MySQL database. This keeps data in one central place so that it can be maintained consistently over time. The database can be protected and backed up using standard MySQL and system administration tools, and in practice it is intended to be accessed only by Administration staff within the education centre.

The application runs on Microsoft Windows and uses a simple Windows Forms graphical interface. Users interact with buttons, tables, and input forms rather than a command-line environment. The system requires only standard office hardware, such as a Windows 10 computer with basic processing power and a local MySQL server.

2. Software Requirements Specification (SRS)

2.1 Introduction

Purpose:
The purpose of the Desktop Information System is to help the education centre store and manage information about Teachers, Administration Staff, and Students. The system replaces manual paperwork with a structured digital solution that makes editing, viewing, and updating data easier and faster.

Project Scope:
The software allows Administration staff to add, edit, view, and delete user records. It supports role-specific data, such as salary, subjects taught, employment type, and working hours. All information is stored in a MySQL database and accessed through a Windows Forms desktop interface.

2.2 Overall Description

Product:
The system keeps personal and role-specific data for each user group. Teaching Staff records include salary and two subjects; Admin Staff records include salary, full-time or part-time status, and weekly working hours; Student records include three subjects. Core functions include adding new records, showing all data, filtering by user group, editing existing entries, and deleting records when needed. The system uses a simple Windows Forms interface to display data in a table and provide forms for input.

Users:
The primary user is the Administration staff member responsible for maintaining all user information. They have full access to add, edit, delete, and filter records. Teachers and Students do not interact with the system directly; they are represented only as data entries.

Operational Environment:
System runs on Microsoft Windows and operates as a standalone desktop application. It connects to a MySQL database hosted on localhost or the same local network. The environment is controlled, and access is restricted to authorised staff.

2.3 System Features

Description:
The User Management feature provides tools for creating, viewing, updating, and deleting user records. The interface adapts based on role selection so that only relevant fields appear. DataGridView displays all users with filtering options for each role.

Functional Requirements:
• The system shall allow the Admin to add new users with role-specific details.
• The system shall display all user records on a table with filtering by role.
• The system shall allow the Admin to edit selected user records.
• The system shall validate required fields before saving.
• The system shall delete records only after confirmation.
• The system shall store and retrieve data from a MySQL database

2.4 User Interface Requirements

The system uses a simple Windows Forms graphical interface. The layout provides buttons for navigation, clear forms for input, error messages for invalid data, and confirmation prompts to prevent accidental deletion.

2.5 Platform Requirements

The application runs on Windows 10 or later and requires the .NET Desktop Runtime. It also requires MySQL Server running on the local machine or network for data storage.

2.6 Quality Attributes

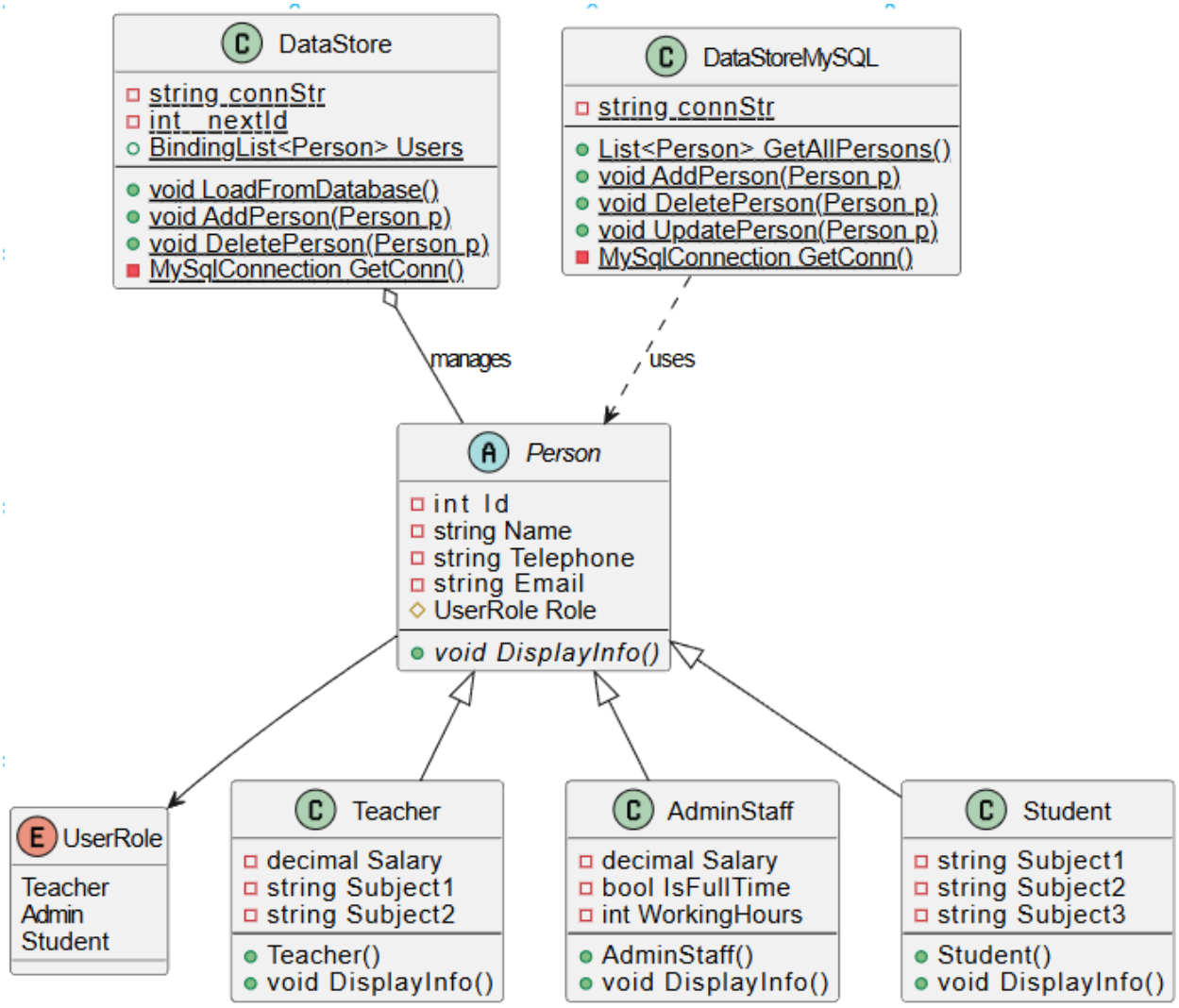
Performance:
The system should load screens and database queries quickly to support daily administrative tasks.

Security:
Access to the system and database must be controlled to ensure user data is not viewed or modified by unauthorised individuals.

Safety:
The system should prevent data loss, avoid crashes, and show clear warnings before performing sensitive actions such as deletion.

3. UML diagrams

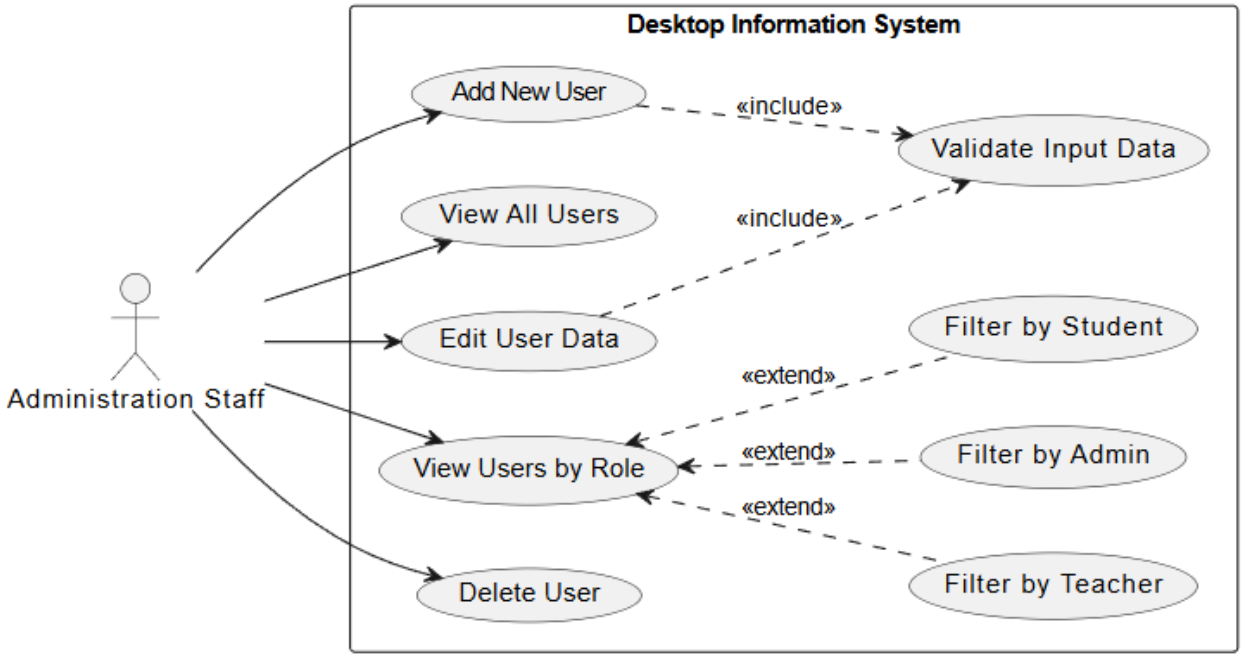
Class Diagram



The Class Diagram shows the object-oriented structure with an abstract Person base class containing common attributes (Id, Name, Telephone, Email, Role). Three derived classes Teacher, AdminStaff, and Student inherit from Person and add role-specific properties: Teachers have Salary and two subjects; AdminStaff have Salary, IsFullTime status, and WorkingHours; Students have three subjects.

The UserRole enumeration defines the three roles. DataStore and DataStoreMySQL classes manage database operations. This design demonstrates encapsulation, inheritance, and polymorphism principles.

Use Case Diagram



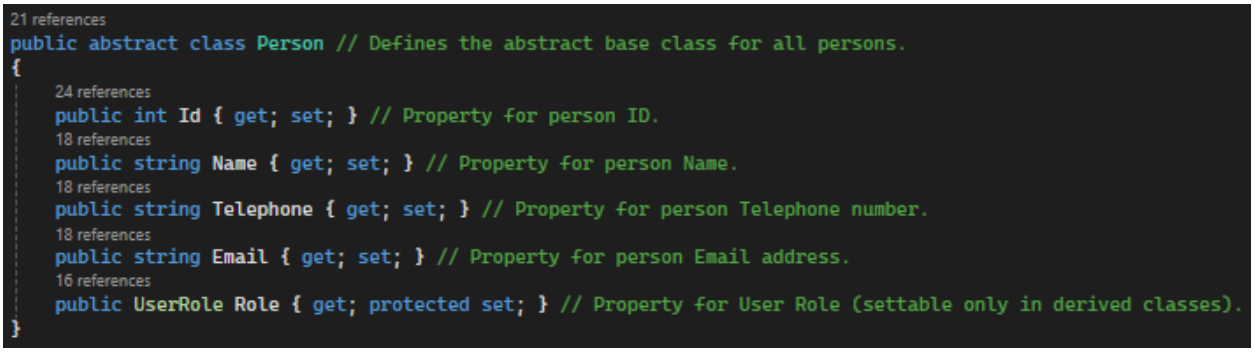
The Use Case Diagram illustrates system functionality from the Administration Staff perspective. The main use cases include Add New User, View All Users, View Users by Role, Edit User Data, and Delete User.

The Validate Input Data use case is included in both Add and Edit operations to ensure data integrity. View Users by Role extends into three filtering options: Filter by Teacher, Admin, and Student.

4. OOP Feature

1.1 Abstraction

The purpose of abstraction in the Person base class is to define a clear and simple model of a system user (Id, Name, Telephone, Email, Role) while hiding the concrete details of each role. This allows the rest of the system to work with a generic “person” concept without depending on how Teacher, AdminStaff, or Student are implemented internally.

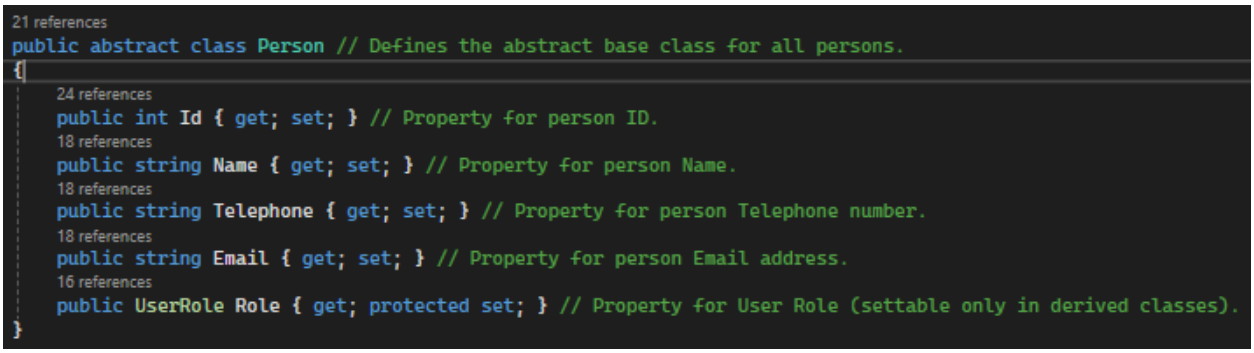


Program.cs

1.2 Encapsulation

1.2.1 Encapsulated state via properties

The purpose of encapsulating state through properties is to control how object data is read and modified. By using C# properties for fields such as Name, Email, or Role the system can enforce validation rules, keep data consistent, and change the internal implementation later without affecting other classes.



Program.cs

1.2.2 Hiding internal details in data store classes

The purpose of hiding internal details in data store classes is to protect sensitive configuration and keep all database logic in one place. Private fields and helper methods ensure that only the data access layer can manage database connections, while the rest of the application uses a clean, high-level API.

```
11 references
public static class DataStore // Defines static class for database operations and data storage.
{
    private static string connStr = // MySQL connection string initialization.
        "Server=127.0.0.1;Database=educationdb;User=root;Password=;";

    private static int _nextId = 1; // Private field to track the next available ID.

    9 references
    public static BindingList<Person> Users { get; } = // Public static BindingList for UI data binding.
        new BindingList<Person>(); // Initializes the BindingList.

    3 references
    private static MySqlConnection GetConn() // Helper method to create a new MySQL connection object.
    {
        return new MySqlConnection(connStr); // Returns the new connection.
    }
```

Program.cs

1.2.3 Encapsulated UI state in forms

The purpose of encapsulating UI state inside forms is to keep the behaviour of each window consistent and self-contained. Private fields prevent external classes from changing the form's internal state directly, so UI logic is controlled only through well-defined event handlers and methods.

```
public static class DataStoreMySQL
{
    // Private static field for the database connection string.
    private static string connStr =
        "Server=localhost;Database=educationdb;Uid=root;Pwd=;";

    // Helper method to create a new MySqlConnection object.
    4 references
    private static MySqlConnection GetConn()
    {
        return new MySqlConnection(connStr);
    }
```

DataStoreMySQL.cs

1.3 Inheritance

1.3.1 Domain inheritance

The purpose of domain inheritance is to reuse common attributes for all user types and model real-world relationships. Teacher, AdminStaff, and Student inherit from Person, so they automatically share Id, Name, Telephone, Email, and Role, while each subclass adds its own specific fields. This reduces duplication and makes the design closer to how people are organised in the education centre.

```
public class Teacher : Person // Defines Teacher class, inheriting from Person.
{
    12 references
    public decimal Salary { get; set; } // Teacher-specific property: Salary.
    12 references
    public string Subject1 { get; set; } // Teacher-specific property: Subject 1.
    12 references
    public string Subject2 { get; set; } // Teacher-specific property: Subject 2.

    4 references
    public Teacher() // Constructor for Teacher.
    {
        Role = UserRole.Teacher; // Sets the role to Teacher.
    }
}

15 references
public class AdminStaff : Person // Defines AdminStaff class, inheriting from Person.
{
    12 references
    public decimal Salary { get; set; } // AdminStaff-specific property: Salary.
    14 references
    public bool IsFullTime { get; set; } // AdminStaff-specific property: Full-time status.
    12 references
    public int WorkingHours { get; set; } // AdminStaff-specific property: Working Hours.

    4 references
    public AdminStaff() // Constructor for AdminStaff.
    {
        Role = UserRole.Admin; // Sets the role to Admin.
    }
}
```

```
public class Student : Person // Defines Student class, inheriting from Person.
{
    12 references
    public string Subject1 { get; set; } // Student-specific property: Subject 1.
    12 references
    public string Subject2 { get; set; } // Student-specific property: Subject 2.
    12 references
    public string Subject3 { get; set; } // Student-specific property: Subject 3.

    4 references
    public Student() // Constructor for Student.
    {
        Role = UserRole.Student; // Sets the role to Student.
    }
}
```

Program.cs

1.3.2 UI inheritance: forms inherit from

The purpose of UI inheritance is to take advantage of the built-in behaviour provided by the Windows Forms framework. By inheriting from Form, classes such as Main, UserForm, and UserListForm automatically gain standard window features and only need to implement the extra controls and logic required for this application.

```
public partial class Main : Form // Defines the 'Main' class, which is a 'partial' class inheriting from 'Form' (making it a window/form)
{
    // Constructor for Main form
    1 reference
    public Main() // The constructor method, which is called when an instance of the Main form is created
    {
        InitializeComponent();  // Method generated by the Visual Studio designer to set up the form's controls and layout
    }
```

Main.cs

```
public partial class UserForm : Form // Defines the 'UserForm' class, a 'partial' class inheriting from 'Form'
{
    private readonly Person _editingPerson; // Private field to hold a reference to the Person object being edited (null in Add mode)
    private readonly bool _isEdit;          // Private field indicating if the form is in Edit mode (true) or Add mode (false)

    // Flag to prevent double-saving (event being triggered multiple times, e.g., double-clicking)
    private bool _isSaving = false; // Flag to prevent the save logic from executing multiple times concurrently

    // Form constructor for ADD new user
    2 references
    public UserForm() // The default constructor used when adding a new user
    {
        InitializeComponent();  // Initializes the form's controls and layout (UI elements)

        // Populate the role ComboBox with enum values (Teacher, Admin, Student)
        cmbRole.DataSource = Enum.GetValues(typeof(UserRole)); // Fills the ComboBox with all values from the UserRole enumeration

        // Event handlers for Full-time/Part-time checkboxes
        chkFullTime.CheckedChanged += chkFullTime_CheckedChanged; // Attaches the custom handler to the Full-time checkbox change event
        chkPartTime.CheckedChanged += chkPartTime_CheckedChanged; // Attaches the custom handler to the Part-time checkbox change event

        // Update the UI according to the selected role
        UpdateRoleFields(); // Calls the method to show/hide fields based on the initial role selection (usually the first enum value)
    }
```

AddUserform.cs

```
public partial class UserListForm : Form // Declares UserListForm class that inherits from Form
{
    private Person _editingPerson;   // Private field to store the currently selected or edited Person object

    1 reference
    public UserListForm() // Constructor method that initializes the UserListForm
    {
        InitializeComponent(); // Initializes all form controls defined in the designer

        // Configure the DataGridView settings for displaying user data
        gridUsers.ReadOnly = true; // Sets the grid to read-only mode so users cannot edit cells directly
        gridUsers.AllowUserToAddRows = false; // Prevents the empty new row from appearing at the bottom
        gridUsers.AllowUserToDeleteRows = false; // Disables row deletion through the grid interface
        gridUsers.SelectionMode = DataGridViewSelectionMode.FullRowSelect; // Makes it so clicking any cell selects the entire row
        gridUsers.RowHeadersVisible = false; // Hides the leftmost gray column with row numbers

        // Configure ComboBox for filtering users by role
        if (cmbRoleFilter != null) // Checks if the ComboBox control exists before configuring it
        {
            cmbRoleFilter.DropDownStyle = ComboBoxStyle.DropDownList; // Makes the ComboBox non-editable by user typing
            cmbRoleFilter.Items.Clear(); // Removes any items that might have been added in the designer
            cmbRoleFilter.Items.Add("All"); // Adds option to show all users regardless of role
            cmbRoleFilter.Items.Add("Teacher"); // Adds Teacher role filter option
            cmbRoleFilter.Items.Add("Admin"); // Adds Admin role filter option
            cmbRoleFilter.Items.Add("Student"); // Adds Student role filter option
            cmbRoleFilter.SelectedIndex = 0; // Sets initial selection to the first item which is All
            cmbRoleFilter.SelectedIndexChanged += cmbRoleFilter_SelectedIndexChanged; // Subscribes to the selection change event to trigger filtering
        }

        // Initially hide the edit panel when the form first loads
        if (panelEdit != null) // Checks if the edit panel control exists
            panelEdit.Visible = false; // Sets the panel visibility to false to hide it

        // Assign event handlers to button controls
        if (btnEditSelected != null) // Checks if the Edit button exists
            btnEditSelected.Click += btnEditSelected_Click; // Subscribes the click event to the edit handler method

        if (btnDeleteSelected != null) // Checks if the Delete button exists
            btnDeleteSelected.Click += btnDeleteSelected_Click; // Subscribes the click event to the delete handler method

        if (btnSaveEdit != null) // Checks if the Save button in the edit panel exists
            btnSaveEdit.Click += btnSaveEdit_Click; // Subscribes the click event to the save changes handler method
```

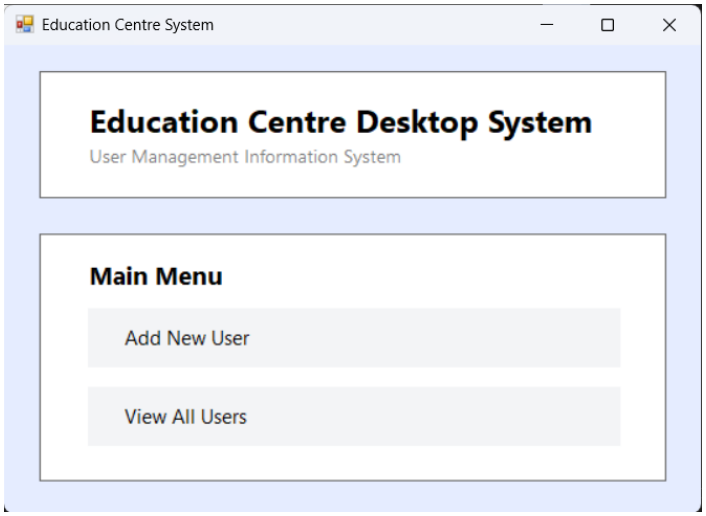
```
    // Assign event handlers for Full-time and Part-time checkboxes to ensure mutual exclusion
    if (chkFullTimeEdit != null) // Checks if the Full-time checkbox exists
        chkFullTimeEdit.CheckedChanged += chkFullTimeEdit_CheckedChanged; // Subscribes to handle when Full-time is checked or unchecked
    if (chkPartTimeEdit != null) // Checks if the Part-time checkbox exists
        chkPartTimeEdit.CheckedChanged += chkPartTimeEdit_CheckedChanged; // Subscribes to handle when Part-time is checked or unchecked

    // NEW: Handle row selection in DataGridView to automatically show View Mode panel
    gridUsers.SelectionChanged += gridUsers_SelectionChanged; // Subscribes to the event that fires when user selects a different row

    this.Load += UserListForm_Load; // Subscribes to the form Load event to fetch and display data when form opens
}
```

UserListForm.cs

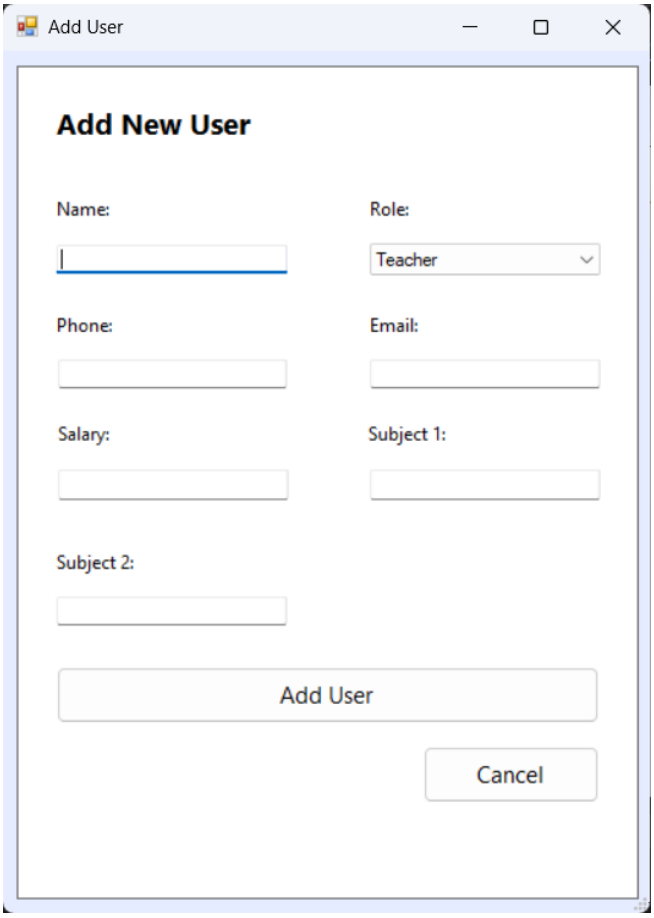
5. Inovation

GUI

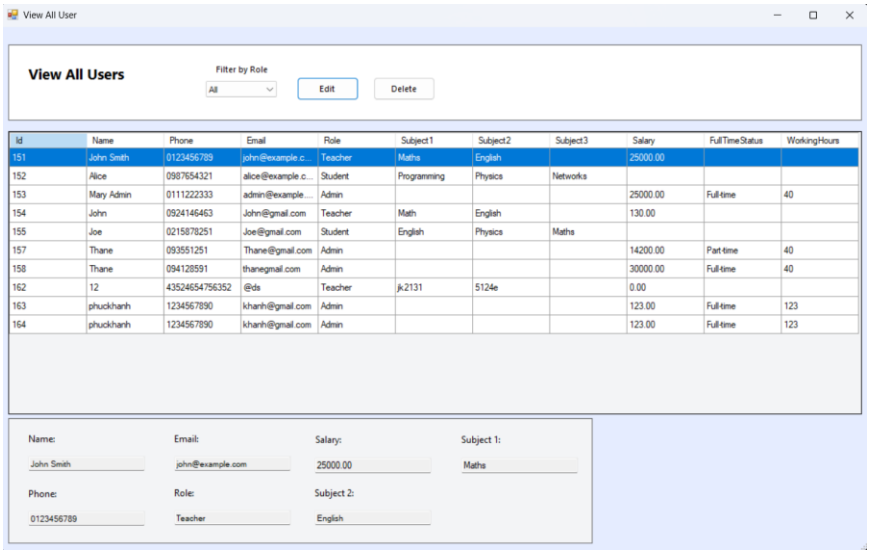
The Graphical User Interface (GUI) is designed for simplicity and user-friendliness, offering a clean layout with role-specific forms. Each user role (Teacher, Admin, Student) is presented with relevant fields such as salary and subjects for teachers and just subjects for students making the interface intuitive and easy to navigate.



Main.cs



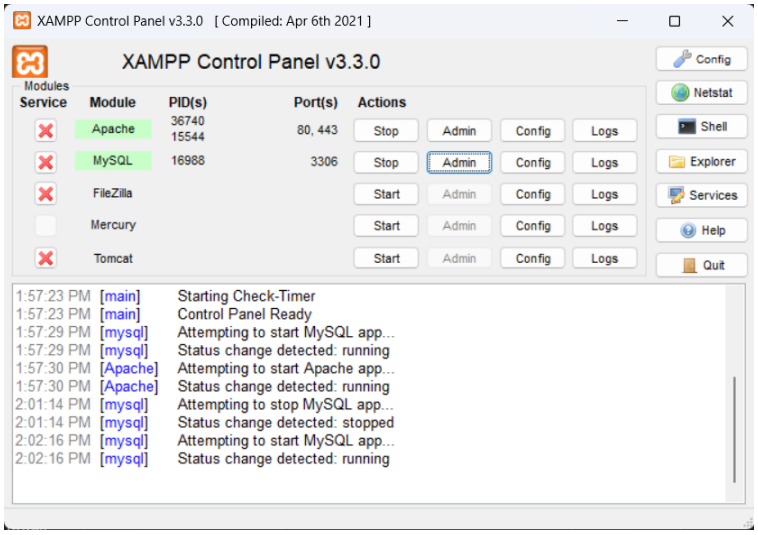
AddUserForm.cs



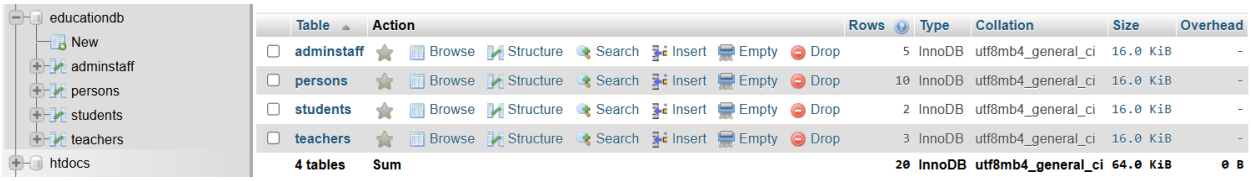
UserListForm.cs

DataBase

The database system is built using object-oriented principles, where the Person class serves as the base class, and the Teacher, Admin, and Student classes inherit from it. This structure enables efficient management of different user types. Data is stored securely in a MySQL database, ensuring fast retrieval and updates.



Xampp



MySQL

6. TestCase

No.	Description	Expected Output	Input	Actual Output	Evidence	Test Result
1	Start application when MySQL is running normally	Main form opens without error, data from DB (if any) is loaded into DataStore.Users	MySQL server ON, database and tables exist, run program	Application started successfully and loaded users into DataStore.Users without any error message.	Figure 1	Pass
2	Start application when MySQL is	MessageBox shows “Cannot load data from	Stop MySQL service or use wrong password	Application showed a connection error message;	Figure2	Pass

	stopped / wrong config	MySQL: …", Main still opens but user list is empty	in connection string, run program	Main form opened, and no users were loaded.		
3	Auto-seed 3 sample users when DB is empty	On first run with empty DB, DataStore.Users contains 3 users (Teacher, Student, Admin) and they are saved to DB	Clear all records from tables, run program	Application created and saved 3 default users, and they appeared in the user list.	Figure3	Pass
4	Open "View All Users" form	UserListForm opens and DataGridView shows all users currently in DataStore.Users	In Main form, click "View All Users" button	UserListForm opened and the grid displayed all users from the data store.	Figure4	Pass
5	Add valid Teacher	New Teacher is saved to DB (persons + teachers), appears in DataStore.Users, and success message is shown	In Main, click "Add User" → Role = Teacher, enter valid Name, Phone, Email, Salary, Subject1/2, click Save	Teacher user was created, stored in DB, and shown in the list with a success message.	Figure5	Pass
6	Add valid Student	New Student is saved (persons + students), appears in list	"Add User" → Role = Student → enter Name, Phone, Email, Subject1–3, Save	Student user was added to DB and appeared correctly in the user list.	Figure6	Pass
7	Add valid full-time AdminStaff	Admin is saved (persons + adminstaff) with IsFullTime = true and correct WorkingHours	"Add User" → Role = Admin → enter Name, Phone, Email, Salary, tick Full-time, WorkingHours = 40, Save	AdminStaff was created with Full-time status and WorkingHours = 40, visible in grid.	Figure7	Pass

8	Missing Name when saving	MessageBox “Please enter Name, Phone and Email.” is shown and user is not saved	Add User, leave Name empty, fill Phone and Email, click Save	Warning message appeared and the user was not created.	Figure8	Pass
9	Missing Phone when saving	Same warning; user not saved	Add User, fill Name and Email, leave Phone empty, Save	Warning was shown; user was not created.	Figure9	Pass
10	Email without “@” and user chooses No	Warning about invalid email is shown; clicking “No” cancels save	Add User, Email = "abc.gmail.com", Save → in confirmation, choose No	Application kept the form open and did not create a new user.	Figure10	Pass
11	Teacher: empty Salary	decimal.TryParse fails, Salary defaults to 0, user is still saved without crash	Add Teacher, leave Salary blank, fill other required fields, Save	Teacher user was created with Salary = 0 and no runtime error occurred.	Figure11	Pass
12	Admin: neither Full-time nor Part-time selected	Warning “Please choose Full-time or Part-time.” is shown and user is not saved	Add Admin, leave both checkboxes unchecked, Save	Application displayed the warning and prevented saving the Admin user.	Figure12	Pass
13	Cancel button on Add User	Form closes without saving and data store is unchanged	Open Add User, enter some text, click Cancel	Add User form closed and no new user was added.	Figure13	Pass
14	Filter by Teacher	Only Teacher users are displayed	UserListForm → role filter = Teacher	Grid showed only rows with Role = Teacher.	Figure14	Pass
15	Filter by Admin	Only AdminStaff users are displayed	UserListForm → role filter = Admin	Grid showed only Admin staff entries.	Figure15	Pass
16	Filter by Student	Only Student users are displayed	UserListForm → role filter = Student	Grid showed only Student users.	Figure16	Pass

17	Selecting a row populates view panel	Edit panel becomes visible in view mode, showing full details for the selected user	In UserListForm, click a row in the grid	Panel displayed correct user data and was set to read-only (view mode).	Figure17	Pass
18	Edit Teacher: change Salary and Subjects	After saving, Teacher's Salary and Subjects are updated in DB and grid, with success message	In UserListForm, select a Teacher → Edit → modify Salary and Subject1/2 → Save	Teacher's details were updated in grid and persisted to database.	Figure18	Pass
19	Edit Student: change Subject3	After saving, Subject3 is updated and shown correctly	Select a Student → Edit → update Subject3 → Save	Student's Subject3 displayed the new value and was stored in DB.	Figure19	Pass
20	Edit Admin: change Full-time to Part-time	FullTimeStatus changes to "Part-time" and WorkingHours is updated	Select an Admin → Edit → select Part-time, update WorkingHours → Save	Grid showed "Part-time" and new WorkingHours; database contained updated values.	Figure20	Pass
21	Edit Admin: no Full-time/Part-time selected	Warning is shown and update is not applied	Edit Admin, uncheck both Full-time and Part-time, click Save	Application displayed "Please choose Full-time or Part-time." and did not save changes.	Figure21	Pass
22	Delete user with confirmation Yes	User is removed from DB and grid, success message shown	Select a user in UserListForm → click Delete Selected → choose Yes	User disappeared from the grid and was removed from database; success message displayed.	Figure22	Pass
24	Close application from Main form	Application exits cleanly without runtime errors	On Main form, click window Close (X) button	Program closed successfully and no error messages were shown.	none	Pass

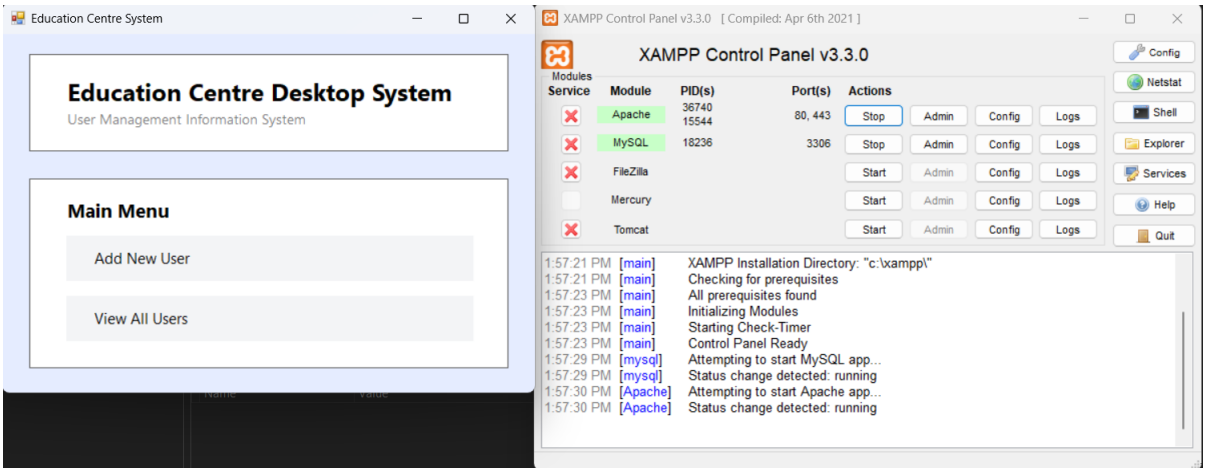


Figure 1

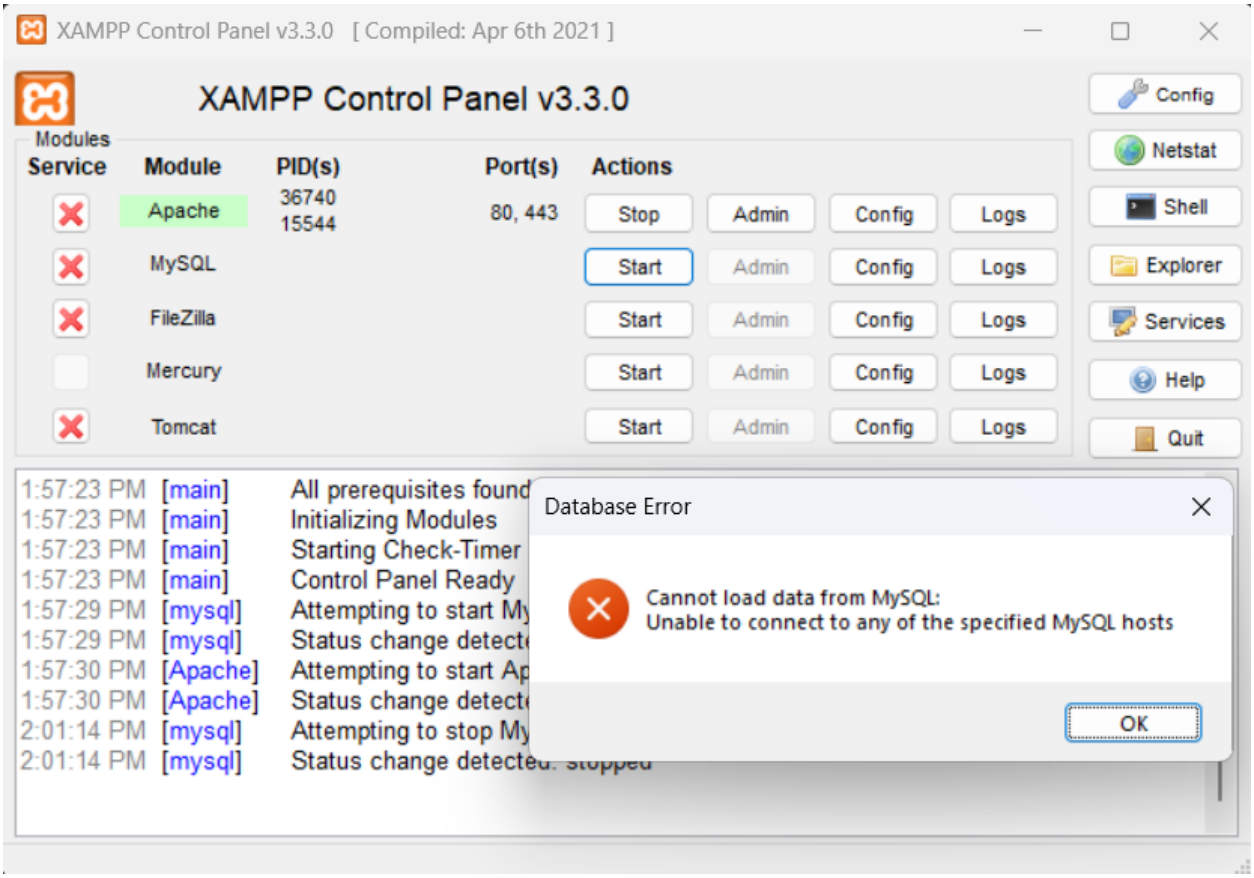


Figure 2

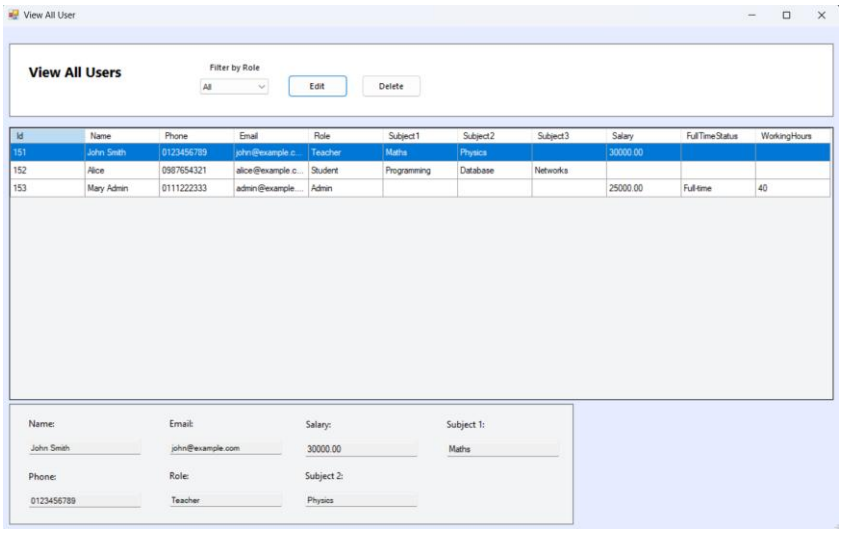
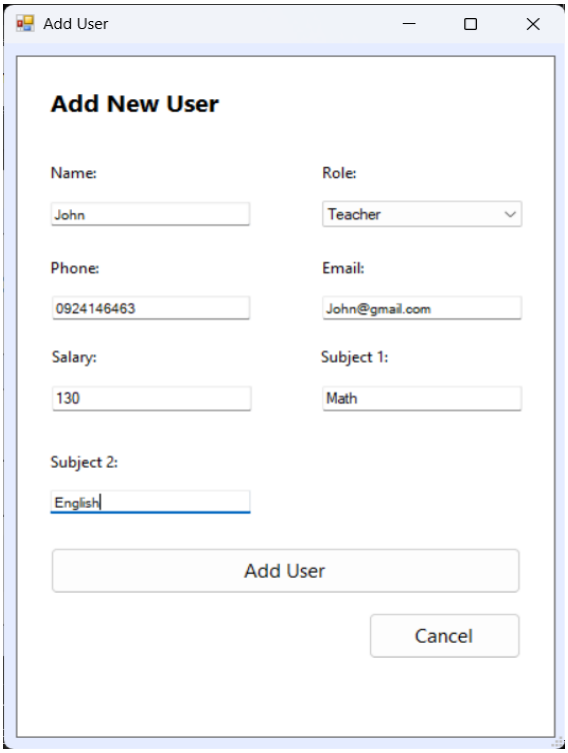


Figure 3,4



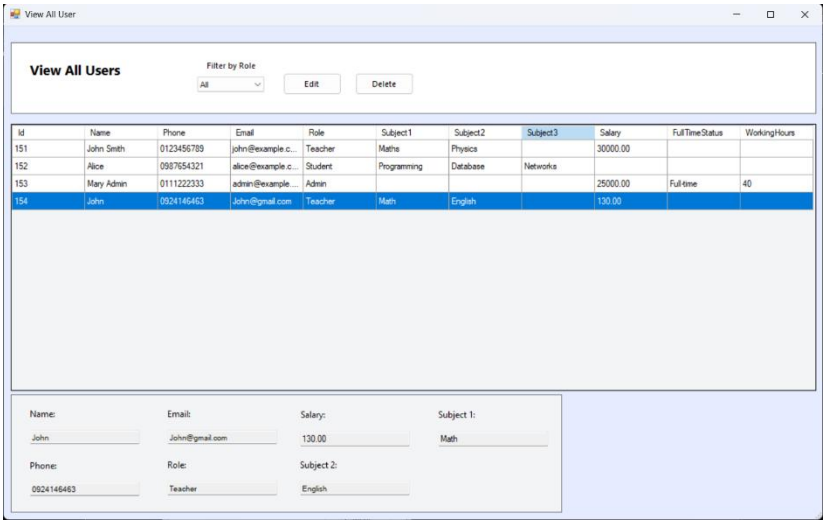
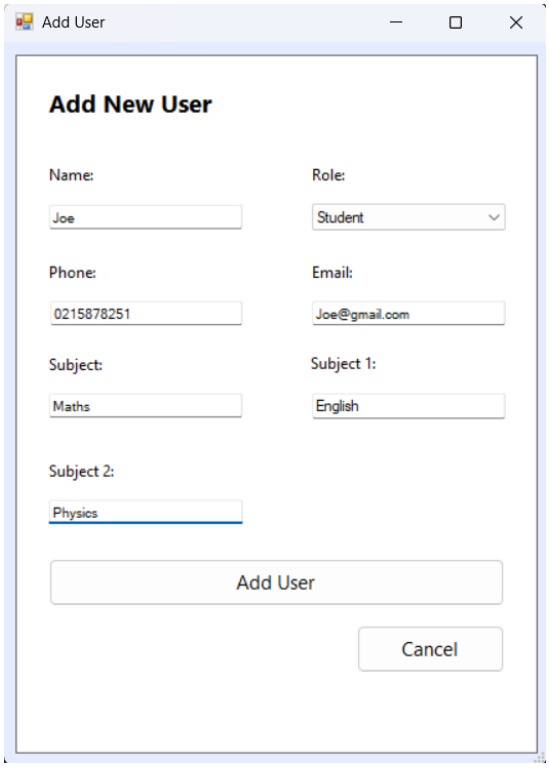


Figure 5



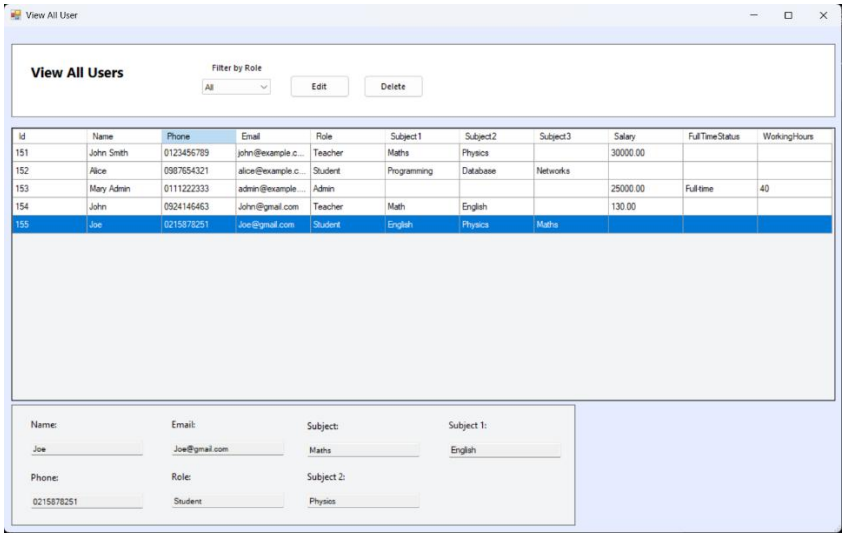
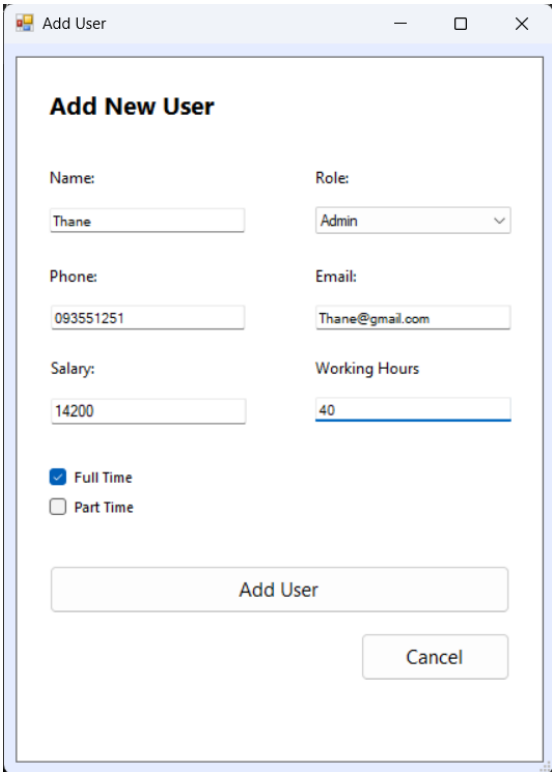


Figure 6



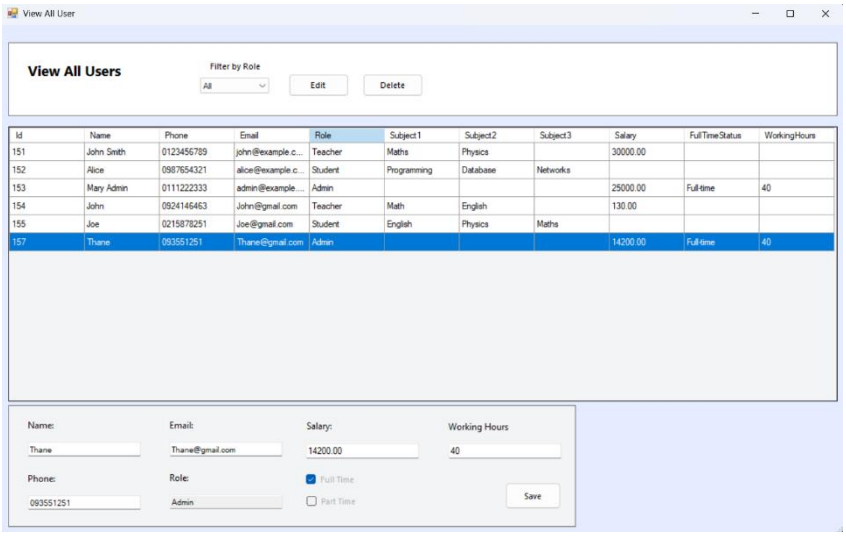


Figure 7

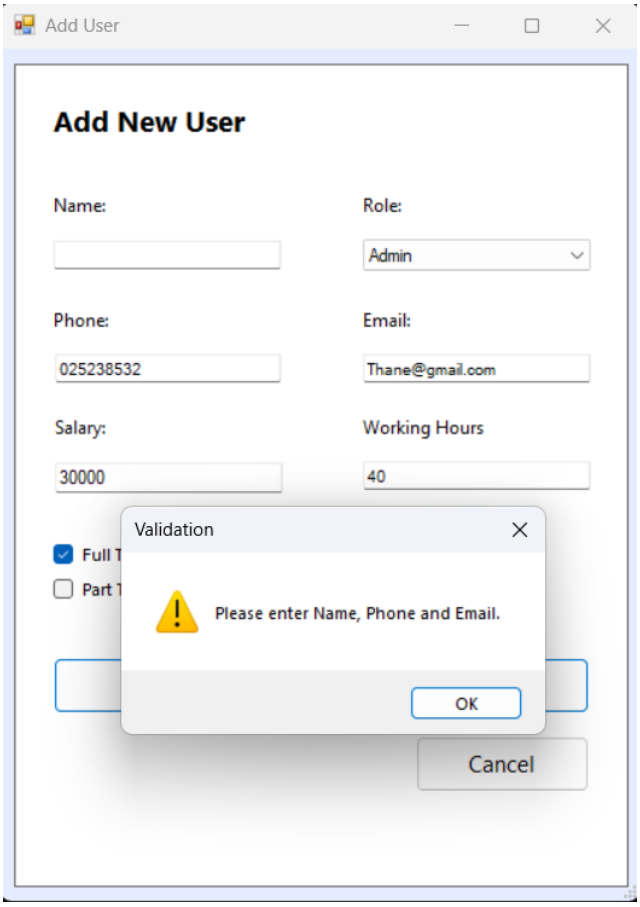

Add User
Add New User
Name:
Role:
Admin
Phone:
Email:
025238532
Thane@gmail.com
Salary:
Working Hours
30000
40
Validation
Please enter Name, Phone and Email.
OK
Cancel

Figure 8,9

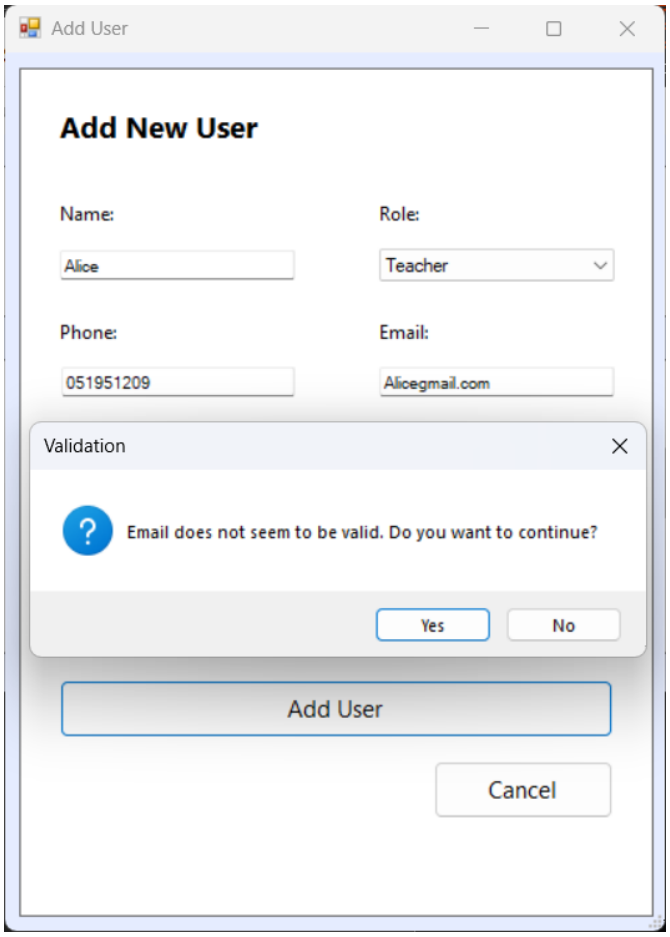


Figure 10

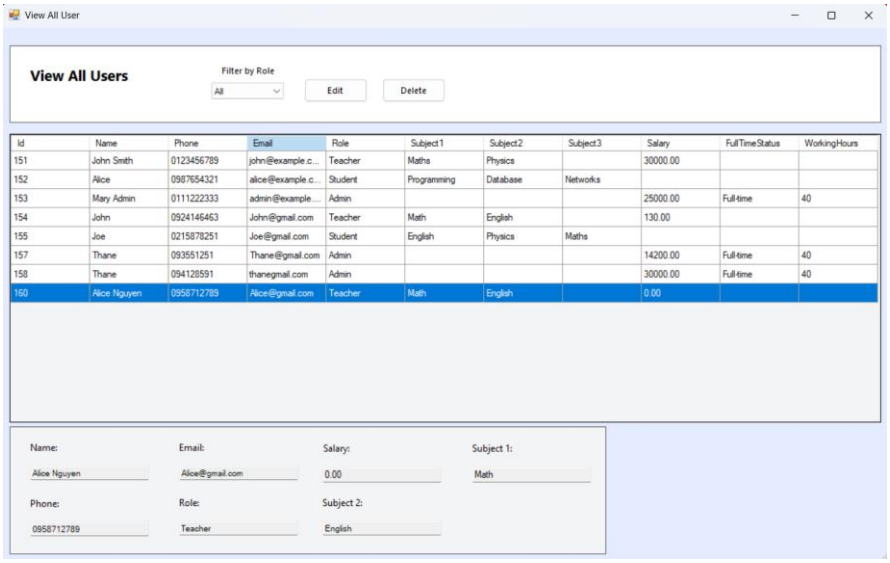


Figure 11

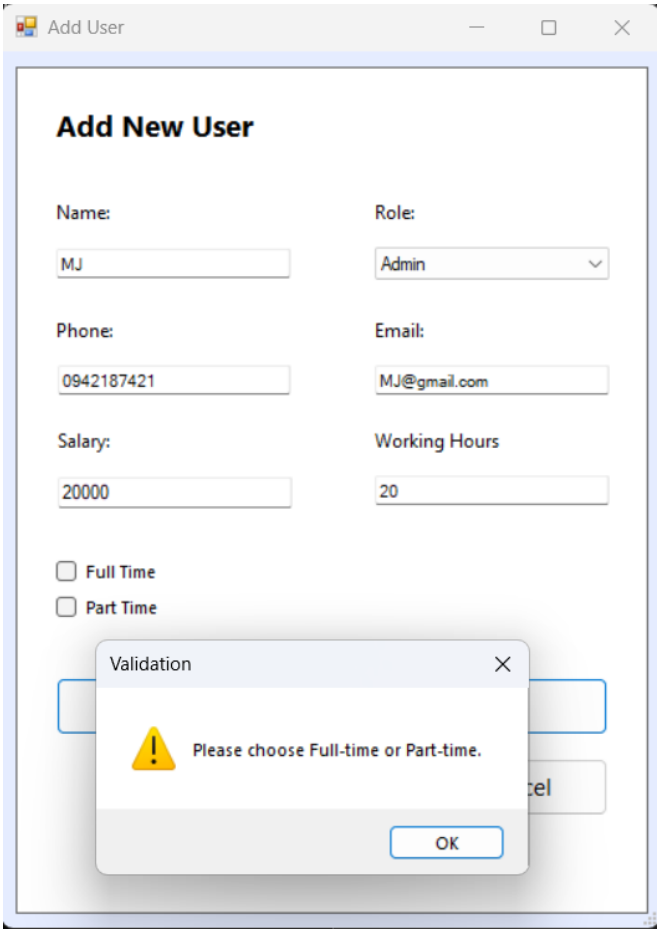

Add User
Add New User
Name:
MJ
Role:
Admin
Phone:
0942187421
Email:
MJ@gmail.com
Salary:
20000
Working Hours
20
Full Time
Part Time
Validation
Please choose Full-time or Part-time.
OK

Figure 12

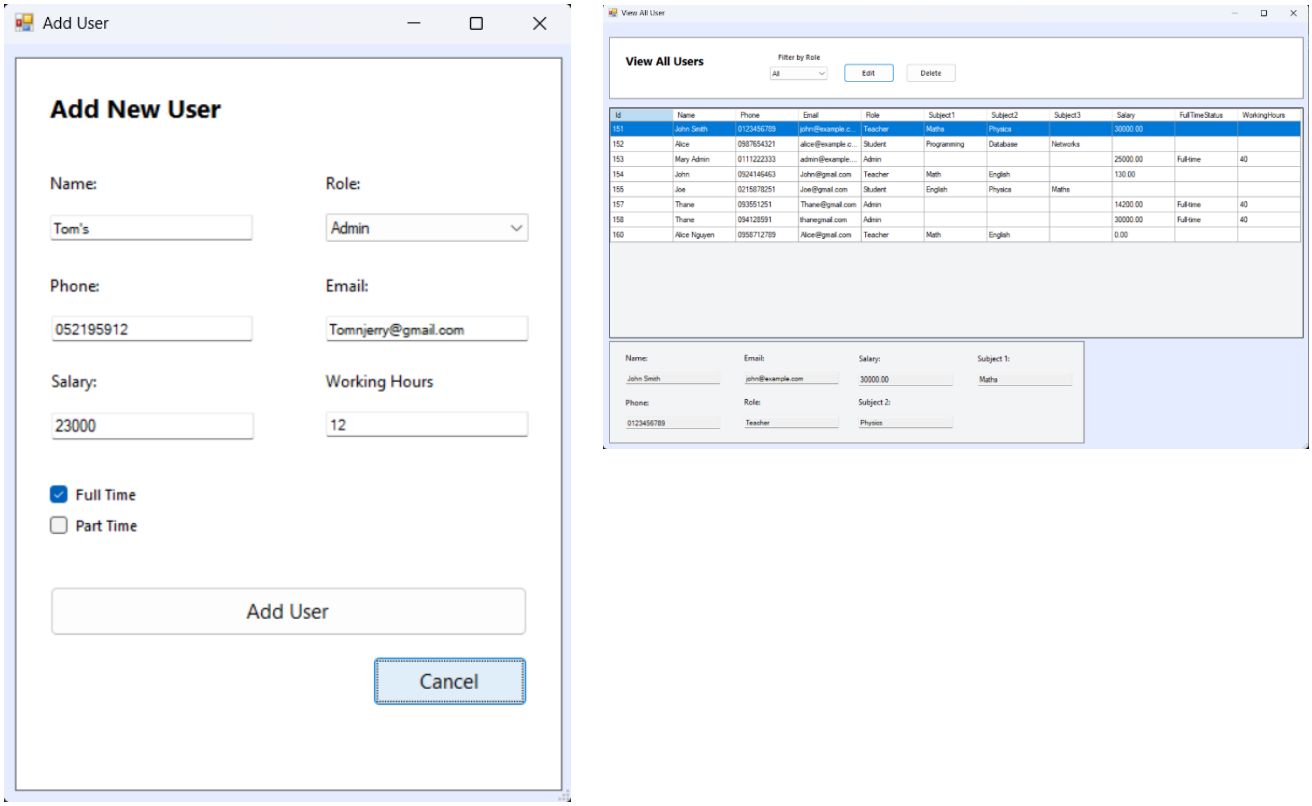


Figure 13

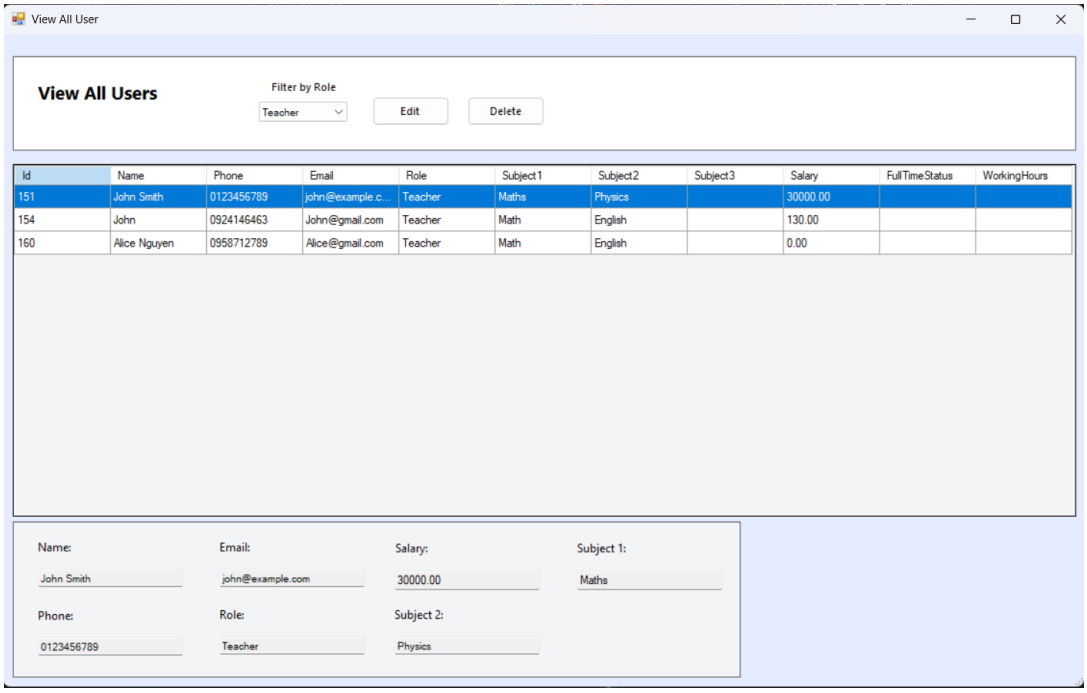


Figure 14

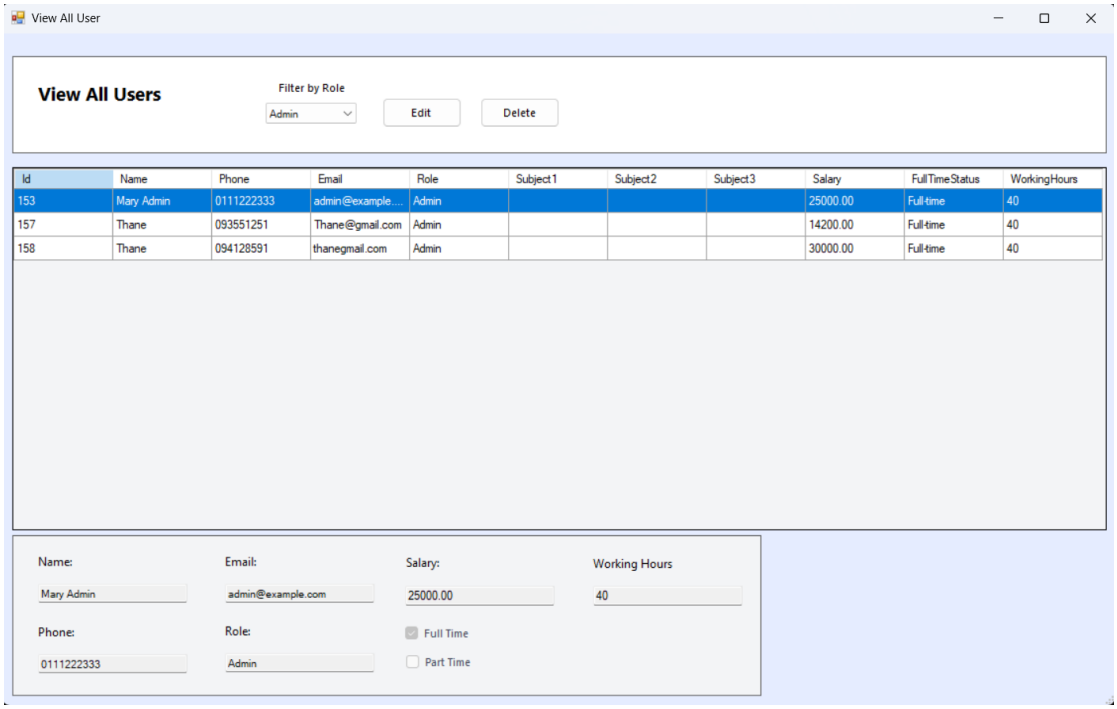


Figure 15

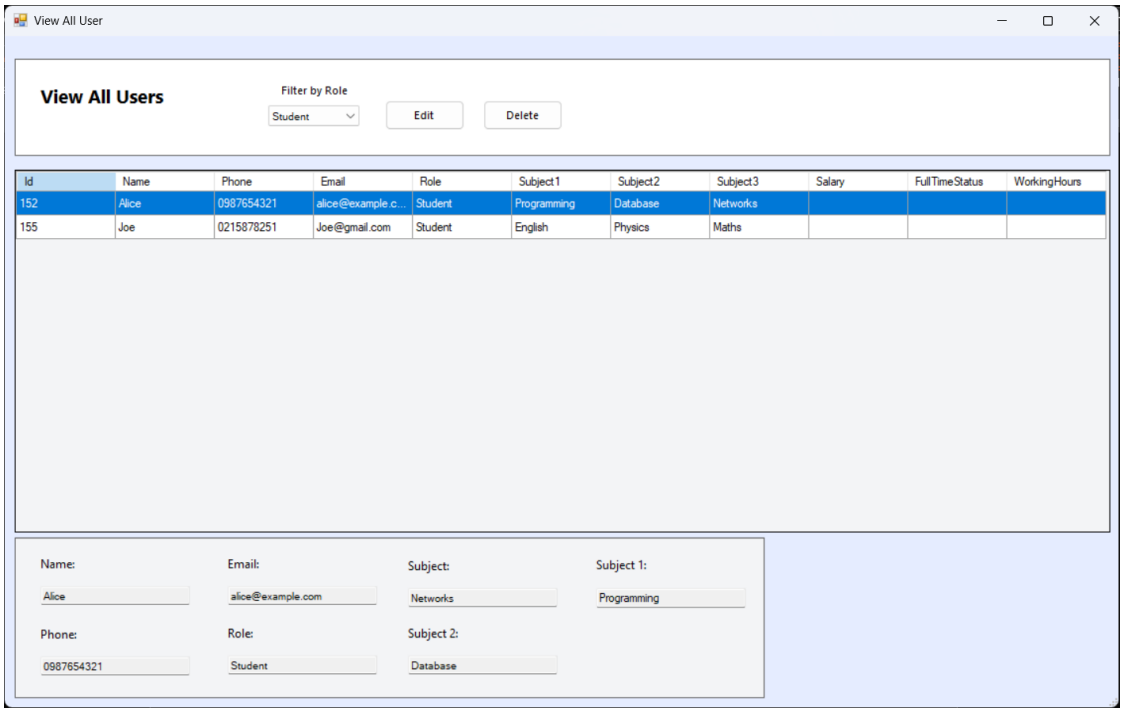


Figure 16

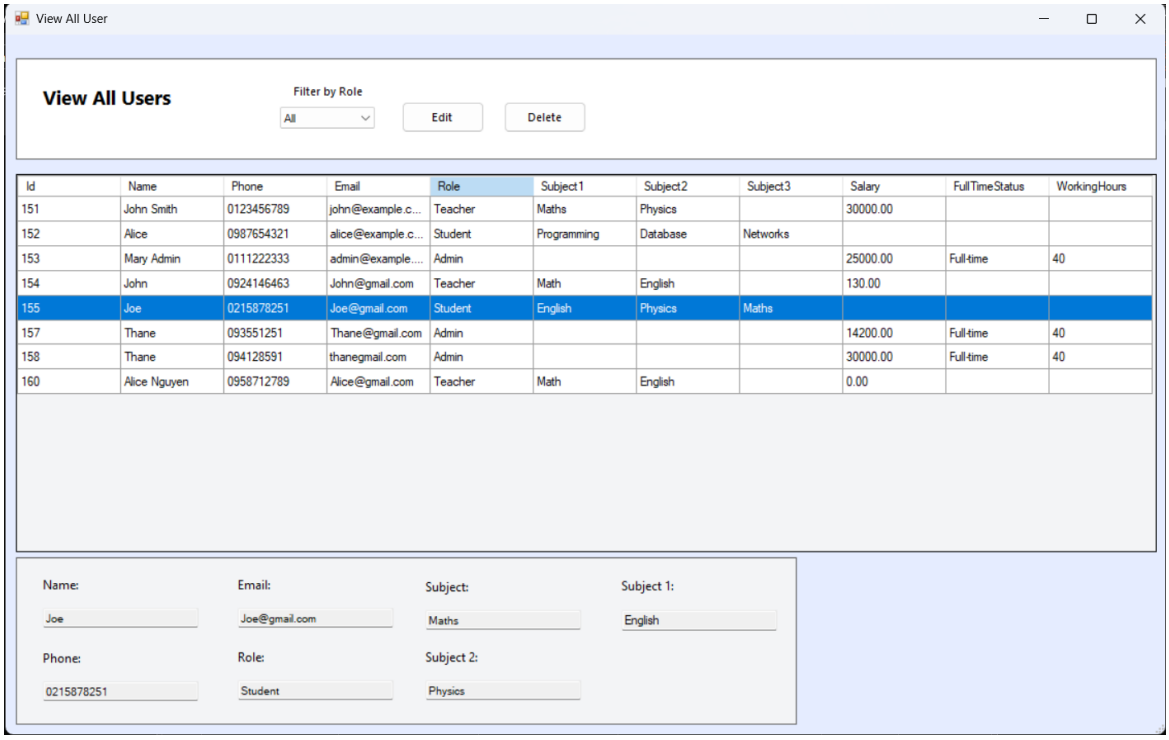


Figure 17

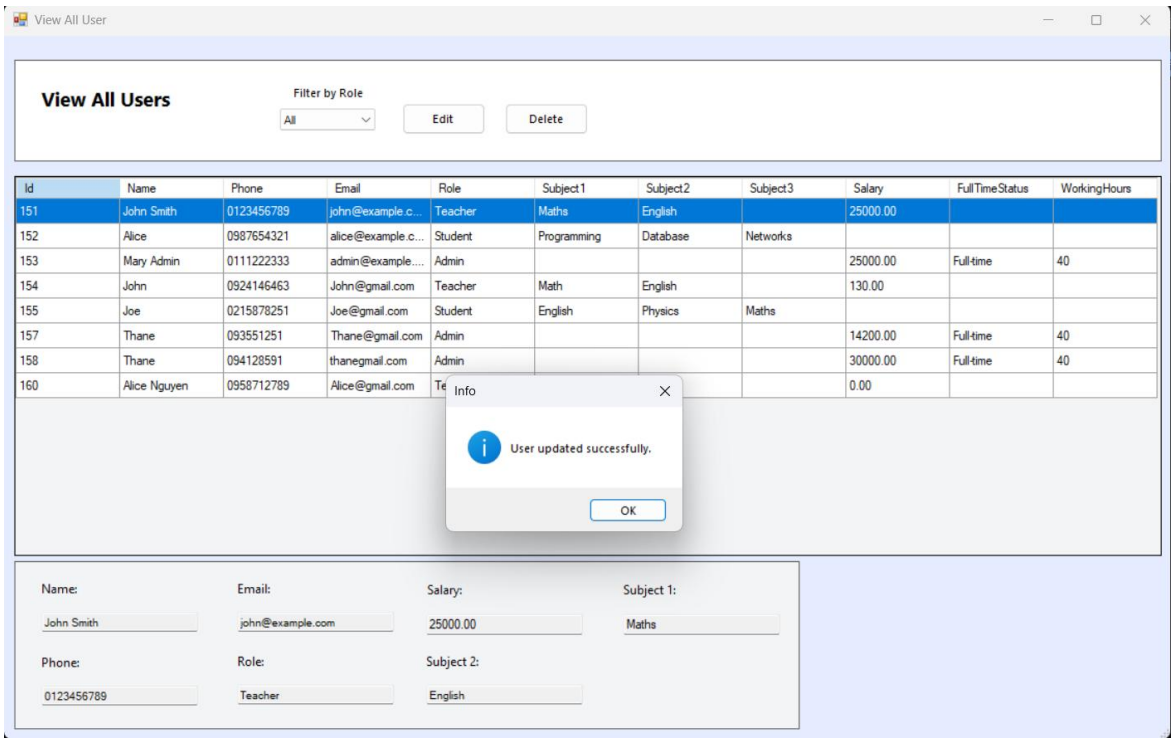


Figure 18

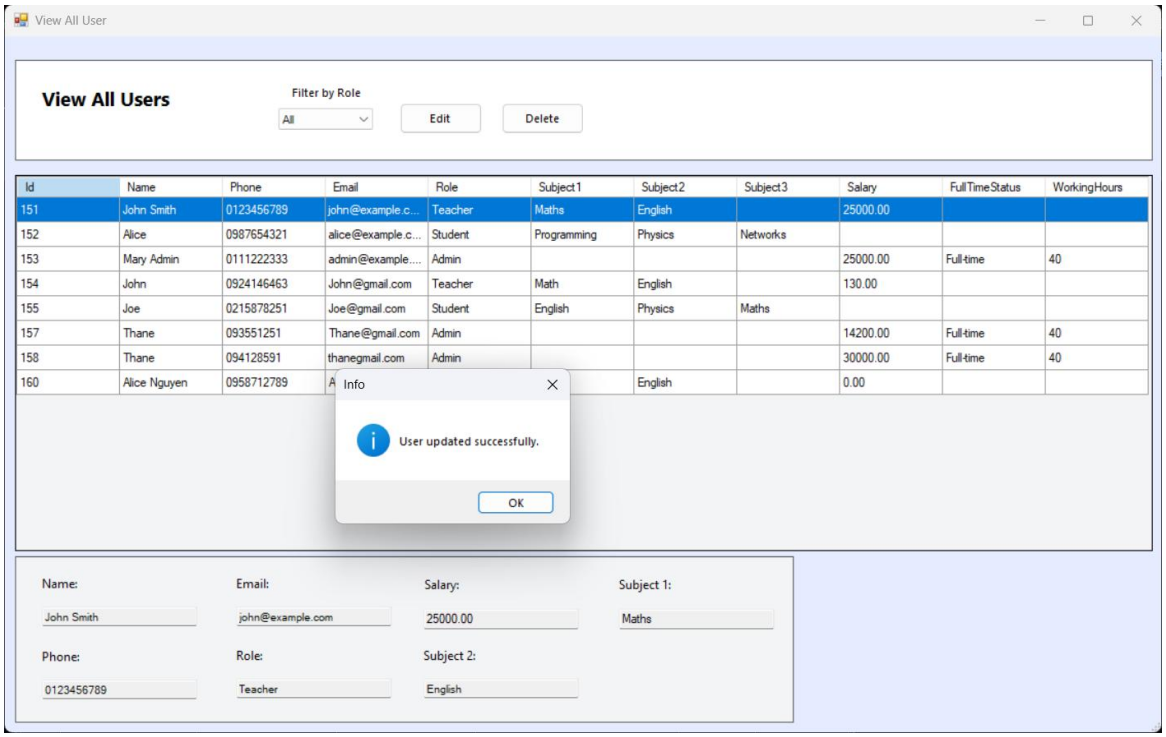


Figure 19

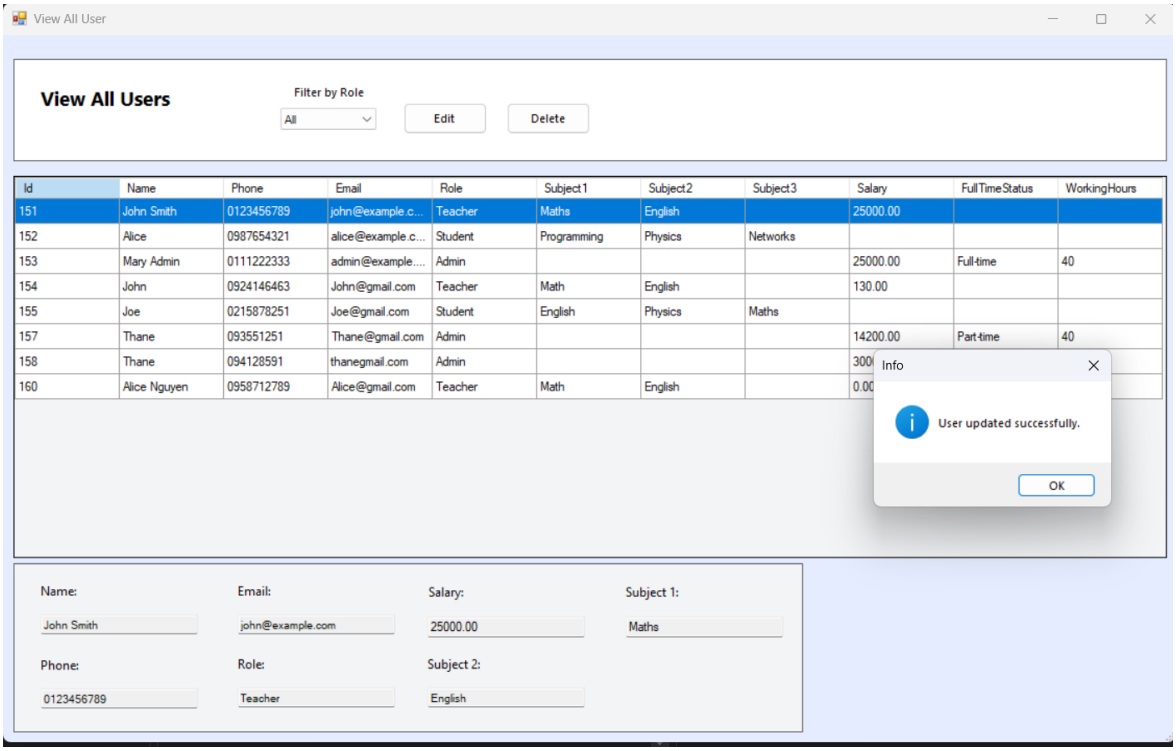


Figure 20

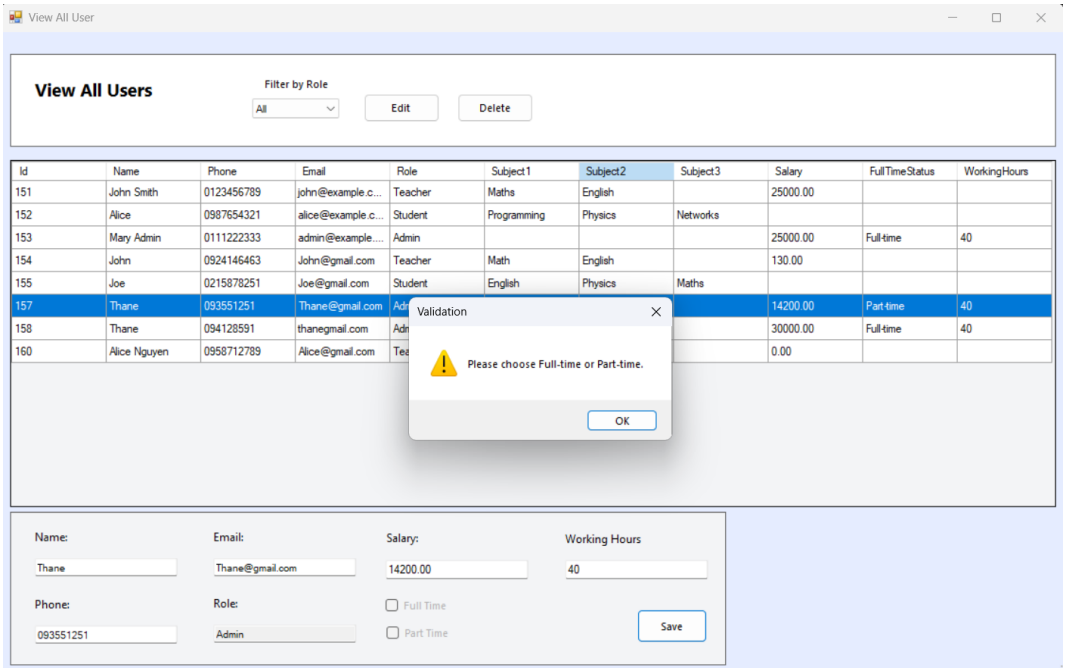


Figure 21

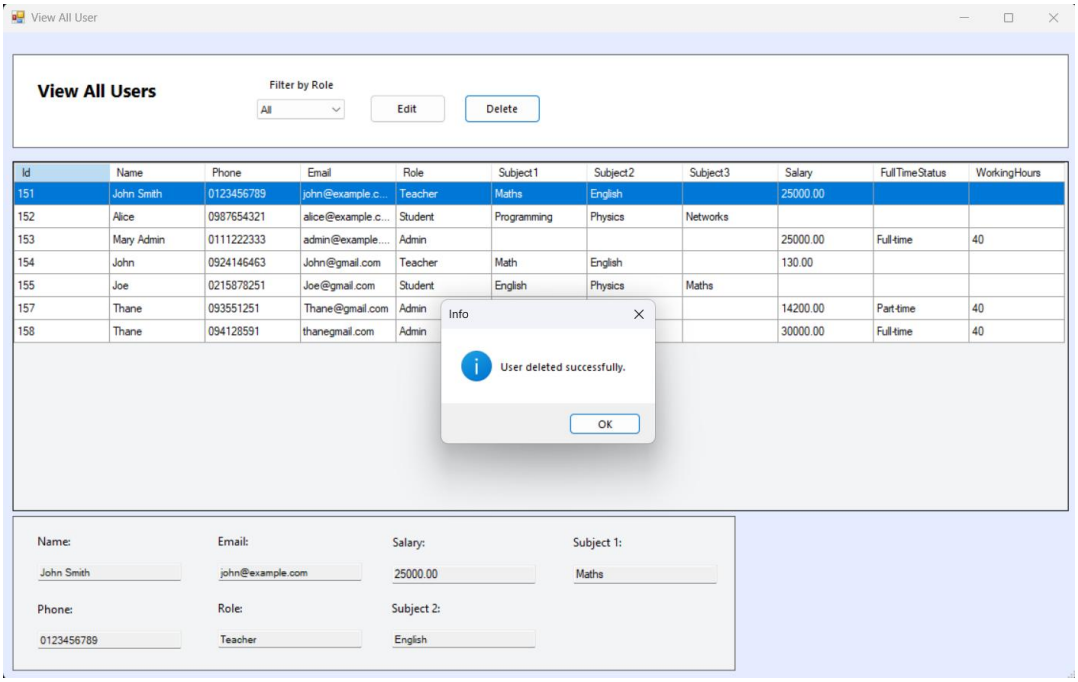


Figure 22

Appendix

To run the system, install **XAMPP** and start **Apache** and **MySQL**. Access the system at http://localhost and configure the database accordingly.

Then use this **SQL** file to set up the database.
https://drive.google.com/file/d/1s7fHhDIPr-iazkKxp3L6gtr__QNMJDbk/view?usp=sharing

(this SQL file is also submitted to Modules as a zip file named COMP1551_001340129_prototype.zip along with Program.cs, the pdf file containing this report)