Introduction

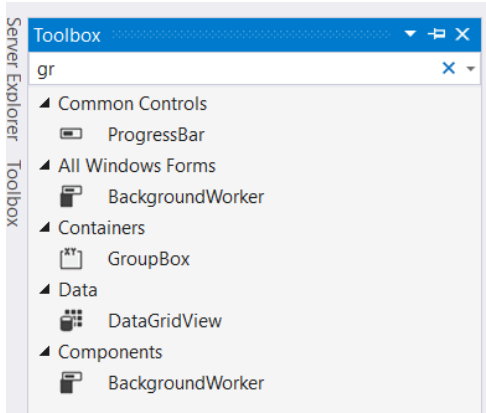
After a week of rigorous coding, Welcome back!

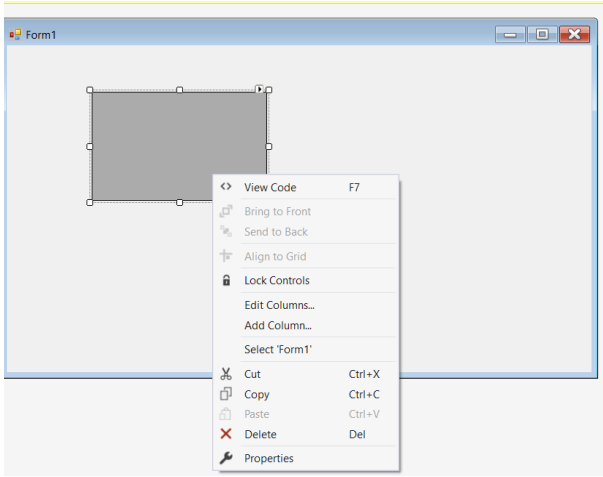
You have learned all about Event-Driven Programming and Windows Forms in the previous lab manuals. Let's move on to the next, new, and exciting concepts.

In contrast to Object-Oriented Programming, students have another kind of programming paradigm known as **Event-Driven Programming**. Event-driven programming is a programming paradigm in which the flow of program execution is determined by events - for example, a user action such as a mouse click, keypress, or a message from the operating system or another program.

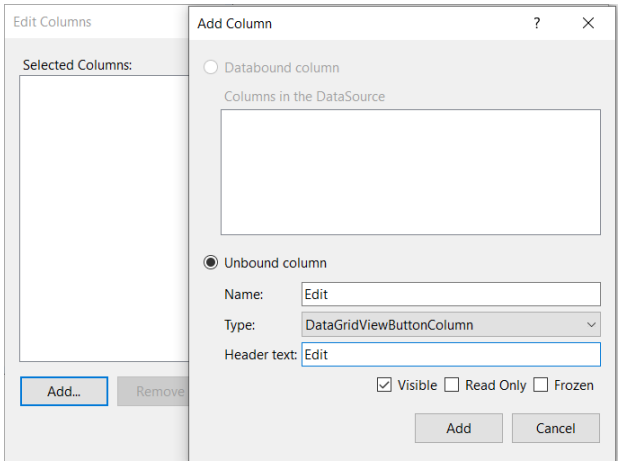
In this Lab, we will learn about the **Grid View** Control component of the forms by incorporating the existing knowledge that we have learned so far.

Creating a Grid View

	Go to toolbox >> **DataGridView**

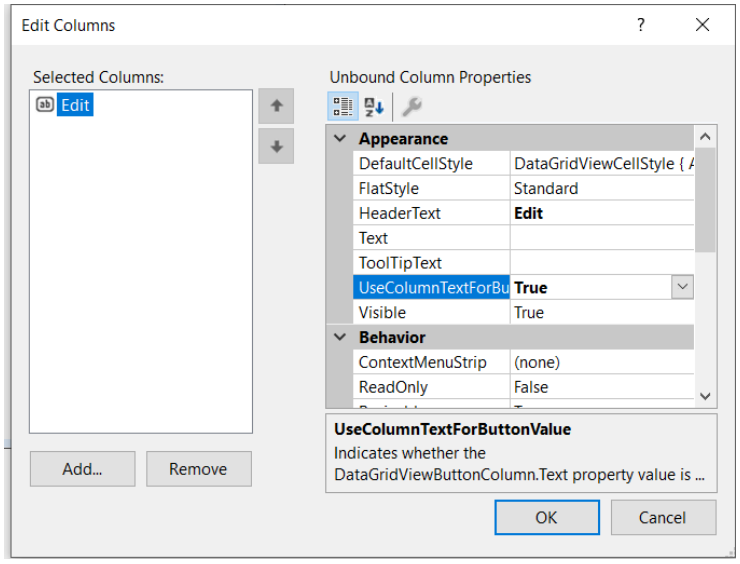


Right Click on the **GridView** and select **Edit Columns.**

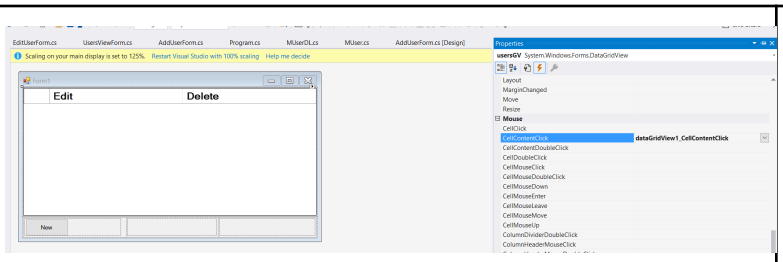
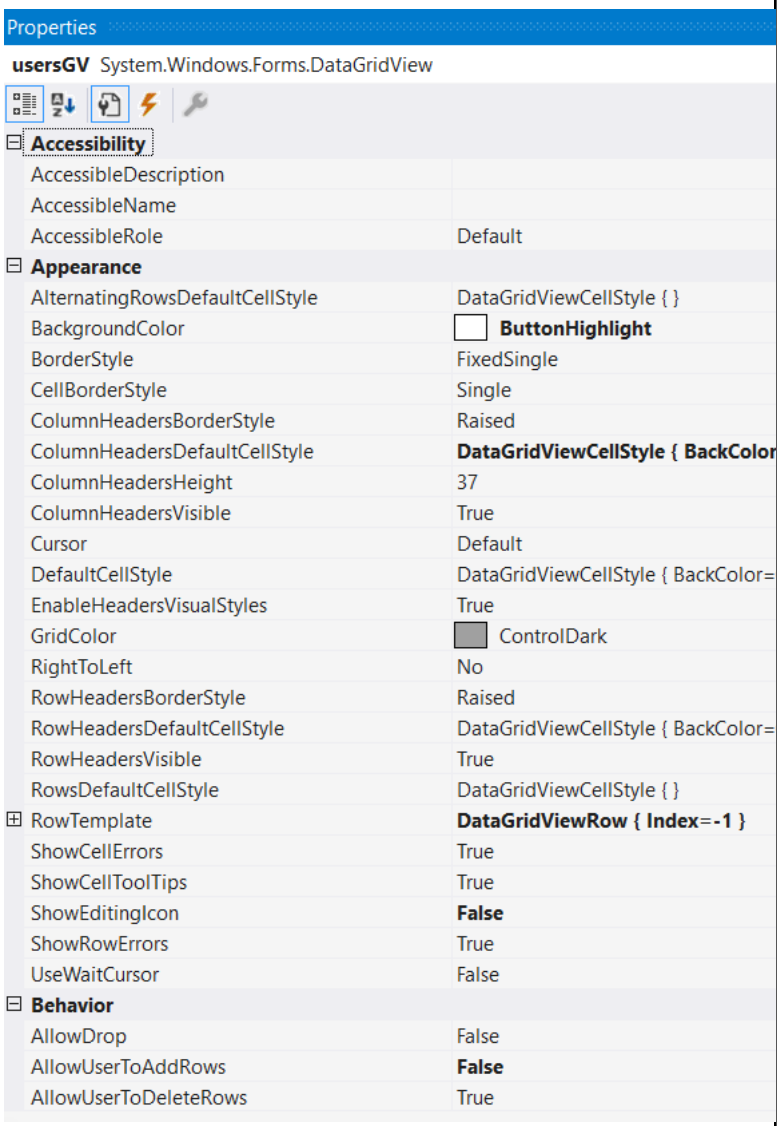


Click on the **Add** button.

- Set the name f the column
- Select the type of the column
- Set the Text of the Header for the Column



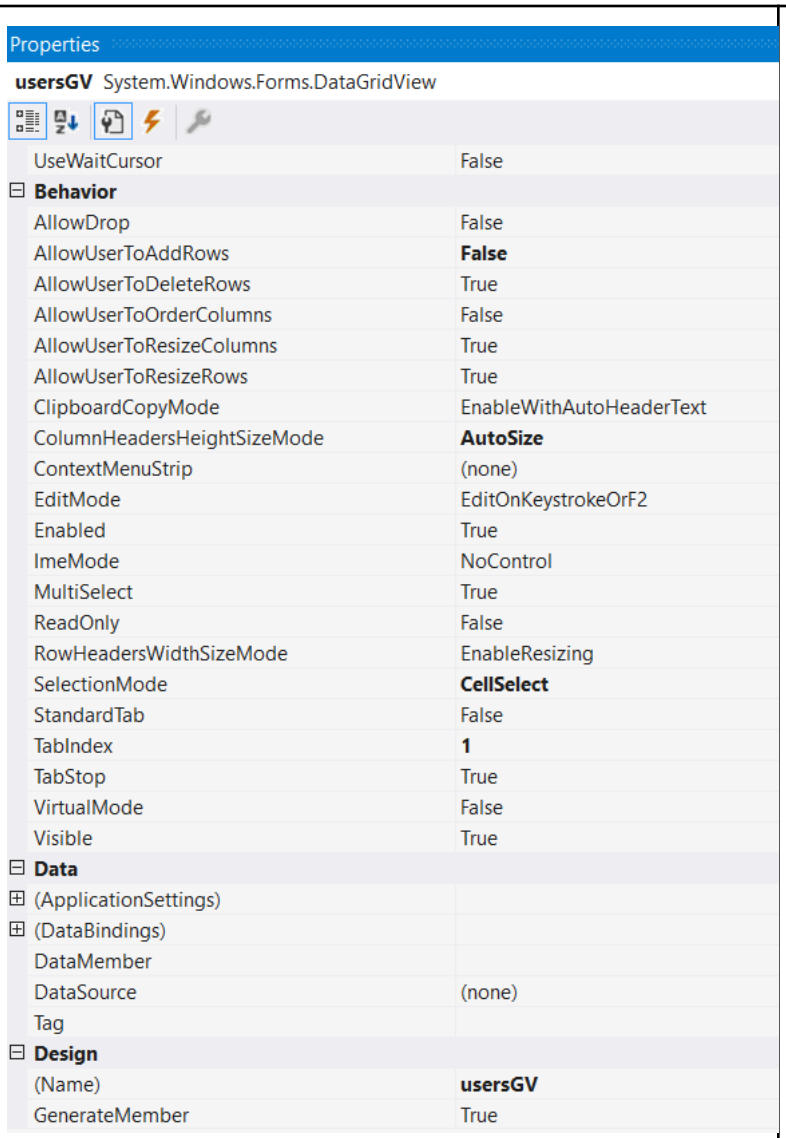
Set the remaining properties through these windows and add the remaining columns in a similar manner.

	Set the Event that triggers every time the user clicks in any of the cells of the grid.
	Set the Properties.

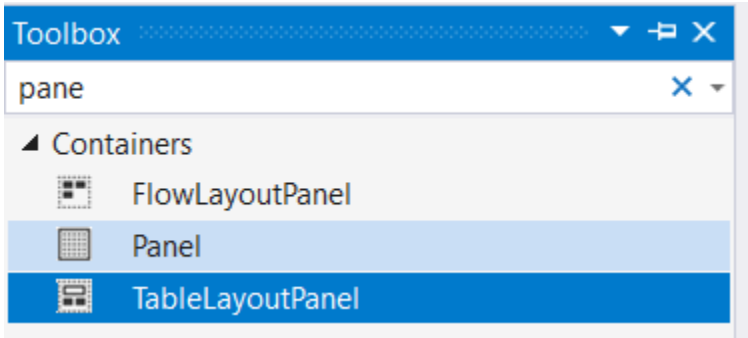
Properties

usersGV System.Windows.Forms.DataGridView

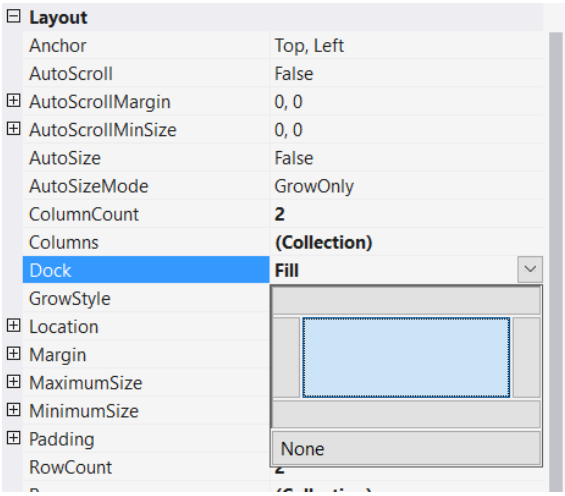
Accessibility	
AccessibleDescription	
AccessibleName	
AccessibleRole	Default
Appearance	
AlternatingRowsDefaultCellStyle	DataGridViewCellStyle { }
BackgroundColor	**ButtonHighlight**
BorderStyle	FixedSingle
CellBorderStyle	Single
ColumnHeadersBorderStyle	Raised
ColumnHeadersDefaultCellStyle	**DataGridViewCellStyle { BackColor**
ColumnHeadersHeight	37
ColumnHeadersVisible	True
Cursor	Default
DefaultCellStyle	DataGridViewCellStyle { BackColor=
EnableHeadersVisualStyles	True
GridColor	ControlDark
RightToLeft	No
RowHeadersBorderStyle	Raised
RowHeadersDefaultCellStyle	DataGridViewCellStyle { BackColor=
RowHeadersVisible	True
RowsDefaultCellStyle	DataGridViewCellStyle { }
RowTemplate	**DataGridViewRow { Index=-1 }**
ShowCellErrors	True
ShowCellToolTips	True
ShowEditingIcon	**False**
ShowRowErrors	True
UseWaitCursor	False
Behavior	
AllowDrop	False
AllowUserToAddRows	**False**
AllowUserToDeleteRows	True



In addition, We can use the Layout Panel for making the desktop application responsive.

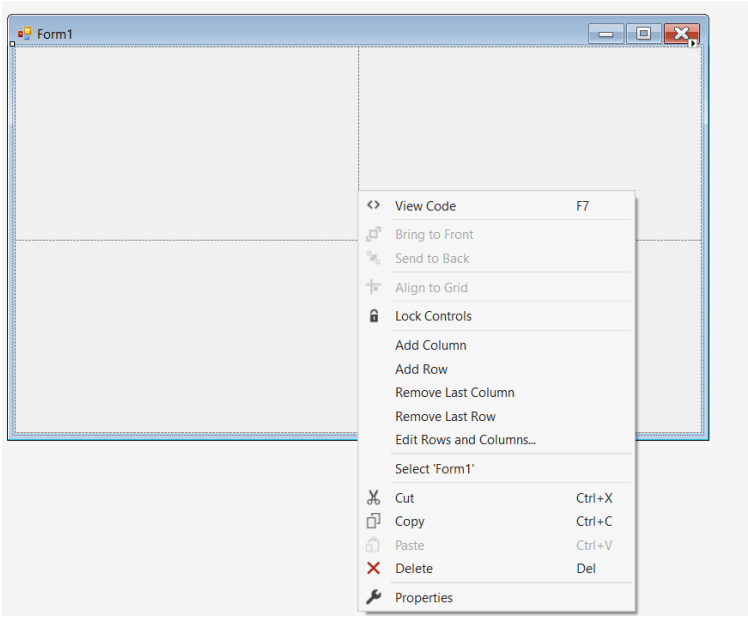


Go to **Toolbox** >> **TableLayoutPanel**



Go to the properties of the panel and set the **Dock** as **Fill**

This will set the Panel on the Complete Screen and the whole display would be divided into four sections.



To add more columns or rows to the panel you can right-click on the panel and select the desired operation.

Drag the borders to adjust the size.

Note: Now you can insert the control components in the panel and set their Dock property accordingly.

Food for Thought:
What if we need to add more than one control component into a single cell of the layout?

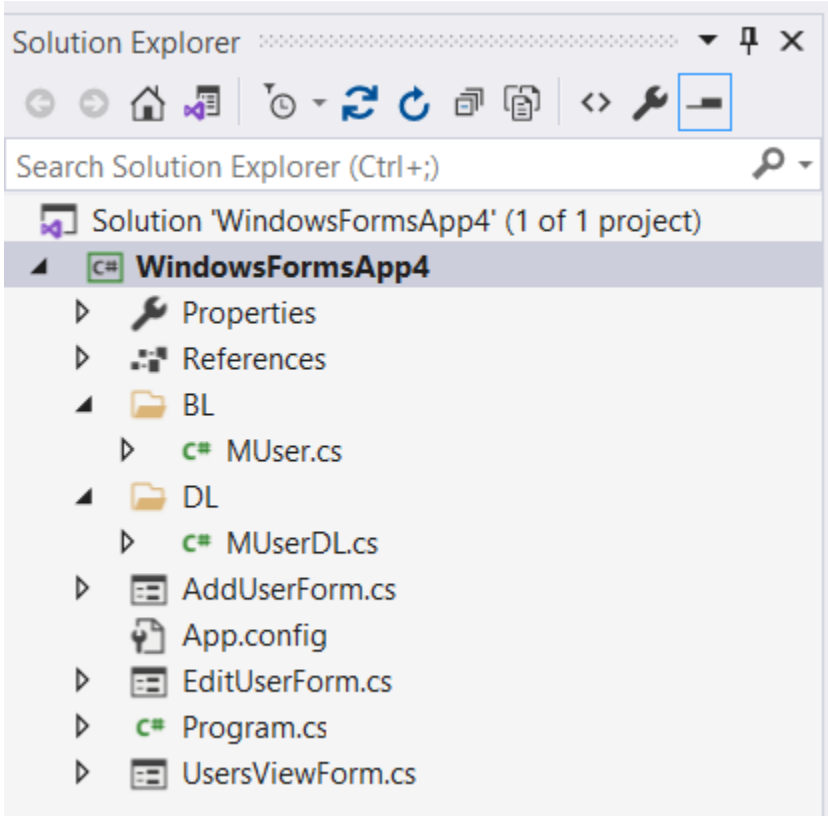
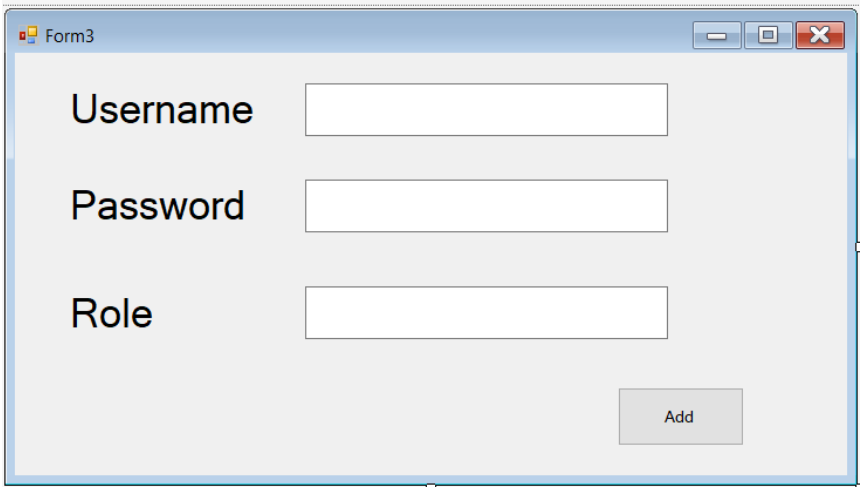
Congratulations !!! You have learned how to implement a grid view and Panel Layout in forms.
Let's jump right into it.

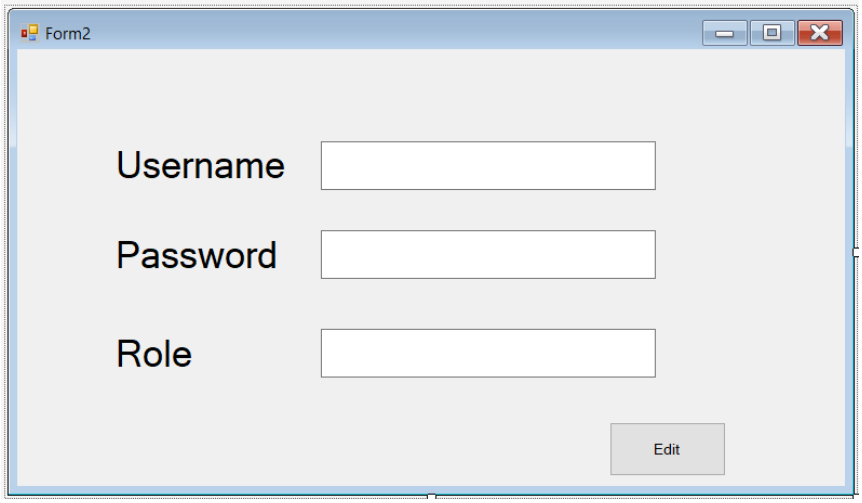
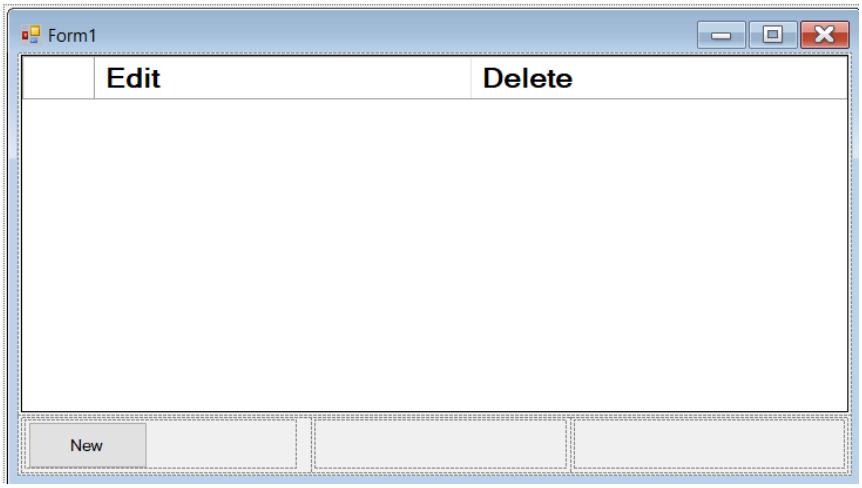
Task 01:

Object Oriented Programming

Lab Manual 10

Create the signInSignUp application by using the Data Grid View.

Sr #	Description	Snapshot
1.	Create three forms by using the same method that we used to create new classes.	
2.	Create the "**addUserForm**" - Insert and Update the labels - Insert and Update the text boxes - Insert and Update the button	

3.	Create the "**EditUserForm**" . - Insert and Update the labels - Insert and Update the text boxes - Insert and Update the button	
4.	Create the "**userViewForm**" - Insert and Update the GridView.	

Note: To rename a control component, it is recommended to Use the prefixes of txt for textboxes, lbl for labels, chk for checkboxes, gv for grid view, cmd for buttons, etc.

Let's Implement the back-end functionality of these forms now.

5.	Define the MUser Class (BL) - Define Constructors - Define getters functions. - Define isAdmin function.	

```
25 references
public class MUser
{
    private string userName;
    private string userPassword;
    private string userRole;

    13 references
    public string UserName { get => userName; set => userName = value; }
    13 references
    public string UserPassword { get => userPassword; set => userPassword = value; }
    9 references
    public string UserRole { get => userRole; set => userRole = value; }

    3 references
    public MUser(string userName, string userPassword, string userRole)
    {
        this.UserName = userName;
        this.UserPassword = userPassword;
        this.UserRole = userRole;
    }

    0 references
    public MUser(string userName, string userPassword)
    {
        this.UserName = userName;
        this.UserPassword = userPassword;
        this.UserRole = "NA";
        this.UserRole = userRole;
    }

    0 references
    public bool isAdmin()
    {
        if (UserRole == "Admin")
        {
            return true;
        }
        else
        {
            return false;
        }
    }
```

6. Define the MUser Class (DL)
   - Define the associated functions.

```
class MUserDL
{
    private static List<MUser> usersList = new List<MUser>();

    internal static List<MUser> UsersList { get => usersList; set => usersList = value; }

    public static void addUserIntoList(MUser user)
    {
        UsersList.Add(user);
    }
```

```
public static MUser SignIn(MUser user)
{
    foreach (MUser storedUser in UsersList)
    {
        if (storedUser.UserName == user.UserName && storedUser.UserPassword == user.UserPassword)
        {
            return storedUser;
        }
    }
    return null;
}
```

```
public static string parseData(string record, int field)
{
    int comma = 1;
    string item = "";
    for (int x = 0; x < record.Length; x++)
    {
        if (record[x] == ',')
        {
            comma++;
        }
        else if (comma == field)
        {
            item = item + record[x];
        }
    }
    return item;
}
```

```
public static bool readDataFromFile(string path)
{
    if (File.Exists(path))
    {
        StreamReader fileVariable = new StreamReader(path);
        string record;
        while ((record = fileVariable.ReadLine()) != null)
        {
            string userName = parseData(record, 1);
            string userPassword = parseData(record, 2);
            string userRole = parseData(record, 3);
            MUser user = new MUser(userName, userPassword, userRole);
            addUserIntoList(user);
        }
        fileVariable.Close();
        return true;
    }
    else
        return false;
}
```

```
public static void storeUserIntoFile(MUser user, string path)
{
    StreamWriter file = new StreamWriter(path, true);
    file.WriteLine(user.getUserName() + "," + user.getUserPassword() + "," + user.getUserRole());
    file.Flush();
    file.Close();
}
```

```
1 reference
public static void deleteUserFromList(MUser user)
{
    for (int index = 0; index < usersList.Count; index++)
    {
        if (usersList[index].UserName == user.UserName && usersList[index].UserPassword == user.UserPassword)
        {
            usersList.RemoveAt(index);
        }
    }

}

1 reference
public static void EditUserFromList(MUser previous, MUser updated)
{
    foreach (MUser user in usersList)
    {
        if (user.UserName == previous.UserName && user.UserPassword == previous.UserPassword)
        {
            user.UserName = updated.UserName;
            user.UserPassword = updated.UserPassword;
            user.UserRole = updated.UserRole;
        }
    }

}

3 references
public static void storeAllDataIntoFile(string path)
{
    StreamWriter file = new StreamWriter(path);
    foreach (MUser storedUser in UsersList)
    {

        file.WriteLine(storedUser.UserName + "," + storedUser.UserPassword + "," + storedUser.UserRole);
    }
    file.Flush();
    file.Close();
}
```

Let us now implement the codes for different events for each form.

7. Creates the
   - **userViewForm**
   - Read the data from the file as soon as the form starts.
   - Additionally, provide the functionality in case of **click event** is triggered.

```
3 references
public partial class UsersViewForm : Form
{
    private string path = "data.txt";
    1 reference
    public UsersViewForm()
    {
        InitializeComponent();
    }
    1 reference
    private void Form1_Load(object sender, EventArgs e)
    {
        MUserDL.readDataFromFile("data.txt");
        usersGV.DataSource = MUserDL.UsersList; // introspection
    }

3 references
public void dataBind()
{
    usersGV.DataSource = null;
    usersGV.DataSource = MUserDL.UsersList;
    usersGV.Refresh();
}
1 reference
private void button3_Click(object sender, EventArgs e)
{
    AddUserForm myform = new AddUserForm();
    myform.ShowDialog();          //Show dialog does not allow to edit the previous form
    MUserDL.storeAllDataIntoFile(path);
    dataBind();
}
```

		1 reference
		`private void dataGridView1_CellContentClick(object sender, DataGridViewCellEventArgs e)`
		`{`
		`MUser user = (MUser)usersGV.CurrentRow.DataBoundItem;`
		`if (usersGV.Columns["Delete"].Index == e.ColumnIndex)`
		`{`
		`MUserDL.deleteUserFromList(user);`
		`MUserDL.storeAllDataIntoFile(path);`
		`dataBind();`
		`}`
		`else if (usersGV.Columns["Edit"].Index == e.ColumnIndex)`
		`{`
		`EditUserForm myform = new EditUserForm(user);`
		`myform.ShowDialog();`
		`MUserDL.storeAllDataIntoFile(path);`
		`dataBind();`
		`}`
		`}`

8.	Creates the
	- **editUserForm**
	- Read the data from the file as soon as the form starts.
	- Additionally, provide the functionality in case of **click event** is triggered.
		4 references
		`public partial class EditUserForm : Form`
		`{`
		`private MUser previous;`
		1 reference
		`public EditUserForm(MUser previous)`
		`{`
		`InitializeComponent();`
		`this.previous = previous;`
		`}`
		1 reference
		`private void Form2_Load(object sender, EventArgs e)`
		`{`
		`txtUsername.Text = previous.UserName;`
		`txtPassword.Text = previous.UserPassword;`
		`txtRole.Text = previous.UserRole;`
		`}`
		
		1 reference
		`private void button1_Click(object sender, EventArgs e)`
		`{`
		`MUser updated = new MUser(txtUsername.Text, txtPassword.Text, txtRole.Text);`
		`MUserDL.EditUserFromList(previous, updated);`
		`this.Close();`
		`}`

	Creates the
	- **editUserForm**
	- Read the data from the file as soon as the form starts.
	- Additionally, provide the functionality in case of **click event** is triggered.
		4 references
		`public partial class AddUserForm : Form`
		`{`
		1 reference
		`public AddUserForm()`
		`{`
		`InitializeComponent();`
		`}`
		
		1 reference
		`private void button1_Click(object sender, EventArgs e)`
		`{`
		`MUser user = new MUser(txtUsername.Text, txtPassword.Text, txtRole.Text);`
		`MUserDL.addUserIntoList(user);`
		`this.Close();`
		`}`

Congratulations !!!! you have the signInSignUp project by using the windows forms and GridViews.

Good Luck and Best Wishes !!

Happy Coding ahead :)